Capitolul 1.
Noțiuni de bază

Capitolul este destinat în principal prezentării unor elemente introductive, absolut necesare pentru păstrarea caracterului de sine stătător al lucrării, în Liceu anumite noțiuni deosebit de importante fiind predate destul de diferit.

1. Noțiuni de bază în Informatică

Începem cu o scurtă trecere în revistă a câtorva concepte care vor fi utilizate intensiv în carte în special ca suport teoretic pentru exemplele alese. Pentru detalii se mai pot consulta <2>, <3>, <10>, <19>, <20>, <31>, <36>, <38>.

1.1. Predarea unor noțiuni fundamentale

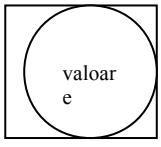
Temele şi domeniile abordate în tratarea disciplinelor de Informatică sunt desigur stabilite prin *obiectivele cadru şi de referință* specifice. Dar, aşa cum nu putem aborda nici un domeniu al matematicii (de exemplu), fără cunoaşterea unor noțiuni fundamentale (cum ar fi cele privind teoria mulțimilor, teoria numerelor etc.), nici în Informatică nu ne putem dispensa de conceptul de **algoritm**. *Prin algoritm (**imperativ**) se înțelege ansamblul de transformări (metode) ce se aplică asupra unui set de date de **intrare** şi care determină obținerea într-un timp finit şi după o succesiune precisă de paşi, a unui set de date de **ieşire*** (<14>, <23>, <24>). Aceasta nu este o definiție, ci o descriere a unui concept *de bază*. Spre deosebire de matematica clasică (în care noțiunile de bază, nedefinite ci doar descrise, sunt relativ simple: *mulțime, punct, plan* etc.), noțiunile informatice similare sunt mult mai complicate (în afară de *algoritm*, mai amintim: *bază de date, program concurent*, *site*, *cip*, etc.). Un accent deosebit trebuie pus pe caracteristicile algoritmilor: *generalitatea (universalitatea), determinismul şi finitudinea, eficacitatea* (<23>, <32>). Să precizăm totuşi că introducerea oricărei noțiuni (chiar nefundamentale), ar trebui să urmeze următoarele etape:

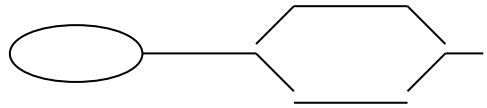
- *Etapa de elaborare şi motivație* (inițială). Fundamentată şi eficient integrată într-un sistem, o noțiune *cere* noi domenii de aplicare. Prin urmare atrage după sine

(motivează) introducerea unor noi noțiuni sau furnizarea unor noi rezultate, până când aria de extindere se îngustează.

- *Etapa de formare* a noțiunii. Ilustrată prin exemple, argumentată teoretic şi, de dorit, *demonstrată* matematic, o noțiune se constituie ca un util şi puternic mijloc de *producție* pentru domeniul pentru care a fost elaborată. Rămâne doar să-l *exploatăm* adecvat. Didactic, acest aspect cuprinde argumentarea ştiințifică a noțiunii introduse şi reliefarea unor noi, posibile domenii de aplicabilitate.
- *Etapa de consolidare,* prin operare cu noțiunea. O noțiune poate fi considerată asimilată dacă ea devine şi instrument de dobândire a unor cunoştințe şi dacă elevii pot opera cu această noțiune în situații noi.

De exemplu, în privința reprezentării algoritmilor, optăm pentru folosirea pseudocodului sau a altor tipuri de „diagrame". Nici un efort metodic nu este prea mare pentru a avea o reuşită deplină în înțelegerea şi abordarea noțiunilor de algoritm şi de reprezentare a acesteia. Noțiunile ulterior introduse vor apare în mod firesc, căpătând caracteristicile unor înlănțuiri cauzale. De aceea este necesar ca în mintea elevilor să existe o *ordonare* a noțiunilor, o corelare firească a lor, o motivație, pentru că numai peste cunoştințe bine asimilate se pot aşterne în mod eficient cunoştințe noi. Pentru a-l cita pe Domnul profesor I. Maxim, elevul trebuie să înțeleagă că ordinea în care se predau noțiunile nu este întâmplătoare şi că el trebuie să facă un efort de asimilare, care va fi răsplătit prin reuşite viitoare. Unele teme de predare pot fi organizate în *spirală* (ceea ce presupune o reîntoarcere la acelaşi conținut, dar pe un nivel superior). Acest mod de planificare corespunde sistemului *concentric propriu-zis* (*concentric calitativ)* şi sistemului *concentric cantitativ* (*concentric liniar)*. *Sistemul concentric calitativ* desemnează modul de organizare a cunoştințelor în programele de învățământ, manuale şi lecții, în aşa fel încât noțiunile (cunoştințele) se însuşesc prin reluări, restructurări, reinterpretări, până la formarea lor completă. *Sistemul concentric cantitativ* este modul de organizare a cunoştințelor în programele şcolare, manuale şi lecții (inclusiv pe **INTERNET**), constând în reluarea adăugită şi detaliată a materiei parcurse anterior, reluare reclamată nu atât de dificultatea înțelegerii noțiunilor, cât mai ales de nevoia lărgirii cunoştințelor în succesiunea claselor şi treptelor şcolare. Trebuie astfel făcută diferența dintre noțiunea de ***variabilă***, aşa cum este ea cunoscută din matematica clasică şi cea de variabilă în sensul limbajelor de programare imperative (D. Barron, <7>), noțiune care poate fi reprezentată ca (de unde poate rezulta şi interpretarea corectă a asignării) (?????de reparat):





Nume Atribute Referinţă

Intuitiv vorbind, pentru a parcurge drumul de la realitatea de modelat la implementarea pe calculator, trebuie înţelese, cel puţin la nivelul descriptiv, şi alte noţiuni, cum ar fi cele de *problemă, complexitate, corectitudine/verificare*, etc. O **problemă** este un concept caracterizat prin *enunţ*, mulţime de informaţii de intrare (*instanţe* ale problemei), mulţime de informaţii de ieşire (*răspunsuri* ale problemei). Ca urmare, rezolvarea unei probleme înseamnă că pentru fiecare instanţă trebuie să se furnizeze (într-un timp finit) un anumit răspuns. Dacă acest răspuns este doar de tipul *DA* sau *NU*, atunci avem de-a face cu o **problemă de decizie**. Soluţia adoptată pentru această a treia cale de descriere a unei mulţimi are avantajul de a avea şi o caracteristică de natură *(semi)algoritmică*. Acceptăm astfel ***paradigma imperativă*** propusă de D. Knuth (<23>), *Algoritm = Date + Operaţii*. Mai exact, un ***algoritm (imperativ)*** *reprezintă o secvenţă finită de paşi (instrucţiuni), care descriu operaţii precise asupra unor informaţii (date) iniţiale (de intrare) sau intermediare (de lucru, temporare), în vederea obţinerii unor informaţii (rezultate) finale (de ieşire)*. Paşii ***se execută*** *(operaţiile se efectuează în mod concret) în ordinea scrierii lor în secvenţă*. Un algoritm **calculează o funcţie** sau **rezolvă o problemă**. Intuitiv, datele de intrare reprezintă elemente din domeniul de definiţie al funcţiei de calculat (sau *informaţiile iniţiale* din realitatea în care îşi are originea problema pe care vrem să o rezolvăm), iar datele de ieşire sunt elemente din codomeniul funcţiei (respectiv, *soluţiile* problemei). Un algoritm **se termină** *pentru toate intrările admise, prin urmare* ***există întotdeauna un ultim pas****, a cărui execuţie marchează de obicei şi obţinerea rezultatelor de ieşire*. Din motive tehnice, vom lua uneori în considerare şi *algoritmi care nu se termină pentru toate intrările*, pe care-i vom numi **semialgoritmi (proceduri)**. Un (semi)algoritm poate fi descris sub mai multe forme, printre care se numără şi **pseudocodul** (limbaj intermediar între limbajul natural şi un limbaj de programare comercial). Prin urmare, **algoritmul** *Alg* **rezolvă problema P,** dacă având la intrare orice instanţă a problemei, acesta se termină având ca rezultat un element din mulţimea de răspuns. Există şi **probleme semirezolvabile**. Diferenţa faţă de problemele rezolvabile este aceea că algoritmul care le rezolvă poate să nu se termine pentru fiecare instanţă. Există de asemenea şi **probleme nerezolvabile (nedecidabile)**, cu alte cuvinte probleme pentru care nu există algoritmi care să le rezolve. În limbajul curent a intrat şi termenul de **problemă netratabilă**, pentru a desemna

o problemă rezolvabilă, dar într-un timp practic inaccesibil (*exponențial* sau mai mare). Astfel, două dintre *măsurile (teoretice, globale) de complexitate* des întrebuințate sunt *complexitatea timp* şi *complexitatea spațiu*. Ideea este aceea că un (orice) pas elementar (instrucțiune) al (a) unui algoritm se execută într-*o unitate de timp* (pentru spațiu, *fiecare dată elementară* se memorează într-*un registru* sau *locație de memorie*, acesta/aceasta ocupând *o unitate de spațiu*), criteriul numindu-se *al costurilor uniforme*. Există şi *criteriul costurilor logaritmice*, în care orice informație de *lungime* i, se prelucrează (respectiv, se memorează) în numărul de unități de timp (unități de spațiu) egal cu $\lfloor \log(i) \rfloor + 1$ (dacă i = 0, se convine să luăm log(i) = 0; $\lfloor\ n \rfloor$ notează *partea întreagă inferioară* a numărului n). Intuitiv, timpul luat de execuția unui algoritm *Alg* este dat de *numărul de instrucțiuni (paşi/operații elementare) efectuate* (să-l notăm cu t^{Alg}), iar spațiul (notat cu s^{Alg}) este dat de *numărul de locații (elementare) de memorie (internă, a calculatorului) ocupate* în cursul execuției. Sigur că *totul se raportează la lungimea* n_F *a fiecărei intrări* $F \in \mathbf{IN}$ şi ne interesează de fapt $\sup\{t^{Alg}(F) \mid F \in \mathbf{IN}$ şi $n_F = n \in \mathbf{N}\}$, margine superioară pe care o vom nota cu $t^{Alg}(n)$ (respectiv $s^{Alg}(n)$). Această abordare (în care se caută *cazul cel mai nefavorabil*), ne permite să fim siguri că pentru fiecare intrare de lungime n, timpul de execuție al lui *Alg* nu va depăşi $t^{Alg}(n)$. Cum determinarea acelui supremum este de multe ori destul de dificilă, ne vom mulțumi să studiem aşa-numita *comportare asimptotică* (sau *ordinul de creştere*) a (al) lui $t^{Alg}(n)$, adică ne vor interesa doar anumite margini ale sale, cum ar fi marginea sa superioară. Formal, pentru fiecare $f : \mathbf{N} \rightarrow \mathbf{N}$, notăm

$O(f) = \{g \mid g : \mathbf{N} \rightarrow \mathbf{N}$, *există* $c \in \mathbf{R}$, $c > 0$ *şi există* $k \in \mathbf{N}$, *astfel încât pentru fiecare* $n \geq k$ *avem* $g(n) \leq c \bullet f(n)\}$ şi vom spune ***că fiecare*** $g \in O(f)$***, este de ordinul lui*** f, ceea ce se mai notează şi cu $g = O(f)$. Astfel, există probleme care au *complexitatea (timp, asimptotică)* $O(2^n)$, sau, pe scurt, complexitate exponențială, deoarece există (măcar) un algoritm *Alg* care rezolvă problema şi pentru care $t^{Alg}(n) = O(2^n)$. Similar, vom vorbi de *algoritmi polinomiali* ($t^{Alg}(n) = O(p(n))$, unde p(n) desemnează un polinom în n, de orice grad), sau de *algoritmi liniari* (p(n) de mai sus este un polinom de gradul I). Pentru detalii pot fi consultate <1>, <8>, <14>, <16>, <26>, <37> (vom reveni şi noi prin câteva exemple în ultima secțiune a acestui capitol). După cum am mai precizat, pentru că noțiunea de algoritm este dată printr-o *descriere* şi nu prin utilizarea *genului proxim* şi a *diferenței specifice* (în sensul logicii aristotelice clasice, ca subdisciplină a *Filozofiei*), avem mai întâi nevoie de *metode de reprezentare* a algoritmilor. O primă formă de reprezentare este desigur ***limbajul natural***. O

altă formă de reprezentare a algoritmilor este ***limbajul pseudocod***. Limbajul pseudocod, față de limbajul natural, este o formă de reprezentare mai exactă, permițându-se în plus orice nivel de detaliere. Nu există un limbaj pseudocod *standard* care să permită reprezentarea convenabilă a tuturor algoritmilor, forma unui asemenea limbaj putând fi influențată chiar de limbajul de programare în care urmează a fi implementat algoritmul. Anumite instrucțiuni şi structuri de informație (<4>, <7>), nu lipsesc de obicei din nici un limbaj:

- O mulțime de *operații elementare*: **atribuirea** unei valori pentru o variabilă „internă”; **citirea** unei valori pentru o variabilă (aceasta fiind tot o atribuire, de un tip mai special); **scrierea** valorii curente a unei variabile „în exterior”.
- O mulțime de *structuri de control*: **structura secvențială**; **structura alternativă**; **structurile** de tip **repetitiv**.
- Clase de structuri de date: numere, şiruri de caractere, tablouri, arbori, liste etc.

Un posibil limbaj pseudocod poate fi generat atunci pornind cu „instrucțiunile elementare”:
var := expresie (operația de atribuire a valorii expresiei din dreapta semnului „:=”, variabilei din stânga semnului „:=”; evaluarea unei expresii, indiferent de tipul acesteia este o operație de un nivel inferior celui elementar şi nu va fi luată în discuție); **citeşte var** (operația de introducere din exterior a unei valori şi atribuirea acesteia variabilei **var**); **scrie var** (operația de afişare în exterior a valorii curente a variabilei). Un **bloc** (Bloc, Bloc1, Bloc2 de mai jos) de operații va fi format dintr-o operație **elementară** de tipul celor enumerate mai sus, sau dintr-o **secvență** (succesiune) de blocuri (intuitiv, acestea se vor „executa” în ordinea textuală în care apar). Acum putem spune că o *structură de control alternativă* poate avea una dintre formele:

a) forma incompletă;

Dacă (condiție) **atunci**
    Bloc
Sfdacă

sau

b) forma completă

Dacă (condiție) **atunci**
    Bloc1
altfel
    Bloc2
Sfdacă

O *structură de control repetitivă* va fi:

a) cu test la intrarea în ciclu

Câttimp (condiție) **execută**

Bloc

Sfcâttimp

sau

b) cu test la ieşirea din ciclu

Repetă

Bloc

Pânăcând

(condiție)

Grafic:

- pentru reprezentarea unei operații de atribuire se va folosi

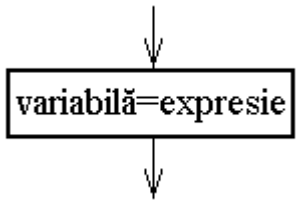


Figura 1

- pentru reprezentarea unei operații de citire se va folosi

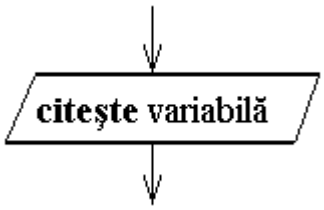


Figura 2

- pentru reprezentarea unei operații de scriere se va folosi

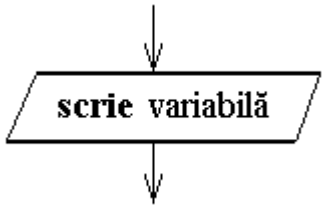


Figura 3

- pentru reprezentarea unei structuri secvențiale se va folosi

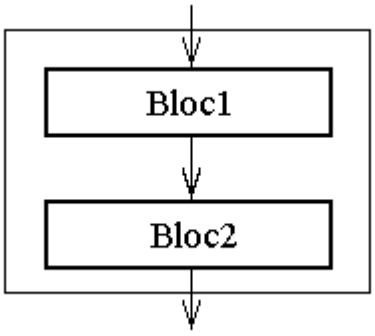


Figura 4

- pentru reprezentarea unei structuri de control alternative incomplete se va folosi

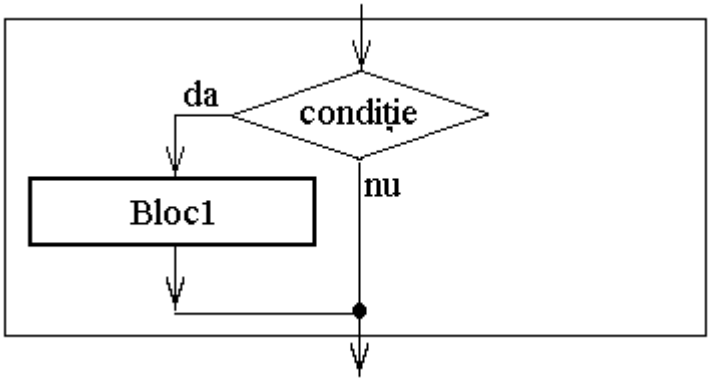


Figura 5

- unei structuri de control alternative complete se va folosi

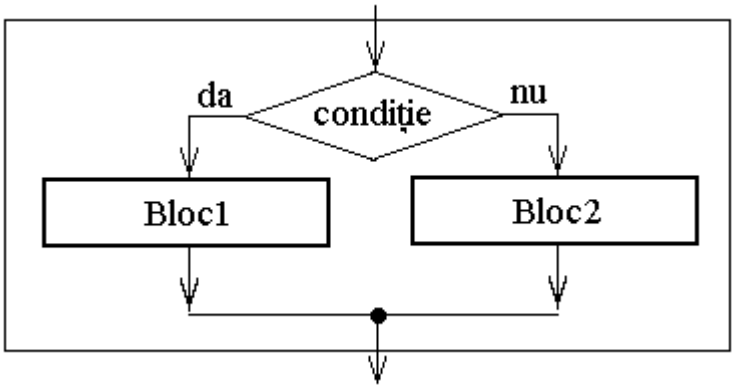


Figura 6

– pentru reprezentarea unei structuri de control repetitive cu test la intrarea în ciclu se va folosi

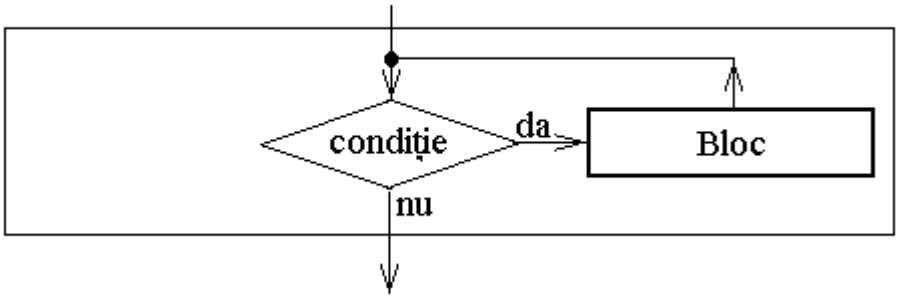


Figura 7

– pentru reprezentarea unei structuri repetitive cu test la ieşirea din ciclu avem

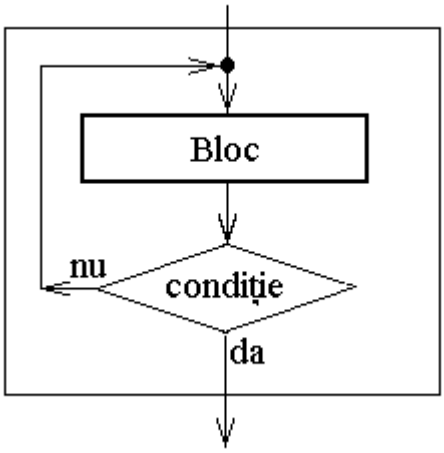


Figura 8

.

Se observă că orice operație sau structură reprezentată mai sus poate fi asimilată cu un bloc care are o singură intrare şi o singură ieşire. Prin urmare, chiar un algoritm, la nivelul cel mai redus de detaliu poate fi privit ca un bloc unic (*schemă logică*):

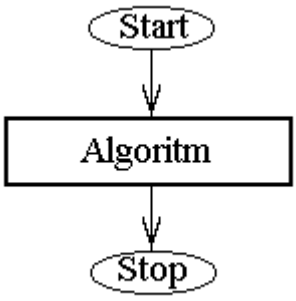


Figura 9

În 1966 (<9>) s-a demonstrat că *orice algoritm (imperativ) poate fi reprezentat folosind numai structurile de control: secvențială, alternativă şi repetitivă*. Rezultatul obținut a condus în acel moment la apariția unor noi viziuni de proiectare a algoritmilor, cum ar fi proiectarea modulară şi structurată. Din acelaşi motiv vom folosi pe parcursul lucrării, în caz că anumite confuzii pot fi evitate, şi alte instrucțiuni „cunoscute" sau limbaje pseudocod apropiate până la identificare de limbajele de programare comerciale. Fără a intra în detalii, următoarea schemă calculează cel mai mare divizor comun a două numere nenule (presupuse a fi naturale în mod implicit):

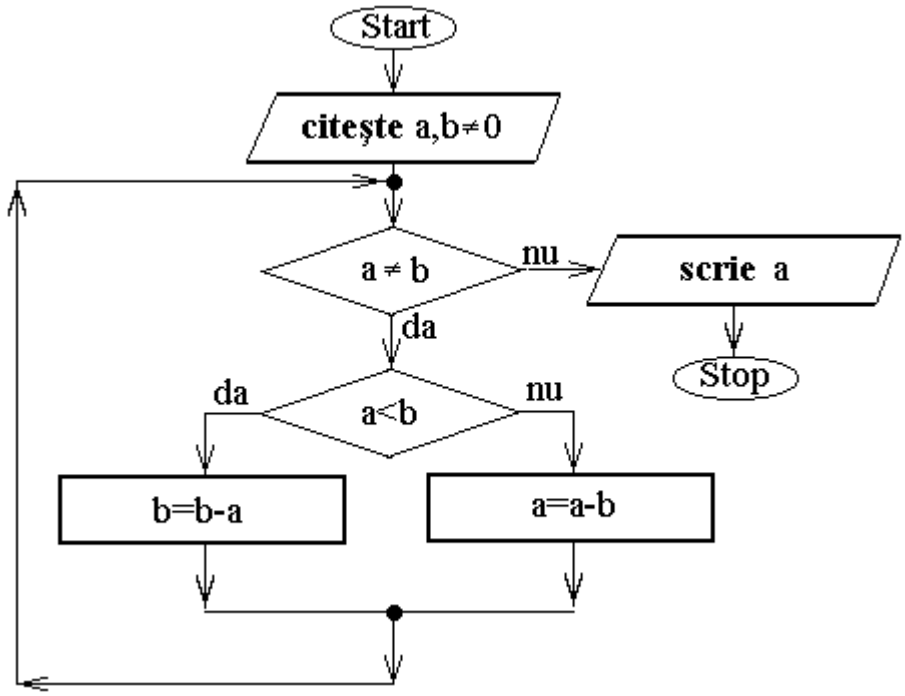


Figura 10.

Se observă că în orice algoritm rezultatul final este condiționat de datele inițiale şi, mai mult, că succesiunea în care se execută operațiile elementare depinde de datele de intrare şi de rezultatele intermediare obținute în urma execuțiilor anterioare. Datele inițiale, rezultatele intermediare şi deciziile luate în structurile de control alternative şi repetitive determină astfel o *traiectorie* (<31>, <32>) a execuției operațiilor (prelucrărilor), aceasta putând fi reprezentată printr-un graf orientat (*digraf*).

Pentru algoritmul anterior vom avea:

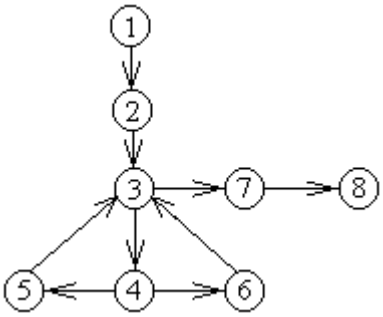


Figura 10.a.

Prin urmare, orice traiectorie de prelucrări induce în digraful asociat algoritmului, *un drum* de la nodul inițial (etichetat cu 1) asociat primei operații din algoritm (**Start-Început**), la nodul final (etichetat cu 8) asociat ultimei operații din algoritm (**Stop-Sfârşit**).

1.2. Metode de elaborare (proiectare) a algoritmilor

Elaborarea unui (nou) algoritm pentru rezolvarea unei (clase de) probleme a constituit mult timp o formă de manifestare a inteligenței, o exprimare a capacității de sinteză şi analiză, a bagajului de cunoştințe şi experiență ale celui care îl elabora punându-se în evidență caracterul de creativitate, de artă chiar a acestei activități. Reuşitei standardizării reprezentării algoritmilor i s-a alăturat dorința de standardizare a elaborării algoritmilor. Cu toate succesele obținute în acest sens, activitatea de elaborare a algoritmilor beneficiază încă de o doză substanțială de libertate de exprimare a experienței şi creativității. Primele metode de elaborare a algoritmilor au avut perioade mai lungi sau mai scurte de *priză la mase*, dar o analiză atentă a eficienței (complexității) algoritmilor elaborați au etalat avantaje şi neajunsuri, care au condus la o ierarhizare a acestor metode. În cele ce urmează, vom prezenta succint cele mai utilizate metode de elaborare a algoritmilor. Pentru alte detalii se pot consulta <17, 19, 20, 23, 24, 25>.

1.2.1. Metoda *divide et impera*

Metoda *„împarte şi stăpâneşte”*, a fost sugerată de ideea firească de rezolvare a unei probleme complexe prin divizarea acesteia în două sau mai multe subprobleme de acelaşi tip cu cea inițială, mai simple, prin rezolvarea cărora (folosind soluțiile deja obținute), se permite obținerea soluției problemei inițiale. Această *divizare* poate fi aplicată succesiv noilor

subprobleme, până la nivelul de detaliu la care obținerea soluțiilor subproblemelor este facilă. În mod natural totul se finalizează cu reconstituirea „de jos în sus" a soluțiilor parțiale. O reprezentare grafică sugestivă a metodei este prezentată mai jos:

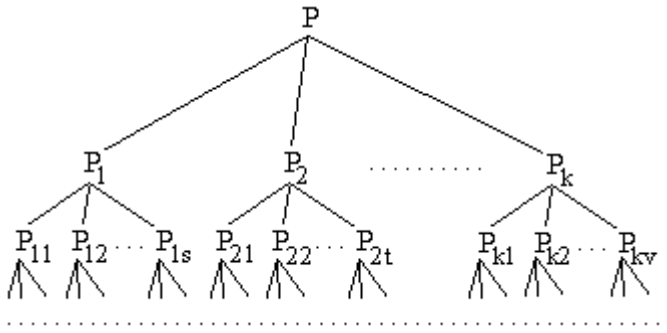


Figura 11

Problemă. Să considerăm $n \geq 1$ elemente $a_1, a_2, \ldots a_n$ şi un subşir al acestuia $a_p, a_{p+1}, \ldots a_q$, cu $1 \leq p < q \leq n$ asupra căruia avem de efectuat o prelucrare oarecare (procedura Prelucrare).

Soluție. Metoda *divide et impera* de rezolvare a acestei probleme presupune împărțirea şirului determinat de capetele acestuia (procedura Divide), (p,q), în două subşiruri (p,m) şi $(m+1,q)$, $p \leq m < q$ sau $(p,m-1)$ şi (m,q), $p < m \leq q$, asupra cărora să se poată efectua mai uşor prelucrarea. Prin prelucrarea celor două subşiruri se vor obține rezultatele β şi γ care *combinate* (procedura ObținSoluțieFinală) vor conduce la soluția α a problemei inițiale. Împărțirea în subşiruri poate continua până la gradul de detaliu care permite obținerea imediată a soluției prelucrării unui subşir. Metoda este ilustrată de procedura de mai jos. Parametrii procedurii DivideEtImpera au urmatoarea semnificație:

p - primul parametru, care reprezintă indexul primului element al şirului;

q - al doilea parametru, care reprezintă indexul ultimului element al şirului;

d - numărul de elemente din şir pentru problema cea mai simplă (elementară, până la care se face divizarea).

```
Procedura DivideEtImpera (p,q,α)
    Dacă (q-p<d) atunci
        Prelucrare (p,q,α)
    altfel
        Divide (p,q,m)
        DivideEtImpera (p,m,β)
```

DivideEtImpera (m + 1,q,γ)

ObținSoluțieFinală (β,γ,α)

Sfdacă

Sfârşit DivideEtImpera

În cele mai frecvente cazuri, procedurile *Divide, ObținSolțieFinală* şi *Prelucrare* sunt compuse dintr-un număr redus de instrucțiuni, nemotivându-se descrierea şi apelul lor separat ca proceduri în corpul procedurii *DivideEtImpera*.

Exemplu. Să se *testeze apartenența* unui element la un şir ordonat crescător.

Rezolvare. Aplicând metoda *divide et impera* vom împărți şirul în două subşiruri. În funcție de elementul *k* (căutat), mai mic sau mai mare decât elementul de diviziune, vom renunța la prelucrarea (căutarea) unuia dintre subşiruri, rezultatul prelucrării fiind deja cunoscut. Vom repeta prelucrarea numai pentru subşirul rămas până când se va ajunge la un şir despre care se poate afirma că este gata prelucrat. In continuare prezentăm algoritmul sub formă de pseudocod (tip **Pascal**), deşi sub o formă nu foarte elegantă.

Intrare. Considerăm că şirul a fost declarat ca un tablou unidimensional, notat cu nSir (dacă şirul conține elemente numere reale şi nu mai mult de 100, atunci o posibilă declarație în C poate fi int nSir[100];). Avem de asemenea nevoie de indexul primului şi ultimului element din şir; notați cu p respectiv q. Valoarea căutată va fi memorată în variabila k.

Ieşire. Vom returna valoarea indexului elementului din şir în cazul în care există soluție şi o valoare negativă în caz contrar. Valoarea returnată este memorată în nIndex.

Observație. Comentariile din cadrul descrierii algoritmului vor fi prefixate cu //, adoptând notația din C/C++. Şirul nSir se consideră că este „vizibil” în cadrul procedurii care urmează.

Procedura DivideEtImpera

// Date de intrare: p, q şi k

// Date de ieşire nIndex

Inițializări.

nIndex := -1 // Presupun că nu există soluție

Început.

Dacă (q-p = 0) **atunci**

// S-a ajuns la o problemă elementară, care se poate rezolva.

// Subşirul conține un singur element.

// Aici este codul ce ar trebui pus în procedura *Prelucrare*

```
        Dacă (nSir[p] = k) atunci
            nIndex = p    // Am obținut soluția problemei elementare.
        altfel
            nIndex = -1   // Nu exista soluție
        Sfdacă
        //Ieşire din procedura DivideEtImpera
    altfel
        // Se împarte problema curentă în subprobleme
        // Calculăm jumătatea intervalului
        m := (p + q) / 2              // se calculeaza partea întreagă
        // Stabilim noul subşir pentru a relua procedura
        Dacă  (nSir[m] > = k) atunci
            q := m
        altfel
            p := m+1
        Sfdacă
        Reluare procedura DivideEtImpera pentru noul subşir
    Sfdacă
Sfârşit
```

Observație. Merită a fi evidențiate procedurile care reliefează metoda *divide et impera* în acest caz:

- procedura Prelucrare este reprezentată de următorul cod:

```
Dacă (nSir[p] = k) atunci
    nIndex = p    // Am obținut soluția problemei elementare.
altfel
    nIndex = -1   // Nu exista soluție
Sfdacă
```

- procedura Divide, prin:
  m := (p + q) / 2
- procedura ObținSoluțieFinală, prin:
  Valoarea lui nIndex.

1.2.2. Metoda backtracking

Backtracking-ul constituie una dintre metodele cele mai des folosite pentru căutarea soluției „optime” pentru o problemă atunci când mulțimea soluțiilor posibile este cunoscută sau poate fi generată. O verificare „necontrolată” printr-o parcurgere după o metodă oarecare a mulțimii soluțiilor posibile este costisitoare ca timp de execuție. Ordinul de complexitate al unui astfel de algoritm este exponențial. Se impune astfel a se evita generarea şi verificarea tuturor soluțiilor posibile.

Problemă. Se consideră $n \geq 2$ mulțimi nevide şi finite $A_1, A_2, \ldots A_n$ şi $m_1, m_2, \ldots m_n$ cardinalele acestor mulțimi. Considerăm o funcție $f: A_1 \times A_2 \times \ldots \times A_n \rightarrow R$. O soluție a problemei este un n-*uplu* de forma $x = (x_1, x_2, \ldots x_n) \in A_1 \times A_2 \times \ldots \times A_n$ care *optimizează* (conform unor criterii specificate) funcția f.

Soluție. Mulțimea finită $A = A_1 \times A_2 \times \ldots \times A_n$ se numeşte spațiul *soluțiilor posibile* ale problemei. *Condiția de optim* pe care trebuie să o îndeplinească o soluție este exprimată printr-un set de relații între componentele vectorului x, relații exprimate prin forma funcției f. O soluție posibilă, care optimizează funcția f, adică satisface *condițiile interne* ale problemei se numeşte *soluție rezultat*, sau mai simplu, *soluție a problemei*. Construirea unei soluții constă în determinarea componentelor vectorului x. Construirea primei soluții începe întotdeauna cu construirea primului element al vectorului x (normal!). La un moment dat se va alege un element dintr-o mulțime, pe care convenim să o numim *mulțimea curentă* şi, presupunând că elementele fiecărei mulțimi A_i $(1 \leq i \leq n)$ sunt ordonate, elementul care se adaugă la vectorul x îl vom numi *elementul curent*. Următorul algoritm (prezentat în limbaj natural) descrie metoda backtracking la nivel conceptual:

Pas1. Considerăm prima mulțime, *A1,* ca *fiind mulțime curentă*.

Pas2. Trecem la următorul element din *mulțimea curentă* (când o mulțime devine mulțime curentă pentru prima dată sau prin trecerea de la o mulțime anterioară ei, acesta va fi primul element din acea mulțime).

Pas3. Verificăm dacă un asemenea element există (adică nu s-au epuizat elementele *mulțimii curente*).

a. **Dacă nu există** un asemenea element, atunci mulțime curentă devine mulțimea anterioară celei curente; când o asemenea mulțime nu există, algoritmul se **opreşte** (nu se mai pot obține soluții);
b. **Dacă există**, atunci verificăm dacă elementul curent din mulțimea curentă, împreună cu componentele vectorului x determinate anterior,

pot conduce la o soluție (această verificare stabileşte dacă sunt îndeplinite *condițiile de continuare* a construirii soluției optime):

i. **Dacă „Da”** (condițiile de continuare sunt îndeplinite), următoarea mulțime devine mulțime curentă, şi se continuă cu **Pas2**;
ii. altfel se continuă cu **Pas3**.

Etapele în detaliu ale acestui algoritm pot fi următoarele:

B1. Definesc mulțimile A_i , i=1,2,...,n. Fiecare mulțime are m_i elemente, i=1,2,...,n, iar modul de memorare al acestor mulțimi îl alegem ca fiind coloanele matricii A[m.n] (coloana i din această matrice reprezintă mulțimea A_i, iar m este cel mai mare număr dintre m_1, m_2, ... , m_n).

B2. Completez cu informațiile necesare lipsă matricea A.

B3. Memorez numărul maxim de elemente pentru fiecare mulțime A_i, i=1,2,...,n în vectorul nr_elemente (de exemplu nr_elemente[2] va conține valoarea lui m_2).

B4. Definesc vectorul soluție x[n] (n reprezintă aici numărul maxim de elemente pentru x).

B5. Completez elementele lui x cu o valoare care nu este în A_i (am notat în cazul de față cu *nimic* această valoare - vezi şi semnificația lui null, nil din limbajele de programare).

B6. Definesc vectorul indecşilor, notat *index* (de exemplu index[1] va păstra indexul elementului selectat din mulțimea A_1 şi care se găseşte în vectorul soluție), pentru fiecare mulțime şi îl inițializez cu –1 (o valoare care nu poate reprezenta un index corect, deci *nimic* în acest caz nu poate reprezenta elementul -1).

B7. Începem procesul de construcție al soluției (variabila i păstrează indexul mulțimii curente şi ia valori de la 1 la n) (?????-mai de verificat aici):

B7.1. i = 1; // luăm prima mulțime, A_1, adică A[.,1]
index[i] = 1; // punctează la primul element din A[index[i],i]
x[i] = A[index[i],i]; //punem primul element în soluție

B7.2. **Câttimp** (mai am mulțimi de selectat) **execută**
{
// atâta timp cât mai există elemente în A[.,i]
Câttimp (index[i] <= nr_elemente[i]) **execută**
{
Dacă (valid(...)) **atunci** // dacă elementul este corect

```
                        // putem trece la următoarea
                        // mulțime
            Dacă (i==n) atunci // suntem la ultima mulțime!
                afisare_soluție();
            altfel
             {
                i++;    // trecem la următoarea mulțime
                index[i] = 1;   // în anumite cazuri se poate
                                // şi index[i]++
             }
            x[i] = A[index[i],i];   // punem elementul în
                                    // soluție
        }
        // Bucla while s-a terminat; deci mulțimea A[.,i]
        // nu mai are elemente care să participe la formarea
        // soluției. Trebuie să ne întoarcem.
        // Înainte de a schimba valoarea lui i vom inițializa
        // indexul de căutare în această mulțime cu –1.
        // Aceasta înseamnă că o nouă căutare în
        // mulțime se va face din nou de la primul element,
        // şi vom pune nimic în soluție
        index[i] = -1;
        x[i] = nimic;
        i--;     // întoarcerea la mulțimea anterioară
        index[i]++;    // măresc indexul de căutare în mulțimea
                       // curentă
        Dacă (index[i] <= nr_elemente[i]) // verific din nou dacă
                                          // indexul este valid
            x[i] = A[index[i],i];
} câttimp (i != 0);
```

Observație. O modificare minoră (inițializarea lui x[i]) a acestui cod conduce la eliminarea secvenței:

```
if (index[i] <= nr_elemente[i]) // verific din nou dacă
```

```
                                        // indexul este valid
                x[i] = A[index[i],i];
```

Cazuri particulare. Toate mulţimile A_i, i=1,2,...,n au acelaşi număr de elemente care sunt în ordine crescătoare şi sunt numere naturale: {1,2,3,...,n}. Se pleacă iniţial cu vectorul soluţie x[]={0,0,...,0}. Pentru componenta x[i], trecerea la următorul element înseamnă x[i]++, iar la elementul anterior x[i]--. Testul de existenţă al elementelor pentru x[i] este 1 <= x[i] <= n (similar se poate proceda şi în cazul codului pentru problemele permutărilor, aranjamentelor etc.). În codul anterior, funcţia valid() trebuie detaliată şi este dependentă de enunţul problemei. Este evident că între *condiţiile interne* (de optim) şi *condiţiile de continuare* există o strânsă legătură, sincronizarea acestora având ca efect o importantă reducere a numărului de operaţii.

O sinteză a metodei backtracking scoate în evidenţă patru etape principale:

- *etapa* în care unei componente a *vectorului soluţie* i se atribuie o valoare din mulţimea corespunzătoare acesteia, urmată de trecerea la mulţimea (componenta) următoare;

- *etapa* în care atribuirea unei valori pentru o componentă a *vectorului soluţie* se soldează cu un eşec, situaţie care se încercă a fi depăşită prin trecerea la următorul element din mulţimea (curentă) corespunzătoare componentei;

- *etapa* în care elementele mulţimii curente au fost epuizate, situaţie generată de o alegere anterioară nepotrivită, caz în care se impune o revenire la mulţimea anterioară, revenire care poate încheia nefericit (fără găsirea unei soluţii) întreg procesul de căutare a soluţiilor;

- *etapa* revenirii în procesul de căutare a unei noi soluţii după obţinerea unei soluţii, etapă care se realizează prin trecerea la elementul următor din ultima mulţime.

Algoritmul prezentat mai sus conduce la obţinerea unei soluţii (dacă măcar o soluţie există). De fiecare dată, pornind de la ultima soluţie obţinută pot fi determinate următoarele eventuale soluţii optime.

Procedura pseudocod de mai jos realizează acest lucru, pornind de la premiza că cele n mulţimi sunt cunoscute.

Vom nota cu a_{ik} al k-lea element din mulţimea A_i şi vom conveni că valoarea variabilei k este proprie fiecărei valori a variabilei i, adică există câte o variabilă k pentru fiecare valoare a variabilei i, notată tot cu k, în loc de k_i .

```
Procedura backtracking
    i := 1
    k := 0                    {k = 0 are semnificația k1 = 0}
    Repetă
      Repetă
        k := k + 1
        Dacă ( k > mk ) atunci
          k = 0               {k = 0 are semnificația ki = 0}
          i = i – 1           {se realizează „întoarcerea”}
        altfel
          xi = aik
          Dacă (x1, x2, ... xi conduce la optim) atunci
              i = i + 1 se verifică condiția de continuare
          Sfdacă
        Sfdacă
      Pânăcând (i > n sau i = 0)
      Dacă ( i > n ) atunci
        „afişare soluție”
        i = n
      Sfdacă
    Pânăcând ( i = 0 )
Sfârşit
```

Exemplu (*Generarea tuturor permutărilor* unei mulțimi având *n* elemente). Să considerăm mulțimea *A = {1,2, ... ,n }, n>0*. Să se determine toate *n*-uplele de elemente distincte din *A*.

Soluție. Această problemă reprezintă un caz particular a problemei generale prezentate anterior, caz în care toate cele *n* mulțimi sunt egale cu mulțimea *A*. Se aplica metoda backtracking considerând funcția de optim exprimată prin condiția: *elementele vectorului soluție să fie distincte*. Pentru cititorul interesat codul poate fi găsit în [MPI ...].

1.2.3. Metoda greedy

Spre deosebire de metoda backtracking, metoda *greedy* este o metodă ce permite determinarea *unei singure soluții* care corespunde unui anumit *criteriu de optim*, în cazul problemelor în care soluția se construieşte ca o submulțime a unei mulțimi date. Ordinul de

complexitate al unui astfel de algoritm este redus considerabil prin faptul că se încearcă obținerea soluției printr-o singură parcurgere a mulțimii din care se construieşte soluția optimă, cu toate că în practică, înainte de aplicarea metodei, se fac prelucrări asupra acestei mulțimi care măresc ordinul de complexitate.

Problemă. Se dă o mulțime A de cardinal n $(n \geq 0)$ şi o funcție $f: P(A) \rightarrow R$. Să se determine o submulțime $B \in P(A)$ de cardinal k, $B = \{b_1, b_2, \ldots b_k\}$, $(1 \leq k \leq n)$, astfel încât *k-uplul* $(b_1, b_2, \ldots, b_k)$ *să optimizeze* funcția f.

Soluție. Familia părților mulțimii finite A, notată $P(A)$ se numeşte *spațiul soluțiilor* problemei. *Condiția de optim* pe care trebuie să o îndeplinească o soluție este exprimată printr-un set de relații între anumite elemente ale mulțimii A, relații exprimate prin funcția f. O soluție care poate conduce la obținerea unei soluții optime se numeşte *soluție posibilă*. Pot exista mai multe soluții care satisfac condițiile de optim, dar se doreşte obținerea măcar a uneia dintre acestea.

Construirea unei soluții optime constă din determinarea unei succesiuni de soluții posibile care îmbunătățesc progresiv valoarea funcției f, conducând către optim. Soluțiile posibile au proprietatea că orice submulțime a unei *soluții posibile* este o *soluție posibilă*. Prin urmare şi mulțimea vidă poate fi considerată ca o *soluție posibilă*.

Descriere metodă:

Pas 1.
- considerăm submulțimea B, mulțimea vidă;
- se alege un element $a \in A$, neales la un pas anterior;
- verificăm dacă submulțimea $B \cup \{a\}$ conduce la o soluție posibilă
  - dacă da, atunci adăugăm elementul ales la mulțimea B $(B := B \cup \{a\})$.
- se continuă cu **Pas 1** până când nici un element al mulțimii A nu mai poate fi adăugat la B sau adăugarea lui nu mai poate îmbunătăți valoarea funcției f.

Algoritmul prezentat mai sus conduce la obținerea unei soluții (măcar o soluție există întotdeauna), pornind de la mulțimea vidă şi căutând în fiecare pas să îmbunătățim soluția deja obținută. Această tehnică de obținere a soluției, care a dat şi denumirea, oarecum ironică, a metodei (*greedy = lacom*), în cele mai frecvente cazuri conduce la îndepărtarea involuntară

de optim, cunoscut fiind faptul (plastic exprimat prin *lăcomia pierde optimalitatea*), că optimul local nu atrage optimul global.

Acest aspect al tehnicii *greedy* a condus la disocierea algoritmilor elaborați prin metoda *greedy* în:

- algoritmi cu atingerea optimului global;
- algoritmi ale căror soluții converg către optimul global (evident, fără atingerea acestuia în toate situațiile). Această din urmă categorie de algoritmi generează soluții *mulțumitoare* în majoritatea cazurilor, dar şi soluții *catastrofale* în alte cazuri.

Disocierea în cele două categorii se realizează prin modalitatea de alegere a elementelor din mulțimea A. De aceea, este frecvent folosită o prelucrare (reordonare) prealabilă a elementelor mulțimii A care să modifice ordinea alegerii elementelor submulțimii B.

procedura greedy
    k := 0 k este numărul de elemente din B
    B := ∅
    repetă
      alege $a \in A$
      dacă ($B \cup \{a\}$ este soluție posibilă) **atunci**
          k::= k + 1
          B := B $\cup$ {a}
      sfdacă
    până când (nu se mai pot alege elemente din A)
sfârşit

Exemplul care urmează scoate în evidență cele două aspecte ale metodei: atingerea optimului sau numai apropierea de acesta.

Exemplu (funcția *maxim*). Se dă o submulțime A a lui R, cu n elemente şi o funcție f de forma $f(x_1,x_2,...,x_k) = c_1x_1 + c_2x_2 +... + c_kx_k$, $(c_i \in R,\ 0 \le k \le n)$. Să se găsească o submulțime $B \subseteq A$ de cardinal k pentru care funcția f ia valoare maximă.

În programul **Pascal:**

- fişierul de *intrare multime.txt* va avea forma:

$a_1, a_2, ..., a_n$ - mulțimea A

$c_1, c_2, ..., c_n$ - coeficienții funcției f

- *ieşirea* va fi:

Soluția : $x = (b_1, b_2, ..., b_k)$

Valoarea maximă a funcției este v.

Soluție. Această problemă constituie un exemplu ilustrativ complet, pentru cazul în care prin aplicarea metodei *greedy* se obține valoarea optimă a funcției f. Algoritmul necesită o pregătire prealabilă a mulțimii A în vederea aplicării procedurii de alegere succesivă a elementelor submulțimii B:

- se va ordona crescător mulțimea A;
- pornind de la $B := \varnothing$ vom selecta elementele din A astfel:
  - cât timp printre coeficienții c_i ai funcției f există numere negative (cărora nu li s-a asociat un element din A, ca valoare pentru x_i - ul corespunzător), executăm: celui mai mic coeficient neasociat unui element din A, îi ataşăm cel mai mic număr din A încă neselectat;
    - pentru ceilalți coeficienți (pozitivi) ai funcției f cărora nu li s-a asociat un element din A (ca valoare pentru x_i), se alege, pentru cel mai mare coeficient neasociat unui element din A, cel mai mare număr din A încă neselectat.

Vom ilustra algoritmul cu un exemplu numeric.

Exemplu. Fie mulțimea de numere $A = \{ -8,-7,-5,-1,2,3,3, 5,7,8\}$ (deja ordonată) şi funcția f de forma $f(x_1,x_2,x_3,x_4,x_5,x_6,x_7) = 3x_1 + 6x_2 - x_3 - 9x_4 - 9x_5 + 3x_6 + 8x_7$. Soluția problemei va fi un vector $x = (b_1,b_2,b_3,b_4,b_5,b_6,b_7)$ ale cărui componente sunt elemente din A. Succesiunea alegerii valorilor componentelor vectorului x pune în evidență *tehnica greedy*:

- corespunzător celui mai mic element negativ dintre coeficienții funcției f alegem primul element din A, deci $b_4 = -8$;

- corespunzător celui mai mic element negativ dintre coeficienții funcției f pentru care nu s-a ales încă o valoare pentru elementul vectorului x, alegem următorul element din A, deci $b_5 = -7$;

- continuăm alegerea elementelor lui x până când tuturor coeficienților negativi ai funcției f li s-a asociat componenta corespunzătoare în vectorul x. Obținem $x = (b_1,b_2,-5,-8,-7,b_6,b_7)$;

- corespunzător celui mai mare element pozitiv dintre coeficienții funcției f alegem ultimul element din A, deci $b_7 = 8$;

- corespunzător celui mai mare element pozitiv dintre coeficienții funcției f pentru care nu s-a ales încă o valoare pentru elementul vectorului x, alegem elementul anterior celui ales la pasul precedent, deci $b_2 = 7$;

- continuăm alegerea elementelor lui x până când tuturor coeficienților funcției f li s-a asociat componenta corespunzătoare în vectorul x.

Obținem în final $x = (5,7,-5,-8,-7,3,8)$. Valoarea maximă a funcției este $f(5,7,-5,-8,-7,3,8) = 270$.

1.2.4. Metoda programării dinamice

Metoda *programării dinamice*, aşa cum îi arată şi numele, permite determinarea unei soluții pentru o problemă dată, în urma unui şir de decizii şi prelucrări ce se condiționează reciproc, realizând o *dinamică* continuă a procesului de căutare a soluției. Ordinul de complexitate al unui astfel de algoritm este condiționat de modul de organizare a datelor inițiale, a rezultatelor intermediare şi de modalitatea de regăsire a rezultatelor intermediare, obținute anterior momentului unei noi prelucrări a acestora.

Problemă. Noțiunea de algoritm, aşa cum a fost prezentată în lucrare, presupune ca entități distincte, existența unui set de date de intrare şi a unei metode de transformare succesivă a acestora, în vederea obținerii unui set coerent de date de ieşire ca rezultat al tuturor prelucrărilor. Abordarea celor trei elemente ca un sistem presupune existența unor intercondiționări între acestea. Ca metodă de elaborare a algoritmilor de rezolvare a unor clase de probleme, programarea dinamică presupune identificarea acestor corelații, privind problema inițială ca un sistem de miniprobleme care se condiționează reciproc.

Soluție. Pentru o problemă dată, fie S_0 *starea sistemului* format din datele de intrare şi de lucru (intermediare), precum şi din corelațiile care există între acestea. O decizie d_1 de transformare a datelor orientată în direcția obținerii unei *soluții optime* pentru problemă

produce o *prelucrare* a stării S_0 determinând transformarea acesteia într-o nouă stare S_1. Suntem în acest moment puşi în faţa uneia sau mai multor probleme similare cu cea iniţială şi care - printr-o nouă *decizie* (comună) de prelucrare - conduc la o nouă stare. Schimbarea stării sistemului va continua până la obţinerea unei stări finale din care se deduce o *soluţie optimă* a problemei iniţiale.

În general, fiecare nouă decizie de transformare a stării sistemului depinde de deciziile luate anterior (acestea au generat starea curentă a sistemului) şi nu este unic determinată ca în cazul metodei greedy, de exemplu.

Fie $d_1, d_2, ..., d_{n-1}, d_n$ o secvenţă de decizii optime care determină trecerea succesivă a sistemului din starea iniţială S_0 în starea finală S_n, prin intermediul stărilor $S_1, S_2, ..., S_{n-1}$.

O modalitate naturală de abordare a problemei constă din luarea succesivă de decizii optime de prelucrare în ordinea $d_1, d_2, ..., d_{i-1}$, pornind de la starea iniţială S_0. Decizia următoare d_i, depinde de şirul de decizii optime deja luate $d_1, d_2, ..., d_{i-1}$. Spunem în acest caz că se aplică metoda spre *înapoi* (sfârşitul şirului de decizii).

Dacă se poate stabili starea sistemului S_n din care s-ar deduce soluţia optimă a problemei, este de dorit să se determine o decizie d_n precum şi o stare S_{n-1} din care să se ajungă în starea S_n în urma aplicării deciziei d_n. Intuitiv spus, se determină *inversa unei decizi*i şi starea sistemului anterioara luării acestei decizii. Fie secvenţa de decizii optime $d_{i+1}, d_{i+2}, ..., d_n$ care duc sistemul din starea S_i în starea finală S_n. O nouă decizie d_i care să ducă sistemul din starea S_{i-1} în starea S_i va depinde de şirul de decizii $d_{i+1}, d_{i+2}, ..., d_n$. Spunem în acest caz că se aplică metoda spre *înainte* (începutul şirului de decizii).

A treia modalitate de abordare sugerează determinarea unei stări intermediare, S_i , şi a două decizii optime d_i şi d_{i+1}, având două subşiruri optime de decizii:

- $d_{i+2}, d_{i+3}, ..., d_n$ care duc sistemul din starea S_{i+1} în starea finală S_n prin intermediul stărilor $S_{i+1}, S_{i+2}, ..., S_{n-1}$;
- $d_1, d_2, ..., d_{i-1}$, şir de decizii optime care determină trecerea sistemului din starea iniţială S_0 în starea S_{i-1}, prin intermediul stărilor $S_1, S_2, ..., S_{n-2}$. Spunem în acest caz că se aplică metoda *mixtă* (*explozivă*).

Cele trei modalităţi de abordare au la bază *principiul optimalităţii*. Dacă $d_1, d_2, ..., d_n$ este un şir optim de decizii care determină trecerea sistemului din starea iniţială S_0 în starea finală S_n, atunci sunt adevărate următoarele afirmaţii:

- $d_{i+1}, d_{i+2}, ..., d_n$ este un şir optim de decizii care determină trecerea sistemului din starea S_i în starea finală S_n, $\forall i, 0 \le i \le n-1$;

- $d_1, d_2, ..., d_i$ este un şir optim de decizii care determină trecerea sistemului din starea iniţială S_0 în starea S_i, $\forall i, 1 \leq i \leq n$;

- $d_{i+1}, d_{i+2}, ..., d_n$ şi $d_1, d_2, ..., d_i$ sunt şiruri optime de decizii care determină trecerea sistemului din starea S_i în starea finală S_n şi respectiv din starea iniţială S_0 în starea S_i, $\forall i, 1 \leq i \leq n$.

Principiul optimalităţii sugerează stabilirea unor relaţii de recurenţă.

În concluzie, rezolvarea unei probleme prin metoda programării dinamice presupune identificarea unor caracteristici ale problemei care o fac rezolvabilă prin această metodă:

- problema se poate descompune în subprobleme de acelaşi tip cu aceasta;

- subproblemele nu sunt distincte, se intercondiţionează reciproc (altfel s-ar putea aplica tehnica *divide et impera*, mult mai eficientă din punct de vedere al consumului de memorie);

- necesitatea satisfacerii principiului optimalităţii, care implică stabilirea relaţiei de recurenţă prin care se exprimă intercondiţionarea subproblemelor.

În cele ce urmează, vom prezenta un exemplu de abordare a unor probleme prin metoda programării dinamice. Sunt punctate caracteristicele importante ale metodei, chiar dacă problema aleasă poate să fie considerată drept necaracteristică.

Problemă. Să se determine *termenul de rang* k *din şirul lui Fibonacci*, pentru un număr natural k dat.

Intrare: k, de la tastatură.

Ieşire: pe ecran, de forma *Termenul de rang* k *din şirul lui Fibonacci este* v.

Soluţie (metodă). În şirul lui Fibonacci, primii doi termeni sunt $a_0 = 1$ şi $a_1 = 1$. Relaţia de recurenţă $a_k = a_{k-1} + a_{k-2}$, $\forall k>2$, arată că un termen se obţine ca suma ultimilor doi termeni anteriori lui.

Vom folosi metoda *înapoi* plecând de la starea iniţială $u = 1, v = 1$ (primii doi termeni), care reprezintă şi starea din care se deduce soluţia problemei pentru $k = 1$.

Decizia de trecere la o nouă stare determină următoarele prelucrări:

- aplicarea relaţiei de recurenţă (calculul sumei $s = u + v$), care respectă principiul optimalităţii;
- obţinerea noii stări prin atribuirea valorilor $u = v$ şi $v = s$.

Se obţine starea $u = 1, v = 2$.

Aceasta este starea nou obținută (succesoare).

Sunt respectate caracteristicile problemelor care sunt rezolvabile prin metoda programării dinamice:

- soluția unei probleme este obținută din soluția problemei rezolvate anterior (se determină termenul de rang *k* din termenii de rang *k-1* şi *k-2*);

- este satisfăcut principiul optimalității (*o soluție optimă pentru problema anterioară conduce la soluția optimă a problemei curente*).

1.3. Analiza complexității şi corectitudinii algoritmilor

Este evident că pentru rezolvarea unei probleme, dacă aceeaşi metodă de proiectare este folosită de către mai multe persoane, algoritmii realizați pot să difere. Cu atât mai mult acest lucru este posibil atunci când metodele sunt diferite. Aşa cum am mai precizat, vom trata analiza complexității timp/spațiu prin câteva exemple concrete. Să punctăm şi faptul că spațiul de memorie real utilizat de un program care implementează un algoritm este format şi dintr-o parte constantă, independentă de datele de intrare (în care se află memorat de exemplu codul executabil), a cărui dimensiune este de obicei ignorată. De asemenea, timpul necesar introducerii valorilor de intrare şi extragerii rezultatului este ignorat. Vom începe cu un exemplu didactic. Deoarece pseudocodul folosit va fi foarte apropiat de **Pascal**, considerațiile de complexitate pot fi destul de „la obiect”.

Problemă. Să se calculeze suma primelor n numere naturale.

Rezolvare. Primul algoritm propus se bazează pe ideea de a construi o funcție care să calculeze succesiv sumele 0, 0 + 1, 0 + 1 + 2, ... funcție care va întoarce în final valoarea sumei 1 + 2 + 3 + ...+ n:

```
function suma(n : byte):word;
var i : byte;
    s : word;
begin
    s := 0;
    i := 1;
```

```
    while (i <= n) do
        begin
            s := s + i;
            i := i + 1;
        end;
   suma := s;
end;
```

Funcția va ocupa un spațiu de memorie fix pentru parametru, variabilele locale, pentru adresa de revenire şi evident cu codul. Nu există spațiu variabil suplimentar, deci $s^{Alg}(n) = O(1)$.

Al doilea algoritm presupune construirea unei funcții recursive care calculează suma după relația de recurență $s(n) = s(n-1) + n$, cu $s(0) = 0$:

```
function suma(p : byte):word;
begin
    if ( p = 0 ) then
      suma := 0 ;
    else
       suma := suma(p-1) + p;
end;
```

Pentru fiecare apel al funcției vor fi ocupați 5 octeți; unul pentru memorarea parametrului p, unul pentru valoarea funcției şi 2 octeți pentru adresa de revenire. Se fac n apeluri recursive, deci spațiul de memorie variabil este de 5n octeți. Algoritmul care foloseşte funcția recursivă foloseşte mai mult spațiu efectiv (real) de memorie decât în cazul primului algoritm, $s^{Alg}(n) = O(n)$.

Putem admite chiar că notația asimptotică determină o clasificare a algoritmilor impusă de valoarea ordinului de complexitate, clasificare pe care am putea-o scrie sub forma:

$$O(1) \leq O(\log n) \leq O(n) \leq O(n \log n) \leq O(n^2) \leq O(n^k) \leq O(2^n), \ \forall k > 2,$$

ideea fiind aceea că un algoritm din $O(1)$ este mai bun decât unui din $O(\log n)$ etc. Reprezentarea grafică a funcțiilor care determină ordinul de complexitate prezentată în figura de mai jos este edificatoare. Într-o evaluare care poate fi de exemplu găsită în <21> sau <30>, *dacă* $t^{Alg}(f(n)) = O(2^n)$, *pentru* $n = 40$, *unui calculator care face 1 bilion* (10^9) *de operații pe secundă, îi sunt necesare aproximativ 18 minute. Pentru* $n = 50$, *acelaşi program va rula 13*

zile pe acest calculator, pentru $n = 60$*, vor fi necesari peste* 310 *ani, iar pentru* $n = 100$ *aproximativ* $4 \cdot 10^{13}$ *ani*.

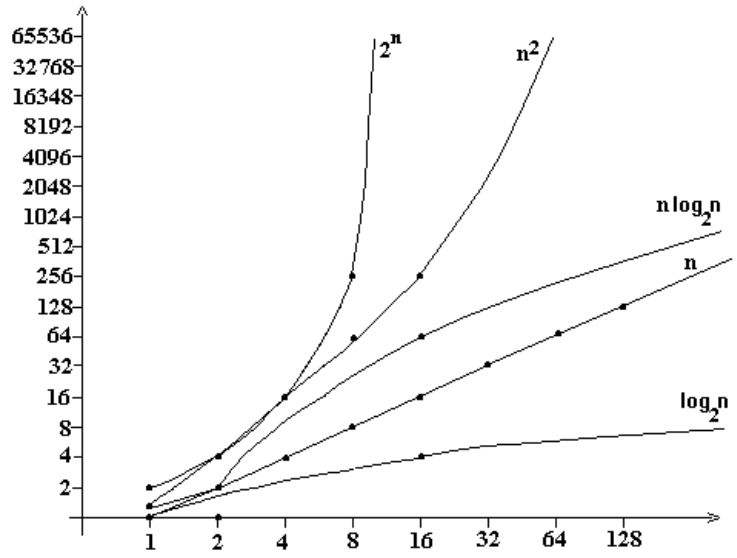


Figura 12.

Algoritmii polinomiali de grad mare nu pot fi utilizați în practică, chiar dacă viteza de execuție a calculatoarelor moderne poate întrece adesea cele mai optimiste previziuni. Astfel, pentru $O(n^{10})$, *pe un calculator care execută 1 bilion de operații pe secundă sunt necesare* 10 *secunde pentru* $n = 10$*, aproximativ* 3 *ani pentru* $n = 100$ *şi circa* $3 \cdot 10^{13}$ *ani pentru* $n = 1000$. Şi următorul tabel poate fi util (<31>):

$O(n)$ (liniar)	$O(\log(n))$ (logaritmic)	$O(n.\log(n))$ (log-liniar)	$O(n^2)$ (pătratic)	$O(2^n)$ (exponențial)	$O(n!)$ (factorial)
1	0	0	1	2	1
2	1	2	4	4	2
4	2	8	16	16	24
8	3	24	64	256	40326
16	4	64	256	65536	20922789888000
32	5	160	1024	4294967296	26313.10^{33}

Evaluarea complexității unui algoritm ca o funcție de dimensiunea datelor de intrare este o problemă dificilă, ea necesitând anumite cunoştințe de matematică superioară chiar dacă se rezumă de cele mai multe ori doar la analiza cazurilor extreme. Deşi în cazul cel mai defavorabil numeroşi algoritmi nu ar putea fi practic utilizați, aceştia au totuşi o comportare acceptabilă în suficiente cazuri reale. O altă posibilitate este analizarea *complexității medii* a

algoritmilor, ceea ce presupune cunoaşterea repartiţiei probabilistice (<12>) a datelor de intrare. În cazurile simple în care putem caracteriza datele de intrare cu precizie, dacă notăm cu D spaţiul datelor de intrare, cu $p(d)$ probabilitatea apariţiei datei $d \in D$ la intrarea algoritmului şi cu $t(d)$ numărul de operaţii elementare efectuate de algoritm pentru o intrare d din D, atunci *complexitatea medie* este dată de suma $\sum p(d)t(d)$. Vom apela la două exemple simple, tratate de I. Maxim în <30>. Un prim exemplu va prezenta un algoritm (implementat în limbajul **Pascal**), a cărui complexitate nu depinde decât de volumul datelor de intrare şi nu de alte caracteristici atipice (se pot consulta şi **Capitolul 6** sau **Anexa 1**).

Sortarea prin selecţie (cu alegerea minimului). Să se ordoneze crescător elementele vectorului a cu n componente, folosind *metoda alegerii elementului minim încă neselectat* din şirul iniţial.

```
for i := 1 to n-1 do
   begin
      min := a[i] ;
      poz := i;
      for j := i + 1 to n do
         if a[j] < min then
            begin
               min := a[j] ;
               poz := j;
            end;
      a[poz] := a[i] ;
      a[i] := min;
   end;
```

La o iteraţie a ciclului **for** după variabila i se determină minimul din subşirul $a_{i+1}, ..., a_n$ şi elementul minim este plasat pe poziţia i, elementele de la 1 la $i-1$ fiind deja plasate pe poziţiile lor definitive. Pentru a calcula minimul dintr-un şir de k elemente sunt necesare $k-1$ operaţii elementare (se presupune primul element din şir ca fiind cel minim, apoi se fac $k-1$ comparaţii şi eventual atribuiri până la epuizarea elementelor şirului). În total se fac $(n-1) + (n-2) + ... + 2 + 1 = n \cdot (n-1)/2$ comparaţii, deci ordinul de complexitate timp este $O(n^2)$. Să subliniem

faptul că timpul de execuție, în sensul situației cele mai defavorabile, nu depinde de ordinea inițială a elementelor vectorului.

În următorul exemplu vom analiza complexitatea atât în cazul cel mai defavorabil cât şi în medie.

Sortarea prin inserție directă. Să se ordoneze crescător elementele unui vector considerând în fiecare moment că se ordonează un subşir obținut din cel anterior (deja ordonat), prin adăugarea unui nou element. Algoritmul porneşte de la subşirul cu un singur element (care este deja ordonat) şi, odată cu adăugarea unui nou element pe următoarea poziție din şir, acesta este „promovat" până când noul subşir devine din nou ordonat.

```
for i := 2 to n do
    begin
        j :=  i;
       while (a[j-1]  > a[j] ) and ( j > 1 ) do
           begin
               k := a[j-1] ;
              a[j-1] :=  a[j];
              a[j] := k;
              j := j-1;
           end;
    end;
```

Analizăm complexitatea asimptotică a algoritmului în funcție de n, dimensiunea vectorului a ce urmează a fi sortat. La fiecare iterație a ciclului **for** elementele $a_1,a_2,...,a_{i-1}$ sunt deja ordonate şi trebuie să interschimbăm elementele de forma *a[j]* cu cele de forma *a[j-1]* (inițial $j = i$) până când noul şir va deveni ordonat. În cazul cel mai defavorabil, când fiecare element adăugat la şir este mai mic decât cele adăugate anterior, elementul *a[i]* adăugat va fi deplasat până pe prima poziție, deci ciclul **while** se execută de *i-1* ori în cadrul fiecărei execuții a lui **for**. Considerând astfel drept operație elementară compararea elementului *a[j-1]* cu *a[j]* şi interschimbarea acestor elemente cât timp *a[j-1] >a[j]*, vom avea în cazul cel mai defavorabil executate $1 + 2 + ...+ (n-1) = n\cdot (n-1)/2$ operații elementare, deci complexitatea algoritmului este $O(n^2)$.

Să analizăm comportarea algoritmului în medie. Pentru aceasta, vom considera că orice permutare a elementelor şirului are aceeaşi probabilitate de apariţie (orice ordine iniţială este egal probabilă). Atunci:

- probabilitatea ca valoarea a_i, nou adăugată la şirul $a_1, a_2, ..., a_{i-1}$ să fie plasată în final pe o poziţie oarecare k, din $a_1, a_2, ..., a_i$ ($1 \le k \le i$), este aceeaşi adică $1/i$;

- numărul mediu de operaţii elementare (interschimbări de elemente), pentru ca elementul a_i să ajungă pe poziţia k va fi $(i-k)\frac{1}{i}$, adică numărul de schimbări ce se efectuează, înmulţit cu probabilitatea ca aceste schimbări să aibă loc;

- numărul mediu total de operaţii elementare pentru un *i* fixat, va fi

$$\sum_{k=1}^{i} \frac{i-k}{i} = \frac{1}{i}\sum_{k=1}^{i}(i-k) = \frac{1}{i}(i^2 - \frac{i(i+1)}{2}) = \frac{i-1}{2}$$

- pentru a sorta cele *n* elemente sunt necesare

$$\sum_{i=1}^{n} \frac{i-1}{2} = \frac{1}{2}(\frac{n(n+1)}{2} - n) = \frac{n(n-1)}{4}$$

operaţii elementare. Deci complexitatea algoritmului în medie este tot de $O(n^2)$.

Încheiem acest capitol cu câteva scurte consideraţii asupra altor termeni (*formulări, concepte*) care sunt esenţiale şi care ar merita poate să fie tratate independent în (sub)secţiuni separate. Nu insistăm deoarece cunoştinţele de logică matematică presupuse a le avea un licean sunt insuficiente pentru un asemenea cadru.

Foarte important în practică este *studiul coerent al* ***terminării şi corectitudini****i* programelor (şi nu numai *verificarea a posteriori a acestora* prin utilizarea diverselor *date de test*). Ideea este că trebuie să ne asigurăm, dacă se poate printr-o demonstraţie formală, că programele concepute *se termină* pentru orice *instanţă admisibilă a datelor* şi *că ele execută ceea ce vrem, înainte ca ele să fie executate* (<33>). Presupunând astfel că *informaţiile de intrare* admisibile sunt cele care satisfac o anumită condiţie, exprimată printr-un predicat P (*precondiţie*), rămâne să arătăm că *programul se termină pentru orice asemenea instanţă* şi că, în acest caz, informaţiile de ieşire satisfac un alt predicat Q (*postcondiţie*). Nu este foarte dificil de a trata în acest mod programele care nu conţin *bucle,* aceste construcţii sintactice fiind singurele posibile generatoare de informaţii *necontrolabile* sau de *execuţii infinite.* Pentru a *stăpâni ciclurile*, se folosesc *predicatele invariante*. Un predicat invariant R asociat unei bucle, este *adevărat* înainte de prima execuţie a acesteia şi satisface în plus condiţia că,

dacă este *adevărat înainte* de o anumită execuție a corpului buclei, atunci va fi *adevărat* şi *după* terminarea acestei execuții. Un invariant pentru bucla exterioară din algoritmul de mai sus al sortării prin selecție poate fi exprimat prin afirmația: *şirul* $a_1,...,a_i$ *este ordonat crescător*. Se poate consulta şi <37> sau <42> pentru alte detalii.

Exemplu *(aflarea elementului minimal al unui şir de numere naturale)*. Această problemă mai este întâlnită pe parcursul lucrării, fiind rezolvată deja în alt context (de exemplu în algoritmul de sortare prin selecție amintit). Soluția de mai jos este adaptată scopului de moment, algoritmul fiind descris acum prin pseudocodul:

Intrare: $(a_1, a_2, ..., a_k)$, $k \in N$, $k > 1$.

Ieşire : min.

Metodă :

Pas1. min := a_1

Pas2. **Pentru** i de la 2 la k **execută**

Pas3. **Dacă** (min > a_i) **atunci**

Pas4. min := a_i

Sfdacă

Sfpentru

Precondiția P : *vectorul de intrare (să spunem,* a*), are cel puțin* 2 *componente şi acestea sunt numere naturale*.

Postcondiția Q : *numărul* min *conține cel mai micnumăr dintre elementele vectorului de intrare*.

Invariantul R, al unicei bucle prezente în program : min *reprezintă cel mai mic element al subsecvenței* $a_1, a_2, ..., a_i$.

Terminare. În acest caz, lucrurile sunt simple datorită faptului că programul conține o unică operație de ciclare şi aceasta are un număr finit de paşi, cunoscut aprioric ($k - 1$).

Corectitudine. Presupunem deci că pornim cu o secvență oarecare de numere naturale, având cel puțin două elemente (P este satisfăcută). Aplicăm algoritmul. Trebuie să arătăm că la finalul execuției (în acest moment bucla **Pentru** s-a executat până la, inclusiv, valoarea $i = k$), valoarea găsită pentru varabila *min* satiface Q, adică *min* este mai mică sau egală decât orice valoare a_i din vectorul inițial. Vom arăta întâi că R este invariant al buclei. Într-adevăr, după execuția **Pas1** (şi înainte de execuția **Pas2**, adică înainte de prima execuție a corpului buclei **Pentru**), trebuie ca *min* să fie cel mai mic element (în sensul ordinii standard pe **N**) al (sub)secvenței formată doar din a_1, ceea ce este evident. Să presupunem acum că R este satisfăcut înainte de cea de-a j-a execuție a corpului buclei şi să arătăm ca el este adevărat şi după această execuție. Prin urmare, ştim că *min* conține cea mai mică valoare a secvenței $a_1, a_2, \ldots, a_{j-1}$ şi executăm corpul buclei (**Pas3** şi **Pas4**) pentru $i=j$. Trebuie arătat că în urma acestei execuții variabila *min* va conține cea mai mică valoare a secvenței $a_1, a_2, \ldots, a_j$. Din nou, afirmația este evidentă. *În concluzie*, după terminarea execuției buclei, *min* va conține cea mai mică valoare a secvenței $a_1, a_2, \ldots, a_k$.

Observație. Pentru algoritmi mai complicați şi care conțin cicluri cu un număr necunoscut (aprioric) de paşi (de tipul **while**), s-ar putea să fie nevoie să se utilizeze anumite *trucuri tehnice*, cum ar fi *salvarea variabilelor de intrare în anumite variabile de lucru* şi considerarea unor predicate invariante mai puternice (adică, afirmații care să implice Q , nu să coincidă cu acesta). Chiar şi aşa, folosind cunoştințe elementare de logică clasică, tot trebuie să se ştie ce înseamnă termeni ca *valoare de adevăr, teoremă directă, contrară, reciprocă, raționament, sferă, diferență specifică*, *modus ponens*, *tertium non datur*, *model*, *sistem deductiv*, *teorie logică* etc. Din lipsă de spațiu indicăm doar câteva referiri bibliografice: <2>, <11>, <26>, <36>.

Cunoaşterea unei *teorii generale a structurilor de date* ar fi benefică, la fel ca şi lucruri despre *compilatoare*, *programare neimperativă*, *sisteme de operare şi medii distribuite*, *sisteme multimedia* etc. Din nou vom apela doar la anumite referințe bibliografice: <5>, <6>, <18>, <19>, <20>, <22>, <33>, <40>. Studiul exemplelor din **Capitolul 6** şi **Anexa 1** poate fi de asemenea util.

Capitolul 2

În acest capitol vom trata câteva probleme globale ale învățământului noului mileniu, cu accentul pe actualitatea din România. Strategiile acceptate astăzi la nivel național pot influența ierarhiile şi importanța utilizării *principiilor*, *metodelor* şi *obiectivelor didactice*. Autorii rămân îndatorați domnului profesor I. Maxim pentru ideile de bază sugerate în construcția acestui capitol, idei expuse în cartea anterioară, comună, menționată. Titlurile secțiunilor care urmează nu reprezintă o chintesență a cursurilor cu acest nume ținute sub egida Facultății de Psihologie şi Științe ale Educației a Universității (sau a **DPPD**), ci doar o reflectare a acestora în domeniul Informaticii, cu tenta personală a autorilor.

1. Teoria curriculum-ului

Credem că este necesară o participare ***activă*** a fiecărui cadru didactic în îndeplinirea obiectivelor generale ale învățământului conform ideii că sistemul educațional românesc trebuie să răspundă prompt atât la cerințele pedagogice cât şi la cele implicând transformarea societății. Una dintre cele mai importante părți ale acestui sistem este **curriculum-ul şcolar** privind conținuturile învățării. În sensul politicii educaționale, *curriculum-ul defineşte sistemul de procese decizionale, manageriale şi de monitorizare care preced, acompaniază şi urmează proiectarea, elaborarea, implementarea, evaluarea şi revizuirea permanentă şi dinamică a setului de experiențe de învățare oferite de şcoală* (<45>, <46>).

1.1. Curriculum-ul național în Informatică

Curriculum-ul național (numit şi **nucleu**) cuprinde sistemul documentelor de tip regulator şi normativ, în cadrul cărora se consemnează experiențele de învățare recomandate elevilor prin şcoală. Conform acestei accepțiuni, curriculum-ul desemnează ansamblul experiențelor de învățare pe care şcoala îl oferă tinerilor, cu scopul de a-i asista în descoperirea şi valorificarea maximă a propriilor disponibilități şi interese şi, în acelaşi timp înseamnă ceea ce întreprind elevii în şcoală sub îndrumarea profesorilor în materie de învățare şi dezvoltare personală. Curriculum-ul Național reprezintă ansamblul experiențelor de

învățare prin care orice instituție şcolară asigură realizarea idealului educațional şi a finalităților învățământului**.** Acesta impune în primul rând fixarea *cadrului de referință* ca document regulator înglobând anumiți indicatori care vor asigura coerența (în termeni de procese şi produse), a întregului sistem curricular. *Planul-cadru de învățământ* este un document regulator care delimitează ariile curriculare, obiectele de studiu şi alocarea de timp minimă şi maximă aferente acestora, pe nivele de învățământ. *Programele şcolare* stabilesc programele analitice, insistându-se asupra conținutului particular (acestea fiind realizate pe clasele şi disciplinele prevăzute în planul-cadru). În consecință, există *Metodologii de aplicare* ale acestora, reprezintând ghiduri de implementare, reglementări suplimentare etc. O discuție mai vastă asupra **Manualelor alternative** şi asupra **curriculum-umui la alegerea şcolii** ar fi benefică dar necesită un spațiu tipografic mult prea mare. Introducerea unui Curriculum Național a fost însoțită de o serie de concepte noi, atât la nivelul documentelor regulatoare cât şi la nivelul programelor, acestea fiind prezentate succint (neexhaustiv şi, din motive obiective, poate nu în ultima formă) în cele ce urmează. Trebuie să vorbim mai întâi de **profilul de formare** al unui absolvent. Acesta ar trebui să sintetizeze principalele cunoştințe, capacități şi atitudini dezirabile obținute la capătul parcursului şcolar obligatoriu, în concordanță cu aşteptările societății față de tânărul absolvent. În termeni operaționali, de la un absolvent de învățământ obligatoriu se aşteaptă (<15>, <45>):

- Să comunice eficient în situații reale.
- Să înțeleagă sensul apartenenței la diverse tipuri de comunități (locală, națională, europeană, etc.).
- Să demonstreze flexibilitate, capacitate de adaptare şi de integrare în medii diverse.
- Să rezolve probleme, să ia decizii şi să-şi exprime opiniile, folosind gândirea critică şi creativă.
- Să folosească în mod eficient tehnologiile relevante pentru viața de toate zilele.
- Să înțeleagă fenomenele esențiale ale naturii înconjurătoare şi ale mediului social imediat.
- Să contribuie la structurarea şi ocrotirea unei vieți sociale de calitate.
- Să aplice şi să-şi valorifice propriile experiențe de învățare, în scopul dezvoltării unui ansamblu personal de atitudini şi al identificării viitoarei orientări profesionale.

- Să-şi formeze capacitățile şi motivațiile proprii învățării permanente.

Prin **ciclu curricular**, se exprimă un concept bazat pe stadiul de dezvoltare psiho-pedagogică al elevilor şi care oferă un set coerent şi clar de obiective de învățare, reflectate la nivelul programelor şcolare. Specificitatea dominantelor curriculare ale fiecărui ciclu în parte este importantă atât pentru proiectarea curriculum-ului, cât şi pentru profesori, elevi, părinți etc. **Curriculum-ul nucleu** şi **Curriculum-ul la decizia şcolii** sunt cele două segmente principale care concură la formarea Curriculum-ului Național. *Curriculum-ul nucleu* este (general) obligatoriu pentru toate şcolile şi toți elevii, reprezentând segmentul prin care învățământul public speră să asigure egalitatea şanselor. Acesta reprezintă unicul sistem de referință pentru examinarea externă admisă în sistem şi constituie baza standardelor naționale de performanță şi de evaluare. *Curriculum-ul la decizia şcolii* vizează zona opțională a *Curriculum-ului Național* şi se concretizează prin:

- Segmentul opțional al disciplinelor obligatorii.
- Disciplinele propriu-zis opționale.

Avantajele acestui mod de abordare a organizării curriculare sunt:

- descongestionarea materiei;
- creşterea posibilităților de opțiune pentru elevi şi profesori;
- asigurarea parcursurilor individuale de învățare;
- creşterea posibilităților unității şcolare în a-şi determina propriul curriculum;
- posibilitatea utilizării flexibile a segmentului neobligatoriu din programe în funcție de nevoile locale de educație şi formare;
- obligativitatea stabilirii unor standarde coerente de performanță;
- obligativitatea formării resurselor umane (profesori şi manageri).

În continuare, putem spune că **ariile curriculare** reprezintă grupaje de discipline, precum şi de domenii şi obiecte opționale, fiind neschimbate pe întreaga durată a şcolii (segmentului şcolar). Ponderea lor pe cicluri şi clase variază în timp. În acest sens, **obiectele de studiu** sunt părți ale ariilor curriculare şi pot fi obligatorii sau opționale. **Programele şcolare** stabilesc obiectivele şi conținuturile propriu-zise ale învățării la nivelul obiectelor de învățământ. Acestea reglează atât procesul de predare-învățare cât şi realizarea manualelor şi altor materiale suport destinate procesului de predare-învățare. Programa şcolară ar trebui să cuprindă, printre altele: modelul curricular al disciplinei; obiectivele cadru ale disciplinei; obiectivele de referință; activitățile de învățare recomandate; conținuturile sugerate pentru autorii de manuale; standardele de performanță pe ciclu curricular (*ciclul achizițiilor*

fundamentale, ciclul de dezvoltare, ciclul de observare/orientare). Pe baza planului-cadru gândit la nivel național, este evident că fiecare şcoală îşi poate decide propria schemă orară, în funcție de proiectul curricular pe care-l realizează. Se poate astfel contura *personalitatea şcolii*, într-o societate care-şi propune să respecte şi să valorizeze diversitatea, în contextul respectării standardelor internaționale, ale unei educații de calitate şi acordării unor şanse realmente egale tuturor tinerilor.

1.2. Elaborarea ofertei curriculare

Oferta curriculară generală este o componentă a culturii curriculare naționale, o parte integrantă a curriculum-ului național. Proiectarea şi elaborarea curriculum-ului nu este apanajul unei elite, un sistem educațional puternic trebuind să fie preocupat de formarea inițială şi continuă a unui contingent numeros şi bine pregătit de cadre didactice capabile de a proiecta şi elabora un curriculum. O asemenea ofertă trebuie să urmărească compatibilitatea sistemului de învățământ (românesc) cu alte sisteme de învățământ performante din lume, în acelaşi timp cu mărirea impactului sistemului de învățământ asupra reformei societății româneşti. Oferta curriculară a fiecărei unități şcolare este un atu important pentru rezonanța şcolii în microclimatul socio-economic în care s-a integrat. Pornind de la finalitățile fiecărui ciclu de pregătire, oferta curriculară a unei unități şcolare trebuie să țină cont de următoarele elemente: nivelul de studiu; profilurile şi specializările existente; baza didactico-materială; resursele umane; preferințele părinților şi elevilor; specificul local; standardele ocupaționale; contextul socio-cultural etc. Decizia privind modul în care va fi abordat acest segment de curriculum aparține deopotrivă ofertanților (şcolii, adică profesori, manageri) cât şi beneficiarilor (elevi, părinți, reprezentanți autorizați ai comunității locale). Curriculum-ul la decizia şcolii poate fi astfel orientat spre:

- ***Curriculum-ul nucleu aprofundat***, util în cazul claselor mai slabe. Acesta poate cuprinde numărul maxim permis de ore din plaja orară pentru o atingere integrală şi efectivă a obiectivelor şi conținuturilor din trunchiul comun.
- ***Curriculum-ul extins***, util în cazul claselor foarte bune. Acesta cuprinde de asemenea numărul maxim de ore din plaja orară, dar în scopul extinderii obiectivelor şi conținuturilor din trunchiul comun;

- ***Curriculum-ul elaborat în şcoală***, util pentru a exploata resursele şi tradiţiile locale, sau pentru a întâmpina cerinţele şi exigenţele de instruire ale elevilor (părinţilor, comunităţii, etc.).

Putem spune că aspectele formative şi informative cuprinse în *curriculum-ul la dispoziţia şcolii* vor constitui obiectul evaluării interne. Didactica pentru învăţământul gimnazial şi liceal se referă, în principal, la obiectivele formative (generale) ale studiului informaticii şi legăturii acesteia cu alte discipline, metodele şi mijloacele didactice specifice disciplinei, principiile didactice clasice aplicate în predarea informaticii, planurile de învăţământ, programele şcolare, manualele şcolare ş.a.m.d. Ultimele trei puncte le vom analiza chiar în capitolul de faţă.

1.3. Planurile de învăţământ

Planurile cadru precizează disciplinele de învăţământ în succesiunea lor pe ani de studiu şi tipuri de şcoli sau nivele. Sunt stabilite, pentru fiecare disciplină în parte, numărul de ore pe săptămână şi ani de studiu, precum şi numărul de ore de aplicaţii practice de laborator, acolo unde este cazul. Acestea au un caracter unic şi obligatoriu pentru fiecare stadiu de pregătire sau tip de şcoală, cuprinzând obiectele de studiu din fiecare clasă (perioadă de studiu). Într-un moment de *reorganizare a mecanismului de achiziţii de cunoaştinţe*, conceperea unui curriculum este o întreprindere greu de realizat ţinând cont de faptul că se impune renunţarea la lucruri depăşite dar şi păstrarea unor soluţii viabile. La baza elaborării planurilor cadru stau următoarele principii (atenţie, nu este vorba despre *principiile didactice*, care vor fi discutate în **Capitolul 3**):

a) *Principiul selecţiei culturale* (alegerea domeniilor cunoaşterii şi gruparea lor în arii curriculare), care vizează armonizarea dintre particularităţile personalităţii elevului, aptitudinile şi interesele sale personale (exprimate prin opţiunea pentru o anumită filieră şi specializare), diversitatea domeniilor cunoaşterii, sau perenitatea componentelor de bază ale personalităţii şi a valorilor asociate acestora, proprii unei societăţi democratice.

b) *Principiul coerenţei*. Acesta vizează caracterul omogen şi echilibrat al parcursului şcolar, având în vedere integrarea verticală şi orizontală a ariilor curriculare în cadrul fiecărei filiere, profil şi specializare, exprimată în raporturile procentuale între ariile curriculare şi disciplinele de studiu.

c) *Principiul funcționalității* presupune organizarea parcursului şcolar pe cicluri curriculare, care să respecte caracteristicile de vârstă, interesele şi motivațiile elevilor. Acest principiu, coroborat cu strategiile de organizare internă a curriculum-ului, a determinat structurarea procesului de învățare pe cicluri curriculare (periodizări ale şcolarității), care se suprapun peste structura sistemului de învățământ, cu scopul de a focaliza obiectivele majore ale fiecărei etape şcolare şi de a regla prin modificări curriculare procesul de învățământ. Ciclurile curriculare asigură continuitatea în momentul trecerii de la o treaptă de şcolarizare la alta prin conexiuni explicite la nivelul curriculum-ului, corelarea structurii curriculare cu vârsta psihologică, transferul de metode şi procedee didactice.

d) *Principiul egalizării şanselor*, care vizează oferta de oportunități echivalente de continuare a şcolarizării în condițiile unor parcursuri şcolare diferențiate. Acest principiu solicită un raport adecvat între trunchiul comun şi disciplinele *la decizia şcolii*.

e) *Principiul flexibilității parcursurilor individuale*, este concretizat prin oferta de pachete opționale, la nivel central sau local pentru fiecare arie curriculară.

Noul curriculum face loc opțiunilor elevilor, permite o reală instruire multidisciplinară, o interdisciplinaritate efectivă în abordarea conținuturilor, orientând formarea elevilor în direcția celor patru capacități de bază ale viitorului specialist în informatică, dintre care pot fi amintite:

- formarea gândirii algoritmice (capacitatea de *abordare sistemică* a problemelor);
- capacitatea de abstractizare;
- capacitatea de comunicare imediată şi eficientă;
- capacitatea de exploatare a facilităților oferite de tehnologiile informaționale moderne.

Descentralizarea curriculară încurajează parcursurile individuale de învățare şi spulberă mitul obligativității şi uniformității parcurgerii conținuturilor, oferind cadrelor didactice o flexibilitate de decizie şi acțiune. În acest sens, organizarea planului de învățământ pentru gimnaziu ca un plan cadru, cu arii curriculare conținând un trunchi comun (discipline şi ore comune pentru toți elevii) şi un curriculum la dispoziția şcolii (cu parcursuri diferențiate în funcție de resursele unității şcolare), au plasat cumva disciplinele de informatică într-o postură mai „marginalizată”.

1.4. Programele şcolare (analitice, curriculare)

Programele şcolare au trecut printr-un proces complex de elaborare şi revizuire în viziune curriculară, presupunând o reproiectare interactivă a obiectivelor, conținuturilor, activităților de învățare şi a principiilor şi metodelor de evaluare. Programele şcolare (analitice) stabilesc *conținutul disciplinelor (de informatică)*, pentru fiecare an de studiu şi materie, pe nivele, filiere, profiluri şi specializări, precum şi pe forme de învățământ. Acestea precizează **ce** cunoştințe, priceperi, deprinderi trebuie să-şi însuşească elevii în anul de studiu respectiv şi care este succesiunea în care trebuiesc dobândite acestea. Programele sunt elaborate de către Comisia de specialitate a **MECT**, sub coordonarea *Consiliului Național pentru Curriculum* (**CNC**), din care fac parte cadre didactice cu experiență din învățământul universitar, profesori de liceu cu rezultate deosebite în activitatea de la catedră, inspectori şcolari din inspectoratele județene şi ai **MECT**, psihopedagogi şi cercetători din *Institutul de Ştiințe ale Educației* (**ISE**). Realizate în concordanță cu noile planuri cadru, urmărind o descongestionare rațională a conținuturilor, actualele programe şcolare reprezintă o adevărată revoluție didactică în ceea ce priveşte conceptele de formare a competențelor de nivel superior, de învățare în clasă, de studiu în grup, de învățare asistată de calculator, de autodocumentare etc. O programă şcolară adecvată este rezultatul unui exercițiu colectiv, desfăşurat sub semnul unui profesionalism specific, dar nu exclude inovația curriculară locală, la nivelul individului sau colectivului didactic. Este imperios necesară parcurgerea următoarelor etape:

- elaborarea individuală (propuneri de programă), care se face de către colectivele de catedră ale unităților de învățământ sau de către cadrele didactice, individual;
- analiza propunerilor şi elaborarea colectivă; aceasta presupune stabilirea formei finale a unei propuneri de programă de către echipe de lucru, stabilite, de regulă, de către Comisia de specialitate a **MECT**;
- prezentarea şi argumentarea în fața Comisiei de specialitate a **MECT** a formei stabilite de către echipele de lucru.

După o perioadă de câteva săptămâni, răgaz în care fiecare membru analizează programa rezervată subcomisiei din care face parte, aceasta este rediscutată şi i se aplică modificările necesare. Programele astfel finalizate sunt supuse aprobării **CNC**. Elaborarea programelor şcolare trebuie să îndeplinească anumite cerințe de ordin ştiințific, psihologic, didactic şi metodic. Programa şcolară pentru un obiect de studiu trebuie să conțină măcar:

- O notă informativă cu privire la scopurile şi obiectivele predării, indicații relative la ordonarea materiei şi repartizarea orelor pe capitole, subcapitole, teme ş.a.m.d.

- Îndrumări cu privire la folosirea manualelor, materialului bibliografic etc.
- Materia de predat, detaliată şi prezentată în succesiunea abordării ei, numărul de ore afectat lucrărilor scrise, recapitulărilor, evaluării, ore la dispoziţia profesorului ş.a.m.d.
- Modalităţi de folosire a manualelor şi a materialului documentar complementar (culegeri de probleme, documentaţii de firmă, manuale de utilizare ş.a.m.d.).

Programa actuală nu prevede un număr fix de ore pentru fiecare temă în parte aceasta fiind lăsată la aprecierea profesorului, în funcţie de particularităţile claselor şi de condiţiile specifice de predare. Preocupările privind elaborarea programelor şcolare pentru disciplinele de Informatică sunt îndreptate spre îmbunătăţirea programelor în sensul punerii de acord a conţinuturilor cu cerinţele sociale. Astfel, se doreşte să se realizeze o pregătire a elevilor în direcţia satisfacerii cerinţelor necesare integrării rapide a absolvenţilor în activitatea economică. Dinamica conţinuturilor este o cerinţă esenţială pentru programa şcolară a disciplinelor de informatică, necesară „menţinerii pasului" cu progresele realizate în domeniu. Trebuie să remarcăm şi faptul că există o tendinţă (cu efecte nu tocmai benefice după opinia noastră) de realizare a unei programe la un nivel ştiinţific foarte ridicat. Nu trebuie scăpat din vedere nici un moment faptul că o programă şcolară, concepută şi aprobată, este obligatorie pentru toţi elevii, iar aceştia nu sunt toţi foarte dotaţi şi motivaţi. Elaborarea curriculum-ului este însă un proces continuu, care marchează perioade de schimbări profunde, dar care ţinteşte îndelungi perioade de stabilitate. Noul curriculum îşi propune să realizeze stabilitatea printr-un echilibru între componenta naţională (care vizează trunchiul comun) şi componenta locală (care vizează oferta curriculară a şcolii). Elaborarea curriculum-ului local devine astfel o componentă esenţială a activităţii didactice, o rezultantă a eforturilor reunite a conducerii şcolii, cadrelor didactice, elevilor, părinţilor precum şi a altor parteneri sociali viabili.

2. Teoria instruirii

Vom începe această secţiune cu o luare de atitudine asupra unei probleme suficient de controversate.

2.1. Manualele şcolare

Crearea pieţei libere a manualelor şcolare şi trecerea de la manualul unic la cel alternativ a însemnat un pas înainte prin înlăturarea unei politici de monopol. Dezideratul

realizării a (cel puțin) trei manuale alternative la fiecare disciplină este o intenție de a atinge un standard minim de diversitate şi calitate. Manualul şcolar reprezintă mijlocul didactic **de bază** folosit în procesul de învățământ, este *profesorul la purtător* al elevului şi principalul material bibliografic al acestuia (deşi în Informatică lucrurile nu stau întotdeauna tocmai aşa). Manualul exemplifică (printre altele) detaliat conținutul programelor şcolare, funcția lui principală fiind desigur aceea de informare a elevului, mijlocul său principal de documentare. Autorii de manuale trebuie să țină seama că acesta ar trebui nu numai să-l ajute pe elev să învețe (Informatică), ci şi să-l obişnuiască cu munca/studiul individual. Manualul trebuie să îndrume elevul spre o gândire independentă şi să-l determine la continuarea efortului creator. Trebuie în acelaşi timp să nu-l inhibe, sau să-l orienteze (voit sau nu) în anumite direcții. Desigur că o mare parte dintre funcțiile manualului pot fi preluate în învățământul de Informatică de către calculator. Acesta poate fi privit şi ca manual şi ca profesor, exercitându-şi atât funcția de comunicare de cunoştințe cât şi cea de verificare, dar nu încă în mod exhaustiv şi pe aceea de evaluare permanentă a gradului (relativ) de acumulare a cunoştințelor. Dacă discutăm despre *documentațiile de firmă,* opinăm că acestea nu se ridică (şi nici nu trebuie) la nivelul didactic al manualelor şcolare, rolul lor fiind pur informativ şi destinat nu neapărat specialiştilor sau viitorilor cunoscători. Tehnicile de învățare şi evaluare a nivelului atins, corectarea deprinderilor şi completarea cunoştințelor (simultan cu descoperirea lipsurilor), nu le poate realiza decât „profesorul". Manualul are încă un rol deosebit şi anume acela de *măsură a gradului de profunzime în abordarea noțiunilor, precum şi a domeniului ca atare.* El ne poate indica până la ce grad de complexitate şi de detaliu trebuie întreprins demersul didactic. Apariția manualelor alternative scoate în evidență, odată în plus, diversitatea punctelor de vedere în această privință. Rolul manualului ca mijloc de comunicare de cunoştințe se diminuează continuu în învățământul modern, locul lui fiind luat de alte mijloace didactice, mai eficiente, mai atractive: mijloacele audiovizuale specifice Informaticii, calculatoarele cu echipamente periferice speciale, sisteme multi-media, utilizarea unor suporturi de mare capacitate şi cu posibilități rapide de acces şi de (re)găsire a informației (casete audio-video, CD-uri, DVD-uri, teletextul, INTERNET-ul, telefonia mobilă etc.). Credem că profesorul şi manualul, ca *surse didactice consacrate*, nu pot fi decât parțial înlocuite. Toate mijloacele anterior enumerate sunt doar auxiliari mai mult sau mai puțin eficienți, în funcție de domeniul şi disciplina abordată. Transformările societății româneşti din ultimii ani, dezvoltarea şi răspândirea Informaticii, impun o pregătire diversificată a tinerilor în acest domeniu. Disciplina **Informatică**, din cadrul profilului *Matematică-Informatică* trebuie astfel să asigure dobândirea unor cunoştințe de informatică la nivel de cultură

generală, necesare continuării studiului şi a unor cunoştinţe cu caracter aplicativ utile în societatea în care vor trăi. În cadrul acestui profil sunt în general prevăzute (la disciplina *Tehnologia Informaţiei*, din cadrul trunchiului comun), 2, 3, 4, respectiv 4 ore (în clasele a IX-a, a X-a, a XI-a, respectiv a XII-a). Pornind de la faptul că nu există domeniu de activitate unde să nu se prelucreze şi să nu se transmită informaţii atât în cadrul domeniului respectiv cât şi spre exteriorul lui, *informaţia este foarte preţioasă, ea trebuie stocată, prelucrată şi transmisă în condiţii care să asigure corectitudine şi exactitate, adică la un nivel profesional*. Indiferent de profesia pe care o va alege un tânăr, cu siguranţă va avea nevoie de **cunoaşterea modului de utilizare a unui instrumentar informatic**. Volumul cunoştinţelor şi deprinderilor necesare va depinde desigur de domeniu, de exigenţele şi cerinţele concrete. Este însă o nevoie stringentă de iniţierea tinerilor din toate şcolile în utilizarea calculatoarelor la un nivel profesional, pe care îl numim azi, doar, ***nivel de cultură generală***. **Dezvoltarea gândirii algoritmice** este un prim obiectiv la realizarea căruia Informatica contribuie esenţial şi eficient. Asemenea Matematicii, Informatica *dezvoltă gândirea* (raţionamentul) care în şcoală, dar şi în viaţa de zi cu zi, are un rol esenţial în procesul de învăţare, în formarea caracterului şi a personalităţii. Aceasta nu se leagă doar de cunoştinţele de programare, ci şi (aşa cum am menţionat deja) de cunoştinţele referitoare la gestionarea bazelor de date, la utilizarea editoarelor de texte etc. Informatica, prin specificul ei, este esenţial legată de lucrul individual cu un calculator şi contribuie la **dezvoltarea deprinderii de a lucra individual**. Pe de altă parte, prin intermediul reţelelor de calculatoare este posibil un schimb de informaţii mult mai eficient decât prin orice altă metodă clasică. **Educarea elevilor în spiritul unei activităţi desfăşurate în grup**, în colaborare, se finalizează prin predarea *Informaticii orientată pe proiecte*. Realizarea unor aplicaţii complexe impune lucrul în grup, modularizarea programului şi păstrarea contactelor cu ceilalţi membri ai grupului. În şcoală se pot crea condiţii similare lucrului din viaţa reală unde activităţile nu se desfăşoară izolat. Aplicaţiile, proiectele, dar şi producţia propriu-zisă sunt întrepătrunse cu o serie de faze de lucru în care calculatorul este un instrument de neînlocuit. Obişnuirea elevilor cu responsabilităţi privind finalizarea propriei munci şi asigurarea înlănţuirii unor elemente realizate în paralel, îi va pregăti pentru o activitate pe care cu siguranţă o vor întâlni în viitor. **Educarea elevilor pentru realizarea unor produse utilizabile, dezvoltarea spiritului inventiv şi creator,** apare ca un obiectiv impus de sistemul economic în care trăim. Indiferent de conţinutul aplicaţiei, *ceea ce realizează elevul, trebuie să funcţioneze*, trebuie să fie utilizabil. Altfel spus, *trebuie să aibă toate calităţile unui produs comercial.* Datorită implicaţiei pe care Informatica o are azi în toate profesiunile, rezultă caracterul ei

interdisciplinar. Informatica nu poate fi privită numai ca o disciplină independentă şi nu poate fi ţinută între bariere create artificial. În diverse domenii de activitate, rezolvarea problemelor concrete impune foarte des o fază de modelare. Informatica este printre puţinele discipline care oferă instrumentar adecvat pentru **învăţarea modelării**. De asemenea, Informatica pune la dispoziţie cele mai spectaculoase **posibilităţi de simulare virtuală** care este o parte a modelării (**neclasică şi necostisitoare**). Elevii trebuie să înţeleagă **conexiunile dintre Informatică şi societate** şi să fie capabili să se adapteze dinamicii schimbărilor determinate de aceste conexiuni.

Prezentăm în continuare o comparaţie între caracteristicile curriculum-ului anterior şi cele ale curriculum-ului actual.

Curriculum-ul anterior:

- A fost centrat pe conţinuturi.
- Formularea obiectivelor viza în mod direct atestarea profesională a absolventului.
- Conţinuturile învăţării erau aceleaşi pentru toţi elevii.
- Absenţa cooperării între elevi în realizarea unei aplicaţii era o regulă.
- Existau conţinuturi didactice fixe, neadaptabile la resursele locale.

Curriculum-ul actual permite**:**

- Centrarea pe raţionalizarea activităţilor de învăţare, în funcţie de obiectivele cadru şi de obiectivele de referinţă.
- Formularea obiectivelor este realizată în termeni de competenţe şi de capacităţi individuale.
- Curriculum-ul la decizia şcolii oferă o paletă largă de activităţi prin care elevul îşi poate acoperi sfera proprie de interese.
- Se permite încurajarea cooperării între elevi prin activităţi de grup cu asumarea de roluri individuale pentru realizarea unei aplicaţii.
- Conţinuturile sunt adaptabile la resursele locale.

În cele de mai jos prezentăm doar o parte (nu neapărat actualizată) a ceea ce există şi a ceea ce se doreşte a se întâmpla în anii următori. Pentru informaţii suplimentare şi „puneri la zi” se pot consulta adresele: http://cnc.ise.ro, http://www.edu.ro/pinf.html, http://www.edu.ro/ptehinf.html, http://www.coe.int/, http://www.edu.ro/download/strategiepreuniv.pdf,

Organizarea pe ani de studiu a disciplinei *Informatică*:

An de studiu	Trunchiul comun		Curriculum la decizia şcolii (Propuneri)
	1 oră curs	1 oră laborator	2 ore laborator
Clasa a IX-a	Algoritmi şi elemente de limbaje de programare	Sisteme de operare şi aplicații de programare	Programare avansată Limbaj de programare concret Limbaj de asamblare concret Tehnoredactare (Word, Latex) Foi de calcul HTML Grafică pe calculator Sisteme de operare Aplicații multimedia
Clasa a X-a	Metode şi tehnici de programare	Programare modulară Recursivitate Metoda Backtracking	Programare avansată Limbaj de programare concret Limbaj de asamblare concret Tehnoredactare (Word, Latex) Foi de calcul HTML Grafică pe calculator Sisteme de operare Aplicații multimedia Programare orientată pe obiecte
Clasa a XI-a	Teoria grafurilor	Alocare dinamică Aplicații elementare cu grafuri şi arbori	Programare avansată Limbaj de programare Limbaj de asamblare Tehnoredactare Foi de calcul HTML Grafică pe calculator Sisteme de operare Aplicații multimedia Programare orientată pe obiecte Teoria grafurilor

			Rețele de calculatoare Programare pe INTERNET Aplicații de proiectare
Clasa a XII-a	Baze de date	Proiectarea şi întreținerea bazelor de date	Programare avansată Limbaj de programare Limbaj de asamblare Tehnoredactare Foi de calcul HTML Grafică pe calculator Sisteme de operare Aplicații multimedia Programare pe obiecte Teoria grafurilor Rețele de calculatoare Programare pe Internet Aplicații de proiectare Baze de date Criptografie Analiză numerică

Metodologia de aplicare a programei pentru disciplina *Informatică* trebuie să țină cont de faptul că studiul Informaticii la profilul *Matematică-Informatică* are atât un caracter teoretic cât şi practic fiind organizat după cum urmează:

- în trunchiul comun, din totalul de (două) ore aprobate, o oră cu caracter teoretic se poate desfăşura în clasă sau laborator cu întreg colectivul clasei, iar cea de a doua oră cu caracter practic se va desfăşura în laboratoarele de informatică, pe grupe de 10-15 elevi, fiecare grupă fiind asistată de un „profesor”;
- în curriculum-ul *la decizia şcolii* cele două ore se vor organiza în laborator pe grupe de 10-15 elevi fiecare grupă fiind asistată de unul sau doi profesori în funcție de specificul modulului ales (de exemplu, modulul de *Grafică pe calculator* poate fi predat de o echipă formată dintr-un profesor de desen şi unul de specialitate).

Profilul *Matematică-Informatică* poate funcționa în licee care dispun de minim un laborator de Informatică dotat corespunzător. Numărul de laboratoare trebuie să asigure acoperirea orelor de laborator solicitate atât de trunchiul comun cât şi de curriculum-ul la decizia şcolii. Programa pentru disciplina Informatică, profilul *Matematică-Informatică,* este orientată pe obiective, profesorul având posibilitatea de a alege activitățile specifice atingerii acestora. Conținutul învățării pentru curriculum-ul obligatoriu este conceput astfel încât să asigure un bagaj minim de cunoştințe şi deprinderi din domeniul informaticii în timp ce curriculum-ul la decizia şcolii poate oferi module derivate din materia studiată, teme care nu sunt incluse în programa de trunchi comun sau teme integratoare pentru arii curriculare cu aplicabilitate în Informatică.

Extensiile de programă marcate cu * pot fi abordate în orele din trunchiul comun sau în curriculum-ul la decizia şcolii în funcție de nivelul elevilor şi de dotarea existentă. Ceea ce urmează sunt doar exemplificări (deduse din indicațiile **MECT**).

OBIECTIVE CADRU (pentru modulul *Informatică*)

1. **Dezvoltarea deprinderilor de utilizare a unui sistem de calcul şi a unor produse soft de largă răspândire.**
2. **Dezvoltarea gândirii algoritmice, a spiritului inventiv şi creator.**
3. **Dezvoltarea deprinderilor necesare activităților individuale şi în echipă.**
4. **Conştientizarea conexiunilor dintre Informatică şi societate.**

INFORMATICĂ

Clasa a IX-a

A. Obiective de referință şi exemple de activități de învățare

Dezvoltarea deprinderilor de utilizare ale unui sistem de calcul şi ale unor produse soft de largă răspândire

Obiective de referință

1. Să cunoască componentele unui calculator şi rolul lor.

2. Să cunoască modul de utilizare a unui calculator.
3. Să definească noțiunea de sistem de operare şi funcțiile lui.
4. Să cunoască elementele de interfață dintre sistemul de operare şi utilizator.
5. Să utilizeze un program de asistență a sistemului de operare studiat (ex. Norton Commander, File Manager, Explorer, etc.)
6. Să cunoască şi să folosească accesoriile sistemului de operare.
7. Să creeze documente simple cu ajutorul unui editor de text.
8. Să creeze imagini simple cu ajutorul unui editor de imagini.

Exemple de activități de învățare

1. Prezentarea structurii unui sistem de calcul.
2. Explicarea funcțiilor unui sistem de calcul.
3. Explorarea resurselor fizice şi logice ale unui sistem de calcul.
4. Clasificarea componentelor fizice şi logice în funcție de rolul lor.
5. Explicarea funcționării diferitelor componente hard şi soft.
6. Exerciții de utilizare a tastaturii şi mouse-lui.
7. Utilizarea unui calculator (ca sistem independent sau în rețea).
8. Învățarea unor comenzi necesare deschiderii/închiderii sesiunii de lucru.
9. Prezentarea noțiunii de sistem de operare, a funcțiilor lui şi a rolului acestuia în funcționarea calculatoarelor.
10. Prezentarea modului de organizare a datelor pe suport magnetic şi a principalelor operații asupra fişierelor şi directoarelor.
11. Prezentarea elementelor de interfață dintre sistemul de operare şi utilizator şi folosirea Help-ului pentru autoinstruire.
12. Prezentarea facilităților oferite de programele de asistență a sistemului de operare;
13. Exemplificarea şi exersarea modului de utilizare a unui program de asistență a sistemului de operare.
14. Prezentarea accesoriilor sistemului de operare.
15. Prezentarea editorului de text ales.
16. Redactarea documentelor tip scrisoare, cerere, adeverință, curriculum vitae etc.
17. Prezentarea editorului de imagini ales.
18. Combinarea elementelor de text şi a imaginilor în acelaşi document.

Dezvoltarea gândirii algoritmice, a spiritului inventiv şi creator

Obiective de referinţă

1. Să descrie în termeni algoritmici anumite activităţi.
2. Să analizeze enunţul unei probleme (să identifice datele de intrare şi datele de ieşire. Să aleagă tipul datelor, să descopere relaţiile existente între date).
3. Să reprezinte un algoritm cu ajutorul schemelor logice şi/sau pseudocodului.
4. Să respecte principiile programării structurate în procesul de elaborare a algoritmilor.
5. Să utilizeze un mediu de programare (**Pascal** sau **C**).
6. Să implementeze algoritmii reprezentaţi prin scheme logice şi/sau pseudocod în limbaj de programare.
7. Să prelucreze date structurate la nivel de componente şi la nivel de structură.
8. Să lucreze cu fişiere text.

Exemple de activităţi de învăţare

1. Discuţii despre activităţi cotidiene şi modelarea acestora sub forma unei secvenţe bine definite de paşi.
2. Combinarea unor activităţi elementare (paşi) pentru obţinerea anumitor activităţi complexe în funcţie de scopul propus.
3. Explicarea conceptului de algoritm şi a caracteristicilor algoritmilor.
4. Explicarea diferenţei existente între informaţii care se materializează în date concrete şi cele care determină calea de rezolvare a unei probleme.
5. Descrierea unui algoritm în limbaj natural.
6. Prezentarea obiectelor cu care operează algoritmii (date, variabile, operaţii, expresii).
7. Descompunerea datelor în date de intrare şi de ieşire (eventual, de lucru).
8. Descrierea etapelor rezolvării unei probleme din punct de vedere algoritmic.
9. Prezentarea elementelor constitutive ale unei scheme logice şi a funcţiilor asociate lor.
10. Prezentarea vocabularului legat de pseudocod şi scheme logice.
11. Exersarea descrierii acestora cu ajutorul schemelor logice şi în pseudocod.
12. Prezentarea structurilor de bază: structura liniară (secvenţa), alternativă şi repetitivă.
13. Exersarea scrierii unor algoritmi simpli, folosind structuri lineare, alternative şi repetitive.
14. Prezentarea mediului de programare (facilităţi de editare, de compilare şi de rulare).

15. Familiarizarea elevului cu noțiunea de limbaj de programare şi cu modalitatea de descriere a acestuia (diagramă de sintaxă sau metalimbaj).
16. Prezentarea şi exemplificarea elementelor de bază ale limbajului de programare.
17. Activități de dezvoltare a deprinderilor de abstractizare şi organizare a informațiilor în diverse structuri de date.
18. Utilizarea intrării şi ieşirii standard.
19. Exersarea scrierii unor programe simple.
20. Codificarea structurilor de control învățate (structura liniară; instrucțiunile de atribuire şi instrucțiunile compuse; structura alternativă; structura repetitivă).
21. Exerciții de transpunere a paşilor unui algoritm în structuri de control specifice.
22. Proiectarea/modelarea unui algoritm şi implementarea acestuia.
23. Folosirea facilităților mediului în depanarea programelor.
24. Prezentarea structurilor de date standard.
25. Exerciții ce dezvoltă deprinderile de abstractizare, necesare definirii, creării şi descrierii unor structuri statice de date.
26. Implementarea structurilor de tip tablou, înregistrare etc.
27. Exerciții de prelucrare a datelor pe suport extern, în fişiere text.
28. Exerciții de manipulare a fişierelor text.
29. Evidențierea analogiilor şi diferențelor între citirea/scrierea utilizând dispozitivele standard de intrare/ieşire şi fişiere text.

Dezvoltarea deprinderilor necesare activităților individuale şi de echipă

Obiective de referință

1. Să participe la munca în echipă şi să execute în cadrul grupei sarcinile ce îi revin în funcție de aptitudinile individuale.
2. Să participe activ la dezbateri, să respecte părerea celorlalți, să asculte cu răbdare şi să exprime propriile păreri argumentate.
3. Să finalizeze individual sau în echipă o aplicație.

Exemple de activități de învățare

1. Identificarea conținuturilor activității ce urmează să se desfăşoare.

2. Dezbateri pe tema fixării rolurilor în echipă în funcție de interesele şi aptitudinile individuale.
3. Formularea unor probleme care să poată fi realizate în grupuri de elevi pe baza unor discuții preliminare şi analizarea problemei.
4. Discuții de abordare a problemelor care apar pe parcursul desfăşurării activităților.
5. Verificarea înțelegerii rezolvării unei probleme în ansamblul ei de către toți membrii grupului.
6. Prezentarea şi dezbaterea aplicațiilor realizate.
7. Încurajarea discuțiilor purtate între elevi, exprimarea şi ascultarea părerilor fiecăruia.
8. Educarea elevilor în ideea că activitatea unui informatician se finalizează cu un produs care trebuie să funcționeze conform condițiilor impuse de beneficiar, să fie însoțit de o documentație, să fie verificat, testat şi evaluat după nişte criterii standardizate.
9. Discuții cu elevii asupra necesității validării datelor.
10. Evidențierea importanței realizării unor produse program fiabile, cu interfață *prietenoasă*.

Conştientizarea conexiunilor dintre Informatică şi societate

Obiective de referință

1. Să cunoască impactul social, economic, etic şi moral al Informaticii.
2. Să identifice probleme din cadrul altor discipline în care intervin cunoştințe de programare sau de utilizare a calculatorului.
3. Să cunoască legislația în vigoare cu privire la dreptul de autor al produselor soft şi la măsurile de protecție a datelor.
4. Să perceapă interdependența între dezvoltarea instrumentelor de calcul, respectiv a conceptelor informaticii şi nevoile societății, prin perspectiva istoriei.

Exemple de activități de învățare

1. Discutarea impactului social, economic, etic şi moral al Informaticii.
2. Enumerarea unor domenii în care sunt folosite instrumente informatice.
3. Lansarea în execuție a unor aplicații specifice altor discipline.
4. Implementarea unor algoritmi simpli cu aplicații în matematică, fizică etc.

5. *Discutarea articolelor incluse în legea copyright-ului şi a semnificaţiilor acestora.
6. *Cunoaşterea prevederilor legale cu privire la accesarea neautorizată sau distrugerea intenţionată (virusarea) datelor.
7. *Lecţii despre evoluţia instrumentelor de calcul, despre inventatori şi informaticieni celebri.

B. Conţinuturile învăţării

Vom exemplifica acest lucru tot la nivel general, conform indicaţiilor **M.E.C.T.** acceptate în acest moment. Toate acestea pot fi comparate cu cerinţele discutate în **Prefaţă** şi **Introducere**, fiind oricum foarte perisabile în timp.

1. Rolul şi funcţiile sistemelor de calcul

1.1. Structura unui sistem de calcul.

1.2. Funcţiile unui sistem de calcul.

1.3. Măsuri de protecţie a muncii în timpul lucrului cu calculatorul.

2. Principalele funcţii ale unui sistem de operare (Ms-Dos, Windows, Unix)

2.1. Concepte de bază şi caracteristici generale ale unui sistem de operare (**SO**)

2.2. Elemente de interfaţă (elemente specifice de comunicare dintre utilizator şi **SO**).

2.3. Programe de asistenţă (Norton Comander, File Manager/Windows Explorer, WinSCP, etc.).

2.4. Editoare de texte (Edit, Notepad, Wordpad etc.).

2.5. Editoare de imagini (Art Studio, Paint, AdobePhotoShop), formate de imagini.

2.6. Accesorii de sistem (agendă, calculator, set de caractere, elemente multimedia etc.).

2.7. Utilizare sistem (scandisk, arhivatoare, programe antivirus).

3. Algoritmi

3.1. Enunţul unei probleme, date de intrare şi de ieşire, etapele rezolvării unei probleme.

3.2. Noţiunea de algoritm, caracteristici.

3.3. Obiectele cu care lucrează algoritmii (date, variabile, expresii, operaţii).

4. Principiile programării structurate

4.1. Structuri de bază; descrierea acestora cu ajutorul schemelor logice şi în pseudocod.

4.2. Aplicații.

4.2.1. Probleme care prelucrează date numerice.

4.2.2. Probleme care prelucrează date nenumerice.

5. Elementele de bază ale limbajului de programare

5.1. Noțiuni introductive.

5.1.1. Structura programelor.

5.1.2. Descrierea sintaxei cu ajutorul diagramelor de sintaxă.

5.2. Vocabularul limbajului.

5.2.1. Setul de caractere.

5.2.2. Identificatori.

5.2.2. Separatori şi comentarii.

5.3. Tipuri simple de date (standard).

5.4. Constante.

5.5. Variabile.

5.6. Expresii.

5.7. Citirea/scrierea tabelelor.

6. Structuri de control

6.1. Structura liniară (secvențială); instrucțiunile de atribuire şi instrucțiunea compusă.

6.2. Structura alternativă; instrucțiuni de decizie şi selecție.

6.3. Structuri repetitive; instrucțiuni repetitive.

7. Mediul limbajului de programare studiat

7.1. Prezentare generală.

7.2. Editarea programelor sursă.

7.3. Compilare, rulare, **depanare.*

8. Tipuri structurate de date

8.1. Tablouri.

8.2. Şiruri de caractere.

8.3. Tipul înregistrare.

8.4. *Alte tipuri specifice limbajului.

9. Fişiere

9.1. Tipul fişier (definire, operații).

9.2. Fişiere text.

10. Aplicații practice

10.1. Etape în realizarea unei aplicații.

10.2. Cerințe în realizarea programelor.

10.2.1. Interfață prietenoasă.

10.2.2. Protecție la date incorecte.

10.2.3. Criterii de optimalitate.

10.3.Exemple de aplicații.

10.3.1. Determinare minim/maxim.

10.3.2. Probleme de divizibilitate.

10.3.3. Sortare.

10.3.4. Căutare.

10.3.5. Probleme de prelucrare a datelor din fişiere text.

10.3.6. Prelucrări de şiruri de caractere.

Putem trece la prezentarea pe scurt a celuilalt modul important, evident tot pe un exemplu concret.

OBIECTIVE CADRU (pentru modulul *Tehnologia Informației*)

1. Utilizarea surselor informaționale şi a mijloacelor de procesare în scopul preluării, prelucrării şi prezentării informației
2. Înțelegerea dezvoltării tehnicii şi a implicațiilor tehnologiei informației asupra mediului şi societății
3. **Valorificarea termenilor de specialitate în comunicare**
4. Dezvoltarea capacității de cooperare în scopul realizării unei aplicații

Clasa a VIII-a

A. Obiectivele de referință şi exemple de activități de învățare

Utilizarea surselor informaționale şi a mijloacelor de procesare în scopul preluării, prelucrării şi prezentării informației

Obiective de referință

1. Să comunice ideile în forme cât mai variate.
2. Să înțeleagă nevoia de a întreba şi să conştientizeze rolul factorului uman în obținerea unor informații corecte.
3. Să identifice mijloacele tehnice de preluare, prelucrare şi transmitere a informației * *şi să descrie modul de utilizare a acestora*.
4. Să clasifice şi să pregătească informația în vederea prelucrării ei şi să distingă diferitele calități ale datelor în procesul de prelucrare.
5. Să aleagă cele mai potrivite căi şi aplicații pentru obținerea şi/sau transmiterea datelor.
6. Să pună în evidență operații specifice procesului prelucrării manuale de date. 7. 7. Să identifice oportunitatea recurgerii la prelucrări automate realizate cu tehnologie modernă.
8. Să dea dovadă de simț practic, etic şi estetic în abordarea unor aplicații.

Exemple de activități de învățare

1. Prezentarea informației prin diversele moduri de reprezentare (text, imagine, grafică, sunet, coduri numerice etc.).
2. Antrenarea elevilor în discuții care să se finalizeze cu identificarea cât mai multor forme de reprezentare a informațiilor.
3. Selectarea elementelor necesare realizării unui anumit obiectiv având în vedere urmările posibilelor erori.
4. Prezentarea mijloacelor tehnice de preluare, prelucrare şi transmitere a informației (telefon, copiator, aparate de fotografiat, televizor etc.).
5. Familiarizarea cu modul de utilizare a aparatelor de tehnologie modernă, identificarea modului în care acestea influențează viața.
6. Diverse exemplificări de prelucrare a datelor în care să se evidențieze diferența dintre datele de intrare şi cele de ieşire - date care urmează să fie prelucrate, date rezultate din procesul de prelucrare (sondaje de opinie, chestionare, tabele, grafice, sinteze etc.).
7. Interpretarea, analiza şi afişarea informației verificând totodată corectitudinea şi plauzibilitatea ei.
8. Realizarea unor prelucrări manuale de date urmărind finalitățile, formele şi etapele fiecărei aplicații.

9. Exemplificarea unor aplicații care, în lipsa echipamentelor tehnologice, nu s-ar putea realiza la nivelul cerințelor actuale.
10. Compararea diferitelor produse (texte, desene) realizate de elevi în funcție de forma şi conținutul acestora.

Înțelegerea dezvoltării tehnicii şi a implicațiilor tehnologiei informației asupra mediului şi societății

Obiective de referință

1. Să prezinte şi să explice semnificația informației în societate.
2. Să cunoască impactul social, economic, etic şi moral al utilizării tehnologiei informației.
3. Să interpreteze efectele inovațiilor şi ale descoperirilor din domeniul tehnologiei informației (ca avantaje şi dezavantaje).
4. Să enumere domeniile de cunoştințe din cadrul altor discipline în care se pot utiliza instrumente de tehnologia informației.
5. Să utilizeze instrumentele tehnologiei informației şi comunicării (editare de text, comunicare prin poşta electronică, etc.).
6. Să folosească instrumentele ajutătoare pe parcursul elaborării temelor de lucru individuale.
7. Să cunoască legislația în vigoare cu privire la dreptul de autor al produselor soft şi la măsurile de protecție a datelor.
8. Să cunoască istoricul instrumentelor informatice şi a Informaticii ca ştiință.

Exemple de activități de învățare

1. Discuții purtate pe teme care conştientizează noțiunea de informație pe bază de exemple din viața elevilor.
2. Exemplificarea contextelor sociale în care informația contribuie la ameliorarea standardelor, luarea deciziilor etc.
3. Punerea în evidență a unor situații practice în care informațiile trebuie stocate în vederea prelucrării, modalitățile de păstrare necesitând reprezentarea acestora sub o formă concretă.

4. Selectarea elementelor necesare realizării unui anumit obiectiv având în vedere urmările posibilelor erori.
5. Discutarea impactului social, economic, etic şi moral al utilizării tehnologiei informației;
6. enumerarea unor domenii în care sunt folosite instrumente de tehnologia informației;
7. studiu de caz, investigație pentru sesizarea aspectului calitativ al muncii de prelucrare automată a datelor.
8. Discuții privind avantajele şi dezavantajele care apar ca urmare a utilizării tehnologiei informației.
9. Exerciții de identificare a oportunității utilizării tehnologiei informației pentru îmbogățirea cunoştințelor şi mărirea eficienței activităților întâlnite în cadrul altor discipline.
10. Redactarea unei scrisori, expedierea ei prin poşta electronică.
11. Utilizarea editorului de text, a foilor de calcul, a aplicațiilor tip baze de date, INTERNET-ului pentru activități întâlnite în cadrul altor discipline.
12. Exemplificarea folosirii facilităților de autodocumentare specifice (Help) puse la dispoziție de mediul de lucru.
13. Discutarea articolelor incluse în legea copyright-ului şi a semnificațiilor acestora.
14. Istorisiri despre evoluția instrumentelor tehnologiei informației, despre personalități celebre din domeniu.

Valorificarea termenilor de specialitate în comunicare

Obiective de referință

1. Să utilizeze corect simbolurile, prescurtările şi terminologia specifică domeniului informatic.
2. Să recunoască semnificația şi sfera de utilizare a unor termeni din domeniul tehnologiei informației.

Exemple de activități de învățare

1. Găsirea unor sinonime pentru termenii de specialitate asimilați care să probeze înțelegerea acestora.
2. Analiza etimologiei cuvintelor noi.

3. Crearea unui vocabular de termeni de specialitate.
4. Realizarea unor clase de noțiuni asociate domeniilor de utilizare a tehnologiei informației.

Dezvoltarea capacității de cooperare în scopul realizării unei aplicații

Obiective de referință

1. Să participe la munca în echipă şi să execute în cadrul grupei sarcinile ce îi revin în funcție de aptitudinile individuale.
2. Să participe activ la dezbateri, să respecte părerea celorlalți, să asculte cu răbdare şi să exprime propriile păreri argumentate.

Exemple de activități de învățare

1. Identificarea conținuturilor activității ce urmează să se desfăşoare.
2. Dezbateri pe tema fixării rolurilor în echipă în funcție de interesele şi aptitudinile individuale.
3. Formularea unor probleme simple care să poată fi realizate în grupuri de elevi pe baza unor discuții preliminare şi analiza problemei.
4. Discuții de abordare a problemelor care apar pe parcursul desfăşurării activităților.
5. Verificarea înțelegerii rezolvării unei probleme în ansamblul ei de către toți membrii grupului.
6. Prezentarea şi dezbaterea aplicațiilor realizate.
7. Se vor încuraja discuțiile purtate între elevi, exprimarea şi ascultarea părerilor fiecăruia.

Desigur că putem continua cu **Conținuturile învățării**, dar credem că este suficient.

2.2. Structura lecției

Forma fundamentală de organizare individualizată a procesului de instruire este **lecția**, indiferent de durata sa temporală. La conținutul propriu-zis al unei lecții se adaugă aplicarea metodelor de predare pe care profesorul le va alege cât şi obiectivele pe care acesta şi le

propune. Nu poate fi considerată lecție ceva care nu *leagă* ceea ce s-a studiat înainte, cunoştinţele anterior dobândite, de cunoştinţele care trebuiesc transmise în continuare. Lecția are un caracter unitar prin conținutul ei, prin procedeele ce se aplică, prin gradul de participare a elevilor la procesul instructiv-educativ. Aşa cum *preambulul* trebuie să conțină o prezentare clară a ceea ce urmează, orice lecție trebuie încheiată printr-un *rezumat*, o *recapitulare* a întregului volum de cunoştinţe abordate pe întreg cuprinsul lecției şi o *fixare,* prin care să se finalizeze activitatea întreprinsă. Ar trebui anticipate necesitatea introducerii unor *noi noțiuni* şi *planul de abordare* a lecțiilor următoare. Considerăm că o asemenea **unitate de învățare ar trebui să dureze 90-100 minute, fără întreruperi**. Lecția nu este numai o formă de organizare a activității de predare/învățare şi o succesiune de etape bine stabilite şi (de dorit) realizate. Evenimentele imprevizibile, apariția unor particularități specifice care trebuie „stăpânite" sunt inevitabile. O cerință metodică este clasificarea lecțiilor: *de comunicare/transmitere de cunoştinţe, de studiu individual, de descoperire, de verificare, de recapitulare* etc. Delimitările nu sunt însă stricte, fiecare lecție fiind o împletire (care se doreşte armonioasă) de metode şi tehnici care concură la realizarea obiectivelor propuse, raportul în favoarea uneia sau altei dintre metode fiind greu de stabilit în final şi cu atât mai mult inițial. Vom puncta totuşi câteva dintre **momentele esențiale** ale desfăşurarii unei lecții, subliniind relativitatea acestora (ca dimensiuni de timp; ca importanță; ca ordine):

- *Momentul organizatoric* impune, în primul rând, verificarea prezenței şi a condițiilor de desfăşurare (existența materialului didactic necesar, incluzând aici calculatoare, soft etc.). Ideal ar fi ca aceasta să fie făcută în pauza dintre ore şi de către un *personal specializat*. Din acest motiv, **pauzele ar trebui să fie de minimum 20 minute**. Ideală ar fi verificarea temei de acasă şi identificarea dificultăților întâmpinate în efectuarea ei.
- Elevii sunt apoi *ascultați* din materia predată în lecția anterioară, căutându-se să se înlăture *anomaliile de înțelelegere* apărute în procesul de asimilare.
- Se *predă* lecția nouă (sau, echivalent, dacă este vorba de fixarea a ceva anterior).
- Se *fixează* cunoştinţele (noi) prin (alte) exerciții.
- Se stabileşte *tema pentru acasă*.

O lecție poate fi apreciată ca necorespunzătoare dacă, de exemplu, se „pierde timpul" cu momentul organizatoric, inclusiv cu verificarea temei de acasă şi cu măsurile luate de profesor privitoare la nerealizarea acesteia. Cel mai mult timp trebuie afectat comunicării

cunoştinţelor noi şi fixării acestora prin exerciţii. Tema pentru acasă nu trebuie dată în grabă (în pauză sau când se sună).

Observaţie. Volumul de muncă necesar efectuării temelor pentru acasă trebuie să se înscrie în limite rezonabile (există suficiente recomandări legale pentru sarcinile suplimentare). Un număr mai mare de exerciţii duce la lucru de mântuială, copieri, abandonarea întregii teme, refuz faţă de abordarea temei. Tema trebuie să fie în corespondenţă cu posibilităţile elevilor şi legată de însuşirea şi aplicarea cunoştinţelor predate. Ea trebuie dată diferenţiat, atunci când între elevii aceleaşi clase sunt diferenţe mari în ceea ce priveşte capacitatea sau pregătirea lor. Tema trebuie să fie însoţită de explicaţii ajutătoare, de indicaţii potrivite. Când tema presupune artificii de calcul sau cere o pricepere deosebită, trebuie ca elevilor să li se atragă atenţia asupra acestui aspect (de exemplu, prin exerciţii cu *, adică dificile). Mulţi elevi învaţă pe de rost metodele de rezolvare a unor probleme şi îşi formează şabloane pe care le aplică automat. Cu siguranţă şi algoritmii importanţi, rezultat al analizei şi cercetării îndelungate, trebuie reţinuţi, dar uzând-se de *logica internă* a acestora. Profesorul are (şi) obligaţia să înveţe pe elevi cum să-şi facă tema, nu să creeze un climat care determină ca necesitate apariţia *meditatorului* (mediator păgubos între elev şi profesor şi nu o prelungire a acestuia din urmă, în cazuri extreme). Temele de acasă îşi ating scopul doar dacă pot fi controlate în permanenţă de către profesor.

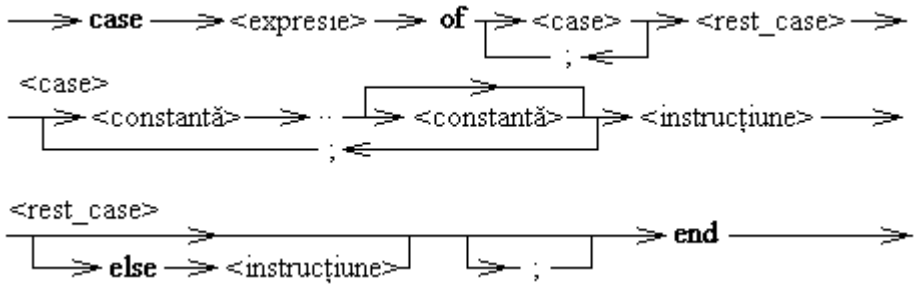
2.3. Calitatea cunoştinţelor asimilate

Procesul de comunicare al cunoştinţelor trebuie să aibă drept efect formarea de reprezentări corecte despre lucruri şi fenomene reale, însuşirea de noţiuni care ajută la înţelegerea legilor care reglementează raporturile dintre fenomenele realităţii şi care permit exprimarea acestor raporturi într-un mod clar. *Formalismul excesiv* este unul dintre pericolele care pândesc procesul instructiv/educativ şi el se manifestă prin:

- *Lipsa* legăturii evident exprimate dintre formă şi conţinut.
- *Memorarea* mecanică a cunoştinţelor şi predominarea formei exterioare asupra esenţei conţinutului (schimbarea notaţiei poate provoca uneori adevărate tragedii).
- *Predominarea* memorării asupra înţelegerii.
- *Supremaţia* şablonului asupra inventivităţii.
- *Ruperea* teoriei de practică.

Evitarea formalismului excesiv se realizează mai ales printr-o înțelegere deplină a fenomenului abstractizării, o urmărire şi o conştientizare a scopului, a însemnătății abordării temei şi o subliniere a consecințelor realizării ei. Trebuie să limităm folosirea şabloanelor, chiar dacă există situații identice care se repetă. *Raționamentul logic trebuie să ne însoțească paşii în permanență*.

Exemplu. Dacă ne referim la prezentarea sintaxei unui limbaj de programare, utilizarea *diagramelor de sintaxă* (<41>) poate fi un impediment în înțelegerea limbajului (deşi experiența proprie ne-a furnizat şi câteva aplicări cu succes). Iată cum este prezentată sintaxa structurii alternative multiple de tip *CASE* printr-o astfel de diagramă:



Ca alternativă profesorul poate opta pentru varianta:

case (<expresie>) **of**
    val_1: <instrucțiune$_1$>;
    val_2: <instrucțiune$_2$>;
    .....
    val_n: <instrucțiune$_n$>
else
    <instrucțiune>
end;

unde *<expresie>* este o expresie de tip ordinal, $val_1, val_2, ..., val_n$ sunt valori sau intervale de valori de acelaşi tip cu *<expresie>*, iar *<instrucțiune$_1$>, <instrucțiune$_2$>,..., <instrucțiune$_n$>, <instrucțiune>* sunt instrucțiuni simple sau compuse. *<instrucțiune>* poate fi şi vidă, caz în care lipseşte şi cuvântul rezervat **else**.

Un alt impediment în calea înțelegerii (generat de abstractizare) se constată la elevii slabi care la construirea primelor programe într-un limbaj de programare, din dorința lor justificată de a menține pasul cu ceilalți elevi din clasă, depun un efort suplimentar învățând pur şi simplu programele făcute în clasă pe de rost. Dacă acest viciu de tehnică de învățare nu este depistat la timp şi înlăturat, în special prin scrierea de către elev sub supravegherea profesorului de programe simple (dar altele de fiecare dată), cu greu va mai putea fi corectat.

2.4. Formarea limbajului de specialitate

Limbajul este desigur un instrument prin care oamenii comunică, fac schimb de informații, idei, se înțeleg între ei. Acesta este nemijlocit legat de gândire, înregistrând şi fixând în cuvinte rezultatele unor activități. Este implicit necesară o vorbire corectă, o exprimare lipsită de ambiguități, în orice domeniu şi cu atât mai mult în Informatică, unde limbajul natural este un intermediar important în cadrul interfeței cu mijloacele de calcul. Formarea limbajului de specialitate este (şi) o consecință a unui proces de instruire de lungă durată. Profesorul trebuie să înlăture în permanență orice greşeală de exprimare şi să clarifice orice neînțelegere a unor noțiuni, să reformuleze corect orice afirmație legată de noțiuni şi fenomene incorect exprimate. Supravegherea încă din clasele mici duce la formarea unui reflex critic, marcat de o atenție sporită atunci când cineva se exprimă incorect şi remarcarea celor mai subtile şi ascunse erori de exprimare şi interpretare. Formarea unei exprimări corecte scrise şi orale se realizează prin:

- Exprimarea corectă a profesorului însuşi, care constituie (ab initio) un model pentru elevi. De aceea profesorul trebuie să-şi formuleze cu grijă afirmațiile.
- Supravegherea permanentă a exprimării elevilor şi corectarea continuă a greşelilor lor.
- Încurajarea libertății de exprimare, cu argumentarea raționamentelor. Deseori elevii răspund telegrafic sau numai „încep" să se exprime. Cum aceştia o fac mai greoi, ei sunt uneori întrerupți şi profesorul continuă ideea formulând-o prin prisma înțelegerii şi raționamentului său. Acest mod de abordare a dialogului elev-profesor are efecte negative în legătură cu formarea limbajului de specialitate şi utilizarea lui de către elev. În plus, prin intervenția prematură a profesorului, elevului i se întrerupe firul raționamentelor sale, el făcând cu greu față efortului de a urmări şi înțelege raționamentul profesorului.

Este astfel absolut necesară asigurarea unei anumite „libertăți individuale", chiar în detrimentul unor riscuri de confuzii momentane. Putem vorbi astfel mai în amănunt despre

exprimarea fluentă în limbajul de specialitate şi exerciţiul oral. Prezentarea orală a soluţiilor unor probleme înainte de abordarea lor strict ştiinţifică are menirea de a lămuri în totalitate aspectele neclare ale problemelor. Limbajul natural este o formă de reprezentare a algoritmilor des uzitată, prin urmare o prezentare în limbaj natural a oricărei activităţi ce urmează a fi desfăşurată, clarifică şi uşurează multe situaţii limită. **exerciţiul oral** are o însemnătate deosebită din punct de vedere educativ, el educând atenţia, capacitatea de concentrare, prezenţa de spirit, iniţiativa creatoare. Exerciţiile orale îmbunătăţesc randamentul multor activităţii şi contribuie esenţial la formarea limbajului de specialitate. Sunt necesare totuşi anumite cerinţe şi precauţii în folosirea exerciţiilor orale. Astfel, ele trebuie:

- Să fie alese cu grijă şi să nu presupună un nivel ridicat de abstractizare sau acumularea unui volum mare de informaţii noi.
- Să fie prezentate gradat, să nu presupună calcule mentale lungi şi complicate.
- Să poată fi folosit un bogat material intuitiv/ilustrativ.

În final, profesorul trebuie să pună accentul pe aspectele care au şansa de a deveni ambigue.

2.5. Caietele elevilor

De regulă, elevii şi profesorii acceptă că există un caiet care conţine partea teoretică şi aplicaţiile ilustrative, iar un alt caiet este destinat exerciţiilor individuale. Dar, ce notează elevii în caiete? Dacă există manual, la predare elevii trebuie să noteze doar exemplele ilustrative şi nu partea teoretică. Ei notează doar concluzii şi o schemă simplificată a lecţiei. Când profesorul expune materia altfel decât în manual, elevii trebuie să o noteze complet. Pe de altă parte, notarea în caiete trebuie să cuprindă doar ceea ce profesorul scrie pe „tablă” (calculator personal, teletext, telefon mobil etc.). Explicaţiile orale lungi şi complicate, chiar dacă trebuie evitat a fi scrise în caiete, îşi au rolul lor. Astfel de notări sunt grele pentru elevii din clasele inferioare şi efortul lor se canalizează în direcţia notării şi nu a înţelegerii noţiunilor predate. Trebuie exclusă ideea copierii textelor din manuale pe caiete, exceptând situaţia în care se realizează o sinteză şi o sistematizare a lecţiei din manual. O atenţie specială trebuie acordată *Caietului de aplicaţii practice de laborator*. Datorită caracterului aplicativ al anumitor ore, există tendinţa de a se nota puţin şi de cele mai multe ori secvenţe izolate şi necorelate între ele. Un caiet de aplicaţii practice de laborator ar trebui să conţină la fiecare lecţie:

- Un rezumat al cunoştinţelor teoretice necesare realizării aplicaţiei practice concrete.

- Enunțul problemei a cărei rezolvare constituie obiectul activității, cu observații asupra „mediului concret”.
- Algoritmul de rezolvare, descris în limbaj natural/pseudocod /schemă logică.
- Rezolvarea implementată, sau acea parte din soluție care constituie esența rezolvării (programul sau secvențele cele mai importante, cu precizarea în clar a ceea ce s-a realizat în acea etapă);
- Un rezumat al cunoştinţelor noi dobândite în urma rezolvării problemelor.

Chiar dacă ideea copierii pe caiete a programelor întocmite la orele de aplicații practice de laborator poate fi supusă unor critici severe, aceste texte sursă constituie totuşi *biblioteca la purtător* a elevului, cea mai rapid accesibilă, cu condiția ca programele să fie însoțite de explicații corespunzătoare. Sursele programelor fără enunțul problemelor şi specificațiile de programare sunt *texte moarte*. Marele dezavantaj al metodei constă în timpul pierdut cu copierea pe „caiete” (nu excludem mijloacele electronice moderne), dar acesta este compensat de obținerea unui text sursă testat, corect şi reprezentând o implementare verificată.

3. Teoria evaluării

Credem că este benefic să ne oprim puțin asupra recapitulărilor, înainte de a vorbi mai în detaliu despre evaluare.

3.1. Repetare, recapitulare, evaluare

Repetarea materiei parcurse serveşte la împrospătarea cunoştinţelor dobândite, ajută la formarea de noi corelații, reluarea materiei predate într-un cadru mai general, uneori cu completarea unor cunoştinţe. Repetarea (lecției sau chiar lecțiilor anterioare) se poate face înainte de predarea unei lecții noi, la sfârşitul unui capitol, al unui semestru sau al anului şcolar, sau chiar cu ocazia necesității de a se susține un examen suplimentar. Recapitularea ar trebui să se realizeze după un plan dinainte stabilit. A *evalua* rezultatele şcolare înseamnă a determina, a cuantifica măsura în care obiectivele programului de instruire au fost atinse, precum şi eficiența metodelor de predare-învățare folosite. Procesul de învățământ cuprinde,

oricum, următoarele etape: **predare, învățare, evaluare**. Deşi aceste etape se desfăşoară separat, proiectarea lecțiilor nu poate fi făcută fără a avea în vedere toate aspectele legate de acestea, ele întrepătrunzându-se. După cum ştim (sau, în acest moment, bănuim), proiectarea unei lecții începe prin a se stabili obiectivele acesteia, şi are la bază programa şcolară a disciplinei. Profesorul va trebui să se încadreze în numărul de ore stabilit prin programă. Procesul de învățământ se desfaşoară într-un cadru organizat şi bine definit. Evaluarea este o componentă foarte importantă a procesului de învățământ. A *evalua* rezultatele şcolare înseamnă a determina, a cuantifica măsura în care obiectivele programului de instruire au fost atinse, precum şi eficiența metodelor de predare-învățare folosite, pe scurt înseamnă *a evalua randamentul şcolar*. Acțiunile efectuate în procesul de evaluare se referă la „măsurare, cuantificare", „interpretare rezultate" şi evident la adoptarea unor decizii corespunzătoare. Activitatea de măsurare sau cuantificare se realizează cu ajutorul unor procedee specifice cunoscute sub numele de **metode şi instrumente de evaluare**. Interpretarea şi aprecierea rezultatelor evaluării este strâns legată de metodele şi instrumentele de evaluare folosite, precum şi de factori externi ce țin de vârsta elevilor, mediul de dezvoltare al acestora etc. În mod normal, aprecierea rezultatelor evaluării va conține două elemente importante: elevii evaluați pe de o parte, disciplina şi profesorul titular pe de alta parte. De fapt, evaluarea, componentă esențială a procesului de învățământ, îndeplineşte funcții bine definite:

- *Funcția de constatare şi diagnosticare* a performanțelor obținute de elevi, explicate prin factorii şi condițiile care au condus la succesul sau insuccesul şcolar şi care sunt de o mare diversitate (psihologică, pedagogică, socială etc.). Se permite depistarea lacunelor şi greşelilor elevilor precum şi înlăturarea acestora, la timp.
- *Funcția de reglare şi perfecționare* a metodologiei instruirii pe baza informațiilor obținute din explicarea factorilor şi condițiilor care au determinat rezultatele la învățătură.
- *Funcția de predicție şi decizie* care vizează desfăşurarea în viitor a activității didactice şi performanțele viitoare ale elevilor.
- *Funcția de selecție şi clasificare* a elevilor în raport cu rezultatele şcolare obținute, aceasta permițând clasificarea şi/sau ierarhizarea elevilor.
- *Funcția formativ-educativă*, de ameliorare a metodelor de învățare folosite de elevi, de stimulare şi optimizare a învățării.
- *Funcția de perfecționare* a întregului sistem de invățământ.

Ca orice altă activitate didactică, evaluarea cunoştinţelor elevilor trebuie să respecte normele impuse de **M.E.C.T.** În general, aceste norme (directive, prevederi) stipulează o evaluare ritmică pe parcursul semestrelor. Funcţie de vârsta şi particularităţile psihointelectuale ale elevilor şi de specificul fiecărei discipline, instrumentele de evaluare pot fi:

a) probe (lucrări) scrise;
b) probe (verificări) orale;
c) activităţi practice;
d) referate şi proiecte;
e) interviuri;
f) alte instrumente stabilite de catedrele/comisiile metodice si aprobate de director sau elaborate de Minister sau de Inspectorate.

Aplicarea uneia sau alteia dintre formele de evaluare depinde în mare măsură de forma raspunsului şi de caracteristicile disciplinei respective. Cele mai răspândite forme de evaluare sunt cele orale şi cele scrise, privite la modul clasic. Aceste forme de evaluare pot fi aplicate *individual* sau *frontal*. Încercând să comparăm cele două metode, vom constata avantaje şi dezavantaje de fiecare parte. În cadrul acestei comparaţii intervine şi personalitatea cadrului didactic precum şi specificul disciplinei. O disciplină tehnică impune în general un mod de gândire şi exprimare direct, înţeles uneori ca un *mod algoritmic* de prezentare a cunoştinţelor, în timp ce o disciplină „netehnică" cere abilităţi suplimentare de exprimare a cunoştinţelor. La nivel de discuţii aceste comparări (evaluare orală, evaluare scrisă) par să fie normale şi corecte, dar practica demonstrează că nu putem renunţa la nici una dintre ele în favoarea alteia.

Verificarea orală, cea mai frecvent folosită, are anumite avantaje care o impun. În primul rând favorizează dialogul, elevul putând să-şi argumenteze răspunsurile şi să participe la o confruntare de idei cu întreaga clasă, iar profesorul poate detecta cu uşurinţă erorile şi poate interveni şi corecta „pe loc". Verificarea orală are însă şi numeroase limite: întrebările nu au toate acelaşi grad de dificultate; unii elevi sunt emotivi şi se blochează (mai ales atunci când sunt ironizaţi de către profesor); răspunsurile lor stârnesc ilaritate în clasă; timpul nu permite o verificare completă a conţinutului predat. Mai mult, comportamentul şi starea psihică a profesorului poate influenţa notarea. Majoritatea specialiştilor din doemniu consideră verificarea orală ca fiind încărcată de un grad înalt de subiectivism. Subiectivă sau nu, această metodă este printre puţinele care dau posibilitatea cadrului didactic de a corecta deficienţele de limbaj şi de abordare a cunoştinţelor, dând posibilitatea elevului de a exersa şi a expune direct noţiunile invăţate. Elevul nu are mult timp la dispoziţie pentru a-şi alege cuvintele

potrivite şi este oarecum obligat să redea cunoştinţele in mod *direct*. De asemenea această metodă dă posibilitatea cadrului didactic de a face o incursiune printre cunoştinţele elevului şi a sesiza din timp noţiunile care creează dificultăţi în înţelegerea lor corectă. Considerăm ca un mare avantaj al acestei metode crearea deprinderii de comunicare a elevului cu lumea exterioară. Un mare dezavantaj al acestei metode este ca profesorul nu are posibilitatea de a testa decât un numar restrâns de elevi, iar in cadrul unei programe aglomerate (materie multă, ore alocate puţine) profesorul se vede in situaţia de a o folosi foarte puţin, iar elevii *pot percepe disciplina respectivă ca fiind ruptă de realitate*. În fond, fiecare poate citi o carte şi fără a i se povesti dinainte acţiunea dar nu fiecare este autodidact, nu poate „puncta" esenţialul.

Verificarea scrisă se utilizează sub forma unor *lucrări de scurtă durată*, lucrări *tip obiectiv*, lucrări de una sau două ore, *semestriale* (care sunt dinainte anunţate şi pregătite şi în clasă), *lucrări scrise tip examen*. Cercetările au dovedit că evaluarea formativă în formă scrisă, după fiecare capitol, combinată cu verificările orale este deosebit de eficientă şi stimulativă. Probele scrise sunt preferate de către elevi şi profesori pentru că asigură un grad mai mare de obiectivitate la notare, oferă elevilor mai emotivi sau celor care gândesc mai lent posibilitatea de a se exprima fără a fi influenţaţi de factori perturbatori, asigură evaluarea unui număr mare de elevi, întrebările au acelaşi grad de dificultate şi favorizează realizarea comparării rezultatelor. Dezavantajele metodei sunt marcate de faptul că profesorul nu poate interveni şi corecta pe loc erorile descoperite, el urmând să o facă în clasă la discutarea lucrărilor. Elevii nu pot fi corectaţi dacă fac anumite confuzii sau când răspunsul nu este complet. Răspunsurile incomplete pot genera şi diferenţe de apreciere şi notare. Metoda de evaluare prin verificare scrisă presupune in general un grad mai mare de obiectivitate din partea cadrului didactic în momentul aprecierii lucrarii, dar are marele dezavantaj ca *rupe* comunicarea dintre elev si profesor. Această metodă îşi dovedeşte eficacitatea în momentul când este utilizată împreună cu verificarea orală sau cu metoda interviului. Verificările scrise pot fi date din lecţia curentă sau din cadrul unui capitol. Cand verificarea scrisă este din cadrul mai multor lecţii (un capitol etc.), cadrul didactic trebuie să anunţe elevii în timp util (*lucrări anunţate*), eventual să puncteze ceea ce se urmăreşte în mod special în cadrul testului respectiv. La urma urmei, profesorul se bucură când elevii răspund bine cerinţelor sale si îi poate aprecia cu note bune, iar elevii capătă încredere în forţele lor şi, de asemenea, în profesor. Profesorul nu va fi perceput astfel ca „un vânător" de elevi ce nu-şi pregătesc lecţiile. Verificările scrise pot viza expunerea unei anumite tematici (în general o lecţie sau două) sau pot fi alcătuite sub formă de teste grilă. Un rol important în reuşita acestei metode îi

revine cadrului didactic, responsabil cu alegerea subiectelor şi formularea corectă a întrebărilor. Între cele două forme de verificări scrise există o diferență foarte mare. Expunerea în scris a unei anumite tematici cere din partea elevului un efort suplimentar, trebuie să prezinte tematica în timpul alocat, deci activitatea de sintetizare aparține elevului. Verificarea scrisă cu ajutorul *testului grilă* presupune alegerea judicioasă a întrebărilor şi răspunsurilor posibile din partea profesorului, astfel încât să acopere materia anunțată pentru test, să dea posibilitatea elevilor să se încadreze în timpul alocat testului. Este recomandat ca întrebările cu răspunsuri multiple să fie separate de cele cu un singur raspuns şi de asemenea „anunțate" în cadrul testului. Strategia de notare pentru un test grilă trebuie anunțată de către profesor (de exemplu, dacă se acceptă răspunsuri parțiale şi în ce condiții nu se acceptă aceste răspunsuri). Este ştiut că elevii îşi redactează răspunsurile şi în funcție de strategia profesorului. Dacă elevii ştiu că sunt acceptate răspunsuri parțiale la un test grilă (test grilă cu mai multe răspunsuri posibile), atunci aceştia ar putea încerca completarea tuturor răspunsurilor în speranța obținerii unui punctaj cât mai mare.

Să analizăm în continuare următoarele scenarii pentru teste grilă cu mai multe răspunsuri posibile. Fiecare întrebare din testul grilă este notată cu 15 puncte şi profesorul anunță că acceptă şi răspunsuri parțiale, fără a mai specifica şi altceva. În aceste condiții o întrebare ce conține patru răspunsuri, poate fi abordată de către elevi prin selectarea tuturor celor patru răspunsuri. Calculul din partea elevului este simplu. Nu am fost anunțați că erorile se penalizează, deci ar trebui să obțin punctajul maxim. Bineînțeles că este ipotetic acest lucru şi că în realitate nu se poate admite să se întâmple. Pe aceeaşi problema, profesorul anunță următoarea strategie de notare: pentru fiecare răspuns incorect selectat se scade ponderea unui răspuns corect din valoarea testului. Calculul din partea elevului poate fi: la o întrebare de 15 puncte cu patru răspunsuri posibile, dacă trei sunt corecte atunci prin selectarea tuturor răspunsurilor obtin 10 puncte, dacă sunt două corecte obțin 0 (zero) puncte şi atunci voi selecta doar trei întrebări la toate testele ce conțin patru răspunsuri posibile, asigurându-mi astfel un minim de 5 puncte pe întrebare. Şi astfel de scenarii pot continua. Deşi par simple, testele grilă se dovedesc a fi destul de dificile în cazul în care nu se acceptă la notare (în fapt acesta este mecanismul de funcționare al testelor grilă) decât testele care au fost rezolvate corect. Primul test grilă aplicat la o clasă va crea surprize mari atât pentru elevi cât şi pentru profesor, de aceea profesorul trebuie să fie conştient că este nevoie să-şi pregătească elevii pentru un asemenea eveniment. Testele grilă prezintă deci următoarele **avantaje** „imediate":

1. Obiectivitate şi uşurință în notare.
2. Răspunsul se poate da în timp scurt.

3. Se poate acoperi o mare parte din materia predată

Dezavantajele ar fi:

1. Nu se poate pune în evidență raționamentul făcut de elev.
2. Există posibilitatea „ghicirii" răspunsului (valori prea mari, neconforme cu tipul de rezultat asteptat, etc.).

Realizarea testelor grilă cer, de asemenea, profesorului să respecte anumite condiții: itemi clar formulați, într-un item să nu existe o indicație a răspunsului, „lungimea" opțiunilor să nu constituie un criteriu de selectare etc.

Examinarea prin *probe practice* este caracteristică disciplinelor cu pronunțat caracter aplicativ, iar Informaticii cu atât mai mult. Ea se poate desfăşura în forme variate, de la realizarea de programe simple sau editări de texte sau grafică pe durata unei ore, lucrându-se individual sau în grup, până la aplicații complexe, realizate într-un interval mai lung de timp. Sunt verificate şi evaluate cunoştințele teoretice necesare realizării lucrării, cât şi deprinderile şi dexteritățile necesare executării ei. Este necesară şi formarea la elevi a *capacității de autoevaluare*, prezentându-le criteriile de apreciere, ceea ce va mări încrederea elevului în propriile sale forțe şi va înlătura orice urmă de suspiciune. Deşi imperfect, sistemul actual de evaluare permite o ierarhizare a elevilor în „clase" după criterii reale de competență, oferă informații edificatoare asupra nivelului de cunoştințe al fiecărui elev, stimulează elevul să învețe. Putem face o şi clasificare a formelor de evaluare în funcție de *timpul* când se aplică acestea. Luând în considerare acest ultim criteriu de clasificare putem vorbi despre:

a) Evaluarea *inițială*, care conduce la formarea unei imagini despre bagajul de cunoştințe cu care elevul „porneşte la drum". Trebuie să ne asigurăm asupra a ceea ce cunoaşte elevul înainte de a-l învăța alte lucruri. Această formă de verificare creează şi o imagine asupra posibilităților de progres ale elevului, asupra capacității lui de învățare, în funcție de care se va stabili programul de instruire. În general, evaluarea inițială este aplicată întregii clase, profesorul având astfel posibilitatea să-şi adapteze programul de instruire.

b) Evaluarea *formativă (continuă)* este forma de evaluare pe care profesorul o aplică pe întreaga durată a programului de instruire în cadrul lecțiilor şi la încheierea unui capitol. Această formă de verificare oferă permanent informații cu privire la eficiența programului de instruire şi permite profesorului să ia cele mai potrivite măsuri de prevenire a insuccesului şcolar, ajutând totodată la ameliorarea metodelor de predare-învățare. Verificarea ritmică oferă, pe baza mecanismului de feed-back continuu, semnalele necesare atât elevului cât şi profesorului, fiind un veritabil metronom al activității didactice.

c) Evaluarea *sumativă (cumulativă)* este forma tradițională de evaluare realizată la sfârşitul unui semestru sau an şcolar şi cuprinde întreaga materie conform programei şcolare, pe intervalul de timp la care se aplică verificarea. Rezultatele acestei forme de verificare nu reflectă întotdeauna adevăratul nivel de performanță al elevilor, dar prin faptul că determină o recapitulare şi o abordare globală a materiei parcurse, are efecte pozitive în direcția dezvoltării capacității de cuprindere şi de sinteză a elevului. Superioară prin caracterul ei predictiv, evaluarea formativă trebuie totuşi completată şi cu celelalte forme. Rezultatele şcolare sunt obiectivate în cunoştințele acumulate, în priceperi şi deprinderi, capacități intelectuale, trăsături de personalitate şi de conduită ale elevilor. Aprecierea cât mai obiectivă a rezultatelor la învățătură presupune şi urmărirea unor anumite criterii, cum ar fi:

a) *Criteriul raportării rezultatelor la obiectivele generale şi operaționale*, prevăzute în programa şcolară. Prin aceasta se scoate în evidență calitatea şi eficiența programului de instruire. Obiectivele pedagogice permit, pe lângă orientarea metodologică şi o verificare şi apreciere exactă a rezultatelor elevilor (astfel încât doi profesori care apreciază aceeaşi performanță să nu realizeze diferențe de notare decât foarte mici). În acest sens, obiectivele îndeplinesc funcția de criteriu de referință atunci când se formulează o judecată de valoare asupra rezultatelor şcolare, dar ele sunt influențate şi de factori perturbatori, uneori obiectivi, alteori subiectivi, cum ar fi dotarea materială, nivelul clasei, pretențiile profesorului, etc.

b) *Criteriul raportării rezultatelor la nivelul general atins de populația şcolară evaluată*, care se manifestă câteodată, din păcate, printr-o tendință de apreciere indulgentă a rezultatelor elevilor din clasele mai slabe şi de exigență sporită pentru elevii din clasele considerate mai bune.

c) *Criteriul raportării rezultatelor la capacitățile fiecărui elev* şi la nivelul lui de cunoştințe de dinaintea încheierii programului de instruire. Această formă de evaluare dă măsura progresului şcolar realizat de elevi.

3.2. Practica evaluării activității didactice

Controlul cunoştințelor dobândite de către elevi dă posibilitatea profesorului să dezvolte la aceştia simțul răspunderii, să sesizeze la timp lipsurile, să aprecieze cât mai just munca lor. Controlul trebuie făcut sistematic (dacă se poate, zilnic) şi în mod echilibrat. La fiecare lecție se verifică modul în care a fost înțeleasă şi asimilată lecția nouă, iar dacă lecția are un caracter instructiv, trebuie verificat şi gradul în care cele expuse au fost reținute. Verificarea gradului de asimilare se poate face:

- prin repetarea raționamentelor făcute pe parcursul lecției, cu sprijinul elevului;
- prin întrebări de control;
- prin rezolvarea de probleme noi.

Toate acestea ajută la verificarea rezultatelor muncii reale efectuate în clasă. Verificarea lucrărilor scrise, date ca teme pentru acasă se poate face:

- printr-o *trecere printre bănci* şi o examinare *superficială*, cantitativă;
- prin prezentarea rezolvării (ideea principală) de către un elev şi confirmarea înțelegerii de către ceilalți.

Este important ca verificarea temelor să se coreleze cu răspunsurile la un set de întrebări, dinainte stabilite, vizându-se lecția predată anterior. Aceasta va permite elevilor să combine repetarea „notițelor" cu formarea şi dezvoltarea deprinderilor de a corela noțiunile teoretice între ele şi de a le aplica în practică. O altă formă de verificare, este cea orală cu toată clasa, când se pun întrebări *pentru toți*. Elevii sunt lăsați să gândească, apoi este numit unul dintre ei care să răspundă. Ceilalți sunt îndemnați să completeze răspunsul sau să corecteze greşelile. Această examinare sumară (de regulă) nu se notează, dar în situația în care un elev nu a învățat deloc sau a răspuns constant bine la mai multe întrebări, ar trebui notat. La examinarea orală se pun întrebări care nu necesită desene, notări în caiete sau la tablă, calcule. Examinarea cu scoaterea la tablă (sau cea cu „calculatorul personal") se face cu unul sau mai mulți elevi. În timp ce elevii pregătesc răspunsurile, se poate lucra cu clasa sau verifica tema de acasă. Când elevii răspund, este bine ca profesorul să se asigure că toată clasa este atentă şi pregătită să intervină. Profesorul poate să pună întrebări suplimentare sau ajutătoare atât elevilor ascultați cât şi celor din bănci. Prin întrebări se caută să se pună în evidență aspectele esențiale ale lecției. Profesorul trebuie să-şi pregătească dinainte întrebările şi nu trebuie să transforme verificarea într-o „scoatere cu sila" la tablă şi punerea unui noian de întrebări care duc chiar până la sugerarea răspunsului. Când elevul tace, profesorul nu trebuie să-i sugereze el fraza sau ideea ci să desemneze un alt elev. Intervenția inoportună a profesorului poate conduce la apariția unei ambiguități cu privire la răspunsul şi la cunoştințele elevului. Lucrările de control scrise pot varia ca dimensiune:

- cele scurte (*10 - 15* minute) se dau, de obicei, în a doua parte a lecției şi urmăresc modul de asimilare a lecției noi sau a cunoştințelor predate anterior, dar în corelație cu lecția nouă;

- cele de *1 - 2* ore se dau numai după ce au fost anunțate din timp şi pregătite eventual printr-o lecție de recapitulare; orice procedeu de verificare trebuie însă să îndeplinească anumite condiții, discutate în prealabil cu elevii.

Verificările, în general, trebuie:

- Să aibă un scop precis, care chiar dacă nu este transparent pentru elev, trebuie să fie foarte clar pentru profesor.
- Să dezvolte deprinderea elevului de a raționa rapid şi de a da răspunsuri corecte, precise şi scurte, dar complete.
- Să dezvolte la elevi grija pentru formulări exacte şi exprimări corecte ştiințific şi gramatical.
- Să permită elevilor să aprecieze răspunsurile;
- Să fie operative.

3.3. Aprecierea cunoştințelor elevilor. Măsuri de prevenire a rămânerilor în urmă

Aprecierea se face, în principal, prin notă (calificativ). Ea trebuie să reflecte cât de bine şi cât de conştient şi-a însuşit elevul materia parcursă şi în ce măsură ar fi capabil să utilizeze în continuare cunoştințele dobândite. Există anumite criterii după care se face aprecierea şi notarea. Greşelile pe care le comit elevii la verificare sunt diferențiate (*grave, mici, de neatenție, de înțelegere* etc.). Sunt calificate drept greşeli grave cele legate de necunoaşterea sau neînțelegerea unei noțiuni elementare, nepriceperea în abordarea problemelor. Greşelile de genul celor legate de interpretarea eronată a unor enunțuri sau cele legate de neatenția de moment nu trebuie calificate ca fiind grave. Acestea se manifestă printr-o formă neîngrijită de prezentare, greşeli de exprimare, prescurtări ambigui în lucrările scrise. Profesorul se loveşte deseori de dificultatea aprecierii răspunsurilor. De cele mai multe ori se cade în extreme. De aceea este indicat a se stabili un *barem de notare* pentru fiecare subiect în parte şi o notare a fiecărui răspuns cu un anumit procent din punctajul maxim acordat. În apreciere se manifestă personalitatea profesorului, pretențiile sale, atenția față de lucrurile esențiale sau mărunte, tactul lui pedagogic. Rămânerea în urmă a unui elev reprezintă un pericol pentru orice disciplină. În Informatică, acest lucru se poate manifesta sub forme cumva deosebite. Este evident că un curs de Informatică poate fi privit ca unul accesibil (dacă nu este primul de acest gen). Prin urmare aici contează foarte mult experiența cursantului.

Prevenirea *eşecului şcolar* depinde în mare măsură de ***metodica predării***, de buna organizare a muncii elevilor la clasă şi în special la orele de aplicaţii practice de laborator. Interesul trezit de anumite aplicaţii este esenţial. De aceea trebuie alese probleme atractive, interesante, al căror rezultat (pozitiv) să fie uşor de constatat. Pentru prevenirea eşecului este, de asemenea, important să se sesizeze la timp lipsurile şi să se intervină prompt, înlăturându-se greşelile. Nu gratuit un profesor invocă (uneori în glumă) greşeli *antologice* ale unor elevi. Astfel este posibilă evitarea repetării lor. Se creează în acest mod un *cont de greşeli personale* care este referit la nevoie. În cazul rămânerilor în urmă, se recomandă reluarea unor noţiuni prin lecţii suplimentare şi ore de consultaţie la care elevii întreabă şi profesorii răspund. Se poate recurge şi la teme suplimentare individuale sau colective. Promovarea succesului şcolar nu poate fi realizată decât printr-un ansamblu de măsuri şi strategii la nivelul macrosistemului de învăţământ, al unităţilor şcolare, cu contribuţia activă a profesorilor, părinţilor şi elevilor. La nivelul macrosistemului, reforma învăţământului trebuie să promoveze ideea perfecţionării structurii sistemului de învăţământ în raport cu cerinţele sociale şi cu dinamica mutaţiilor economice şi sociale, prin modernizarea obiectivelor pedagogice, a conţinuturilor (planuri, programe, manuale), a metodologiei şi mai cu seamă a bazei didactico-materiale a procesului de învăţământ. Efortul devine singular dacă bunele intenţii şi iniţiative promovate la nivel macro nu sunt aplicate în unităţile şcolare. Este necesar să se creeze un climat favorabil de muncă, să se stimuleze iniţiativa şi responsabilitatea corpului profesoral, să se accentueze perfecţionarea profesională în raport cu noile cerinţe. Aceasta va avea efecte benefice asupra strategiilor de proiectare, organizare şi realizare a activităţii didactice şi se va reflecta pozitiv în promovarea reuşitei şcolare. Ca un corolar, să punctăm şi următoarele:

- În învatamântul preuniversitar, evaluarile se concretizează, de regulă, prin note de la 10 la 1.
- În clasele din învatamântul primar, aprecierea rezultatelor elevilor se face şi prin calificative sau alte forme de apreciere, conform reglementărilor **M.E.C.T.**
- Numărul de note acordate fiecărui elev, la fiecare disciplină de studiu, exclusiv nota de la teză, trebuie să fie cel putin egal cu numărul de ore săptămânal prevăzut în planul de învăţământ, cu excepţia disciplinelor cu o oră pe saptamână, la care numărul minim de note/calificative este de două.
- Primul pas ar consta deci din a defini ceea ce încercăm să măsurăm/evaluăm, evaluarea fiind o componentă esenţială a procesului de învăţământ şi îndeplinind funcţii bine conturate:

- *Funcţia de constatare şi diagnosticare* a performanţelor obţinute de elevi, explicate prin factorii şi condiţiile care au condus la succesul sau insuccesul şcolar şi care sunt de o mare diversitate (psihologică, pedagogică, socială etc.).
- *Funcţia de reglare şi perfecţionare* a metodologiei instruirii pe baza informaţiilor obţinute din explicarea factorilor şi condiţiilor care au determinat rezultatele la învăţătură.
- *Funcţia de predicţie şi decizie* care vizează desfăşurarea în viitor a activităţii didactice.
- *Funcţia de selecţie şi clasificare* a elevilor în raport cu rezultatele şcolare obţinute.
- *Funcţia formativ-educativă*, de ameliorare a metodelor de învăţare folosite de elevi, de stimulare şi optimizare a învăţării.
- *Funcţia de perfecţionare* a întregului sistem şcolar.

Creşterea eficienţei procesului de predare-învăţare presupune o mai bună integrare a actului de evaluare în desfăşurarea activităţii didactice prin verificarea şi evaluarea sistematică a tuturor elevilor, pe cât posibil după fiecare capitol, prin raportarea la obiectivele generale şi operaţionale ale acesteia, verificarea procesului de instruire şi corelare a notelor din catalog cu rezultatele obţinute de elevi la probele externe (concursuri, olimpiade, examene de admitere etc.). Principalele forme de evaluare întâlnite în practica didactică sunt:

a) Evaluarea *iniţială* care conduce la formarea unei imagini despre bagajul de cunoştinţe cu care elevul porneşte la drum. Trebuie să ne asigurăm de ceea ce cunoaşte elevul înainte de a-l învăţa alte lucruri. Această formă de verificare creează şi o imagine asupra posibilităţilor de progres ale elevului, asupra capacităţii lui de învăţare, în funcţie de care se va stabili programul de instruire.

b) Evaluarea *formativă (continuă)* este forma de evaluare pe care profesorul o aplică pe întreaga durată a programului de instruire în cadrul lecţiilor şi la încheierea unui capitol. Această formă de verificare oferă permanent informaţii cu privire la eficienţa programului de instruire şi permite profesorului să ia cele mai potrivite măsuri de prevenire a insuccesului şcolar, ajutând totodată la ameliorarea metodelor de predare-învăţare. Verificarea ritmică oferă, pe baza mecanismului de feed-back continuu, semnalele necesare atât elevului cât şi profesorului, fiind un veritabil metronom al activităţii didactice.

c) Evaluarea *sumativă (cumulativă)* este forma tradiţională de evaluare realizată la sfârşitul unui semestru sau an şcolar şi cuprinde întreaga materie conform programei şcolare, pe intervalul de timp la care se aplică verificarea. Rezultatele acestei forme de verificare nu reflectă întotdeauna adevăratul nivel de performanţă al elevilor, dar prin faptul că determină o

recapitulare şi o abordare globală a materiei parcurse, are efecte pozitive în direcția dezvoltării capacității de cuprindere şi de sinteză a elevului.

Superioară prin caracterul ei predictiv, evaluarea formativă trebuie totuşi completată şi cu celelalte forme. Rezultatele şcolare sunt obiectivate în cunoştințele acumulate, în priceperi şi deprinderi, capacități intelectuale, trăsături de personalitate şi de conduită ale elevilor. Aprecierea cât mai obiectivă a rezultatelor la învățătură presupune urmărirea unor anumite criterii:

a) *Criteriul raportării rezultatelor la obiectivele generale şi operaționale*, prevăzute în programa şcolară. Prin aceasta se scoate în evidență calitatea şi eficiența programului de instruire. Obiectivele pedagogice permit, pe lângă orientarea metodologică şi o verificare şi apreciere exactă a rezultatelor elevilor, astfel încât doi profesori care apreciază aceeaşi performanță să nu realizeze diferențe de notare decât foarte mici. În acest sens, obiectivele îndeplinesc funcția de criteriu de referință atunci când se formulează o judecată de valoare asupra rezultatelor şcolare, dar ele sunt influențate şi de factori perturbatori, uneori obiectivi, alteori subiectivi, cum ar fi dotarea materială, nivelul clasei, pretențiile profesorului, etc.

b) *Criteriul raportării rezultatelor la nivelul general atins de populația şcolară evaluată*, care se manifestă câteodată, din păcate, printr-o tendință de apreciere indulgentă a rezultatelor elevilor din clasele mai slabe şi de exigență sporită pentru elevii din clasele considerate mai bune.

c) *Criteriul raportării rezultatelor la capacitățile fiecărui elev* şi la nivelul lui de cunoştințe de dinaintea încheierii programului de instruire. Această formă de evaluare dă măsura progresului şcolar realizat de elevi.

Metodele de verificare a randamentului şcolar presupun observarea modului în care învață elevul (logic, mecanic, creativ, ritmic, continuu, în salturi etc.) şi se realizează prin probe orale, scrise şi practice, teste de cunoştințe şi deprinderi.

Verificarea orală, cea mai frecvent folosită, are anumite avantaje care o impun. În primul rând favorizează dialogul, elevul putând să-şi argumenteze răspunsurile şi să participe la o confruntare de idei cu întreaga clasă, iar profesorul poate detecta cu uşurință erorile şi poate interveni şi corecta imediat. Verificarea orală are însă şi numeroase limite: întrebările nu au toate acelaşi grad de dificultate; unii elevi sunt emotivi şi se blochează (mai ales atunci când sunt ironizați de către profesor); răspunsurile lor stârnesc ilaritate în clasă; timpul nu permite o verificare completă a conținutului predat. Mai mult, comportamentul şi starea psihică a profesorului poate influența notarea.

Verificarea scrisă se utilizează sub forma unor lucrări de scurtă durată, lucrări *tip obiectiv*, lucrări de una sau două ore, semestriale (care sunt dinainte anunțate şi pregătite şi în clasă), lucrări scrise tip examen. Cercetările au dovedit că evaluarea formativă în formă scrisă, după fiecare capitol, combinată cu verificările orale este deosebit de eficientă şi stimulativă. Probele scrise sunt preferate de către elevi şi profesori pentru că asigură un grad mai mare de obiectivitate la notare, oferă elevilor mai emotivi sau celor care gândesc mai lent posibilitatea de a se exprima fără a fi influențați de factori perturbatori, asigură evaluarea unui număr mare de elevi, întrebările au acelaşi grad de dificultate şi favorizează realizarea comparării rezultatelor. Dezavantajele metodei sunt marcate de faptul că profesorul nu poate interveni şi corecta pe loc erorile descoperite, el urmând să o facă în clasă la discutarea lucrărilor. Elevii nu pot fi corectați dacă fac anumite confuzii sau când răspunsul nu este complet. Răspunsurile incomplete pot genera şi diferențe de apreciere şi notare.

Examinarea prin *probe practice* este caracteristică disciplinelor cu pronunțat caracter aplicativ, iar Informaticii cu atât mai mult. Ea se poate desfăşura în forme variate, de la realizarea de programe simple sau editări de texte sau grafică pe durata unei ore, lucrându-se individual sau în grup, până la aplicații complexe, realizate într-un interval mai lung de timp. Sunt verificate şi evaluate cunoştințele teoretice necesare realizării lucrării, cât şi deprinderile şi dexterițățile necesare executării ei.

O altă formă de verificare este *evaluarea prin teste* şi care se efectuează la începutul programului de instruire (inițiale), pe parcursul acestuia (progres) şi la sfârşitul programului (finale). Rezultatele acestor teste pot fi prelucrate statistic şi pot conduce la concluzii interesante în legătură cu eficiența metodelor de predare-învățare folosite. Este necesară formarea la elevi a *capacității de autoevaluare*, prezentându-le criteriile de apreciere, ceea ce va mari încrederea elevului în propriile sale forțe şi va înlătura orice urmă de suspiciune. Deşi imperfect, sistemul actual de evaluare permite o ierarhizare a elevilor în clase după criterii reale de competență, oferă informații edificatoare asupra nivelului de cunoştințe al fiecărui elev, stimulează elevul la învățătură.

3.4. Condiția profesorului

Analiza psihologiei *omului de la catedră* a constituit un obiect de studiu permanent pentru cercetători. De exemplu, în <36> se abordează problema condiției psihice a

profesorului (*Decalogul profesorului*), conturându-se un (posibil) profil psihologic al acestuia. În fața elevului, profesorul trebuie să fie (sau cel puțin să pară):

- *Cel mai interesat de subiectul pe care-l abordează*, deoarece pe parcursul unei lecții starea profesorului se transmite elevului. Profesorul nu-şi poate permite să arate dezinteres sau plictiseală față de subiectul pe care-l predă. El trebuie să-l considere şi să-l facă interesant (chiar dacă este simplu, îl cunoaşte şi l-a mai abordat de zeci şi zeci de ori). Profesorul nu poate să dea niciodată semne de rutină sau plictiseală. El va capta interesul elevilor atunci când va dovedi că este cel mai interesat şi încântat de subiectul abordat (numai aşa va stârni şi va menține treaz interesul elevilor).
- *Va cunoaşte cu exactitate subiectul*. Este normal că orice metode am alege, orice mijloace am folosi în predare, nu putem face pe alții să înțeleagă ceva ce nici noi nu înțelegem cu exactitate. Celebra butadă „*am explicat până am priceput şi eu"*, vine să confirme regula. A explica o problemă sau a elucida un aspect pe care nu-l poate înțelege clasa, presupune abordarea aspectului de la nivelul de înțelegere al elevului de nivel mediu din clasă şi amplificarea în spirală, prin paşi care presupun, pe lângă raționament, şi introducerea unor noțiuni noi. Succesiunea etapelor demonstrației este subordonată obiectivului final adică *înțelegerea subiectului*. Orice „ruptură sau forțare" compromite demersul didactic, iar elevii sesizează cu rapiditate aceste momente. O „conjunctură" de raționament poate conduce către aspecte care vor fi abordate în lecțiile viitoare (astfel, *stăpânirea* conținuturilor în ansamblul lor este o condiție *sine qua non* pentru profesor, singura în măsură a realiza conexiunile dintre conținuturi).
- Să ştie că *înainte de a învăța de la altcineva, poți descoperi singur*. Recurgerea la metodele active (bazate pe activitatea proprie a elevului) în însuşirea anumitor concepte, priceperi, deprinderi are un efect stimulator, elimină *şablonismul*, dă frâu liber imaginației creatoare, muncii independente. Desigur că există limite în aplicarea acestui principiu, cunoscuta metodă a specialistului care *încearcă întâi toate posibilitățile, apoi citeşte documentația*, fiind un argument în plus.
- *Profesorul colaborează, nu conduce*. Adică, activitatea în grup are avantajul armonizării ideilor în vederea atingerii obiectivului final, iar profesorul se integrează frecvent în grup, participând de cele mai multe ori ca lider, la soluționarea problemelor. Această postură de lider creează grupului un handicap,

întărit uneori de ideea preconceptă că profesorul cunoaşte cu exactitate modul de rezolvare şi prin urmare, el nu participă la descoperirile echipei, ci doar le supervizează (de aceea, tot ceea ce el sugerează este regulă). Elevului trebuie să i se ofere posibilitatea prezentării şi argumentării ideilor sale, el trebuie lăsat să-şi continue raționamentul (în anumite limite, chiar dacă acesta este greşit), până când se convinge de greşeală. Întreruperea brutală şi fără argumentare transformă elevul din colaborator în adversar, acesta canalizându-şi eforturile în contracarare şi nu în colaborare.

- În procesul instructiv profesorul trebuie să *se coboare la nivelul de înțelegere şi anticipare al elevului*, să se transpună în situația acestuia; relația profesor-elev fiind una de colaborare, în care, cu certitudine, profesorul este cel care ştie şi elevul cel care învață. Premizele colaborării pornesc de la cunoaşterea reciprocă a exigențelor profesorului şi a posibilităților elevului, iar împărțirea forțată şi apriorică în profesori blânzi sau duri, sau în elevi slabi şi buni, este dăunătoare. Profesorul are obligația să cultive elevului încrederea în posibilitățile sale, să-i depisteze punctele slabe şi fără a i le scoate în evidență, să-l ajute să se corecteze. Cea mai dăunătoare atitudine este calificarea unui elev după rezultatele obținute la celelalte discipline. Opțiunile, înclinația, vocația, interesele, perturbările exterioare, pot influența într-un sens sau altul prestația elevului la o disciplină, iar dacă situația o permite calificarea elevilor se va face totdeauna cu etichete pozitive: mai interesați, mai pasionați, mai rapizi, mai originali.
- Să *informeze şi să formeze priceperea de a utiliza informația*. Realizarea acestui deziderat face parte din panoplia de mijloace „externe" a fiecărui cadru didactic. Unii profesori, în funcție şi de disciplină, introduc noțiuni şi teme noi pornind de la necesități reale, de soluționare a unor probleme concrete, iar aceste noțiuni devin mijloace naturale, folosite imediat. Analiza atentă a mijloacelor care ne stau la dispoziție pentru rezolvarea unei probleme, scoate în evidență utilitatea cunoştințelor dobândite anterior, iar *îmbrăcarea* problemelor aparent strict teoretice într-o haină practică, reală, se poate transforma într-o posibilitate de succes.
- *Să dirijeze raționamentul elevului către descoperirea soluției optime*. Elevul trebuie îndreptat pe făgaşul descoperirii, corectându-i-se alegererile şi sfătuind-l să-şi verifice singur paşii, învățându-l să facă analogii, să descopere diferențe, să

intuiască situațiile limită. Elevul trebuie învățat în acelaşi timp să abstractizeze, să aplice rezultatele teoretice ce i-au fost prezentate, să aleagă dintr-o mulțime de soluții metoda cea mai adecvată de rezolvare. Elevul trebuie să fie conştient de faptul că nu este primul şi nici singurul rezolvator al acelei probleme şi că poate să existe o metodă mai eficientă descoperită de alții. În acest fel, acesta va fi preocupat mereu de optimizarea propriilor soluții, i se va forma spiritul critic şi autocritic şi dorința de autodepăşire.

- *Să învețe elevii să-şi argumenteze şi demonstreze corectitudinea soluției* găsite. Argumentele pro şi contra unei soluții trebuie să însoțească fiecare pas al rezolvării. Elevul trebuie obişnuit să-şi *suspecteze* corectitudinea soluției găsite prin analiza cazurilor limită şi să caute în permanență contraexemple. Analiza complexității algoritmilor este un concept care se deprinde şi se aplică după o oarecare experiență.
- *Să formeze elevilor capacitatea de abstractizare şi generalizare*. Posibilitatea adaptării şi aplicării unui algoritm la o clasă de probleme de acelaşi tip, a avut ca rezultat, printre altele, apariția metodelor de elaborare a algoritmilor, înțelegerea problematicii generale şi a metodelor aplicate, particularizarea lor la situații concrete. Crearea unor deprinderi de genul *de la simplu la complex*, este la fel de importantă.
- *Să nu prezinte sau să rezolve o problemă pe care elevul o poate rezolva singur*. Elevul trebuie încurajat să descopere cât mai multe soluții, profesorul care oferă şi pretinde totul aşa cum a oferit devenind de fapt un *dresor de papagali*. Cu răbdare, punând întrebări ajutătoare, dând mici indicații, elevul poate fi ajutat să obțină, sau să creadă cu convingere că a obținut singur rezultatul dorit; încrederea în posibilitățile lui creşte, nu se simte stresat sau presat de asimilarea unei succesiuni amețitoare de noutăți.
- Mai mult ca oricare altul, profesorul *trebuie să fie un bun actor*, un interpret fără partitură, care trebuie să imagineze şi să improvizeze mereu, fără ca *spectatorul* lui fidel, elevul, să sesizeze vreodată acest aspect. Ne vom preface că o soluție prezentată de elev este bună, până când îşi va descoperi singur greşeala, vom suferi alături de el căutând-o pe cea corectă şi ne vom bucura o dată cu el descoperind-o. Profesorul nu poate fi supărat sau trist, nu poate fi melancolic, distrat, inexact. El trebuie să fie mereu bine dispus şi atent. În plus, trebuie să-şi soluționeze singur

toate probleme cu clasa, să nu dea semne de slăbiciune. Cu cât se cunosc reciproc mai mult, cu cât colaborează şi se ajută mai mult, cu cât se înțeleg şi se apreciază mai mult, cu atât profesorul şi elevii se vor apropia mai mult.

3.5. Planificarea activității didactice

Actoria didactică are însă limite. În urma practicii didactice s-a stabilit ca profesorul să prezinte un plan de muncă anual (calendaristic) sau semestrial. Planificarea calendaristică trebuie să conțină eşalonarea conținuturilor disciplinei respective pe durata anului sau semestrului cu indicarea numărului de ore şi a datei stabilită pentru studiul fiecărei teme. În paralel cu lecțiile de comunicare de cunoştințe sau mixte, este necesară planificarea *lecțiilor recapitulative*, iar la sfârşitul semestrului, *lecții de evaluare sumativă*. În planificarea calendaristică se vor face referiri la materialul didactic şi la lucrările practice care vor fi efectuate. Rubricația planificării calendaristice depinde de gradul de detaliu la care se doreşte să se realizeze aceasta. Temele specificate în planificare sunt concretizate în lecții, pentru care profesorul trebuie să întocmească în plus un plan de lecție (**Proiect de tehnologie didactică** etc.) la nivel de detaliu. Pentru o proiectare corectă, profesorul trebuie să țină seama de anumite etape pe care trebuie să le parcurgă şi în care trebuie să răspundă la următoarele întrebări:

1) ***Ce voi face ?***

Înainte de a face orice altceva se vor preciza cu claritate obiectivele educaționale ale activității viitoare.

2) ***Cu ce voi face ?***

Este absolut necesar să se analizeze atent resursele educaționale disponibile pentru a realiza obiectivele stabilite.

3) ***Cum voi face ?***

Se va alcătui strategia educațională potrivită pentru realizarea obiectivelor stabilite.

4) ***Cum voi şti dacă s-a realizat ceea ce trebuia ?***

În orice activitate de altfel este dificil de stabilit dacă s-a atins obiectivul propus. În activitatea didactică este cu atât mai greu. Găsirea unei metodologii satisfăcătoare de evaluare a eficienței activității realizate este o problemă doar parțial rezolvată. Proiectarea didactică

presupune totuşi concretizarea şi detalierea următoarelor etape (şi vom încheia capitolul cu aceste consideraţii):

1) *Precizarea obiectivelor*. Presupune stabilirea în mod precis a *ce deprinderi se doresc a se forma* pe parcursul desfăşurării activităţii didactice. Se va verifica dacă ceea ce s-a stabilit este ceea ce trebuia realizat în raport cu programa şcolară. Se va verifica şi dacă obiectivele stabilite sunt realizabile în timpul disponibil.

2) *Analiza resurselor*. Se va stabili conţinutul activităţii. Se va analiza calitatea materialului uman, dezvoltarea fizică şi psihică a elevilor, particularităţile individuale, motivaţia învăţării, mijloacele materiale. Se vor alege metodele didactice necesare.

3) *Elaborarea strategiei*. Se vor selecta mijloacele de instruire de care este nevoie, combinând metodele, materialele şi mijloacele astfel încât să se amplifice eficacitatea lor didactică. Se va descrie în detaliu *scenariul* activităţii care urmează a fi desfăşurată.

4) *Evaluarea*. Se vor analiza cu atenţie standardele de performanţă şi se va elabora un sistem de metode şi tehnici de evaluare adecvate atingerii scopului propus.

Următoarele capitole vor avea fie un aspect pur metodic, sau vor conţine exemple concrete (planuri/proiecte de lecţie) legate de concretizarea noţiunilor/rezultatelor *în sine*.

CAPITOLUL 3

Principii didactice

În acest capitol sunt prezentate pe scurt câteva dintre principiile didactice generale, simultan cu anumite exemplificări ale aplicării lor în domeniul Informaticii.

1. CLASIFICAREA ŞI CARACTERISTICILE PRINCIPIILOR DIDACTICE

Un model al sistemului de învățământ trebuie să se încadreze în contextul „legilor obiective" care acționează în societate la momentul respectiv. Conținutul, scopul, sarcinile concrete ale predării Informaticii pot fi deduse din planurile de învățământ, precum şi din alte activități specifice (şcolare sau chiar extraşcolare). Aceasta corespunde stadiilor (ciclurilor) de învățare fixate în conformitate cu dezvoltarea intelectuală a elevilor, o atenție prioritară trebuind să fie direcționată spre adaptarea la nou, inclusiv în ceea ce priveşte dezvoltarea bazei materiale. ***Principiile didactice*** reprezintă normele generale care orientează conceperea, organizarea şi desfăşurarea procesului de predare/învățare. Aşa cum este normal, începem prin a puncta câteva dintre caracteristicile generale ale principiilor. Va urma o clasificare şi descriere mai detaliată a acestora. Deoarece din punct de vedere metodic principiile didactice nu sunt independente, am preferat să grupăm (puținele) exemple într-o secțiune separată. Principiile didactice au un:

- *Caracter legic*, ceea ce înseamnă ca ele exprimă raporturile esențiale şi globale care orientează conceperea şi desfăşurarea procesului de învățământ.
- *Caracter obiectiv*, adică se asigură o orientare a procesului de învățământ nefalsificată şi detaşată de impresii, tendințe şi dorințe subiective; procesul de învățământ este de dorit a fi orientat în concordanță cu legile dezvoltării psihice ale individului, precum şi cu legile evoluției societății.
- *Caracter algoritmic*. Se exprimă cerințe şi soluții prin utilizarea unui sistem precis de reguli, care trebuie cunoscute şi respectate cu exactitate dacă se doreşte o orientare eficientă a procesului de învățământ.
- *Caracter dinamic*. Principiile didactice sunt elemente legice, dar deschise înnoirilor şi creativității. Ele trebuie să fie *în pas* cu schimbările şi mutațiile care pot interveni în actul didactic.
- *Caracter sistematic*. Fiecare principiu (ca entitate în sine) intră în relație cu celelalte principii, alcătuind un ansamblu unitar de legități ale cărui componente se condiționează reciproc.

Pentru o bună organizare şi desfăşurare a procesului de învățământ, profesorul trebuie să respecte şi să aplice corect măcar următoarele *principii didactice* clasice:

1. Principiul intuiției.
2. Principiul legării teoriei de practică.
3. Principiul însuşirii conştiente şi active a cunoştințelor.
4. Principiul sistematizării şi continuității cunoştințelor.

5. Principiul accesibilității cunoştinţelor.
6. Principiul însuşirii temeinice a cunoştinţelor.
7. Principiul individualizării şi diferenţierii învăţării.

Vom descrie pe scurt latura aplicativă a fiecărui principiu în zona noastră de interes.

1. Principiul *intuiției*

Acest principiu exprimă necesitatea studierii obiectelor, fenomenelor, proceselor cu ajutorul simțurilor, ținându-se cont de importanța realizării unității dintre senzorial şi raţional. *A transmite cunoştinţe de informatică în mod intuitiv* înseamnă a porni de la contactul direct cu realitatea, pentru ca apoi (prin perceperea acestora) să se ajungă la generalizări. De cele mai multe ori putem face apel la memorie, reprezentări grafice, asemănări, analogii. Instrumentele de tip multimedia moderne au deschis deja căi nebănuite. Folosind acest principiu, este posibil să nu putem descrie *exact* şi *complet* o problemă, într-o singură fază. Putem însă deschide o cale spre înțelegerea acesteia, putem stabili un drum cât de cât sigur spre reveniri ulterioare.

2. Principiul *legării teoriei de practică*

Raportul dintre teorie şi practică depinde în ultimă instanță de dificultatea noțiunilor implicate, de mijloacele tehnice avute la dispoziție, de cunoştinţele anterioare precum şi de capacitățile intelectuale ale clasei de elevi avută la dispoziție, de abilitatea şi experiența cadrului didactic. În informatică, conştientizarea necesității utilizării performante a unor tehnici folosite frecvent astăzi în viața cotidiană (coduri de bare, telefonie mobilă, transmisie audio-video prin satelit, poştă electronică, scanări, etc.) este esențială. Mai mult, *importanța verificării* faptului că elevii sunt într-adevăr în stare să aplice în practică cunoştinţele teoretice acumulate este cu adevărat vitală. Sintetizând, putem spune că aplicarea eficientă a principiului legării teoriei de practică pretinde respectarea consecventă a următoarelor direcții:

- Laboratoarele (cu caracter didactic) precum şi sălile de curs/seminar trebuie să fie dotate (inclusiv în ceea ce priveşte condițiile de lucru) la nivelul cerințelor moderne, anticipându-se condițiile posibile a fi întâlnite la viitoarele locuri de muncă.
- Activitățile practice ale elevilor trebuie să aibă o finalitate şi o aplicabilitate imediată (manifestată de exemplu prin lucrul în echipă la contracte ferme cu unități economice, gen *coaching*, sau prin elaborarea unui raport cu contribuții personale, publicabil în reviste şcolare). Ar fi benefic ca atât recompensele cât şi pedepsele să fie similare cu cele aplicate într-o activitate reală şi nu doar reprezentate de note sau calificative.

- Activitățile serioase cer o fundamentare teoretică, conştientizându-se faptul că partea de teorie este efectiv utilă, ba chiar indispensabilă dacă se doreşte o adaptare „din mers” la cerințe ulterioare.
- Asistența cadrelor didactice trebuie corelată cu apelarea la specialişti „lucrativi” din sfera producției directe, precum şi cu o testare pe cât posibil individualizată şi specifică a elevului.

3. Principiul *însuşirii conştiente şi active a cunoştințelor*

Acest principiu exprimă necesitatea ca procesul de instruire (*acumulare de cunoştințe*) să se facă organizat, prin fixarea unor scopuri, finalități şi termene precise. Înțelegerea semnificațiilor şi conexiunilor esențiale pentru studiul obiectului vizat (Informatica), trebuie realizată printr-un efort de gândire *acțional*. Profesorul trebuie să delimiteze încă de la începutul lecției scopul şi utilitatea practică şi teoretică a temei respective, folosind un bogat material exemplificativ. Se urmăreşte trecerea de la intenție la gândirea abstractă, de la treapta senzorială la treapta rațională şi favorizarea formării de noi *structuri informaționale*. Pentru evitarea unei însuşiri mecanice, se va pune accentul pe metodele active de învățare, pe asigurarea participării permanente şi conştiente a elevilor la desfăşurarea lecțiilor, pe *stimularea muncii creatoare şi independente*. Însuşirea conştientă şi activă a cunoştințelor determină formarea unor atitudini sau condiții favorizante pentru învățare cum ar fi:

- Obținerea unei motivații favorabile şi a satisfacției învățării.
- Asigurarea credibilității adevărurilor şi transformarea lor în convingeri şi deprinderi ştiințifice.
- Sporirea posibilităților de a utiliza în mod concret şi profitabil informația asimilată, oferind potențialului intelectual individual şanse superioare de reuşită, atât pe plan practic/constructiv cât şi pe plan creativ.

4. Principiul *sistematizării şi continuității cunoştințelor*

Scopul oricărei activități de predare este de a *înarma* elevii cu un sistem *armonios* şi corect de cunoştințe. Logica internă a obiectului de predat şi legile generale ale dezvoltării capacităților de cunoaştere individuale impun asigurarea continuității, dar şi necesitatea sistematizării materiei. Noile informații relevante vor fi legate de cele deja introduse şi vor prefigura informațiile ulterioare (respectându-se programa şcolară). Principiul sistematizării se concretizează deci prin expuneri organizate asupra cunoştințelor de asimilat, respectându-se un anumit plan. Pentru a dezvolta continuu gândirea logică a elevilor, pentru a încuraja participarea lor activă, pentru a le crea deprinderi de sistematizare şi generalizare a

celor învățate, profesorul trebuie să-şi folosească la maximum disponibilitățile creatoare şi talentul pedagogic în pregătirea expunerilor. *Activitatea individuală conştientă* a elevului ar trerbui să fie esențială. Cunoştințele nu se pot asimila în salturi, iar deprinderile neexersate se pierd (în special în Informatică, unde rata de *perisabilitate* a acestora este foarte ridicată). Dacă dorim un învățământ de masă eficient şi asigurarea unei pregătiri ritmice a elevilor, trebuie să fie acceptat şi un control permanent şi riguros al profesorului asupra modului şi stadiului de însuşire a cunoştințelor de către elevi. Putem recomanda aplicarea câtorva reguli generale:

- *Secvențele* de cunoştințe transmise trebuie să fie coerente şi unitare, ordinea fiind determinată de conexiuni logice clare.
- Învățarea trebuie făcută ritmic, la intervale optime, asigurându-se simultan restructurarea şi reorganizarea „pachetului" de cunoştințe.
- În privința instrumentelor specifice pentru controlul realizării acestor obiective putem cita: utilizarea de rezumate, conspecte, sinteze, planuri de perspectivă, clasificări, tabele, scheme, statistici etc.
- Controlul şi evaluarea periodică a calității receptării trebuie să fie o modalitate de reglaj dar şi de autoreglaj.

5. Principiul *accesibilității cunoştințelor*

Cunoştințele predate pot fi asimilate de elevi numai dacă sunt accesibile ca volum şi conținut. O temă este accesibilă atunci când corespunde particularităților psihologice de vârstă ale elevilor cărora le este adresată, este o continuare firească a celor acumulate anterior şi *corespunde capacității lor reale de muncă.* Conform acestui principiu, respectarea programei şcolare, în ideea că ea a fost „civilizat" construită, apare ca fiind esențială. De asemenea, demersul instructiv-educativ trebuie adaptat condițiilor concrete ale clasei, stabilindu-se un raport optim între efortul solicitat elevului şi ajutorul care i se acordă în procesul de învățare. După cum am evidențiat deja, în Informatică acest aspect este cu atât mai important cu cât condițiile de lucru se pot schimba cu rapiditate chiar pe parcursul unui aceluiaşi semestru. Respectarea particularităților psihologice de vârstă nu înseamnă a scuti elevii de efortul intelectual necesar dezvoltării gândirii abstracte. În acest scop recomandăm:

- Folosirea unor demersuri gradate de predare/învățare, de genul: de la simplu la complex, de la uşor la greu, de la particular la general, de la concret la abstract.
- Conştientizarea elevilor asupra faptului că efortul personal este absolut esențial pentru *înțelegerea corectă şi de durată* a celor studiate.

- Asigurarea unui studiu ritmic pentru a evita golurile de cunoştinţe şi eforturile ulterioare de înţelegere şi asimilare.
- Asigurarea unui control activ şi a unei evaluări permanente, în scopul eficientizării maxime a actului didactic.

6. Principiul *însuşirii temeinice a cunoştinţelor*

Acest principiu reclamă cerinţa fixării materialului de specialitate studiat, astfel încât elevii să-l poată reproduce şi utiliza creator atât în rezolvarea temelor şcolare curente cât şi în activitatea practică viitoare. Expunerile trebuie făcute intuitiv, accentuându-se esenţialul şi evitându-se supraîncărcarea. Fixarea cunoştinţelor nu trebuie realizată printr-o repetare succintă a celor expuse ci trebuie să se bazeze pe o receptare logică, raţională, cu ajutorul căreia să se poată *identifica* esenţialul. O asemenea însuşire temeinică poate fi obţinută prin diverse modalităţi de *recapitulare*: *curentă, de sistematizare şi sinteză, de preîntâmpinare a uitării celor deja învăţate, de asigurare a fixării în memorie a sistemului de cunoştinţe fundamentale*. Putem din nou recomanda respectarea câtorva reguli:

- Predarea trebuie să fie intuitivă şi accesibilă.
- Însuşirea cunoştinţelor trebuie să fie direcţionată spre o asimilare logică şi conştientă, urmându-se un studiu sistematic.
- Elevii trebuie stimulaţi în ideea participării active şi continue la lecţii.
- Este de dorit să se asigure *motivaţia învăţării*, în strânsă legătură cu anumite aspiraţii individuale.

7. Principiul *individualizării şi diferenţierii învăţării*

Exprimă necesitatea adaptării strategiei instructiv/educative atât la particularităţile psihofiziologice ale fiecărui elev în parte cât şi la particularităţile (relativ comune) ale unei grupe omogene de elevi, în vederea dezvoltării lor ca personalitate şi profesionalism. *Individualizarea învăţării* se referă la valorificarea cât mai bună a posibilităţilor şi eforturilor individuale, atât pentru persoanele înzestrate cât şi pentru cele mai puţin înzestrate. Se recomandă:

- Elaborarea de sarcini instructive (teme, lucrări etc.) individualizate pentru fiecare elev în parte (în funcţie de aptitudinile, înclinaţiile, opţiunile, nivelul de dezvoltare intelectuală, coeficientul de inteligenţă)
- Cerinţa ca oricare dintre sarcinile anterior specificate să fie identificată prin *fişe de lucru individuale*, cum ar fi:
    - Fişe de *recuperare* (pentru cei rămaşi în urmă).

- Fişe de *dezvoltare* (pentru elevii foarte buni).
- Fişe de *exerciţii*, destinate tuturor, în scopul formării unor priceperi şi deprinderi aprofundate.
- Fişe de *autoinstruire*, destinate în special însuşirii unor tehnici de învăţare individuală şi independentă.
- Fişe de *evaluare generală*, pentru constatarea nivelului general de pregătire.

Consultaţiile speciale, individualizate, nu pot fi evitate. *Diferenţierea învăţării* exprimă însă necesitatea de a adapta conţinutul strategiilor educaţionale în funcţie de particularităţile comportamentului individual (sau de grup) ale elevilor (cum ar fi promovarea aptitudinilor specifice pentru anumite materii). Această diferenţiere va răspunde atât satisfacerii nevoilor destinate tratării unor particularităţi psihologice individuale, cât şi satisfacerii unor cerinţe sociale privind pregătirea şi utilitatea existenţei unor specialişti. Aici ar fi utile: crearea de şcoli şi/sau profile specializate; relaxarea învăţământului prin introducerea mai multor discipline opţionale şi facultative; intensificarea activităţilor de coordonare directă profesor-elev (consultaţii, discuţii, mese rotunde, cercuri de profil, etc.); cunoaşterea cât mai completă a fiecărui elev, atât ca individualitate cât şi ca fiinţă socială; îmbinarea judicioasă a tratării individuale şi diferenţiate cu cea globală, de grup, în care se rezolvă sarcini de echipă; utilizarea învăţământului asistat; conştientizarea elevilor privind posibilităţile proprii de formare/dezvoltare intelectuală.

2. EXEMPLU

Pentru ilustrarea aplicării tuturor principiilor (şi nu numai) vom încheia acest capitol printr-un exemplu global. **Problema turnurilor din Hanoi** este, considerăm noi, suficient de edificator şi de complex putând fi folosit în plus şi pentru:

1. Prezentarea unor noţiuni informatice generale: cuvânt, limbaj formal; graf (arbore); stivă (listă, coadă).
2. Înţelegerea metodei backtracking;
3. Înţelegerea derecursivării automate în sens iterativ (parte a construcţiei compilatoarelor).
4. Înţelegerea unor tehnici de prelucrare a imaginilor.
5. Introducerea câtorva consideraţii de corectitudine şi complexitate a algoritmilor.
6. Introducerea câtorva concepte de programare nestandard, cum ar fi programarea funcţională.

Enunțul problemei. În oraşul Hanoi există 3 (trei) turnuri de aur care au în vârf un număr de n discuri de diamant. Fiecare disc are propria sa dimensiune, dimensiunile (adică diametrele), fiind diferite între ele. Discurile sunt plasate inițial pe un singur turn, de jos în sus în ordinea descrescătoare a diametrelor (discul cu diametrul maxim găsindu-se la bază). Se cere să se deplaseze cele n discuri de pe turnul inițial pe un altul (folosind, eventual, ca suport intermediar şi pe cel de-al treilea).

Restricții. Mutarea discurilor trebuie făcută într-un număr succesiv de paşi independenți, la fiecare pas mutându-se un singur disc de pe un turn pe altul; se mută întotdeauna discul din vârf (adică cel care are diametrul minim de pe turnul respectiv); nu se poate aşeza un disc cu diametrul mai mare peste unul cu diametrul mai mic.

Soluție. Ca un prim comentariu să remarcăm faptul că *enunțul recursiv* este foarte simplu, deşi *ideea* unui algoritm iterativ general pentru această problemă nu este deloc transparentă. Propunem alegerea următoarelor notații care vor simplifica exprimarea ulterioară a soluției:

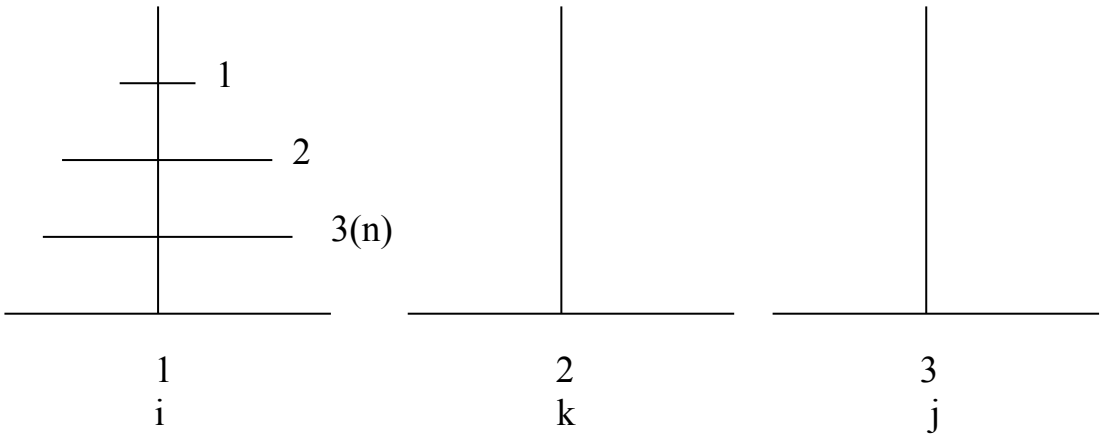
1. Pentru **turnuri**: i, j, k $\in$ {1,2,3}, valori diferite între ele, unde i reprezintă turnul „de plecare", j este turnul de sosire iar k este „turnul intermediar". În acest caz, putem observa că avem $k = 6 - i - j$ = al_treilea(i, j).
2. **Discurile** vor fi notate cu 1, 2, ..., n în funcție de dimensiune (n este discul de dimensiune maximă).
3. **Mutările** vor fi desemnate prin triplete de tipul <a, b, c>, ceea ce va însemna că se deplasează discul c (cel mai din vârf) de pe turnul a pe turnul b (în vârf). Desigur că a,b $\in$ {1,2,3} iar c $\in$ {1,2,...,n}.
4. **Succesiunea** mutărilor va fi indicată prin „·".

Exprimarea problemei ca o funcție definită recursiv (în sens matematic). Dacă M este numele funcției (care depinde de: turnul sursă, turnul destinație, numărul de discuri mutate), putem defini:

$$M(i, j, n) = M(i, k, n-1) \cdot <i, j, n> \cdot M(k, j, n-1)$$

Intuitiv, pentru a deplasa n discuri de pe turnul i pe turnul j, se deplasează întâi n-1 discuri de pe turnul i pe turnul k şi în final se deplasează cele n-1 discuri rămase de pe turnul k pe turnul j. În cadrul unei lecții concrete, se pot da explicațiile de rigoare cu privire la funcția recursivă şi la faptul că un limbaj de programare funcțional este un limbaj care lucrează cu liste şi liste de cuvinte. Faptul că definiția recursivă este corectă rezultă imediat prin inducție.

În final, se obține valoarea funcției ca o secvență finită de paşi (cuvânt) de tipul <i, j, n>. Acest lucru rezultă din faptul că, aplicând în mod repetat definiția lui M, în egalitatea precedentă n scade la fiecare repetare.



Observație. M(p, q, 0) va reprezenta cuvântul vid (punctul din definiția lui M poate fi considerat ca reprezentând operația de *concatenare*, în sensul obişnuit al *teoriei limbajelor formale*).

Acum, să precizăm că pentru derecursivarea algoritmului vom folosi o stivă. Inițial, stiva este goală. În reprezentarea grafică, ordinea mutărilor este dată de numărul încercuit. Elementele stivei denotă:

i) M(...) - apelul recursiv al funcției M.

ii) M(... 0) - se ignoră apropo de orice acțiune (de fapt acest simbol va fi şters ulterior).

iii) <...> - se efectuează o mutare normală.

Operațiile generale care se efectuează asupra stivei sunt:

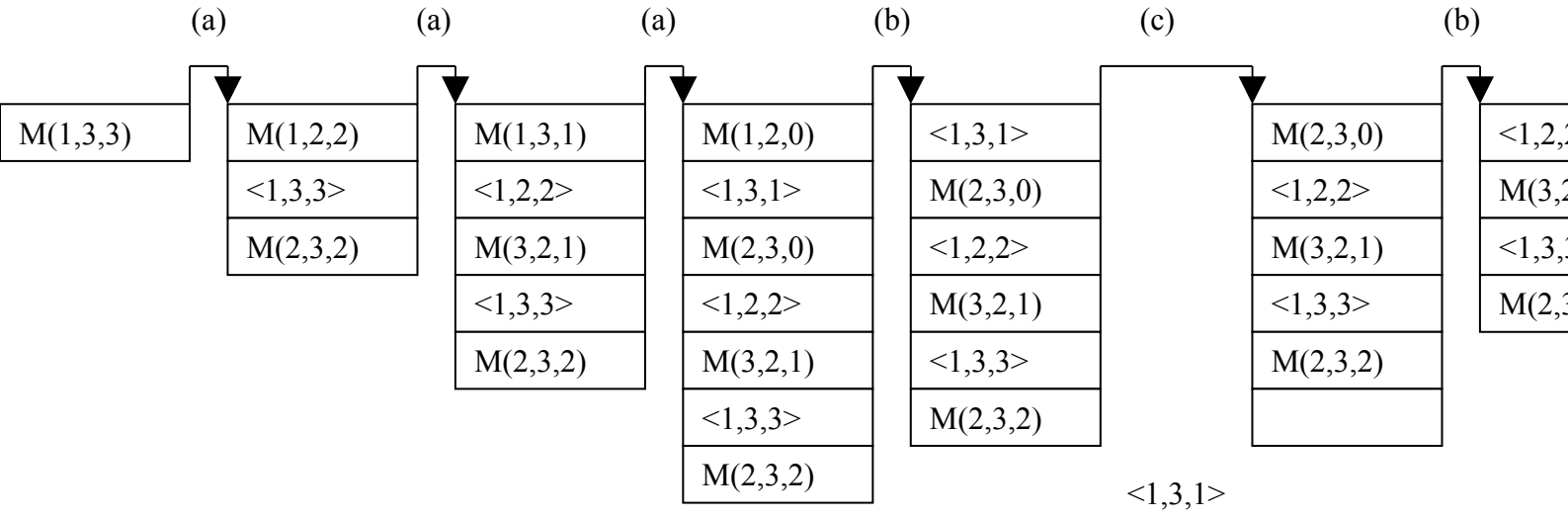
- **În cazul i)**. Dacă vorbim de un apel al funcției M cu ultima poziție diferită de zero, atunci conținutul „capului" se şterge şi acesta se înlocuieşte cu 3 celule noi. Restul conținutului stivei „coboară".

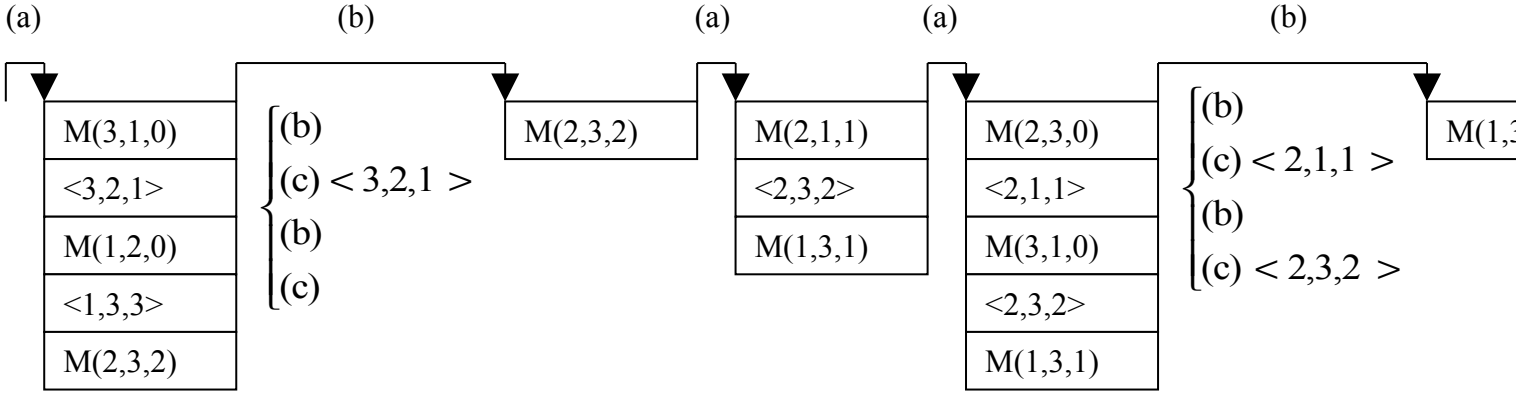
- **În cazul ii)**. Conținutul capului stivei se şterge şi restul conținutului urcă în stivă.
- **În cazul iii)**. Se *execută în mod efectiv* mutarea indicată, se trece aceasta pe *lista de ieşire* (care va constitui în final soluția problemei) şi apoi se procedează ca mai înainte.

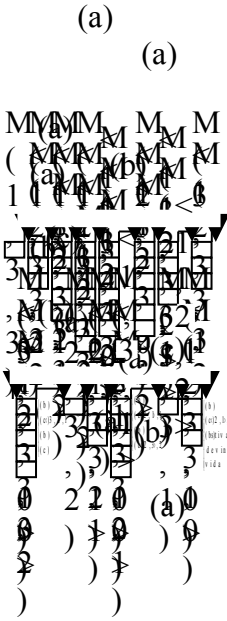
Procesul se termină şi se obține soluția finală doar în momentul în care când stiva redevine goală. În exemplul detaliat de mai jos considerăm cazul i = 1, j = 3, k = 2, n = 3. **Ceea se găseşte în final, ca succesiune de mutări este:**
<1,3,1>·<1,2,2>·<3,2,1>·<1,3,3>·<2,1,1>·<2,3,2>·<1,3,1>.

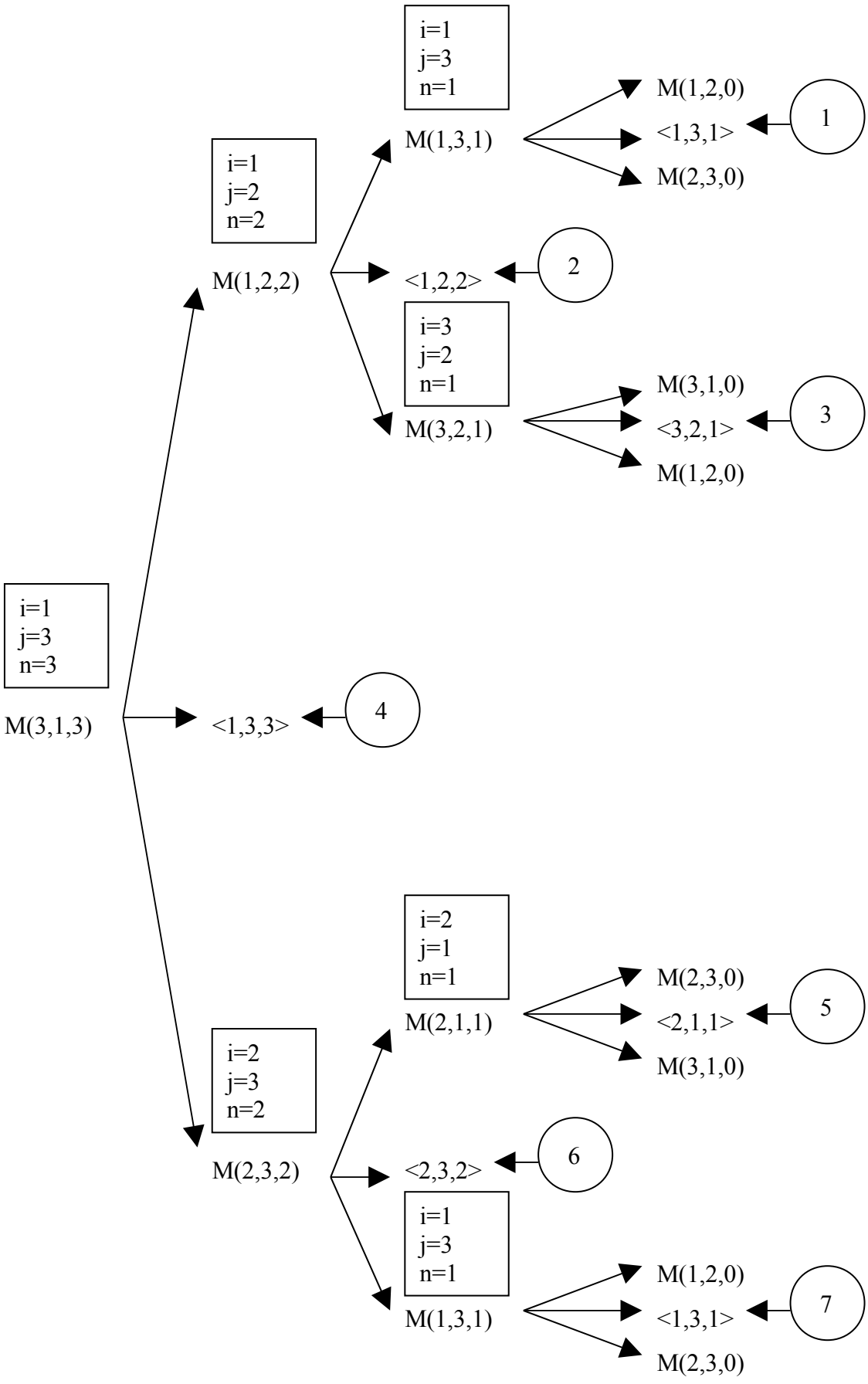
Observație. Numărul de noduri în graful general este $1+2^0 \cdot 3+ 2^1 \cdot 3 + ... +2^{n-1} \cdot 3$, dacă sunt n discuri şi 3 turnuri. Numărul exact de mutări poate fi calculat imediat.

Imaginea stivei şi a grafului prin care se reprezintă backtracking-ul sunt prezentate în continuare.


(a)
(a)
(a)
(b)
(c)
(b)
M(1,3,3)
M(1,2,2)
<1,3,3>
M(2,3,2)
M(1,3,1)
<1,2,2>
M(3,2,1)
<1,3,3>
M(2,3,2)
M(1,2,0)
<1,3,1>
M(2,3,0)
<1,2,2>
M(3,2,1)
<1,3,3>
M(2,3,2)
<1,3,1>
M(2,3,0)
<1,2,2>
M(3,2,1)
<1,3,3>
M(2,3,2)


(a)
(b)
(a)
(a)
(b)
M(3,1,0)
<3,2,1>
M(1,2,0)
<1,3,3>
M(2,3,2)
(b)
(c) < 3,2,1 >
(b)
(c)
M(2,3,2)
M(2,1,1)
<2,3,2>
M(1,3,1)
M(2,3,0)
<2,1,1>
M(3,1,0)
<2,3,2>
M(1,3,1)
(b)
(c) < 2,1,1 >
(b)
(c) < 2,3,2 >


(a)
(a)


i=1
j=3
n=3
M(3,1,3)
<1,3,3>
4
i=1
j=2
n=2
M(1,2,2)
<1,2,2>
2
i=1
j=3
n=1
M(1,3,1)
M(1,2,0)
<1,3,1>
M(2,3,0)
1
i=3
j=2
n=1
M(3,2,1)
M(3,1,0)
<3,2,1>
M(1,2,0)
3
i=2
j=3
n=2
M(2,3,2)
<2,3,2>
6
i=2
j=1
n=1
M(2,1,1)
M(2,3,0)
<2,1,1>
M(3,1,0)
5
i=1
j=3
n=1
M(1,3,1)
M(1,2,0)
<1,3,1>
M(2,3,0)
7

CAPITOLUL 4

Obiective didactice

În acest capitol vom trata o parte importantă a predării oricărei discipline şi anume *obiectivele didactice generale* şi *particulare*. Deoarece obiectivele nu sunt independente, am preferat ca şi în acest capitol să dăm exemple globale.

1. OBIECTIVELE MAJORE ALE STUDIULUI INFORMATICII

Ţinând cont de faptul că prezenta lucrare se adresează profesorilor şi viitorilor profesori de Informatică, am încercat, fără a avea pretenţia că am atins toate aspectele, să formulăm o ierarhie a obiectivelor cadru şi operaţionale ce trebuie atinse prin studiul disciplinelor de informatică în liceu, jalonând astfel etapele de pregătire ale elevilor. **Obiectivele cadru** au un grad ridicat de generalitate şi complexitate şi se referă la formarea unor capacităţi şi aptitudini specifice disciplinei şi sunt urmărite pe o întreagă perioadă de şcolarizare. **Obiectivele de referinţă**, specifică rezultatele aşteptate ale învăţării şi urmăresc în special progresul realizat în acumularea de cunoştinţe şi în formarea deprinderilor, de regulă pe perioada unui an de studiu. Transformările care au loc în societate, dezvoltarea şi răspândirea Informaticii, pătrunderea rapidă în viaţa economică, socială şi în învăţământ a celor mai noi realizări în domeniul hardware-ului şi software-ului, impun o diversificare a pregătirii elevilor de liceu în acest domeniu. Învăţământul preuniversitar trebuie să asigure în primul rând dobândirea unor cunoştinţe de Informatică la nivel de cultură generală. Totuşi, cunoştinţele de tehnologia informaţiei, utilizarea calculatoarelor în rezolvarea problemelor profesionale în diversele domenii ale vieţii economice, reprezintă o cerinţă a integrării în diferitele domenii profesionale ale momentului. Din acest motiv, este posibil să admitem şi introducerea/predarea în liceu (la un nivel corespunzător) a unor elemente de programare mai complexă (C++, WEB, reţele etc.). Astfel, în funcţie de filieră şi specializare, elevii trebuie să dobândească, până la un anumit nivel de aprofundare, un sistem de cunoştinţe, relativ la prelucrarea informaţiei cu ajutorul calculatoarelor personale. Pentru realizarea acestui obiectiv pedagogic considerăm că este necesar ca elevii:

i) Să dobândească **cunoştinţele necesare înţelegerii principalelor aspecte legate de noţiunea de informaţie** (culegere, prelucrare, stocare, transmitere).

ii) Să-şi formeze şi modeleze **modul de gândire şi abordare a problemelor**. Asemenea tuturor ramurilor ştiinţei, Informatica dezvoltă gândirea, având un rol esenţial în procesul

de învățare, în formarea caracterului şi a personalității. În plus, Informatica formează şi dezvoltă o manieră sistemică de abordare, provoacă o analiză progresivă a detaliilor, o rezolvare în contextul general a problemelor particulare. Aceasta este **gândirea algoritmică**, practică, diferită cumva de gândirea teoretică şi abstractă. Această manieră de abordare a problemelor leagă cunoştințele de programare de contextul *bazei de date* pe care o prelucrează şi de cel al soluțiilor pe care le va obține. Formarea unei gândiri algoritmice, analitice şi sistematice şi a unui mod de lucru ordonat are consecințe deosebite în evoluția viitoare a elevului şi este un obiectiv esențial al studiului informaticii în învățământul preuniversitar.

iii) Să-şi formeze şi să-şi dezvolte **deprinderi de a munci individual şi în echipă**. Cu riscul de a ne repeta, trebuie să subliniem că, chiar dacă munca în Informatică este aparent individuală, activitatea colectivă este esențială în conceperea şi realizarea bazelor de date mari şi a produselor software de dimensiuni medii sau mari. Se impune formarea la elevi a acelor deprinderi elementare de lucru cu calculatorul, care oferă şansa unei învățări în ritmul propriu al fiecăruia, dar şi posibilitatea asimilării lucrului în echipă. Acesta va fi un element de esențial de integrare socială şi va conduce la formarea unor trăsături de caracter care poate oferi o alternativă „civilizată" individualismului. În viața reală activitățile nu se desfăşoară izolat, de aceea se impune realizarea unor aplicații complexe care necesită lucrul în grup, modularizarea programului şi păstrarea contactelor cu ceilalți membri ai grupului. Se realizează astfel asumarea responsabilității, cu privire la finalizarea propriei munci şi asigurarea condițiilor de finalizare a activității celorlalți membri ai colectivului. Conducerea rațională a activității de proiectare şi programare, dezvoltarea intuiției, face ca elevul să capete încredere în propriile-i forțe.

iv) Să capete deprinderi care-l vor ajuta să devină un utilizator profesionist, adică să dobândească cunoştințele necesare exploatării resurselor hardware şi software puse la dispoziție de tehnologia informatică actuală. Pentru aceasta, elevul trebuie să-şi formeze o cultură generală informatică, care presupune identificarea şi înțelegerea principalelor componente ale calculatorului, funcționarea rețelelor de calculatoare, să dobândească deprinderile necesare de utilizare a noilor produse software. Punctăm din nou că pentru atingerea acestui ultim obiectiv cadru, trebuie urmărite, în mod diferențiat, măcar următoarele obiective de referință:

- **Cunoaşterea** până la un anumit nivel de detaliu, a sistemele de operare/mediilor de programare cele mai des folosite (**MS-DOS**, **Windows**, **Unix**, **Linux** etc.).
- **Cunoaşterea** structurii şi arhitecturii sistemelor de calcul şi a noțiunilor elementare de hard, care să le permită să-şi facă o impresie precisă despre caracteristicile tehnice ale oricărui calculator.
- **Cunoaşterea** unui limbaj de programare de nivel înalt (**Pascal**, **C**, **Prolog**, **Java** etc.) şi a noțiunilor elementare despre limbajele de asamblare (măcar în liceele de specialitate), a limbajului **VisualBasic** sau, de ce nu, **Logo** (chiar în gimnaziu).
- **Cunoaşterea** tehnicilor de proiectare a produselor program cu caracter ştiințific, a metodelor de elaborare a algoritmilor, a algoritmilor fundamentali, a tehnicilor de

optimizare a algoritmilor (elevii ar trebui să aibă şi capacitatea de apreciere destul de exactă a complexității algoritmilor).

- **Cunoaşterea** unor noțiuni privind analiza şi proiectarea aplicațiilor de gestiune economică şi cunoaşterea unui sistem de gestiune a bazelor de date, procesoare de calcul tabelar etc.
- **Cunoaşterea** celor mai uzitate programe utilitare, editoare de texte şi editoare grafice, pachete de programe de compresie (arhivare), programe antivirus, noțiuni primare de inginerie de sistem etc.
- **Cunoaşterea** principalelor modalități de exploatare ale facilităților oferite de rețele (locale şi interconectate), servicii Internet, documente Html, facilități multimedia etc.

v) F**ormarea unei conduite şi a unei moralități profesionale** este un obiectiv esențial. În Informatică, respectarea strictă a **eticii profesionale** este o necesitate din motive de respectare a legii copywrite-ului. Elevii trebuie să conştientizeze impactul social al dezvoltării Informaticii care poate chiar modifica societatea, de aici rezultând necesitatea înțelegerii rolului pe care îl are aceasta în schimbările din viața socială, economică, a aspectelor etice ce derivă din aceste schimbări, a avantajelor şi riscurilor impuse de utilizarea calculatoarelor. Elevii trebuie să cunoască prevederile legale cu privire la dreptul de autor, confidențialitatea informațiilor, protecției bazelor de date, efectele dezvăluirii informației sau distrugerii ei prin spargeri de parole de protecție, virusare, transfer neautorizat etc. Formarea trăsăturilor de caracter nu se poate realiza fără o cunoaştere a istoricului dezvoltării Informaticii ca un domeniu al culturii, fără o cunoaştere a realității şi a perspectivelor, fără impunerea respectului față de valorile materiale şi umane, a respectului față de munca colegului sau a colectivului din care elevul face sau va face parte. Acest aspect trebuie avut în vedere pe toată durata şcolarizării elevului şi nu trebuie să apară ca un scop în sine, ci ca un element de coloratură, în contextul predării altor noțiuni. Formarea unor trăsături ale personalității elevilor, exprimate şi ilustrate prin însăşi produsele lor informatice fac din imaginea unui text sursă, din modul de organizare a instrucțiunilor în program, o oglindă fidelă a personalității intelectuale şi sociale a elevului.

2. PRECIZAREA OBIECTIVELOR

Succesul oricărei activități didactice este condiționat de claritatea şi ordonarea obiectivelor pe care acesta le urmăreşte. Mai mult decât în oricare alt domeniu, procesului de învățământ informatic îi este caracteristică *intenționalitatea*, orientarea către realizarea unor obiective, spre producerea unor schimbări şi transformări care să poată fi controlate şi dirijate. În acest spirit, cea mai importantă condiție pentru reuşita predării Informaticii este structurarea, conştientizarea şi ierarhizarea unor obiective generale şi specifice, adaptate particularităților de vârstă ale elevilor, conținutului

cunoştinţelor şi pregătirii ştiinţifice şi metodice a elevilor. *Un **obiectiv didactic** este o descriere a unui ansamblu de comportamente şi performanţe de care elevul trebuie să se arate capabil*. Un obiectiv este o intenţie comunicată printr-o declaraţie ce descrie modificările pe care dorim să le provocăm elevului. Obiectivele integrează organic comportamentul, adică activitatea vizibilă manifestată de elev cât şi activitatea mentală mai puţin vizibilă. **Obiectivele generale** ale predării Informaticii au anumite determinări care trebuie să pună în evidenţă:

- Importanţa Informaticii în lumea contemporană, în ştiinţă, în tehnică sau economie.
- Necesitatea învăţământului de Informatică şi rolul acestuia în formarea culturii generale şi nu numai.
- Necesitatea dezvoltării capacităţii intelectuale şi a gândirii algoritmice.
- Necesitatea formării elevului pentru activităţile viitoare, ca utilizator al calculatoarelor, la diferite nivele.

Fixarea obiectivelor generale ale Informaticii trebuie să răspundă la măcar la următoarele două întrebări:

- **De ce se predă informatica în şcoală?**
- **Ce se urmăreşte prin includerea ei în planul de învăţământ?**

Obiectivele predării ştiinţelor informatice în şcoală includ cu siguranţă:

- Trezirea interesului pentru studiul acestora.
- Formarea priceperilor şi deprinderilor de bază în utilizarea şi exploatarea calculatoarelor.
- Stimularea creativităţii.
- Integrarea utilizării Informaticii în modul de gândire şi de viaţă al elevului.

În afara obiectivelor sale generale, Informatica participă prin mijloace ce-i sunt proprii la modelarea personalităţii, nu numai sub aspect intelectual ci şi sub aspect estetic şi moral (*estetic*: programarea este o artă iar personalitatea autorului se manifestă prin opera sa; *moral*: activitatea în domeniul Informaticii nu se poate desfăşura în afara unei etici profesionale sănătoase, dacă ne gândim doar la pericolul hacker-ilor şi la relaţia defectuoasă a acestora cu cyber-space-ul virtual de pe Internet). Dintre **obiectivele specifice**, putem desprinde anumite **obiective derivate** care pot fi la rândul lor structurate pe trei nivele:

- Nivelul obiectivului (elev).
- Nivelul subiectului (profesor).
- Nivelul acţiunii comune.

La nivelul elevului, obiectivele derivate sunt:

- Integrarea şi asimilarea cunoştinţelor cuprinse în programă.
- Memorarea activă a acestor cunoştinţe.
- Dezvoltarea judecăţii deductive şi inductive.
- Conştientizarea procedeelor ce stau la baza raţionamentelor.
- Formarea capacităţii de analiză şi sinteză.

- Formarea capacității de structurare şi planificare.
- Formarea capacității de abordare a unei probleme complexe.

La nivelul subiectului (profesor), obiectivele derivate se referă la capacitatea de apreciere a fenomenelor şi rezultatelor.

Nivelul acțiunii are în vedere asimilarea de către elev a noțiunilor şi aplicarea lor în practică. Pe baza acestor considerații se pot elabora şi delimita **obiectivele operaționale** ale fiecărui capitol, lecție ş.a.m.d. cu detalierea fiecărei componente. Formularea obiectivelor operaționale trebuie făcută în termeni comportamentali cât mai precişi, care să excludă formulări vagi. Formularea obiectivelor operaționale presupune:

- Identificarea *performanței finale* care trebuie realizată.
- Descrierea în detaliu a *condiției esențiale în care poate să se producă* comportamentul respectiv.
- Precizarea *nivelului de performanță* la care trebuie să se ajungă pentru a fi acceptată ca atare.

În acelaşi timp, trebuie să se cunoască:

- Cine va dirija modelarea unui comportament dorit ?
- Ce comportament observabil va dovedi că obiectivul a fost atins?
- Care va fi produsul (performanța) acestui comportament?
- În ce condiții trebuie să aibă loc comportamentul?
- Pe baza căror criterii apreciem că produsul este satisfăcător?

3. ANALIZA RESURSELOR

În acest moment trebuie să răspundem la întrebarea: ***Cu ce*** *pot realiza obiectivele stabilite* ? Sunt necesare:

- O analiză a resurselor psihologice, care necesită cunoştințe de psihologia copilului, a capacității de învățare, a particularităților de vârstă şi natură psihică, motivația învățării etc.;
- O analiză a resurselor materiale.
- O analiză a conținutului învățării.

Programa şcolară determină conținutul învățării, dar acest conținut este prelucrat după două categorii de obiective:

- **informative** (ce va şti elevul?);
- **formative** (ce va putea face elevul?).

Această clasificare trebuie să stea la baza întocmirii planului calendaristic. Întocmirea planificării calendaristice se poate face după următoarea procedură:

- Se va selecta din manual conținutul informativ propus de programă.
- Acest conținut va fi coroborat cu conținutul formativ pretins (priceperi, deprinderi, abilități).
- Ambele vor fi raportate la elementul timp prin stabilirea numărului de ore afectate fiecărei teme.

O **posibilă** rubricație pentru **planificarea anuală** este următoarea:

Planificare anuală

Disciplina : Sisteme de calcul (opțional)
Clasa a IX -a : Profilul matematică-informatică

Disciplina	Capitolul	Semestrul I		Total ore
		I	II	
Sisteme de calcul (opțional)	1. Sisteme de calcul (evoluție, generații de calculatoare, caracteristici)	2		2
	2. Arhitectura generală a unui sistem de calcul	3		3
	3. Sisteme de operare	1		1
	4. Sistemul de operare MS-DOS	1		1
	5. Comenzi interne	3		
	6. Comenzi externe	3		1
	7. Recapitulare	1		6
	8. Fişiere de comenzi		2	2
	9. Sistemul de operare UNIX		1	1
	10. Sisteme multiutilizator		1	1
	11. Comenzi UNIX		4	4
	12. Editoare de text UNIX		3	3
	13. Mediul Shell		3	3
	14. Mecanismul realizării proceselor		2	2
	15. Recapitulare finală		1	1
Total		17	17	34

Pentru realizarea **obiectivelor generale**, odată ce acestea au fost stabilite, este necesară elaborarea unei planificări semestriale pe capitole, detaliate la nivel de lecție. Pentru fiecare capitol se fixează un *obiectiv general (final)*, care orientează formularea **obiectivelor operaționale** pentru fiecare lecție. Aceste obiective se formulează ținând seama de criteriile de conținut (reprezentate prin obiectivele operaționale) şi criteriile relative (analizând performanța unui elev prin comparație cu performanța clasei lui sau altor clase din aceeaşi şcoală sau din alte şcoli). Astfel, criteriile de optimalitate vizează

creşterea performanţei în termeni relativi şi de conținut. O **posibilă** rubricație pentru o **planificare semestrială** poate fi următoarea:

Planificare calendaristică semestrială

Disciplina : Sisteme de operare
Semestrul I

Capitolul / Săptămâna	Nr. ore	Subiectul lecției	Nr. ore Lecție
1. Sisteme de calcul (evoluție, generații de calculatoare, caracteristici) 1-2	2	1.Tipuri de sisteme de operare (evoluție, caracteristici) 2. Suporturi de memorare	1
2. Arhitectura generală a unui sistem de calcul 3-5	3	1. Arhitectura unui sistem de calcul 2. Părțile componente ale unui sistem 3. Structura şi funcționarea unui sistem de calcul	1
3. Sisteme de operare 6	1	1. Sisteme de operare. Exemple, aprecieri, studiul comparativ	1
4. Sistemul de operare MS-DOS 7	1	1. Definiție, funcții, structură	1
5. Comenzi interne 8-10	3	1. Comenzi pentru lucru cu directoare 2. Comenzi pentru lucru cu fişiere 3. Comenzi pentru pregătirea şi utilizarea discurilor	3
Comenzi externe 11-13		1. Principalele comenzi externe 2. Programe utilitare curente	1
Recapitulare 14		1. Recapitulare	1

4. ELABORAREA STRATEGIEI DIDACTICE

Elaborarea strategiei presupune alegerea unui sistem de *forme, metode, materiale şi mijloace*. De selectarea şi combinarea acestora depinde reuşita activității didactice. Selectarea tehnicilor de învățare se face în funcție de materialele didactice, care sunt dependente de metodele utilizate, iar metodele sunt determinate de obiective, conținut şi colectivul de elevi. Astfel, obiectivele operaționale informative vor pretinde metode mai

pasive (profesorul spune - elevul ascultă). Îmbinarea judicioasă a celor „3M" (**M**etode, **M**ateriale, **M**ijloace), asigură succesul lecției. Această corelare este gândită din momentul întocmirii *scenariului didactic* prin care se înțelege o descriere anticipată a desfăşurării pas cu pas a unei lecții. Gradul de detaliu vizează aspectele esențiale ale condiției elevului şi schimbările pe care dorim să le realizăm.

5. MOMENTELE LECȚIEI

Desigur că principalele momente ale unei lecții pot fi sumarizate după cum urmează:

- Captarea atenției.
- Enunțarea obiectivelor.
- Reactualizarea cunoştințelor învățate anterior.
- Prezentarea conținutului noii lecții.
- Dirijarea învățării.
- Asigurarea feed-backului.
- Intensificarea atenției.
- Asigurarea transferului de cunoştințe.

Succesiunea şi importanța lor variază de la lecție la lecție, de la tip la tip de lecție. **Lecțiile pot fi împărțite în trei mari categorii** (vor exista desigur **lecții mixte**):

- De **comunicare** de noi cunoştințe.
- De **fixare** şi formare de priceperi şi deprinderi.
- De **recapitulare** şi sistematizare.

Subliniem încă odată că lecția este în concepția noastră un act de creație care nu se poate încadra în şabloane. Profesorul se bazează doar pe anumite sugestii pentru întocmirea de diverse scenarii. Vom prezenta în continuare un proiect de tehnologie didactică pentru o lecție mixtă.

Proiect de tehnologie didactică

*Disciplina***:** Sisteme de operare.
Profesor: -.
*Clasa***:** a IX –a.
*Data***:** 7.03.2000 (Semestrul I – săptămâna 10).
Tema lecției **:** Comanda Format.

*Obiective***:** Însuşirea şi formarea deprinderilor de utilizare a suporturilor magnetice în diverse situații limită (înainte de orice utilizare, la incidente, etc.). Legarea şi încadrarea acestei deprinderi de cele dobândite anterior.

*Material didactic***:** Calculator echipat cu hard-disc şi unități de disc flexibil de $3^{1/2}$ şi/sau $5^{1/4}$, precum şi suporturile magnetice aferente.

*Metode***:** metoda demonstrației (practice).

Evenimentele lecției care duc la realizarea obiectivelor. Strategia didactică	**Activități ale lecției (*metodă*)**
1. Captarea atenției şi trezirea interesului pentru lecție (1 minut)	Profesorul arată că se va studia o comandă nouă extrem de utilă în folosirea calculatorului fără de care activitatea ar fi aproape imposibilă.
2. Informarea elevului asupra obiectivului urmărit (2 minute)	Profesorul cere elevilor să se gândească la modul în care ar trebui să acționeze dacă doresc să păstreze anumite informații de pe hard-disc sau dacă ar constata un incident care priveşte sistemul de operare.
3. Reactualizarea cunoştințelor dobândite anterior (5 minute)	Fişiere sistem, încărcarea sistemului memorarea fişierelor, păstrarea lor. (*Conversație*)
4. Dirijarea învățării (10 minute)	Profesorul prezintă comanda. Se pleacă de la simplu la complex. Format A: Format A: /V /Q /U
5. Prezentarea materialului stimulativ (calculator şi dischete) (3 minute)	Profesorul prezintă unitățile de disc flexibil precum şi principalele lor caracteristici
6. Asigurarea conexiunii inverse (3 minute)	Ce facem dacă unitatea şi discul flexibil diferă ? /N /T (*Problematizare*)
7. Obținerea performanțelor (10 minute)	Elevii execută formatări cu diverse opțiuni (*Muncă independentă*)
8. Dirijarea învățării (4 minute)	Profesorul prezintă o dischetă defectă. Soluții de reutilizare
9. Evaluarea performanței (3 minute)	Formatarea unei dischete de 720 pe o unitate de 1,2 (programul ATDQ şi variante)
10. Recapitulare (6 minute)	Stabilirea formei generale
11. Tema pentru acasă (4 minute)	Se recapitulează noțiunile predate (fixarea cunoştințelor)

Evaluarea este posibilă numai în cazul în care formularea obiectivelor a fost făcută în termeni comportamentali preciși, care evidențiază performanța aşteptată de la elevi. Este de altfel necesar să organizăm acțiuni care să ne permită să constatăm

realizarea acestei performanţe şi să nu scăpăm din vedere efortul depus pentru obţinerea lor. Evaluarea trebuie făcută după criterii absolut obiective. Fixarea unui obiectiv şi principiu (*ce se face* şi *ce se obţine*), este decisivă pentru construirea unui plan de lecţie. Aceasta este baza pentru construcţia planului de lecţie şi a prezentării (alegerea metodelor şi materialelor didactice necesare).

6. CLASIFICAREA OBIECTIVELOR

Există în realitate două mari categorii de obiective care trebuiesc avute în vedere în momentul proiectării unei lecţii:

- Obiective sub raport stadial.
- Obiective sub raport psiho-pedagogic.

6.1. Obiective sub raport stadial

Aceste obiective, la rândul lor, se pot împărţi în:

Obiective fundamentale (finale). Acestea definesc elementele şi sarcinile rezultate din delimitarea scopului final al educaţiei, cum ar fi cele legate de formarea unei personalităţi puternice, complexe, cu o mare dispoziţie spre iniţiativă şi creativitate. Avem în vedere:

- Formarea *capacităţii de asimilare* a cunoştinţelor de către elevi.
- Formarea *capacităţii de transfer* a cunoştinţelor şi a experienţei deja dobândite la rezolvarea unor sarcini necunoscute, apărute pe parcursul derulării procesului didactic.
- Formarea *limbajului ştiinţific de profil*.
- Formarea *unei atitudini ştiinţifice*; trebuie creat un respect al elevului pentru ştiinţă şi importanţa acesteia în evoluţia sa ulterioară; acesta trebuie să înţeleagă că procesul de cunoaştere nu se încheie într-o perioadă determinată de timp, că procesul de cercetare - pentru a fi eficient - trebuie să prelucreze orice informaţie în mod critic, abţinându-se de a face afirmaţii categorice/definitive.

Obiective intermediare. Asemenea obiective sunt formulate în planul-cadru al procesului de învăţământ (privit ca un sistem complex şi într-o permanentă evoluţie). În primul rând, se urmăreşte *dobândirea unei culturi generale* de bază (în învăţământul preuniversitar), a unei culturi de specialitate (în învăţământul superior), sau chiar a unei meserii (şcoli de profil).

Obiective secvenţiale. Acestea reprezintă obiectivele *specializate*, orientate spre anumite laturi ale procesului de educaţie: intelectuală, tehnologică, profesională, morală, estetică, fizică etc.

Obiective operaţionale. Ele privesc *îndeplinirea concretă a unor activităţi curente*, cum ar fi cele legate de predarea unei lecţii sau de exemplificarea unor teme de laborator.

6.2. Obiective sub raport psiho-pedagogic

Ele reprezintă obiectivele didactice *necesare formării de capacităţi intelectuale (teoretice, practice) şi/sau afective.* Şi acestea le putem subîmpărţi în mai multe categorii:
Obiective cognitive/de cunoaştere. Prin acestea se urmăreşte formarea/dezvoltarea următoarelor capacităţi intelectuale:

- **cunoaşterea:** posibilitatea, în principal, a îndeplinirii sarcinilor legate de memorarea, reproducerea şi recunoaşterea materiei de asimilat;
- **înţelegerea:** se referă la *transpunere*, *interpretare* şi *extrapolare*.
    1. *Transpunerea* înseamnă reformularea unei definiţii/noţiuni/rezultat cu propriile cuvinte; de exemplu, traducerea unui algoritm dintr-o reprezentare oarecare într-un limbaj implementat.
    2. *Interpretarea* înseamnă înţelegerea comportării/evoluţiei unui obiect/sistem dat, într-un context/mediu clar precizat.
    3. *Obiectivele (legate) de extrapolare* au drept consecinţă căpătarea îndemânării de a se evidenţia consecinţe noi, neidentificate încă în procesul anterior.
- **analiza:** demonstrează capacitatea elevului de a gândi discriminativ, profund, deductiv, de a distinge faptele concrete (noi) de ipotezele (generale) de lucru;
- **sinteza:** vizează - în principal - activitatea intelectuală de corelare logică a fenomenelor observate şi a cunoştinţelor asimilate, în vederea realizării unor lucrări cu caracter personal;
- **evaluarea:** implică posibilitatea formulării de către elevi a unor judecăţi de valoare, originale (de natură ştiinţifică, socială, culturală), raportate bineânţeles la cantitatea de informaţii acumulate până în acel moment.

Obiective psihomotorii/acţionale. Asemenea obiective includ formarea de perceperi, capacităţi, deprinderi motorii/practice legate de utilizarea corectă a întregii aparaturi de laborator (*tastatură*, *mouse*, *joy-stick*, etc.). Totul trebuie însuşit într-un mod profesional şi utilizat rapid, precis, cu o bună coordonare a mişcărilor şi implicând agilitate şi suplețe.
Obiective afective (conative). Acestea au scopul de a dezvolta emoţii şi sentimente superioare, contribuind la formarea conştiinţei şi conduitei morale, vizează deci, în mare, formarea intereselor, atitudinilor şi valorilor etico-morale, a personalităţii elevului. Personalitatea poate fi formată începând cu o vârstă foarte fragedă, etapizat şi utilizând idei, norme, practici şi valori deja recunoscute. Deşi nu crearea de asemenea deprinderi reprezintă scopul principal al predării Informaticii în gimnaziu/liceu, rezultatele indirecte pot fi spectaculoase. Să ne amintim doar de *societatea informaţională* şi de faptul că - practic - *Informatica poate deveni un mediu de lucru pentru toate celelalte discipline*.

7. FORMULAREA ŞI OPERAȚIONALIZAREA OBIECTIVELOR

Nu considerăm că este scopul principal al acestei cărți de a intra în detaliile elaborării unui plan de învățământ sau a unei programe analitice pentru o disciplină specifică, fie ea *ştiința calculatoarelor* sau *tehnologia informației şi comunicării*. Acestea fac parte din strategiile (pe termen scurt sau lung) de dezvoltare/promovare a disciplinei şi sunt de competența Consiliilor Profesorale, Inspectoratelor Şcolare, Senatelor Universitare, Ministerului Educației Naționale etc. În momentul în care un plan de învățământ şi o programă analitică sunt însă fixate, **formularea obiectivelor** *este obligația profesorului şi constituie o parte indispensabilă a oricărei planificări didactice generale.* **Operaționalizarea** acestora, presupune în plus faptul că un cadru didactic are o orientare globală şi coerentă asupra întregului proces de învățământ, că el cunoaşte şi aplică în mod curent elementele de metodică, că *procesul în sine de coordonare a învățării în clasă nu mai are secrete*. În urma oricărei lecții, elevii trebuie să dobândească anumite cunoştințe, să aibă abilitatea de a le structura (analiza, sintetiza) în mod creator. Aceştia trebuie să aibă şi posibilitatea de a se *manifesta direct*, intervenția profesorului trebuind să fie mai degrabă discretă. Prin urmare, *operaționalizarea* înseamnă transpunerea scopurilor urmărite de obiectivele formulate în termenii unor operații sau acțiuni sau manifestări observabile şi aflate în concordanță cu cerințele generale. **Obiectivele operaționale** „sunt imediate", putând însă avea în anumite situații şi o finalitate pe un termen mai lung; aceasta în ideea că deprinderile şi cunoştințele dobândite anterior vor trebui să fie completate prin acțiuni viitoare care să contribuie decisiv la includerea lor în sistemul individual de informații şi îndemânări. Dacă, la un moment dat, avem în vedere o cantitate mai restrânsă de date (informații), vom urmări definirea a câte unui obiectiv de recunoaştere, de înțelegere, de aplicare, de reprezentare etc. Dacă această cantitate este mai completă (sau mai complexă), putem adăuga şi un obiectiv general (de genul *formare şi utilitate*). Operaționalizarea obiectivelor trebuie să implice, eventual gradat, etape diferite de dificultate care să precizeze:

- obiectivele în termeni comportamentali observabili;
- sarcinile concrete de învățare, precum şi contextul de realizare;
- informația (*finală*) cerută de obiectiv;
- criteriul de succes şi modul de evaluare.

Considerăm util să încheiem şi acest capitol cu un exemplu general.

8. EXEMPLU

Vom prezenta o implementare a algoritmilor de parcurgere a grafurilor (neorientate) de tip **BFS** (**B**readth **F**irst **S**earch) şi **DFS** (**D**epth **F**irst **S**earch) pornind de la un nod fixat i. Pe scurt, aceşti algoritmi pot descrişi în felul următor. La o mulțime S (inițial, aceasta are un singur element şi anume pe i) deja selectată de noduri se adaugă la fiecare pas un nod nou dintre cele neselectate încă. Noul nod este succesorul unui nod

(ales) j din S. În cazul **BFS**, se parcurge graful *în lățime*, adică se *vizitează* vecinii nodului j care nu sunt în S şi procesul continuă într-un mod similar. În cazul **DFS,** procedeul anterior este aplicat fiilor direcți ai nodului j. *Coada (respectiv stiva)* pot fi utilizate pentru o implementare performantă a algoritmilor **BFS** şi **DFS**. Pentru detalii, se pot consulta <16, 21, 25, 30>.
(?????-trebuie pus)

CAPITOLUL 5

Metode, tehnici, procedee didactice

Sarcinile didactice se realizează cu ajutorul *metodelor, tehnicilor şi procedeelor didactice*. Folosirea judicioasă a acestora are o deosebită importanță pentru reuşita activității „de la catedră". Pe de altă parte, conținuturile fiecărei discipline şi obiectivele pe care şi le propune să le îndeplinească pretind metode specifice. **Adoptarea** şi nu **adaptarea** metodelor de predare ale unor discipline la alte discipline pot conduce la rezultate contradictorii. Aplicarea metodelor, tehnicilor şi procedeelor didactice generează activități de învățare specifice.

1. METODE GENERALE DE ÎNVĂȚARE

Trebuie să avem în vedere care dintre obiectivele operaționale, rezultate în urma studierii obiectivelor cadru şi de referință sunt urmărite prin studiul disciplinelor de Informatică, ce cunoştințe noi vor asimila elevii şi ce cunoştințe deja dobândite în cadrul altor discipline vor fi utilizate. Cert este că Informatica poate *adopta* şi *adapta* metode de predare de la alte discipline, dar acest lucru trebuie să se facă ținându-se cont de: dinamica conținuturilor şi particularitățile metodice ale predării disciplinei; individualizarea învățării Informaticii ca disciplină deschisă şi dinamică; constructivism, care pretinde o participare prioritară conştientă a elevului la procesul de autoinstruire; studiul Informaticii atât ca disciplină autonomă cât şi ca instrument operațional al altor discipline. Dintre metodele de predare specifice, de exemplu, Matematicii, amintim:

- Metoda demonstrației.
- Metoda reducerii la absurd.
- Metoda inducției matematice (structurale).

Aceste metode nu fac obiectul cărții de faţă. Cititorul interesat poate consulta <2>, <10>, <38>. În cele ce urmează se vor analiza **metodele generale** utilizate în predarea Informaticii:

1. **Expunerea (sistematică, a cunoştințelor).**
2. **Conversația.**
3. **Problematizarea.**
4. **Modelarea.**
5. **Demonstrarea folosind materialul intuitiv.**

6. **Exercițiul.**
7. **Învățarea pe grupe mici.**
8. **Munca cu manualul.**
9. **Jocurile didactice.**
10. **Instruirea programată.**

În tratarea acestor metode se vor urmări cu predilecție particularitățile specifice predării disciplinelor de Informatică şi în special, aplicațiile practice de laborator şi contribuția Informaticii la realizarea obiectivelor didactice ale altor discipline din învățământul preuniversitar.

1.1. Expunerea sistematică a cunoştințelor

Dintre formele pe care le îmbracă expunerea sistematică a cunoştinţelor (*povestirea, prelegerea, descrierea, explicaţia, conversaţia etc.*), opinăm că Informatica utilizează cu precădere explicaţia. Elementele explicative domină procesul de instruire informatică, acestea fiind caracteristice atingerii unor obiective de referinţă care cuprind formarea de deprinderi şi abilităţi practice de utilizare a unor produse soft deseori complicate şi dominate de interfeţe *neprietenoase* faţă de utilizator (netransparente). Ceea ce conferă o accentuată notă de adaptabilitate este operativitatea impusă de aplicarea acestei metode prin alternarea expunerii cu demonstraţia practică, elevii fiind astfel scoşi din pasivitatea posturii de simpli receptori. Analogiile cu situaţii cunoscute fac din receptorul pasiv un participant activ la expunere. Expunerea, nici la disciplinele cărora le este caracteristică ca metodă, nu se desfăşoară în condiţii perfect univoce, adică fără alternative şi reveniri. La informatică aceasta se întâmplă cu atât mai puţin. Elevul primeşte în condiţii univoce doar ceea ce i se comunică în funcţie de nivelul de cunoştinţe dobândit, de propriile-i presupuneri, de experienţa sa practică, de nivelul său de gândire, de înţelegerea codului de comunicaţie, ca să nu mai vorbim de oscilaţiile de atenţie. Profesorul trebuie să reproiecteze lecţia prin prisma posibilităţilor elevilor şi cu mijloacele lor de gândire. Accentul trebuie pus pe raţionament, prin argumentări temeinice, prin scoaterea în evidenţă a modului în care trebuie să gândească. Expunerea trebuie să fie însoţită de un control permanent al gradului de receptivitate al clasei, urmărindu-se *mimica* elevilor (edificatoare în special la elevii mici), *satisfacţia înţelegerii* lecţiei sau *îngrijorarea şi neliniştea* în cazul în care elevul a pierdut firul explicaţiei citindu-se pe faţa elevilor. Întrebările, repetiţia, explicaţiile suplimentare, analogiile cu alte noţiuni cunoscute, permit realizarea unui control permanent al receptivităţii la expunere. În Informatică recurgem neapărat la metoda expunerii (explicaţiei) atunci când tema este complet nouă şi printr-o *metodă activă* nu se poate descoperi noutatea, sau metoda activă este ineficientă din punct de vedere al operativităţii. Astfel este necesară această metodă pentru a înţelege noţiunea de algoritm (inclusiv exemplificările clasice), de structură de date (inclusiv modalităţile de reprezentare), de comandă, funcţie sau procedură standard (în legătură cu sistemul de operare sau mediul de programare ales), de raţionament (într-un spaţiu închis ales) şi chiar a modalităţii de prezentare şi introducere a unor programe utilitare, soft-uri de aplicaţie etc. În acest context, pentru prezentarea comenzilor unui sistem de operare, a unui editor de texte (sau grafic), a altor soft-uri mai complicate (prevăzute de programa şcolară), se poate recurge la următoarele (sub)metode:

- Expunerea (la tablă, prin slide-uri pe retroproiector sau prin PowerPoint) cu „desenarea” meniurilor şi prezentarea funcţiilor fiecărei opţiuni, urmând ca elevul (prin aplicaţiile de laborator) să exerseze fiecare funcţie în parte, individual sau în grupe mici de lucru.

- Prezentarea meniurilor şi funcţiilor fiecărei opţiuni simultan cu exersarea acestora în cadrul orelor de aplicaţii practice de laborator.
- Prezentarea meniurilor şi funcţiilor fiecărei opţiuni simultan cu demonstrarea practică în momentul prezentării lor de către profesor, sarcina elevului fiind numai aceea de a urmări şi reţine modul de executare a operaţiilor prezentate de profesor, urmând ca elevul să aplice cunoştinţele dobândite în cadrul orelor de laborator, în aplicaţii ample (integrate, de dorit, într-un mediu economic clar) care necesită utilizarea în mod repetat şi în situaţii diferite a funcţiilor fiecărei opţiuni din meniul discutat.

Fiecare dintre variantele de mai sus au avantajele şi dezavantajele lor. Prima variantă este cea mai des folosită datorită faptului că de regulă profesorul nu are la dispoziție un laborator şi pentru predare (iar aceasta se face cu întreaga clasă). Ea prezintă dezavantajul că elevul *nu vede pe viu* efectul executării fiecărei opțiuni (profesorul fiind nevoit în acest caz să-l descrie în cuvinte), dinamica transformărilor şi efectul video al acestora fiind greu de redat în cuvinte. Singurul avantaj este cel al obținerii de către elev al unui rezumat logic şi coerent după care se va ghida în timpul realizării unor aplicații practice. A doua variantă înlătură dezavantajul neobservării pe viu a efectului executării fiecărei opțiuni, dar atenția elevului este îndreptată spre realizarea practică (simultan cu comunicarea modului de realizare a funcțiilor opțiunilor din meniuri). Astfel, o parte dintre funcții sunt abordate prea „abrupt" sau sunt chiar omise, iar altele sunt exersate prea mult. La acest dezavantaj se adaugă şi reducerea randamentului prin faptul că profesorul trebuie să urmărească modul în care fiecare elev sau grupă aplică funcția prezentată şi să intervină ori de câte ori un elev sau o grupă este în impas. În plus, unii elevi îşi formează mai repede deprinderea utilizării iar alții mai greu, primii fiind tentați să încerce între timp alte opțiuni (chiar neprezentate încă de către profesor), ceea ce creează disfuncționalități în desfăşurarea lecției, aprecierea gradului de asimilare şi chiar formarea unor idei greşite de utilizare (datorate încercărilor individuale, necoordonate). Pe lângă acestea, se pierde din vedere şi realizarea unui rezumat sistematic al modului de utilizare, elevul fiind tentat să exerseze imediat funcția şi uită să-şi noteze „în stil propriu" modul de utilizare al acesteia. Ultima variantă pare să cumuleze toate avantajele celor anterioare prin faptul că elevul urmăreşte şi reține (neavând alte preocupări care să-i distragă atenția) modul în care profesorul execută (corect) şi explică simultan, elevii putând nota tot ceea ce acesta prezintă. Această manieră de expunere înlătură formarea unor deprinderi greşite, mărind randamentul la predare şi asimilarea noilor cunoştințe. Această variantă are însă şi un dezavantaj. Este vorba despre necesitatea existenței unei dotări speciale, care să permită observarea în bune condiții de către toți elevii clasei a ecranului calculatorului pe care profesorul face demonstrația. Utilizarea unui retroproiector sau a unui videoproiector are multe inconveniente (înafară de costul ridicat), printre care faptul că trebuie să existe anumite condiții de mediu specifice în sala se clasă. De exemplu, pentru grupe mici poate fi folosit numai calculatorul ca atare, dacă elevii pot fi aşezați în preajma acestuia astfel încât fiecare să poată observa fără efort ecranul. Indiferent de conținutul lecției, metoda expunerii nu se foloseşte singură decât foarte rar pe parcursul unei ore întregi, aceasta alternând cu alte metode de predare. Pe de altă parte, există o tendință accentuată a cadrelor didactice de a nu-şi propune aprioric folosirea cu precădere a niciunei metode, ceea ce este foarte dăunător.

1.2. Metoda conversației

Metoda conversației se referă la dialogul dintre profesor şi elev, în care profesorul nu trebuie să apară în rolul examinatorului permanent, ci în rolul unui colaborator care nu numai întreabă ci şi răspunde la întrebările elevilor. Prin metoda conversației se stimulează gândirea elevilor în vederea însuşirii, fixării şi sistematizării cunoştințelor şi deprinderilor, în vederea dezvoltării spiritului de colaborare şi de echipă. Se asigură astfel o participare activă din partea elevilor, întrebările putând fi adresate (teoretic) în orice moment al lecției. Metoda conversației este frecvent utilizată în învățarea Informaticii, ea implicând un dialog continuu între elev şi profesor, respectându-se anumite reguli elementare de colaborare constructivă care să nu determine diminuarea demersului didactic, ci să-l amplifice şi să-l consolideze. Conversația didactică poate să îmbrace forme diferite în funcție de anumite criterii. În funcție de *numărul de persoane*, ea poate fi:

- *Individuală*. Se poartă între un elev şi profesor.
- *Colectivă* sau frontală. Întrebările sunt adresate întregii clase, iar răspunsurile „vin" de la diferiți elevi.

După *obiectivele urmărite* în diferite variante de lecții, conversația poate fi:

- *Introductivă*. Aceasta este folosită în momentul captării atenției şi reactualizării cunoştințelor asimilate anterior, pentru a trezi interesul pentru lecția care urmează.
- *Expozitivă*. În timpul prezentării unei noi lecții, ea poate trezi interesul pentru *fixarea noilor cunoştințe*.
- *Recapitulativă*. Este utilizată atunci când se urmăreşte recapitularea şi generalizarea unor rezultate prezentate anterior.
- *Evaluativă*. Este indicată desigur pe parcursul procesului de verificare şi evaluare.
- *Dezvoltată*. Aceasta este destinată prezentării unui nou subiect, nu complet necunoscut.

Caracteristicile principale ale întrebărilor, indiferent de forma de conversație, impun precizie şi vizarea unui singur răspuns. De multe ori se pun întrebări vagi care încep cu *Ce puteți spune despre ...* sau *Ce ştiți despre ...*, care plasează elevul într-un dubiu total în legătură cu conținutul răspunsului. Din aceeaşi gamă face parte şi celebrul îndemn de evaluare *Prezintă subiectul pe care-l cunoşti tu cel mai bine*. Întrebarea nu este normal să conțină răspunsul, sau să ceară un răspuns prin diferit de *da* sau *nu*. Ea contribuie la dezvoltarea gândirii. De asemenea, răspunsurile acceptate trebuie să fie corecte, complete, exprimate în termeni preciși, să oglindească o înțelegere efectivă a problemei abordate. Discuțiile au şi rolul de a corecta greşelile din răspuns. Identificarea cauzei, eliminarea greşelii cât şi posibilitatea reapariției ei sunt foarte importante. Conversația are un rol primordial prin faptul că ajută la formarea limbajului informatic, la dezvoltarea raționamentului logic şi a gândirii elevului. Dificultățile pe care elevul le întâmpină în formarea limbajului de specialitate pot lăsa urme în plan afectiv, se pot repercuta asupra dezvoltării intelectuale a acestuia. De aceea se impune o analiză amănunțită a cauzelor acestor dificultăți, iar scoaterea lor în evidență trebuie relevate prin examinări (scrise, orale, reprezentări schematice, utilizarea simbolurilor specifice). A fi la curent cu dificultățile de limbaj pe care le au elevii la anumite vârste şcolare şi la un anumit stadiu de însuşire a disciplinei înseamnă în primul rând să nu se abuzeze de termeni de specialitate (înlocuindu-i cu termeni sinonimi din vocabularul curent sau explicându-le sensul, dacă un alt înțeles al termenului este accesibil). Dificultatea formării vocabularului de specialitate constă şi în faptul că aceste cuvinte noi sunt introduse în acelaşi timp cu introducerea noțiunilor noi, ceea ce face ca îmbogățirea limbajului informatic să se facă simultan cu *dezvoltarea şi formarea gândirii informatice*. Stăpânirea limbajului se reflectă în rezolvarea problemelor şi înțelegerea textelor şi documentațiilor de specialitate. Nestăpânirea acestuia provoacă inhibiție, imposibilitatea comunicării sau chiar o comunicare şi o înțelegere defectuoasă, făcându-l pe elev timid, incoerent sau chiar ridicol în exprimare. Această metodă mai are şi următoarele subdirecții:

- *Euristică*. Nu există reguli precise, se bazează doar pe întrebare/răspuns, în functie de evolutia concretă a dialogului.
- *Tip dezbatere*. Se realizează un schimb de păreri în care este implicat un anumit colectiv. Ar fi bine să fie trase şi nişte concluzii care să nu aibă doar un rol istoric.
- *Catihetică*. Aceasta impune efectuarea unor teste care implică memoria.

Este clar că o conversație se face prin întrebări. În plus, acestea trebuie să satisfacă următoarele condiții (unele din ele rezultând din ceea ce am amintit mai înainte):

- Să fie precise (vizând un singur răspuns).
- Să nu conțină răspunsul şi să aibă un rol instructiv.

- Să stimuleze gândirea şi capacitatea de creativitate a elevilor (*De ce?*, *Din ce cauză?*, *În ce caz?* etc.).
- Să fie formulate prin enunțuri variate şi „atrăgătoare”.
- Să se adreseze întregului colectiv vizat.
- Să conțină întrebări ajutătoare atunci când răspunsul este eronat sau parțial.

Răspunsurile acceptate trebuie să fie nu numai corecte ci şi exprimate în termeni precişi şi să oglindească un anumit nivel de înţelegere. Răspunsurile eronate trebuie corectate imediat, prin discuţii individuale. Cadrul didactic trebuie să dirijeze conversaţia astfel încât ideile să fie bine conturate înainte de a trece la altele, în timp ce lecţia îşi menţine caracterul unitar. În ceea ce priveşte Informatica, recomandăm şi utilizarea unor instrumente ajutătoare ca de exemplu introducerea/exprimarea noţiunilor printr-un limbaj „de programare” (scris/oral), care să implice utilizarea eficientă a simbolurilor (în afară de latura didactică propriu-zisă), ceea ce înseamnă separarea clară a sintaxei de semantică.

1.3. Problematizarea şi învățarea prin descoperire

Predarea şi învățarea prin problematizare şi descoperire presupun utilizarea unor tehnici care să producă elevului conştientizarea „conflictului” dintre informația dobândită şi o nouă informație, determinându-l pe elev să acționeze în direcția lichidării acestuia prin descoperirea unor (noi) proprietăți ale fenomenului studiat. Pedagogic vorbind, conflictele se mai numesc şi *situații problemă* (**problematizare**), putând fi de cel puțin doua tipuri:

- *Contradicții* între posibilitățile existente ale elevului (nivelul intelectual şi de pregătire) şi cerințele, situațiile în care este pus de noua problemă. Aceste conflicte se datorează *imposibilității* elevului de a selecta dintre cunoştințele sale anterioare pe cele potrivite cu valoare operațională de aplicabilitate a viitorului.
- *Incapacitatea elevului* de a integra noțiunile selectate într-un sistem, în acelaşi timp cu conştientizarea faptului că sistemul este pe moment ineficient operațional (lucru care poate fi remediat doar prin completarea informației de bază).

Întrebările frontale sau individuale utilizate în etapa de pregătire a introducerii unei noțiuni, a prezentării unui domeniu nou, întrebări care se adresează capacității de reacționare a individului pot genera noi situații conflictuale de tipul menționat anterior. Pe cât posibil, cadrul didactic trebuie să gestioneze el însuşi apariția situațiilor problemă. La modul ideal, ele trebuie să apară *de la sine* în mintea elevului. Relativ la condițiile pedagogice ale acestor situații conflictuale generate de anumite probleme practice putem spune că problemele trebuie să aibă un sens precis şi să fie enunțate într-un moment „optim" al lecției. Ele trebuie să înglobeze cunoştințe anterior însuşite de elev, să le trezească interesul, să le solicite un anumit efort mental creator. Există părerea că rezolvarea problemei poate fi privită ca un proces prin care elevul descoperă că o combinație de reguli învățate anterior se poate aplica pentru găsirea soluției unei noi situații conflictuale. În acest sens se pot evidenția următoarele **etape** în rezolvarea problemei:

- Prezentarea problemei (verbal, scris, grafic etc.).
- Definirea problemei de către elev în sensul distingerii caracteristicilor esențiale ale situației, însuşirii enunțului, găsirii legăturii între date, informații etc.
- Formularea de către elev a anumitor criterii, ipoteze care pot fi aplicate în vederea găsirii unei soluții.
- Verificarea succesivă a unor asemenea ipoteze, eventual şi a altora noi şi găsirea efectivă a unei soluții (sau a tuturor).

Desigur că în contextul de mai sus expresiile *situație conflictuală*, *problemă*, *rezolvare de problemă* se referă la probleme şi soluții noi, necunoscute încă de elev şi nu la ceva de tipul substituirii de valori numerice în expresii date, execuția unui program dat pentru nişte valori de intrare etc. Utilizarea în predare a acestei metode este totdeauna utilă în momentul în care se şi găseşte rezolvarea conflictului.

Descoperirea apare ca o întregire a problematizării. Se pot pune astfel în evidenţă trei modalităţi principale de învăţare prin problematizare şi descoperire (clasificarea făcându-se după tipul de raţionament folosit):

- Modalitatea *inductivă*.
- Modalitatea *deductivă*.
- Modalitatea *prin analogie*.

În primul caz este vorba de generalizări. Elevul trebuie încurajat să îşi dezvolte propria cale de învăţare, care să nu contrazică lucrurile în care deja „crede”, prin folosirea unor mijloace tehnice şi resurse informaţionale personale. În al doilea caz se foloseşte logica, sau mai exact sistemele deductive (ca metodă de **raţionament**). Putem deriva (obţine) cunoştinţe noi din cunoştinţe vechi (cu ajutorul unor **reguli de inferenţă** specifice). În ultimul caz, se încurajează folosirea unei experienţe anterioare nu numai dintr-un domeniu conex, ci chiar din domenii total diferite.

Problematizarea are astfel interferenţe cu conversaţia, întrebările individuale sau frontale care se adresează gândirii, raţionamentului născând situaţii conflictuale. Generarea *situaţiilor problemă* trebuie produsă astfel încât întrebările să apară în mintea elevului fără ca acestea să fie puse de către profesor. După cum am mai precizat, ca disciplină cu caracter formativ, Informatica îşi propune formarea unei gândiri algoritmice, sistematice şi riguroase, care să promoveze creativitatea, să stimuleze imaginaţia şi să combată rutina. Chiar dacă aparent travaliul informatic se sprijină pe anumite şabloane, acestea reprezintă numai tendinţe utile de standardizare. Procesele care izvorăsc din situaţii reale, care implică calculatorul în rezolvarea unor probleme aparţinând diferitelor sfere ale vieţii de zi cu zi, analiza acestor probleme, alegerea structurilor de date pe care se mulează informaţia oferită de mediul înconjurător, paşii algoritmilor şi programarea în sine, implică folosirea metodei problematizării, iar aplicarea acestei metode necesită formarea unor deprinderi ce nu se obţin decât printr-un exerciţiu îndelungat. Rezolvarea de probleme, ceva curent în învăţarea Informaticii, poate fi privită ca un proces prin care elevul descoperă că o altă combinaţie de reguli învăţate anterior conduc la rezolvarea unei noi situaţii problematice. Formularea de probleme de către elevii înşişi constituie forme ale creativităţii şi presupune că elevii şi-au format deprinderi intelectuale eficace din punct de vedere al generalizării şi aplicabilităţii (*orice soluţie generează o nouă problemă*). Problemele propuse pot fi inspirate din viaţa cotidiană, din cunoştinţele dobândite prin studiul altor discipline, din generalizarea unor probleme de informatică rezolvate anterior, probleme de perspicacitate, jocuri etc. Problematizarea şi descoperirea fac parte dintre metodele *formativ-participative*, care solicită gândirea creatoare a elevului, îi pun la încercare voinţa, îi dezvoltă imaginaţia, îi îmbogăţeşte experienţa. În lecţiile în care se aplică aceste metode profesorul alege problemele, le formulează, dirijează învăţarea şi controlează munca depusă de elev în toate etapele activităţii sale. Această metodă este caracteristică, de exemplu, unor lecţii de aplicaţii practice de laborator, metoda învăţării prin descoperire fiind frecvent aplicată în momentul în care este necesară folosirea programelor utilitare, a soft-urilor de aplicaţie etc. Utilitarele se abordează în funcţie de problemele concrete care urmează a fi rezolvate. Obiectivul imediat

este cunoaşterea şi exploatarea produsului şi nu îmbunătăţirea lui. Concentrarea atenţiei va fi dirijată spre rezolvarea problemei şi nu asupra analizei facilităţilor şi lipsurilor produsului software. Cu siguranţă, în acest caz este deosebit de importantă experienţa dobândită, cunoştinţele şi deprinderile formate în alte situaţii similare de învăţare: lucrul cu meniuri, funcţii comune mai multor utilitare, cunoaşterea structurilor de date, dexteritatea în tehnoredactare etc. Cunoaşterea facilităţilor produsului soft se face în momentul ivirii necesităţii exploatării acestuia şi nu printr-o prezentare a lui ca o înşiruire mai mult sau mai puţin sistematică şi completă de funcţii sau facilităţi. Bineînţeles că este obligatorie o prezentare generală a utilitarului. În contextul altor produse similare, trebuie concepută o viziune de ansamblu din care să se desprindă caracteristicile dominante ale utilitarelor din clasa de respectivă şi să se prezinte particularităţile specifice produsului, cu îmbunătăţiri faţă de versiunile anterioare şi perspective de dezvoltare pentru cele viitoare.

Ca informaticieni, ne interezează (în acest context) şi ceea ce numim **rezolvarea problemelor (*problem solving*)**. Îndemânările achiziționate în legătură cu acest subiect depind în primul rând de cunoştințele specifice acumulate, dar din punct de vedere al psihologiei există acordul că se pot căpăta şi „îndemânări generale”. Procesul cognitiv în ansamblu este foarte complicat, numai pentru explicarea coerentă a acestuia fiind necesară o întreagă carte. Vom sublinia doar câteva elemente cheie şi direcții principale pentru abordarea rezolvării unor probleme. Astfel, când dorim să rezolvăm o problemă cu ajutorul calculatorului, presupunând că enunțul este „acceptat”, trebuie să ne întrebăm în primul rând:

Ce ştim în legătură cu domeniul implicat.
Cum sunt apreciate „pe piață” rezultatele.
Care strategii generale sunt aplicabile.
Care sunt motivațiile suplimentare.

După ce problema a fost enunțată şi sunt furnizate anumite indicații suplimentare, putem trece la alegerea **strategiei concrete de rezolvare**. Aceasta trebuie să fie selectată după un anumit **plan**, să permită un anumit tip de **verificare** şi **generalizare**. De asemenea, trebuie avute în vedere metode sau metodologii prin care să se **interzică** anumite „ramuri” şi să se permită **explorarea** de direcții colaterale. Una dintre strategiile generale poate fi următoarea:

- Pot să rezolv problema (am cunoştințele necesare).
- definesc în mod (semi)formal.
- Caut informațiile suplimentare astfel încât să am o definiție formală concretă (eventual, chiar într-un limbaj de programare concret).

- Fac planul de implementare.
- Îl execut (scriu „programele” şi le „rulez”).
- Verific faptul că ceea ce am făcut este „corect”.
- Generalizez (la alte cazuri, la alte probleme).

Peste tot, cunoaşterea măcar a unei părți din logica formală este indispensabilă.

1.4. Modelarea

Modelarea ca metodă pedagogică poate fi descrisă ca fiind un mod de lucru prin care gândirea elevului este condusă la descoperirea adevărului, folosind un aşa numit *model* şi utilizându-se raţionamentul prin analogie. Modelul şi metoda în sine nu presupun o asemănare perfectă cu cazurile reale iniţial specificate, ci numai o analogie *rezonabilă*. Ea constă în construirea unui sistem *S1* a cărui descriere coincide cu descrierea sistemului original *S* până la un anumit punct. *S1* poate avea o natură diferită şi este în general mai simplificat şi formalizat. Ideea este că investigând sistemul *S1* prin metode specifice legate de o anumită temă de lecţie se pot găsi noi soluţii, care apoi pot fi translatate în concluzii asupra evoluţiei sistemului de bază *S*. Modelarea are o mare valoare euristică colaterală, prin utilizarea ei putându-se dezvolta spiritul de observaţie, capacitatea de analiză şi sinteză, creativitatea. Ideea ar fi să putem determina elevii să descopere singuri modelul. Astfel elevul se obişnuieşte să creeze noi probleme ce trebuiesc rezolvate, să adapteze algoritmi cunoscuţi la situaţii noi etc. Realitatea înconjurătoare este percepută şi înţeleasă pe baza unor modele deja cunoscute. Dezvoltarea deprinderilor de modelare, obişnuirea elevilor cu gândirea logică, se realizează prin prezentarea exactă şi clară a modelelor şi prin transparenţa particularizărilor. Un exemplu edificator îl reprezintă *învăţarea metodelor de elaborare a algoritmilor*. Necesitatea unor formalizări se impune prin rigoarea modului de abordare a problemei, prin sistematizarea organizării informaţiei de intrare, a exactităţii proiectării prelucrării şi prin standardizarea ieşirii. Formalizarea necesită cunoştinţe dobândite în studiul altor discipline, fundamentate teoretic, iar accesibilitatea formalizării este condiţionată de factori specifici nivelului de cunoştinţe dobândite anterior, de categoria de vârstă, de capacitatea de asimilare (a nivelul clasei, de exemplu). Abordarea ponderată a acestor aspecte conduce la dezvoltarea deprinderilor de abstractizare, a gândirii algoritmice şi sistemice. Utilizarea modelelor în realizarea algoritmilor presupune stabilirea unor analogii şi în organizarea datelor de intrare. Învăţarea algoritmilor este legată de cunoaşterea modului de organizare a datelor, de cunoaşterea profundă a structurilor de date posibile a fi prelucrate uşor de către calculator. Etapa cea mai importantă este cea a *descoperirii* algoritmului, urmată de stabilirea *modului de organizare a datelor*, dar importanţa acestui ultim aspect este esenţială în *determinarea performanţelor* produsului program care implementează algoritmul. Modelarea (ca metodă pedagogică) este definită ca un mod de lucru prin care gândirea elevului este condusă la descoperirea adevărului cu ajutorul modelului, graţie raţionamentului prin analogie. *Modelarea similară,* constă în realizarea unui sistem de aceeaşi natură cu originalul, care să permită evidenţierea trăsăturilor esenţiale ale originalului. O gamă variată de probleme sunt rezolvate prin metoda *backtracking*. Pentru implementarea într-un limbaj de programare a unui algoritm elaborat prin *backtracking*, elevul are nevoie de un model reprezentat de un program, cum ar fi cel

de *generare a permutărilor* sau de rezolvare a *problemei celor opt dame* şi prin mici modificări, el poate obține multe alte programe care implementează algoritmi ce rezolvă probleme clasice, cum ar fi: generarea aranjamentelor, combinărilor, problema parantezelor, partițiile unui mulțimi, problema celor opt turnuri etc. Similar se procedează în rezolvarea problemelor care necesită utilizarea stivelor sau a cozilor, folosind operațiile elementare cu elementele acestor structuri dinamice elementare. Pentru detalii se pot consulta **Anexa 1** şi **Capitolul 6**. *Modelarea analogică* nu presupune o asemănare perfectă cu originalul, ci numai folosirea unei analogii. Momentele cunoaşterii în procesul modelării sunt:

- Trecerea de la original la model.
- Transformarea modelului sau experimentarea pe model.
- Transferul pe original a rezultatelor obținute pe model.
- Verificarea experimentală pe original a proprietăților obținute pe model.

Trecerea de la original la model se face prin *simplificare*. Se impune ca simplificarea să nu fie exagerată pentru a nu se omite trăsăturile esențiale ale originalului. Totodată, trebuie să nu se scape din vedere că valoarea modelului va fi apreciată prin prisma eficacității lui, adică a posibilităților pe care le oferă pentru atingerea scopului şi că noile informații obținute pe baza modelului vor fi transferate cu grijă asupra originalului, având în vedere diferența dintre model şi original. Modelul devine astfel purtătorul unei semnificații, informații, care poate fi exprimată printr-un suport material sau ideal. O clasificare a modelelor după natura suportului sub care se vehiculează informația, poate fi:

- *Modele materiale*, care au suport concret şi care se folosesc foarte puțin în învățarea Informaticii: folosirea unui table de şah în rezolvarea problemei celor opt dame determină o rapidă înțelegere a mecanismului metodei backtracking; utilizarea unei stive de monezi de dimensiuni diferite pentru înțelegerea rezolvării problemei turnurilor din Hanoi. Nu trebuie exclusă posibilitatea învățării direct pe obiectul de studiu, caz întâlnit (şi recomandat) în studiul structurii şi arhitecturii sistemelor de calcul, unde prezentarea părților componente ale unui sistem de calcul şi a conexiunilor dintre acestea, în contextul funcționalității ca un ansamblu (sistem), este esențială.
- *Modele ideale (virtuale)*, care se exprimă prin imagini sau sisteme de simboluri sau semne convenționale.

Învățarea Informaticii prin modelare presupune *două etape*. Într-o primă etapă, învățarea se va face pe baza modelelor construite „de profesori", etapă în care se vor analiza trăsăturile modelului şi compararea lui cu originalul. Pentru a reliefa condițiile pe care trebuie să le îndeplinească modelul se vor da şi contraexemple. În a doua etapă, elevii vor fi deprinşi să construiască singuri modele. Importanța descoperirii modelului de către elev constă în faptul că elevul este obişnuit a reprezenta într-o formă standard condițiile impuse de problemă şi adânceşte convingerea sa că Informatica este un domeniu în care rezultatele pozitive se obțin doar printr-o înlănțuire logică de raționamente. Folosirea modelelor nu înseamnă impunerea unor metode care trebuie reținute şi aplicate orbeşte. Se va pune accentul pe înțelegerea paşilor unui algoritm şi se va încuraja prezentarea oricăror metode care exclud modelul şi care se impun prin eleganță şi eficiență. Elevii vor fi încurajați să-şi dezvolte şi să-şi prezinte ideile proprii, contribuind în acest fel la creşterea încrederii în posibilitățile lor, în valoarea ideilor lor. Ei nu trebuie să fie obligați să reproducă ideile altora, să aştepte ca totul să fie prezentat de profesor, să asimileze rețete, ci să descopere metode noi, să le prezinte, analizeze şi perfecționeze printr-o comunicare continuă şi constructivă. Folosirea modelelor în învățare deschide pentru Informatică o impresionantă arie de aplicabilitate (inclusiv utilizarea Informaticii în predarea altor discipline, de la artele plastice la cele mai diverse domenii ale tehnicii).

1.5. Exemplificarea sau demonstrarea materialului intuitiv

Prin *exemplificare* sau *demonstraţie*, în acest caz, înţelegem prezentarea sistematizată şi organizată a unor obiecte, procese, experimente, cu scopul de a uşura înţelegerea intuitivă şi executarea corectă a unor activităţi programate. Cuvântul *intuiţie* din titlu, înseamnă utilizarea oricărui raţionament inductiv, în contextul temei şi bagajului de cunoştinţe ale elevului. Utilizarea intuiţiei împreună cu exemplificarea necesară poate implica folosirea a diverse modalităţi şi tehnici didactice datorită diversităţii materialului de studiu. Exemplificarea sau demonstrarea materialului intuitiv presupune utilizarea obiectelor reale, ca: utilizarea materialului grafic (planşe, scheme); utilizarea retroproiectorului/videoproiectorului şi a materialului pretipărit; utilizarea calculatorului (imagini grafice, multimedia, power point). În acest context putem spune că: *Prin demonstrarea materialului intuitiv se înţelege prezentarea sistematică şi organizată a unor obiecte, procese, etc. sau producerea unor experienţe, fenomene în faţa elevilor, cu scopul de a uşura înţelegerea şi executarea corectă a unor activităţi* <38>. Un rol deosebit îl joacă astfel *intuiţia* (intuiţia este o experienţă mentală; intuiţia înseamnă o simplă observare şi notare a unor fapte; intuiţia poate fi asimilată cu un raţionament de tip inductiv <38>). Intuiţia realizează corelaţia dintre imagine şi cuvânt, fiind atât sursă de cunoştinţe cât şi mijloc de verificare. Informatica nu poate fi desprinsă decât artificial de bazele ei intuitive şi de extinderea ei în realitatea cotidiană. Convertirea principiului intuiţiei în metoda demonstraţiei se realizează în funcţie de materialul intuitiv: machete, grafică, film didactic, televiziune şcolară, software-uri de învăţare. Materialul intuitiv este frecvent folosit în numeroase lecţii cum ar fi de exemplu (se pot consulta şi **Anexa 1**, **Capitolul 6**):

- Învăţarea *algoritmilor de sortare*, unde prin diferite moduri de reprezentare sunt urmărite grafic valorile care se compară şi se schimbă între ele, conducând la ordonarea şirului.
- Învăţarea metodei *backtracking*, unde folosind materialul natural se urmăreşte *formarea* soluţiei prin avansări şi întoarceri repetate.
- Vizualizarea ocupării şi eliberării zonelor de memorie prin *alocarea dinamică* a variabilelor.
- Ilustrarea *modului de lucru cu elementele listelor simplu şi dublu înlănţuite, a stivelor şi a cozilor*.
- Echilibrarea *arborilor binari* (arbori AVL).

Ţinând cont de eficienţa transmiterii informaţiei prin mijloacele vizuale (inclusiv **INTERNET**) şi de orientarea cu predilecţie spre mijloacele de informare rapidă care solicită atât memoria vizuală cât şi cea auditivă şi formarea involuntară a unui public consumator de informaţie audio-video, o orientare a metodelor şi procedeelor didactice în vederea exploatării acestei stări de lucruri creează un avantaj aparte procesului instructiv-educativ. Crearea unor filme (casete video) didactice care să urmărească cu exactitate programa şcolară creează facilităţi de predare multor discipline şi ar permite elevului să poată revizualiza predarea lecţiei. Aceasta ar putea elimina ambiguităţile sau golurile create de momentele de neatenţie din timpul predării şi ar constitui un veritabil *profesor la purtător* al elevului. Este evident că acest mijloc didactic nu poate înlocui (nici măcar suplini) exerciţiul individual şi nici prezenţa efectivă a cadrului didactic. Efortul profesorului este însă cu totul special. Nu este suficient ca un elev să vadă un material, el *trebuie învăţat să vadă*. Poate că în acest moment ar trebui sa aducem în discuţie **euristicile** şi **încurajarea creativităţii**. Conform <(?????Wankat...)>, se pot pune în evidenţă chiar euristici pentru dezvoltarea creativităţii:

- Încercaţi să aveţi cât mai multe idei. Cu cât mai multe, cu atât este posibil să puteţi selecta câteva „bune”.
- „Inversaţi” (reformulaţi, reiteraţi, puneţi-o într-un alt context etc.) problema.
- „Ghiciţi” o soluţie la întâmplare (chiar urmărind un Dicţionar...).
- Gândiţi-vă la ceva distractiv, apropo de utilzarile posibile ale rezolvării.
- Gândiţi-vă la probleme similare şi la soluţiile acestora, chiar în contexte diferite.
- Concepeţi o lista generală „explicativă” de cuvinte cheie, proprietăţi utile, stimulente ş.a.m.d., care au cât de cât legatură cu tema în cauză.

1.6. Metoda exercițiului

La modul cel mai general, exerciţiile pot fi privite ca acţiuni concrete efectuate în mod conştient şi repetat în scopul dobândirii unor priceperi şi deprinderi (mai rar cunoştinţe) noi pentru a uşura anumite activităţi şi a contribui la dezvoltarea unor aptitudini. Avantajele metodei exerciţiului sunt**:**

- Se poate forma o gândire productivă, creatoare, cu implicație financiară.
- Se oferă posibilitatea câştigării unei anumite independențe.
- Se oferă posibilitatea inițierii unui dialog-conversație cu obiective precise asupra unor metode şi soluții.
- Se activează atitudinea critică şi poate creşte discernământul elevilor în privința celor mai bune metode de lucru.
- Se oferă o anumită posibilitate profesorului pentru a analiza şi evalua activitatea sau performanțele generale ale unui elev.

Condiția primordială de reuşită este dată în principal de selecția corespunzătoare a problemelor sau exercițiilor precum şi de activitatea de îndrumare-proiectare. Prin urmare, exercițiile sunt acțiuni efectuate în mod conştient şi repetat de către elev cu scopul dobândirii unor priceperi şi deprinderi şi chiar cunoştințe noi, pentru a uşura alte activități şi a contribui la dezvoltarea altor aptitudini. Însuşirea cunoştințelor de Informatică este organic legată de exersarea utilizării unor soft-uri de aplicație, de rezolvarea unor probleme de programare etc. Nu există lecție în care să nu se aplice această metodă. Alte avantaje ale acestei metode sunt concretizate în rezultatele aplicării ei: formează o gândire productivă; oferă posibilitatea muncii independente; oferă posibilitatea analizei diverselor metode şi soluții de rezolvare a problemelor; activează simțul critic şi autocritic şi îi învață pe elevi să-şi aprecieze rezultatele şi metodele de lucru; oferă posibilitatea depistării şi eliminării erorilor.

Este clar că metoda nu contribuie numai la formarea priceperilor şi deprinderilor de lucru cu calculatorul, ci contribuie substanțial la dezvoltarea unui raționament flexibil şi operant. Pentru profesor alegerea, formularea şi rezolvarea problemelor şi apoi exploatarea rezultatelor obținute constituie o sarcină de importanță deosebită. *Alegerea* problemelor este condiționată de programa analitică, succesiunea prezentării noțiunilor în manuale, metodele de rezolvare ce pot fi folosite şi de elevii cărora li se adresează. *Formularea problemelor* trebuie să țină cont de noțiunile cunoscute de elevi, să fie clară, concisă (neambiguă) şi să folosească limbajul de specialitate numai în măsura în care este cunoscut elevilor. *Rezolvarea* trebuie să aibă în vedere obținerea rezultatelor pe căi clare şi uşor de verificat, reținerea tipurilor de raționamente folosite, deschiderea perspectivei pentru rezolvarea unor probleme analoage sau mai complexe. *Folosirea rezultatelor* obținute trebuie să vizeze lămurirea conținutului activ în cunoaşterea noțiunilor învățate şi adâncirea semnificației lor, asimilarea metodelor de rezolvare şi aplicarea lor la rezolvarea altor probleme. Utilizarea pe scară largă a acestei metode a condus la o clasificare a exercițiilor şi problemelor, clasificare ce are la bază aportul capacităților intelectuale necesare rezolvării lor. În subsecțiunile care urmează insistăm asupra unor particularizări.

1.6.1. Exerciţii şi probleme de *recunoaştere* a unor noţiuni, formule, metode

De exemplu**,** elevilor li se prezintă *metoda backtracking* (**Capitolul 1**). Utilizând-o, se pot descrie algoritmii care generează permutările, aranjamentele, combinările, apoi li se poate cere să genereze toate funcţiile injective, surjective, bijective definite pe o mulţime cu *m* elemente, cu valori într-o mulţime cu *n* elemente.

1.6.2. Exerciţii şi probleme *aplicative* ale unor formule sau algoritmi cunoscuţi.

Cunoscând modul de lucru cu elementele structurilor de date de tip stivă sau coadă, elevilor li se poate propune să rezolve problema parcurgerii „în lăţime" sau „în adâncime" a unui graf oarecare (<16>, <21>, <25>). Exerciţiile aplicative trebuiesc utilizate atât timp cât ele trezesc interesul. Repetarea lor nejustificată poate conduce la efecte contrarii. Contraexemplele însoţite de o analiză amănunţită vin să sublinieze trăsăturile esenţiale. În acelaşi timp, *analiza erorilor* (<17>) este utilă prin faptul că dezvăluie anumite lacune în cunoştinţele elevilor.

1.6.3 Probleme care permit *însuşirea* unor noţiuni

Specifice Informaticii sunt problemele al căror grad de dificultate creşte treptat, o dată cu formarea şi asimilarea noţiunii, fiecare nouă problemă aducând un plus de dificultate. În rezolvarea unei probleme de programare este necesar să se ţină seama de următoarele etape:

- *Analiza* iniţială a problemei prin care se stabileşte formatul şi natura datelor de intrare, intervalele de variaţie a datelor de intrare, a variabilelor de lucru (date intermediare) precum şi formatul şi intervalele de variaţie a datelor de ieşire. Tot în această etapă se va stabili un algoritm (plan) de rezolvare, exprimat, eventual, în limbaj natural, pe baza căruia se va permite fiecărui elev să lucreze independent.
- *Rezolvarea* propriu-zisă a problemei este etapa în care se realizează transpunerea într-un limbaj de programare a algoritmului stabilit în prima etapă. În prealabil, algoritmul este reprezentat în una dintre formele cunoscute, se stabilesc variabilele de lucru, forma lor de alocare, prelucrările ce vor avea loc, apoi se trece la implementarea în limbajul dorit. Dacă rezolvarea se poate face pe mai multe căi, trebuie să se sublinieze, dacă este posibil, calea optimă.

- *Verificarea* soluției sau soluțiilor obținute va permite elevului să-şi dea seama dacă soluția obținută este cea corectă. În această etapă intervine profesorul cu seturi de date de test care să cuprindă dacă este posibil, majoritatea (dacă nu toate) cazurilor existente ridicate de problemă şi în special cazurile critice, *la limită*, ale datelor de intrare.

Aceste etape cuprind în esență: însuşirea enunțului; discutarea problemei şi stabilirea algoritmului de rezolvare; rezolvarea propriu-zisă; verificarea soluțiilor. Ele se pot modifica după natura problemelor. Acolo unde problema permite mai multe căi de rezolvare, profesorul analizează toate aceste căi şi selectează pe cele mai importante propunându-le spre rezolvare pe grupe, comparând rezultatele, avantajele şi dezavantajele fiecărei metode în parte. Se va evidenția în mod obligatoriu *cea mai bună* soluție.

Exemplu. Se cere elevilor determinarea arborelui parțial de cost minim asociat unui graf, prin algoritmul lui Kruskal.

În prima etapă:

Se analizează enunțul.

Se verifică dacă elevii cunosc noțiunea de arbore şi de arbore parțial de cost minim.

Pe tablă se desenează un graf oarecare, se numerotează nodurile şi se stabilesc costurile muchiilor.

Se stabilesc datele de intrare, formatul acestora, tipul lor (deja se gândeşte în direcția implementării într-un limbaj de programare); în acest caz datele de intrare se vor citi dintr-un fişier text cu înregistrări de forma:

n - numărul de noduri ale grafului;

$I, j, a_{i,j}$ - muchia de la nodul i la nodul j are costul $a_{i,j.}$

Fişierul va conține un număr de înregistrări de forma celor de mai sus, egal cu numărul de muchii al grafului.

Aici poate interveni profesorul, solicitând elevilor sau prezentând o formă mai condensată a fişierului de *intrare*, cu înregistrări de forma:

n – numărul de noduri ale grafului;

$1\ i_1\ a_{1,i1}\ i_2\ a_{1,i2}...\ i_k\ a_{1,ik}$ - nodul 1 are vecinii $i_1, i_2, ..., i_k$, cu $i_k>1$, iacosturile acestor muchii sunt $a_{1,i1}, a_{1,i2}, ...,a_{1,ik}$

$2\ j_1\ a_{2,j1}\ j_2\ a_{2,j2}...\ j_t\ a_{2,jt}$ - nodul 2 are vecinii $j_1, j_2, ..., j_t$, cu $j_k>2$, iarcosturile acestor muchii sunt $a_{2,j1}, a_{2,j2}, ...,a_{2,jt}$

.....

$n-1\ l_1\ a_{n-1,l1}$ - nodul $n-1$ poate avea cel mult un vecin mai mare decât el, nodul n.

Dacă un nod nu are vecini mai mari decât el, linia din fişierul de intrare corespunzătoare acelui nod va lipsi.

Vom construi fişierul de intrare pentru graful desenat pe tablă, pe care îl vom folosi ca prim fişier de test.

În a doua etapă:

Se va stabili modul de memorare al datelor de intrare. Elevii vor fi tentați să reprezinte graful printr-o matrice de adiacență (simetrică), iar într-o altă matrice tot simetrică costurile muchiilor, sau în cel mai fericit caz, printr-o singură matrice, atât costurile muchiilor cât şi graful. Aici trebuie să intervină profesorul. El va sublinia risipa de memorie realizată prin acest tip de memorare şi va propune sau va încerca să obțină de la elevi o memorare mai eficientă (printr-un vector) a grafului şi costurilor muchiilor. Se va defini un tip de dată numit muchie:

```
muchie : record
 i,j:byte;
  a:byte;
end;
```

cu semnificația că *i, j* sunt vârfurile muchiei, iar *a* costul ei şi se va aloca un vector de *muchii*, a cărui dimensiune maximă se va stabili împreună cu elevii. Profesorul va prezenta *Algoritmul lui Kruskal*. Apoi se consideră inițial arborele parțial vid. Se va selecta muchia de cost minim neselectată anterior şi care nu formează un circuit cu muchiile deja selectate; procedeul se opreşte după selectarea a *n-1* muchii (<16>, <21>, <30>). Se insistă asupra criteriului de oprire, profesorul având două posibilități: să prezinte el criteriul şi să verifice cu clasa de ce acesta este cel corect, sau să încerce să obțină de la clasă un criteriu de oprire. Următoarea problemă care trebuie abordată este cea a alegerii muchiei. Evident că prin ordonarea crescătoare a vectorului de muchii acestea vor putea fi selectate în ordinea crescătoare a costurilor lor, dar se pune problema eliminării muchiilor care formează circuite. Aici se va obține de la clasă o soluție, ținând cont că determinarea componentelor conexe ale unui graf a fost deja rezolvată. Se va stabili algoritmul de ordonare a vectorului de muchii şi modul de memorare a nodurilor selectate pe parcursul determinării arborelui parțial de cost minim. Tot în această etapă, se va determina o soluție în cazul numeric prezentat în figura de pe tablă.

În a treia etapă, elevilor li se va propune implementarea algoritmului fie cu memorarea datelor de intrare în matrice, fie în vectorul de muchii, pe grupe de lucru. Li se poate cere chiar folosirea de algoritmi de sortare diferiți, aceasta pentru a constata faptul că soluția nu este unică şi în plus li se va cere să determine cauza obținerii de soluții diferite, dar optime. Profesorul va supraveghea implementarea solicitând elevilor verificarea etapă cu etapă a realizării programului, prin afişarea temporară chiar a unor rezultate intermediare. În ultima etapă, elevii vor verifica corectitudinea programului prin folosirea de date de test construite de ei şi prin noi teste propuse de către profesor, dar aceste teste vor fi prezentate sub formă grafică, prin desen pe tablă etc., pentru ca ei să construiască singuri fişierul de intrare. În final se va propune elevilor spre rezolvare probleme care să utilizeze rezultatul obținut sau să folosească tehnici asemănătoare, evident fără a specifica elevilor acest lucru.

Problemă. O localitate având n puncte *vitale*, legate prin străzi a căror lungime se cunoaşte, este complet înzăpezită. Primăria, care nu dispune de rezerve suficiente de combustibili, este obligată să deszăpezească un număr de străzi, astfel încât toate punctele vitale ale localității să fie accesibile din fiecare punct şi să realizeze un consum minim de carburant. Să se determine străzile care trebuiesc deszăpezite ştiind că orice consum de carburant este direct proporțional cu lungimea drumului deszăpezit. Exemplul prezentat subliniază importanța şi consecințele pe care le are asupra modului de rezolvare a unei probleme, modul de organizare a datelor de intrare şi a celor intermediare, de lucru.

O posibilă clasificare a problemelor/exercițiilor (relativ la capacitățile intelectuale pentru rezolvare) ar fi:

Exerciții de recunoaştere a unor noțiuni (unitate curentă de I/E, unitate de disc, memorie internă, comandă externă, programe executabile de tip com. sau exe, HTTP-uri, telnet, etc.).

Exerciții aplicative (programe pentru transcrierea unor formule, pseudocoduri).

Aceste două clase de exerciții sunt recomandate în special pentru fixarea unor cunoştințe deja predate. În acest context poate fi utilă o complicare graduală a enunțului inițial, urmărindu-se memorarea mai bună a formulei sau a ideii algoritmului, cum ar fi: încadrarea acestuia într-un eventual alt tip de probleme cunoscute; complicarea lui în mod progresiv în vederea utilizării sale în alte situații; prezentarea unor cazuri *limită*, care pot conduce la rezultatele eronate.

Exerciții grafice – planşe, vizualizări.

Exerciţii complexe - acestea presupun o analiză mult mai detaliată a problemei în ansamblu şi implică descompunerea problemei în subprobleme, succesiv, până în momentul în care rezolvarea subproblemelor elementare este cunoscută.

În rezolvarea exercițiilor este importantă crearea posibilității îndeplinirii unei independențe (individual, grup, echipă). Pentru formarea unor priceperi sau abilități legate de munca indepdendentă se poate utiliza şi aşa numita formulă a *exercițiilor comentate*. Aceasta constă în rezolvarea exercițiilor de către toți elevii, în timp ce un elev desemnat explică permanent rezultatele obținute. Nu este nevoie ca această explicație să fie utilizată pe calculator. Profesorul poate în orice moment să invite oricare alt elev pentru continuarea explicației (în acest fel, această metodă este deosebit de activă). Discuțiile suplimentare sunt obligatorii în acest caz. Se vor evidenția permanent avantajele şi dezavantajele rezolvărilor propuse, alte metode posibile de rezolvare, idei privind utilizarea acestor rezolvări în lecțiile următoare, particularizări ale lor în lecțiile anterioare.

1.7. Metoda învățării în grupe mici

Activitatea de învățare pe grupe mici se defineşte ca o metodă în care sarcinile sunt executate de grupuri de elevi, grupuri care sunt câteodată autoconstituite şi care se autodirijează. Activitatea în Informatică se desfăşoară în general în echipă, travaliul individual fiind o componentă a muncii corelate din cadrul unui grup de lucru. Tehnicile de organizare a muncii în unitățile de Informatică evidențiază ca o formă de organizare *echipa programatorului-şef*, echipă în care fiecare membru are sarcini bine stabilite (de analiză, programare, implementare, exploatare), sarcini corelate între ele. Este normal ca şi activitatea didactică să recurgă la metode de învățare colectivă, fără a neglija însă munca individuală, ci doar privind-o pe aceasta ca o componentă a muncii în echipă <15>, <39>. Profesorii recunosc, în general, eficacitatea unei astfel de organizări a activității didactice şi o integrează în arsenalul metodic al predării disciplinei. Criteriile de formare a grupelor sunt în funcție de obiectivele urmărite (însuşirea de noi cunoştințe, rezolvare de probleme, etc.): grupuri *omogene*, formate din elevi cu acelaşi nivel de cunoştințe; grupuri *eterogene*, formate din elevi de toate categoriile (foarte buni, buni şi slabi), dar în proporții apropiate; grupuri formate pe criterii *afective* (prietenie, vecini de bancă). Numărul elevilor dintr-un grup poate varia de la *2* la *10*, dar cele mai potrivite grupuri sunt cele formate din *4-6* elevi. La lecțiile de aplicații practice de laborator, grupurile de lucru formate din *4* elevi care dispun de două calculatoare, par a fi cele mai eficiente. Grupuri formate din mai mult de *2* elevi la un calculator se dovedesc a fi neproductive. Este bine ca la întocmirea grupurilor să se stabilească criterii clare de formare şi elevii să fie lăsați să se grupeze singuri, respectând criteriile cerute. Pentru grupurile omogene sarcinile pot fi diferite în funcție de scopul propus. Pentru grupurile eterogene sau create pe criterii afective, sarcinile vor fi aceleaşi la fiecare grup, dar profesorul va rezerva sarcini suplimentare elevilor mai buni din fiecare grup. Etapele pretinse de această metodă de învățare sunt: repartizarea materialului (problemelor) fiecărui grup; munca independentă a grupurilor sub supravegherea profesorului; discutarea în plen a rezultatelor obținute. Activitatea profesorului se concretizează în două etape. Prima este una *proiectivă* în care se pregăteşte materialul de repartizat pe grupe şi materialul în plus pentru elevii buni şi a doua, de *îndrumare/supraveghere* şi de animare a activității grupelor de lucru. Ajutorul acordat grupelor de lucru trebuie să fie dat numai la cerere şi în aşa fel încât profesorul să se situeze pe poziția de colaborator şi nu pe cea de autoritate care îşi impune părerile şi soluția personală. Profesorul va interveni cu autoritate numai în situația în care activitatea grupului se îndreaptă într-o direcție greşită. Când unul sau mai multe dintre grupuri găseşte o soluție, acestea vor fi discutate şi analizate succesiv sau în paralel. Scopul acestei discuții este de a reliefa corectitudinea rezolvării, determinarea celei mai eficiente şi mai elegante soluții şi de a descoperi eventualele erori. Importanța acestor dezbateri pentru dezvoltarea raționamentului este foarte mare, iar rolul profesorului este cel de a incita

şi coordona discuţiile în direcţia obţinerii concluziilor care se impun. Se impută, pe bună dreptate, acestei munci în grup o intensitate şi o productivitate scăzută. Diversificarea sarcinilor grupurilor şi împărţirea sarcinilor între membrii grupurilor atenuează această deficienţă. Dacă prin activitatea în grup se intenţionează dobândirea de noi cunoştinţe prin lucrul cu manualul, documentaţia sau prin testarea unor produse soft, pentru profesor este obligatoriu de a organiza dezbaterile finale care să stabilească dacă elevii şi-au însuşit corect noţiunile şi şi-au format deprinderi corecte. Este de asemenea greşit a se lucra mereu cu grupuri constituite după aceleaşi criterii, pentru că fie că sunt suprasolicitaţi elevii buni din grupurile eterogene, iar elevii slabi se bazează exclusiv pe aportul liderilor de grup, fie că, în grupurile omogene, elevii slabi se complac în postura în care se află şi nu mai încearcă să scape de acest calificativ. Alte câteva probleme pot fi abordate sub un unghi diferit în acest context. Astfel, se pot pune **întrebări** mult mai individualizate (acestea nu ţin neapărat de conţinutul în sine al lecţiei!). Ce întrebări se pun şi modul în care se pun, poate fi mai important decât întrebarea în sine. Apoi, este mai simplă „contactarea" elevilor în timpul lecţiei şi chiar după ea. Susţinem, ca prioritate şi soluţie la anumite probleme locale de învăţământ, aducerea unor **specialişti** care lucrează în lumea reală pentru a preda lecţii de sinteză, lecţii speciale etc.

1.8. Metoda lucrului cu manualul şi documentaţia

Manualele şcolare, purtătoare ale valenţelor formative prin deosebitul lor conţinut metodic şi didactic, reprezintă o limită impusă de programa şcolară din punct de vedere al conţinutului informativ. În Informatică, mai mult decât în alte domenii, manualul este supus perisabilităţii conţinuturilor prin frecvenţa cu care disciplina este receptivă la noutăţile domeniului. Realitatea didactică reliefează faptul că elevul foloseşte pentru învăţarea teoriei doar notiţele întocmite în clasă la predare şi din considerente de comoditate sau de obişnuinţă, foarte puţin (sau deloc) manualele. Acestea sunt consultate în cel mai fericit caz doar pentru citirea enunţurilor problemelor. Atitudinea de reţinere sau de respingere faţă de manual are consecinţe negative atât asupra caracterului formativ, cât şi asupra celui informativ al învăţării. Capacitatea de raţionament al unui copil nu se formează numai după modele de raţionament oferite de profesor, ci şi prin eforturi proprii, prin activitatea proprie de căutare şi comparare cu alte scheme de raţionament. Valoarea acestei metode nu constă numai într-o însuşire temeinică a cunoştinţelor ci şi în formarea unor deprinderi de activitate intelectuală. Mulţi elevi încheie ciclul liceal fără a avea formate deprinderi de lucru cu manualul şi documentaţia, ceea ce le creează serioase probleme de adaptare şi explică eşecurile din primul an de studenţie şi greutatea de adaptare la cerinţele studiului universitar. Metoda muncii cu manualul este un aspect al studiului individual şi se introduce ca metodă, treptat, sub directa îndrumare şi supraveghere a profesorului. Sunt discipline şi profesori care aplică în mod abuziv această metodă. Pe lângă efectele negative asupra învăţării, aceste abuzuri ascund şi alte aspecte care nu fac obiectul acestei lucrări. Înainte de a aborda această metodă, profesorul trebuie să atragă atenţia elevului asupra aspectelor importante ale lecţiei, care trebuiesc urmărite în mod special, cerând elevului să realizeze un rezumat cu principalele idei ce trebuie reţinute. Rolul profesorului nu se limitează numai la a indica lecţia din manual sau documentaţia care trebuie studiată. În timpul studierii de către elevi a noului material, profesorul are un rol activ. El urmăreşte fiecare elev cum îşi întocmeşte *conspectul*, dă îndrumări cu voce scăzută elevilor care-l solicită, verifică planurile întocmite de aceştia, corectând acolo unde este cazul. Profesorul poate să descopere în acest fel anumite lacune în cunoştinţele anterior dobândite ale elevilor şi să intervină ulterior pentru remedierea lor. El se ocupă deopotrivă de elevii slabi şi de cei buni cărora le dă sarcini suplimentare, reuşind astfel să-şi facă o imagine despre stilul de lucru şi ritmul fiecărui elev. După studierea individuală din manual sau documentaţie, urmează discuţii asupra celor însuşite de către elevi. Aceste discuţii au scopul de a preciza problemele esenţiale ale lecţiei, a le sistematiza, a înlătura posibilitatea unor omisiuni din partea elevilor sau chiar a însuşirii eronate a unor noţiuni. Profesorului i se cere o pregătire minuţioasă a materialului pentru a fi în măsură să răspundă prompt la orice întrebare pusă de către elevi. Nu orice lecţie se pretează la a fi însuşită din manual. Metoda se aplică numai lecţiilor care au în manual o redactare sistematică şi

accesibilă nivelului de vârstă şi de cunoştinţe ale elevilor. Metoda poate fi aplicată pentru studiul unor aplicaţii soft, limbaje procedurale (de exemplu HTML) sau în studiul comenzilor sistemelor de operare. Elevilor li se recomandă studiul temei stabilite pentru acomodarea cu noţiunile, apoi profesorul reia prezentarea cu sublinierea aspectelor esenţiale (<31>). Având o asemenea bază, profesorul se poate concentra asupra **discursului** său (ceea ce urmează este într-o strânsa legătură şi cu precedentele metode). Dacă este organizat bine, există urmatoarele avantaje (<(?????-Wankat...)>):

- Urmărirea atentă a audienţei: fiecărui „ascultător" îi poate fi sugerată ideea că este personajul principal, ca el (ea) este cel vizat(ă) în primul rând.
- Noi porţiuni de text pot fi uşor introduse suplimentar, prin referirea la „manual".
- Se prezintă lucruri deja verificate. Nimic nu poate „merge rău", exceptând ... îmbolnăvirea lectorului.
- Stresul fiecărui elev în parte poate fi micşorat, el ştiind ca nu este „destinatarul" special.
- Există posibilitatea unui „feedback" imediat şi anumite principii de învăţare pot fi imediat folosite.
- Există posibilitatea pregătirii prealabile a materialului, cu durată determinată, inclusiv cea a expunerii.
- Posibilitatea de control asupra a ceea ce s-a transmis/recepţionat, cui, când, sub ce formă, precum şi a modului „de reacţie" este foarte mare.

Desigur că există şi dezavantaje. Nu insistăm, pentru că ideea este că fiecare avantaj de mai sus devine un dezavantaj dacă profesorul este un prost gestionar al metodelor şi timpului său. Oricum, se poate ajunge, din partea clasei, la pasivitate, stagnare, plictiseală, lipsă de individualizare etc.

1.9. Metoda jocurilor didactice

Jocurile didactice (şi nu numai) pe calculator au valenţele lor educative. Ca metodă de învăţare, jocurile didactice dau rezultate deosebite în special la clasele mici, dar marele pericol care planează asupra acestei metode de instruire îl constituie acele aplicaţii soft care au o încărcătură educativă redusă, dar prin atractivitate captivează şi reţin atenţia elevului, uneori ore în şir, fără ca acesta să dobândească cunoştinţe sau deprinderi corespunzătoare cu efortul făcut. Un rol aparte se atribuie jocurilor *manipulative*, prin care elevul devine conştient de proprietăţile obiectului studiat, îşi formează deprinderi şi dexterităţi de utilizare a acestuia prin simularea pe calculator a utilajului sau dispozitivului respectiv. Aceste jocuri numite uneori şi *simulatoare*, necesită în cele mai frecvente cazuri echipamente periferice suplimentare, unele specializate pe lângă cele clasice. Amintim în acest caz, utilizarea unor căşti speciale pentru obţinerea efectului de realitate virtuală, echipamente care simulează condiţii de zbor (pentru pilotaj) etc. Alte tipuri de jocuri, numite *reprezentative*, printr-o simbolizare sau abstractizare a unor elemente reale, conduc spre descoperirea unor reguli de lucru (sau joc) cu aceste elemente, dezvoltând în acest fel imaginaţia elevului. Ce altceva reprezintă un produs soft (de exemplu, un editor grafic sau de text) atunci când înveţi să-l utilizezi, decât un joc mult mai serios? Chiar dacă această metodă nu este caracteristică studiului Informaticii, la limita dintre jocul didactic şi învăţarea asistată de calculator se situează o bună parte dintre software-urile de învăţare, atât a Informaticii cât şi a altor discipline (<38>).

1.10. Instruirea programată şi învăţarea asistată de calculator

Instruirea programată poate fi aplicată cu mare succes în momentele în care obiectul primordial al predării îl constituie utilizarea unui mecanism real. În cadrul instruirii programate, esențiale devin probele şi produsele demonstrative, pe care ar trebui să le descriem elevilor. Trebuie avut în vedere ca numărul de ore afectat acestei instruiri programate nu trebuie să fie foarte mare. Acestea trebuie să includă un număr suficient de ore de verificare a cunoştințelor acumulate, evitându-se însă monotonia şi instaurarea plictiselii (este recomandată utilizarea alternativă a altor metode). Trebuie evitată de asemenea şi folosirea metodei pentru un timp îndelungat, lucru care poate conduce în anumite situații, la o izolare socială a elevului. O idee pentru contracararea acestor efecte ar fi creşterea **numărului de ore sau organizarea activităților pe grupuri sau în echipă.** *Instruirea asistată de calculator* este un concept diferit de instruirea programată, doar în modalitatea de utilizare. Există aceleaşi premise şi moduri de utilizare, cu excepția faptului că un sistem de calcul devine principala interfață dintre un profesor şi un elev. Absolut toate noțiunile, conceptele, exercițiile, problemele, evaluările, testările, prezentările legate de o anumită temă în cadrul unei lecții (inclusiv estimarea îndeplinirii obiectivelor) sunt îndepliniri, dirijări, verificări cu ajutorul calculatorului (mediul soft corespunzător). Procesul de predare-învățare şi verificare-evaluare funcționează pe baza *principiului cibernetic comandă-control-reglare (autoreglare)*. Instruirea programată, ca metodă didactică, presupune construirea unor *programe* de învățare, care prin fragmentarea materialului de studiat în secvențe realizează o adaptare a conținuturilor la posibilitățile elevilor, la ritmul lor de învățare, asigură o învățare activă şi o informare operativă asupra rezultatelor învățării, necesară atât elevului pentru autocorectare cât şi profesorului. În elaborarea *programelor* de învățare se au în vedere următoarele operații(<38>):

- Precizarea obiectivelor operaționale în funcție de conținut şi posibilitățile elevilor.
- Structurarea logică a conținutului după principiul paşilor mici şi al învățării gradate.
- Fracționarea conținutului în secvențe de învățare (unități didactice) inteligibile şi înlănțuite logic.
- Fixarea după fiecare secvență a întrebărilor, exercițiilor sau problemelor ce pot fi rezolvate pe baza secvenței informaționale însuşite.
- Stabilirea corectitudinii răspunsurilor sau soluțiilor elaborate; aceasta se poate realiza fie prin alegerea dintre mai multe răspunsuri posibile (trei, patru sau chiar cinci), iar în situația în care nu a fost ales răspunsul corect, se poate recurge la întrebări suplimentare, fie se elaborează un răspuns şi se compară cu cel corect.

Ca orice inovație, instruirea programată a trecut prin câteva faze contradictorii. La început s-a lovit de rezerva tenace a tradiției şi de dificultățile materiale (tehnice), apoi după ce a câştigat teren în conştiința teoreticienilor şi practicienilor s-au exagerat într-o oarecare măsură valențele ei aplicative, *creându-se iluzia descoperirii pietrei filozofale în domeniul pedagogic*. În final, după o analiză lucidă, s-a admis că există părți pozitive şi negative. Criticile aduse instruirii programate sunt atât de ordin psihologic, cât şi de ordin pedagogic şi metodic. *Psihologic*, instruirii programate i se impută faptul că nu ține seama de principiile psihologice ale învățării, vizând învățarea ca o simplă succesiune şi înmagazinare de fapte. De asemenea, se ştie că motivația învățării nu poate fi analizată numai prin prisma reținerii şi învățăririi imediate, făcând abstracție de interesul elevului față de conținut. În plus, elevul lucrând singur sau cu calculatorul, se simte izolat. *Pedagogic* vorbind, fărâmițarea conținuturilor este în detrimentul formării unei viziuni globale, iar valoarea cunoaşterii imediate de către elev a rezultatului obținut are valențe contestabile. *Metodic*, decupajul analitico-sintetic al conținuturilor îngustează elevului posibilitatea formării aptitudinilor de analiză şi sinteză. Aceste critici au determinat mutații serioase în concepția de aplicare a metodei, dar practica didactică dovedeşte că atunci când se cunosc şi se evită cauzele care generează efecte negative, metoda produce rezultate bune. Tendințele de îmbunătățire a aplicării metodei se îndreaptă către alternarea utilizării metodei cu celelalte metode clasice. Inserarea într-o lecție programată a unor metode clasice schimbă determinarea muncii şcolare, repunându-l pe elev în directă dependență cu activitatea profesorului şi dându-i acestuia posibilitatea să verifice gradul de însuşire a cunoştințelor conținute în program. O altă tendință este aceea de a modifica modul de redactare al programului, în special prin mărirea volumului de informație din unitățile logice şi prin separarea părții de verificare, existând situații în care verificarea se va face după câteva ore sau chiar a doua zi. În plus, în program se pot insera secvențe independente, care să necesite timp mai mare de gândire sau de lucru. Izolarea imputată învățării programate poate fi contracarată prin alternarea cu munca în grup sau chiar prin învățare programată în grup, situație în care grupul parcurge în colectiv un program special conceput în acest sens.

Un **exemplu de program de învățare** care convinge prin atractivitate (<31>) este un program de *învățare a tablei înmulțirii*:

- se generează aleator, succesiv, zece perechi de numere naturale de la 1 la 10;
- se afişează pe ecran perechile corespunzătoare sub forma $n_1 x n_2$ şi elevul introduce de la tastatură rezultatul;
- programul afişează un mesaj sau emite un semnal sonor în cazul în care răspunsul este incorect şi repeta întrebarea; dacă nici al doilea răspuns nu este corect, se va afişa răspunsul corect;

- fiecare răspuns este punctat, iar la sfârşit se va afişa nota obţinută; programul poate cere continuarea cu un nou set de zece întrebări.
Perspectiva învăţării asistate de calculator, inclusiv prin intermediul INTERNET-ului, este certă. Ea oferă posibilitatea prezentării programului, verificării rezultatelor şi corectării erorilor, modificând programul după cunoştinţele şi conduita elevului. Calculatorul nu numai că transmite un mesaj informaţional, dar el poate mijloci formarea şi consolidarea unor metode de lucru, de învăţare. Se poate afirma că învăţarea asistată de calculator nu numai că *învaţă elevul*, ci îl şi *învaţă cum să înveţe*. Prin aplicarea acestei metode de învăţare nu se întrevede diminuarea rolului profesorului. Dimpotrivă, sarcinile lui se amplifică prin faptul că va trebui să elaboreze programe şi să le adapteze la cerinţele procesului educativ. Oricât de complete ar fi programele de învăţare asistată de calculator, *profesorul rămâne cea mai perfecţionată maşină de învăţat*. Se poate consulta site-ul **MECT** pentru lucruri suplimentare (adresele le vom furniza şi în **Anexa 2**).

1.11. Studiu de caz (exemplu general)

Acest exemplu poate fi, în fapt, considerat ca un corolar al tuturor capitolelor anterioare. După cum ştim deja, în viziunea planului cadru pentru licee, filiera teoretică, specializarea *Matematică-Informatică*, funcţionează clase cu studiul intensiv al disciplinelor de Informatică. Disciplinele de Informatică sunt componente ale ariei curriculare *Tehnologii* şi se studiază pe trunchiul comun cu *4* ore pe săptămână şi *1-3* ore pe săptămână în cadrul curriculum-ului *la decizia şcolii*. Având în vedere finalitatea aplicativă imediată a conţinutului teoretic, orele din trunchiul comun se desfăşoară consecutiv, în laborator, cu clasa împărţită în două grupe de *10-15* elevi. În cadrul curriculumu-lui *la decizia şcolii* orele se pot organiza pe grupe asistate chiar de doi profesori, în funcţie de specificul conţinuturilor modulului (dacă acesta este interdisciplinar). Unitatea didactică intitulată *Structuri arborescente şi aplicaţii* acoperă o arie de conţinuturi neabordate pe trunchiul comun şi se adresează elevilor *Clasei a X-a*. Necesitatea abordării acestor conţinuturi este relevată de frecvenţa problemelor practice care pot fi rezolvate utilizând aceste structuri de date şi de constatarea faptului că în cadrul concursurilor şcolare, la nivel judeţean, naţional şi internaţional, frecvenţa problemelor care necesită cunoştinţe de teoria grafurilor, în speţă de utilizarea structurilor arborescente, este mare (**Anexa 1, Capitolul 6**).

1.11.1. Obiective cadru

Conştientizarea necesității organizării datelor în structuri arborescente şi formarea deprinderilor de utilizare a acestor structuri. Stimularea creativității şi formarea deprinderilor de simulare şi utilizare a modelelor matematice în rezolvarea problemelor concrete. Implementarea algoritmilor specifici structurilor arborescente şi utilizarea lor în aplicații în vederea optimizării alocării resurselor.

1.11.2. Grupuri țintă

Cursul se adresează elevilor de *Clasa a X-a* cu performanțe peste nivelul mediu al clasei, care au atins obiectivele trunchiului comun şi care dovedesc reale perspective de progres şi disponibilitate la efort, elevilor participanți la concursurile şi olimpiadele şcolare. Se studiază în *semestrul al II-lea*.

1.11.3. Obiective de referință şi activități de învățare

Asimilarea noțiunilor şi rezultatelor teoretice cu privire la structurile de date de tip arbore şi a modului de reprezentare a acestora în memoria calculatorului. Aplicarea cunoştințelor dobândite şi a deprinderilor formate în rezolvarea unor probleme concrete.

Obiective de referință	**Activități de învățare**
3.1 – să cunoască alte structuri de date de tip arbore (de exemplu heap-uri)	- prezentarea structurilor de tip heap, a modului de memorare a şi a dinamici lor; perceperea avantajelor utilizării în aplicații a acestor structuri.
3.2 - diversificarea gamei structurilor cunoscute şi adaptarea lor la specificul aplicațiilor (arbori parțiali, arbori de compresie, arbori de joc)	perceperea necesității adaptării structurilor cunoscute la specificul aplicațiilor; integrarea şi adaptarea la particularitățile aplicației a algoritmilor.

1.11.4. Specificarea conținuturilor

1.
Noțiuni introductive.
1.1 Proprietăți ale arborilor.
1.2 Arbori cu rădăcină.
1.3 Arbori binari şi proprietăți.
2. Reprezentarea arborilor.
1.1 Reprezentarea arborilor binari.
2.2 Operații elementare pe arbori binari (creare, parcurgere).
2.3 Reprezentarea arborilor binari stricți.
3. Arbori asociați expresiilor aritmetice.
4. Structuri de căutare.
4.1 Căutarea secvențială.

4.2 Căutarea binară.
4.3 Căutarea pe arbori binari (de căutare).
4.4 Alte operații pe arbori binari de căutare.
4.5 Arbori binari de căutare optimali.
5. Arbori echilibrați.
5.1 Arbori *Adelson-Velskii-Landis* (AVL).
5.2 Arbori bicolori.
6. Heap-uri.
6.1 *Min-heap*-uri şi *Max-heap*-uri.
6.2 Crearea unui heap.
6.3 Heap-sort.
6.4 Cozi cu prioritate.
7. Arbori parțiali.
7.1 Arbori parțiali de cost minim.
7.2 Algoritmii lui *Kruskal* şi *Prim* pentru determinarea unui arbore parțial de cost minim.
7.3 Arbori parțiali BF.
7.4 Arbori parțiali DF.
8. Arbori de compresie *Huffman* (codul *Huffman*).

1.11.5. Elaborarea standardelor de performanță

Corespunzător obiectivelor specifice, se elaborează standardele de performanță ce se doresc atinse. Astfel, pentru competența *cunoaşterea proprietăților arborilor în general şi a arborilor binari în particula*r, se va elabora standardul:

	Nivel	Descriptor de nivel
	A. (9-10)	-elevul cunoaşte toate proprietățile definitorii ale arborilor ; - înțelege şi utilizează codul lui Pruffer; -operează cu noțiunea de arbore binar; -cunoaşte proprietățile arborilor binari şi modul de reprezentare în memorie a acestora; - este capabil să creeze şi să parcurgă un arbore binar; - implementează optimal în aplicații algoritmii învățați; -intuieşte necesitatea şi este capabil să adapteze organizarea datelor de intrare la cerințele algoritmilor ce vor fi folosiți.

	B. (7-8)	-elevul cunoaşte majoritatea proprietăților definitorii ale arborilor; -înțelege şi utilizează codul lui Pruffer; -operează cu noțiunea de arbore binar; -cunoaşte proprietățile arborilor binari şi modul de reprezentare în memorie a acestora, dar are preferințe pentru anumite metode, care nu sunt totdeauna cele optime; -este capabil să creeze şi să parcurgă un arbore binar; -este capabil să implementeze în aplicații algoritmii învățați; -are unele dificultăți în organizarea datelor de intrare, la implementarea algoritmilor folosiți.

	C. (5-6)	-elevul cunoaşte unele proprietăți definitorii ale arborilor; - înțelege, dar utilizează cu dificultate codul lui Pruffer; - înțelege noțiunea de arbore binar; -cunoaşte unele proprietăți ale arborilor binari, dar foloseşte o singură metodă de reprezentare în memorie a acestora; -are dificultăți în crearea şi parcurgerea arborilor binari; -implementează cu dificultate în aplicații algoritmii învățați; -are dificultăți în organizarea datelor de intrare;
	C. (3-4)	-elevul cunoaşte unele proprietăți definitorii ale arborilor; -înțelege, dar nu poate utiliza codul lui Pruffer; -are lacune în înțelegerea noțiunii de arbore binar; -nu cunoaşte proprietățile arborilor binari şi nu este capabil să folosească nici o metodă de reprezentare în memorie a acestora; -nu este capabil să creeze şi să parcurgă un arbore binar; -are dificultăți în organizarea datelor de intrare.

2. Metode specifice de învățare

Acestea se referă la ramuri (subramuri) particulare ale Informaticii (cum ar fi teoria algoritmilor, logica, etc.). Fără a intra în detalii, invităm cititorul să consulte atât **Bibliografia** cât şi **Capitolul 1.** Metodele nu sunt independente la fel ca obiectivele sau principiile didactice. Ele se pot combina, iar dacă luăm în calcul şi varietatea de obiective şi/sau metode/metodologii specifice, ajungem la un număr impresionant de variante educaționale oră/temă. Din acest motiv, am preferat să furnizăm exemple globale şi nu locale, pentru fiecare metodă în parte. Alte exemple pot fi consultate pe parcursul lucrării, în **Anexa 1, Capitolul 6**. Alegerea problemelor este condiționată de: planul de învățământ; manualele alternative; contextul local; nivelul clasei; materialul didactic disponibil; criteriile de valoare receptate; formularea problemelor (acestea trebuie să țină cont de: conținutul manualelor; noțiunile anterioare pe care le posedă elevii; caracterul fundamental sau legislativ al problemelor). Ca o concluzie parțială şi nici pe departe exhaustivă, tratarea rezolvărilor trebuie să aibă în vedere obținerea rezultatelor pe căi clare (şi, pe cât posibil, verificabile printr-o altă metodă), analiza metodelor utilizate, reținerea tipurilor de raționamente folosite, deschiderea unor perspective pentru probleme similare sau mai complexe. Se urmăreşte cunoaşterea activă a noțiunilor învățate, adâncirea semnificațiilor, asimilarea metodelor de rezolvare, aplicarea lor în rezolvarea altor tipuri de probleme.

CAPITOLUL 6

Liste, Stive, Cozi, Grafuri, Arbori, Sortare/Căutare

Scopul acestui capitol este de a prezenta câteva exemple semnificative de *algoritmi asupra unor structuri de date clasice*, care au fost referiți pe parcursul lucrării, dar nu într-un mod metodic.

1. Liste

Lista este o *multimulțime dinamică*, adică este o colecție cu un număr variabil de elemente, care se pot repeta. Elementele au acelaşi *tip*. În general, tipul elementelor unei liste este un *tip utilizator*. Elementele unei liste se numesc *noduri*. Dacă între nodurile unei liste există o singură

relaţie de ordine, atunci lista se numeşte *simplu înlănţuită*. Dacă între nodurile unei liste există două relaţii de ordine, atunci lista se numeşte *dublu înlănţuită*. În general vom spune că o listă este *n-înlănţuită* dacă între nodurile ei sunt definite *n* relaţii de ordine. În legătură cu listele se au în vedere unele operaţii de interes general:

a) crearea unei liste;
b) accesul la un nod oarecare al listei;
c) inserarea unui nod într-o listă;
d) ştergerea unui nod dintr-o listă;
e) ştergerea unei liste.

1.1. Liste simplu înlănţuite

Între nodurile unei liste simplu înlănţuite este definită o singură relaţie de ordine, *totală*. De obicei această relaţie este cea de *succesor*, adică fiecare nod conţine un pointer a cărui valoare reprezintă adresa nodului următor din listă. Asemănător, se poate defini relaţia de *precedent*. În cele ce urmează ne vom mărgini numai la liste simplu înlănţuite pentru care nodurile satisfac relaţia *succesor*. O asemenea listă se caracterizează prin aceea că există totdeauna un nod şi numai unul care nu are următor (*succesor*, *fiu*), precum şi un nod, unic, care nu este următorul (succesorul) nici unui alt nod. Aceste noduri formează *capetele* listei simplu înlănţuite. Pentru a gestiona nodurile unei liste simplu înlănţuite, vom utiliza doi pointeri spre cele două capete ale listei. Notăm pointerul spre nodul care nu este următorul (succesorul) nici unui alt nod al listei (adică primul nod al listei) cu `pInceputLista` şi cu `pSfarsitLista` pointerul spre nodul care nu are succesor în listă. Aceşti pointeri vor fi utilizaţi în toate exemplele pe care le vom avea în vedere în prelucrarea listelor simplu înlănţuite. Ei pot fi definiţi fie ca variabile globale, fie ca parametri pentru funcţiile de prelucrare a listei, fie ca date membru ale unui obiect. Tipul unui nod într-o listă simplu înlănţuită se poate defini folosind o declaraţie de forma (**C/C++**):

```
typedef struct _tagNod {
      //declaraţii;
      int   nCodUnic;
      struct _tagNod    *pElementUrmator;
} MPI_Nod;
```

Pointerul *pElementUrmator* va conţine adresa spre următorul nod al listei, adică defineşte relaţia *succesor* pentru nodurile listei. Nodul spre care pointează variabila *pSfarsitLista* va avea drept valoare *NULL* pentru *pElementUrmator* (*pElementUrmator = NULL*;).
Pointerii *pInceputLista* şi *pSfarsitLista* se declară în afara oricărei funcţii (de obicei înaintea definirii funcţiei `main` a programului principal, deci variabile globale) prin:

```
MPI_Nod     *pInceputLista, *pSfarsitLista;
```

Pe tot parcursul acestui capitol ne vom mărgini la a descrie ordinea operaţiilor ce trebuie respectată în lucrul cu diverse structuri. Menţionăm de la început că nu urmărim o optimizare a codului, ci o înţelegere corectă a operaţiilor ce trebuie efectuate şi o claritate a codului scris. De asemenea, acest capitol nu constituie o tratare completă a structurilor prezentate ci doar o sinteză a acestora.

1.1.1. Crearea unei liste simplu înlănţuite

Operaţiile ce trebuiesc efectuate la crearea unei liste simplu înlănţuite sunt:

1. Se inițializează pointerii *pInceputLista* şi *pSfarsitLista* cu valoarea *NULL*, deoarece la început lista este vidă.
2. Se rezervă zonă de memorie în memoria *heap* pentru nodul curent.
3. Se încarcă nodul curent cu informațiile suplimentare.
4. Se atribuie pointerului *pSfarsitLista->pElementUrmator* adresa din memoria *heap* a nodului curent, dacă lista nu este vidă. Altfel se atribuie lui *pInceputLista* această adresă.
5. Se atribuie pointerului *pSfarsitLista* adresa nodului curent.
6. *pSfarsitLista->pElementUrmator = NULL;*
7. Procesul se reia de la pasul *2* de mai sus pentru a adăuga un nod nou la listă.

Pentru claritate, acțiunea de creare a unei liste ar trebui tratată în modul următor:

C.1. Lista este vidă şi trebuie creat primul nod al listei.
C.2. Lista nu este vidă şi se adaugă un nou nod la sfârşitul listei.

Pentru C.1., ordinea operațiilor este următoarea:

Se inițializează pointerii *pInceputLista* şi *pSfarsitLista* cu valoarea *NULL*, deoarece la început lista este vidă.

```
pInceputLista = NULL;
pSfarsitLista = NULL;
```

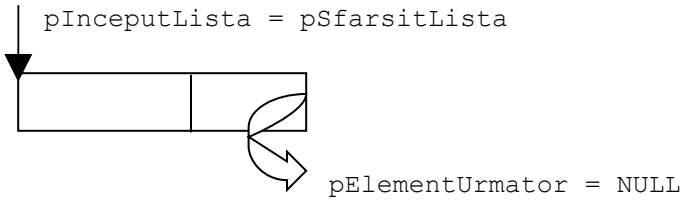
C.1.1. Se rezervă zonă de memorie în memoria heap pentru nodul curent.
C.1.2. Se încarcă nodul curent cu informațiile suplimentare.
C.1.3. Se atribuie pointerului *pInceputLista* şi *pSfarsitLista* adresa din memoria *heap* a nodului curent (pointerii *pInceputLista* şi *pSfarsitLista* au aceeaşi valoare când lista este vidă (valoarea *NULL*) sau lista are un singur element.
C.1.4. Se atribuie valoarea *NULL* pointerului *pElementUrmator*.

Codul ar putea arăta astfel:

```
...
// C1.1.
pInceputLista = NULL;
pSfarsitLista = NULL;

// C1.2.
pTemp = (MPI_Nod*)malloc(sizeof(MPI_Nod));
if (pTemp == NULL)
{
printf ("Memorie insuficienta la crearea listei\n");
exit(1);
}

// C1.3.
// acţiuni specifice de iniţializare a datelor membru din nodul listei

// C1.4.
```

```
pInceputLista = pTemp;
pSfarsitLista = pTemp;

// C1.5.
pSfarsitLista->pElementUrmator = NULL;
```

Grafica intuitivă:



Pentru C.2. ordinea operațiilor este următoarea:

C.2.1. Se rezervă zonă de memorie în memoria heap pentru nodul curent.

C.2.2. Se încarcă nodul curent cu informațiile suplimentare.

C.2.3. Se atribuie pointerului *pSfarsitLista->pElementUrmator* adresa din memoria *heap* a nodului creat.

C.2.4. Se atribuie pointerului *pSfarsitLista* adresa din memoria heap a nodului creat.

C.2.5. Se atribuie valoarea *NULL* pointerului *pSfarsitLista->pElementUrmator*.

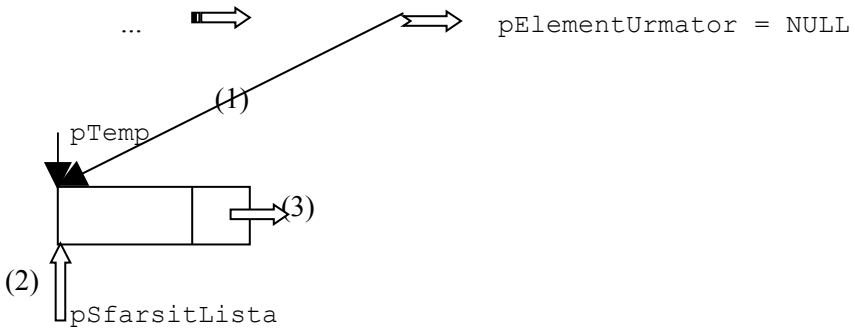
Codul ar putea fi următorul:

```
// C2.1.
pTemp = (MPI_Nod*)malloc(sizeof(MPI_Nod));
if (pTemp == NULL) {
printf ("Memorie insuficienta la crearea listei\n");
exit(1);
}

// C2.2.
// operații specifice de inițializare a nodului

// C2.3. Se face legătura dintre ultimul nod al listei cu noul nod creat
pSfarsitLista->pElementUrmator = pTemp;

// C2.4.  Noul nod creat va deveni ultimul nod al listei
pSfarsitLista = pTemp;
```

//C2.5. Acum pSfarsitLista pointează spre noul nod creat care nu are succesori

```
pSfaristLista->pElementUrmator = NULL;
```

Operația de adăugare a unui nod la o listă existentă poate fi schematizată ca mai jos.



Se alocă în heap memorie pentru noul nod, adresa fiind în *pTemp*.

Ordinea operațiilor care urmează este strictă:

1. realizarea legăturii noului nod cu ultimul nod al listei;
2. noul nod devine ultimul nod al listei;
3. *pElementUrmator* din ultimul nod adăugat ia valoarea *NULL*.

1.1.2. Accesul la un nod al listei simplu înlănțuite

După modul cum este definită relația de ordine în listă, rezultă şi accesul la nodurile listei. Pentru a găsi un nod al listei va trebui să parcurgem lista de la început şi apoi trecem de la un nod la altul folosind pointerul *pElementUrmator*. Pentru a găsi un anumit nod al listei va trebui să definim *criterii de identificare* pentru acesta (de exemplu numărul de ordine al nodului, nodul care conține o anumită informație, etc.).

Parcurgerea toatală a listei pentru a afişa (a efectua anumite operații) poate fi redată cu următorul cod:

```
...
MPI_Nod     *pTemp;
pTemp = pInceputLista;
while (pTemp != NULL)
{
      // Calcule. Adresa nodului curent este în pTemp.
      ...
      // Trec la următorul nod al listei
      pTemp = pTemp->pElementUrmator;
}
...
```

Dacă definim drept criteriu de căutare după o anumită valoare a datei membru *nCodUnic* al structurii *MPI_Nod*, atunci determinarea nodului respectiv se va face parcurgând lista de la început şi comparând valoarea datei membru *nCodUnic* cu valoarea memorată într-o variabilă locală (în general preluată de la tastatură sau rezultată în urma unor calcule anterioare). Presupunem că valoarea este păstrată în variabila *m_nCodUnic*. Codul poate arăta astfel (în cadrul unei funcții):

```
...
MPI_Nod     *pTemp;
int   m_nCodUnic;
```

```
...
pTemp = pInceputLista;
while (pTemp != NULL)
{
      if (pTemp->nCodUnic == m_nCodUnic)
            return pTemp; // în pTemp avem adresa nodului căutat
      pTemp = pTemp->pElementUrmator;
}
return NULL;      //nu există un asemenea nod
...
```

1.1.3. Inserarea unui nod într-o listă simplu înlănțuită

Inserarea unui nod într-o listă simplu înlănțuită se poate face în mai multe moduri:

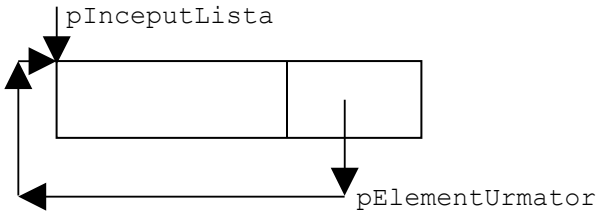
1. inserarea înaintea primului nod;
2. inserarea înaintea unui nod precizat printr-o cheie;
3. inserarea după un nod precizat printr-o cheie;
4. inserarea după ultimul nod al listei – aceasta coincide cu operația de adăugare la sfârşitul listei, descrisă mai înainte.

1.1.3.1. Inserarea unui nod într-o listă simplu înlănțuită înaintea primului ei nod

Adresa primului nod al listei (dacă nu este vidă) este păstrată în pointerul *pInceputLista*. Operațiile care trebuiesc efectuate precum şi ordinea acestora este descrisă în continuare:

1) alocare de memorie pentru noul nod, adresa se obține de exemplu în *pTemp* (dacă operația s-a desfăşurat cu succes se continuă cu (*2*) altfel se renunță la inserare);
2) pointerul *pTemp->pElementUrmator* va păstra adresa următorului nod care este în fapt fostul prim nod al listei, deci valoarea lui *pInceputLista*;
   ```
   pTemp->pElementUrmator = pInceputLista;
   ```
3) pointerul *pInceputLista* va primi ca valoare adresa noului nod creat
   ```
   pInceputLista = pTemp;
   ```

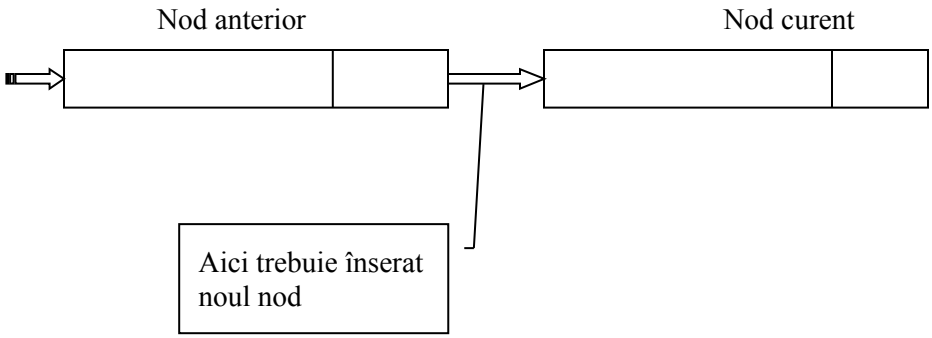
Observație. Dacă se inversează etapele (*2*) cu (*3*) atunci *am pierdut* lista. Pointerul *pSfarsitLista* va puncta spre ultimul element al listei, pointerul *pInceputLista* va puncta spre noul nod creat iar *pInceputLista->pElementUrmator* va puncta tot spre noul nod creat. O încercare de a parcurge lista în acest moment va duce la buclarea programului. Grafic, situația se prezintă astfel:



Dacă lista este vidă această operație coincide cu cea de creare a primului nod al listei.

1.1.3.2. Inserarea unui nod într-o listă simplu înlănțuită înaintea unui nod precizat printr-o cheie

Presupunem că valoarea cheii memorată în nod este în data membru `nCodUnic`. Să reprezentăm grafic ce ar trebui să facem în această situație. Prin nod curent înțelegem nodul din listă care satisface condiția `nCodUnic = m_nCodUnic`:



Judecând după figura de mai sus, ordinea operațiilor care trebuie făcute este următoarea (presupunând că toate operațiile se efectuează cu succes):

1. alocarea memoriei pentru nodul de inserat; adresa va fi păstrată în variabila `pTemp`;
2. păstrarea adresei nodului anterior în variabila `pNodAnterior`;
3. păstrarea adresei nodului care satisface condiția (nodul curent) în variabila `pNodCurent`;
4. atribuirea pointerului `pNodAnterior->pElementUrmator` a adresei nodului ce trebuie jnserat, păstrat în variabila `pTemp`;
5. atribuirea pointerului `pTemp->pElementUrmator` a adresei nodului curent `pNodCurent`;
6. inițializarea datelor pentru noul nod.

Observație. Trebuie avută în vedere posibilitatea că şi primul al listei poate îndeplini condiția de căutare. În acest caz *avem de inserat un nod la începutul listei*.

În continuare, s-ar părea că nu este necesară o altă prelucrare deoarece `pNodCurent = pNodAnterior->pElementUrmator`. Să nu uităm însă că valoarea cheii ne ajută la obținerea nodului curent.

Codul ar putea fi următorul (inserăm acest cod în cadrul unei funcții care returnează `0` în caz de succes sau `1` în caz contrar):

```
...
MPI_Nod     *pTemp, *pNodCurent, *pNodAnterior;
int   m_nCodUnic;
...
pTemp = pInceputLista;
pNodAnterior = pInceputLista;
pNodCurent = NULL;
while (pTemp != NULL){// Caut nodul ce satisface condiția...
      if (pTemp->nCodUnic == m_nCodUnic)
         {
               pNodCurent = pTemp;
               break;
         }
      pNodAnterior = pTemp;
      pTemp = pTemp->pElementUrmator;
```

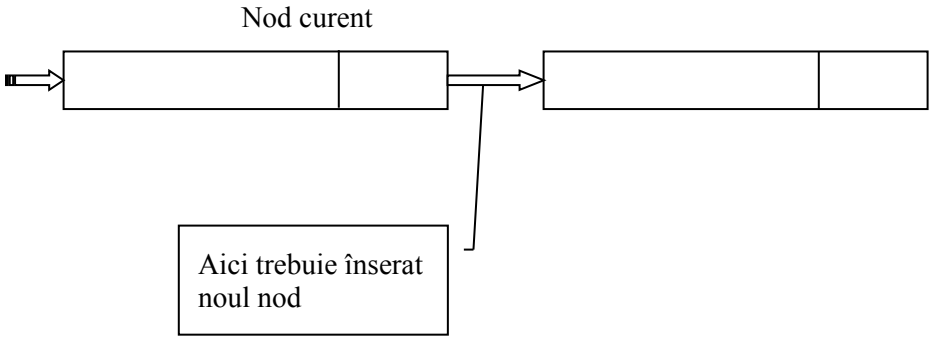
```
}
if (pNodCurent == NULL) {
      printf("Nu exista cheia %d Lista nemodificata...", m_nCodUnic);
      return -1;
   }

if (pNodCurent == pInceputLista)
{// Nodul se inserează la începutul listei. Se va apela funcția care
trateză acest caz. }
else
{ //  aloc memorie pentru noul nod
pTemp =  (MPI_Nod*)malloc(sizeof(MPI_Nod));
if (pTemp == NULL) {
      printf("Nu pot aloca memorie pentru noul nod\n";
      return -1; // Se întoarce un cod de eroare
      }
pNodAnterior->pElementUrmator = pTemp;
pTemp->pElementUrmator = pNodCurent;
// Urmează inițializări pentru nodul inserat
return 0; // Operație încheiată cu succes
```

1.1.3.3. Inserarea unui nod într-o listă simplu înlănțuită după un nod precizat printr-o cheie

Ca mai sus, prin *nod curent* înțelegem nodul din listă care satisface condiția `nCodUnic = m_nCodUnic`. În acest caz avem nevoie de adresa nodului următor, adresă care este păstrată în `pElementUrmator` al nodului curent. Codul pentru determinarea nodului curent este următorul:

```
MPI_Nod     *pTemp;
int   m_nCodUnic;
...
pTemp = pInceputLista;
while (pTemp != NULL)
{
      if (pTemp->nCodUnic == m_nCodUnic)
            return pTemp;
      pTemp = pTemp->pElementUrmator;
}
// Dacă nu există un asemenea nod se întoarce valoarea NULL
return NULL;
...
```

O reprezentare grafică a situației de mai sus este următoarea:



Ordinea operațiilor este:

1. se determină nodul curent, adresa păstrându-se în `pNodCurent`;
2. dacă există nod curent, se alocă memorie pentru nodul ce se va adăuga (presupunem că operația s-a efectuat cu succes); adresa se păstrează în `pTemp`;
3. adresa nodului următor (dacă există), se salvează într-o variabilă temporară, `pTempUrmator`;
4. se actualizează valoarea pointerului `pElementUrmator` din nodul curent, cu adresa noului nod : `pNodCurent->pElementUrmator = pTemp`;
5. se realizează legătura dintre noul nod şi următoarele, `pTemp->pElemntUrmator = pTempUrmator;`
6. dacă `pTempUrmator` este `NULL` (necompletat) atunci înseamnă că inserarea se face la sfârşitul listei şi atunci trebuie actualizată valoarea pointerului `pSfarsitLista` cu adresa nodului adăugat, care se găseşte în `pTemp`; deci `pSfarsitLista = pTemp`.

Observație. Codul poate fi scris imediat *traducând* strict etapele descrise mai sus.

1.1.4. Ştergerea unui nod dintr-o listă simplu înlănţuită

Ştergerea se poate realiza în mai multe moduri. În cele ce urmează avem în vedere următoarele cazuri:

S.1. ştergerea primului nod al unei liste simplu înlănţuite;
S.2. ştergerea unui nod precizat printr-o *cheie*;
S.3. ştergerea ultimului nod al unei liste simplu înlănţuite.

Vom analiza fiecare situaţie în parte punând în evidenţă operaţiile care trebuie efectuate precum şi ordinea acestora. Operaţia comună tuturor cazurilor luate în considerare este cea a eliberării memoriei alocate. Pentru fiecare funcţie (operator) din **C/C++** de alocare de memorie din memoria heap există definită şi funcţia (operatorul) corespunzătoare de eliberare a memoriei ocupate.

1.1.4.1. Ştergerea primului nod al unei liste simplu înlănţuite

Ştergerea primului nod presupune reactualizarea valorii pointerului `pInceputLista` cu valoarea pointerului `pInceputLista->pElementUrmator`. Ordinea operaţiilor este următoarea:

(1) dacă valoarea pointerului `pInceputLista` este `NULL` atunci lista este vidă şi nu avem ce şterge (operaţie terminată);
(2) dacă `pInceputLista = pSfarsitLista` atunci lista are un singur nod şi vom elibera memoria ocupată de acel nod după care vom asigna valorea `NULL` pentru pointerii `pInceputLista` şi `pSfarsitLista` (operaţie terminată), în caz contrar se trece la (*3*);
(3) păstrăm adresa de început a listei într-o variabilă temporară, `pTemp`;
(4) asignăm pointerului `pInceputLista` valoarea pointerului
```
pInceputLista->pElementUrmator;
```
(5) eliberăm zona de memorie a cărei adresă se află în `pTemp`.

Se observă că dacă se execută direct (*4*) se pierde adresa zonei de memorie ce trebuie eliberată.

1.1.4.2. Ştergerea unui nod precizat printr-o cheie

Ştergerea unui nod precizat printr-o cheie (se presupune că nodul care trebuie şters are succesor) presupune refacerea legăturilor dintre nodul precedent şi succesorul nodului şters, precum şi eliberarea zonei de memorie alocate. Presupunem că lucrăm cu următoarele variabile de memorie:

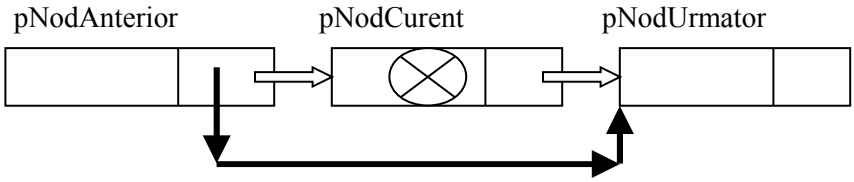
- `pNodAnterior` ce conţine adresa nodului precedent celui ce trebuie şters;
- `pNodCurent` ce conţine adresa nodului ce trebuie şters;
- `pNodUrmator` ce conţine adresa nodului succesor celui ce trebuie şters care se obţine astfel:

```
pNodUrmator = pNodCurent->pElementUrmator;
```

Cu aceste notaţii ordinea operaţiilor este următoarea:

(1) se actualizează valoarea pointerului `pNodAnterior->pElementUrmator` cu valoarea pointerului `pNodUrmator`;
(2) se eliberează zona de memorie dată de `pNodCurent`.

Reprezentarea grafică:



Săgeţile îngroşate indică noua legătură care trebuie stabilită. Nodul *din mijloc* va fi şters.

Determinarea nodului curent se va face cu ajutorul unui cod asemănator celui descris la *căutarea unui nod folosind o cheie*.

Observaţie. Dacă nodul ce trebuie şters este primul nod al listei (`pNodCurent = pInceputLista`) atunci se aplică soluţia indicată în paragraful anterior.

1.1.4.3. Ştergerea ultimului nod al unei liste simplu înlănţuite

Această operaţie presupune următoarele acţiuni:

(1) dacă lista este vidă atunci nu avem ce şterge, operaţia fiind terminată;

(2) determinarea penultimului nod al listei, a cărui adresă o vom păstra în variabila `pTemp`;

(3) eliberarea zonei de memorie a cărei adresă se află în `pSfarsitLista`;

(4) actualizarea valorii pointerului `pSfarsitLista` cu valoarea variabilei `pTemp`;

(5) setarea pe `NULL` a pointerului `pSfaristLista->pElementUrmator`.

Observaţie. Aplicarea efectivă necesită de fiecare dată parcurgerea listei în totalitatea ei, ceea ce pentru liste mari operaţia este consumatoare de timp.

Etapele de mai sus nu tratează cazul când lista are exact un singur nod. În această situaţie, înainte de etapa (*2*) ar trebui testat dacă `pInceputLista = pSfarsitLista`. În caz afirmativ, se execută etapele descrise la ştergerea primului nod al listei. De asemenea trebuie testat mereu dacă lista nu este vidă.

1.1.4.4. Ştergerea unei liste simplu înlănţuite

Ştergerea unei liste simplu înlănţuite se poate face prin aplicarea repetată a acţiunii de ştergere a primului nod din listă. Se repetă acest procedeu până când valoarea pointerului `pInceputLista` devine `NULL`.

1.2. Stive

O **stivă** este o listă simplu înlănţuită gestionată conform principiului *LIFO* (*Last In First Out*). Conform acestui principiu, ultimul nod pus în stivă este primul nod care este scos din stivă.

Operaţiile cele mai importante care se definesc asupra unei stive sunt:

(1) se adaugă un element în stivă (`push`);

(2) scoate un element din stivă (`pop`);

(3) se şterge stiva (`clear`).

Primele două operaţii afectează *vârful* stivei.

Pentru a implementa o stivă printr-o listă simplu înlănţuită, va trebui să identificăm baza şi vârful stivei cu capetele listei simplu înlănţuite. Distingem două posibilităţi:

I.1. nodul spre care pointează variabila `pInceputLista` este *baza* stivei, iar nodul spre care pointează variabila `pSfarsitLista` este *vârful* stivei;

I.2. nodul spre care pointează variabila `pInceputLista` este vârful stivei, iar nodul spre care pointează variabila `pSfarsitLista` este baza stivei.

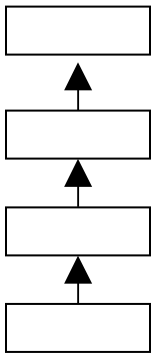
În cazul *I.1.*, funcțiile `push` şi `pop` se identifică cu operațiile de adăugare a unui nod la sfărşitul listei simplu înlănțuite, respectiv cu ştergerea ultimului nod al unei liste simplu înlănțuite. Dacă revenim la operația de ştergere a ultimului nod al unei liste simplu înlănțuite, atunci constatăm că funcția `pop` este ineficientă în acest caz, pentru că nu avem acces direct la penultimul nod al listei simplu înlănțuite. În cazul *I.2.*, funcțiile `push` şi `pop` se identifică cu operațiile de adăugare a unui nod la începutul listei simplu înlănțuite, respectiv de ştergere a primului nod al unei liste simplu înlănțuite. După cum am observat, aceste operații efectuate la începutul listei se realizează fără a fi necesară parcurgerea listei simplu înlănțuite. În concluzie, dacă se implementează o stivă folosind liste simplu înlănțuite este de preferat varianta *I.2*. În ambele situații (*I.1., I.2.*) funcția `clear` – şterge stiva – se implementează la fel ca în cazul ştergerii unei liste simplu înlănțuite.

Observație. O stivă care are un număr maxim cunoscut de elemente poate fi implementată şi ca un vector. De exemplu, o stivă de întregi se defineşte astfel:

```
int stiva[100];
```

caz în care funcțiile `push`, `pop` şi `clear` au o cu totul altă implementare. În acest caz numărul maxim de elemente al stivei va fi `100` (de la `0` la `99`). Va exista un indice, `nIndiceStiva`, prin care vom gestiona vârful stivei. În general, *punerea* unui element pe stivă va însemna verificarea faptului dacă nu se depăşeşte valoarea maximă a indicelui (`99` în acest caz), incrementarea valorii indicelui urmată de actualizarea elementului stivei. Scoaterea unui element din stivă va însemna preluarea valorii curente data de indicele stivei urmată de decrementarea indicelui stivei. În cazul funcției `pop` se va verifica faptul că indicele nu trebuie să devină negativ. Operația de ştergere a stivei este echivalentă cu setarea pe `0` (zero) a indicelui stivei.
O reprezentare grafică a stivei este dată în figura următoare.

Vârful stivei



Baza stivei

1.3. Cozi

O listă simplu înlănțuită gestionată după principiul *FIFO* (*First In First Out*), adică primul nod introdus în listă este şi primul nod care va fi scos din listă, se numeşte **coadă**. Cele două capete ale listei simplu înlănțuite care implementează o coadă sunt şi *capetele* cozii. Operațiile care se definesc asupra unei cozi sunt aceleaşi ca la stive:

CO.1. adăugarea unui element în coadă;
CO.2. scoaterea unui element din coadă;
CO.3. ştergerea cozii.

Implementarea acestor funcţii este aceeaşi cu implementarea funcţiilor de adăugare a unui nod la sfârşitul unei liste simplu înlănţuite, respectiv de ştergere a unui nod de la începutul aceleaşi liste.

Observaţie. Implementarea unei cozi folosind un tablou unidimensional (static) se dovedeşte în acest caz ineficientă. De exemplu, la fiecare extragere a unui element din coadă elementele tabloului trebuie rearanjate (mutate spre stânga).

1.4. Liste circulare simplu înlănţuite

Lista simplu înlănţuită pentru care valoarea pointerului, *pSfarsitLista->pElementUrmator*, este egală cu valoarea pointerului *pInceputLista* (ultimul nod al listei punctează spre primul nod al listei) se numeşte ***listă circulară simplu înlănţuită***.

Din definiţia listei circulare simplu înlănţuite se constată că toate nodurile sunt *echivalente*: fiecare nod are un succesor şi în acelaşi timp este succesorul altui nod. Într-o astfel de listă nu mai există *capete*. Gestiunea nodurilor listei circulare simplu înlănţuite se realizează cu *ajutorul unei variabile ce punctează spre un nod oarecare al listei*. Pentru cele ce urmează vom nota această variabilă cu *pListaCirculara*, definită astfel:

```
MPI_Nod     *pListaCirculara;
```

Operaţiile posibile asupra acestui tip de listă sunt:

1. crearea unei liste circulare;
2. adăugarea unui nod înainte sau după un alt nod care satisface un anumit criteriu de identificare;
3. ştergerea unui nod care satisface un anumit criteriu de identificare;
4. şteregerea completă a listei.

1.4.1. Crearea unei liste circulare simplu înlănţuite

Crearea listei circulare simplu înlănţuite se face analog ca în cazul listei simplu înlănţuite. Pentru început, variabila *pListaCirculara* va avea valoarea *NULL*. Nodurile care se vor adăuga vor fi plasate după nodul spre care pointează *pListaCirculara*. Etapele creării listei circulare înlănţuite sunt:

1. alocarea de memorie pentru nodul care se va crea, adresa este în *pTemp*;
2. noul nod creat va puncta spre nodul următor celui gestionat de *pListaCirculara*, adică, *pTemp->pElementUrmator = pListaCirculara->pElementUrmator*;
3. se va face atribuirea variabilei *pListaCirculara->pElementUrmator* a adresei nodului creat, *pListaCirculara->pElementUrmator = pTemp*.

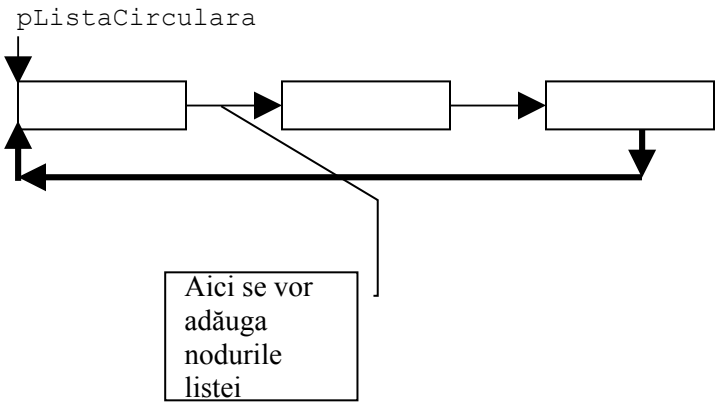
Un cod parţial poate arăta astfel:

```
// Alocare în memoria heap a noului nod

pTemp =  (MPI_Nod*)malloc(sizeof(MPI_Nod));
if (pTemp == NULL)
{     printf("Memorie insuficienta. Alocare esuata...");
}
```

```
else
{
      if (pListaCirculara == NULL)  // Lista vidă
      {
        pListaCirculara = pTemp;
            pListaCirculara->pElementUrmator = pTemp;
      // inițializări informații nod
            ...
      }
      else
   {
   // Inserare după nodul identificat de pListaCirculara
   pTemp->pElementUrmator = pListaCirculara->pElementUrmator;
   pListaCirculara->pElementUrmator = pTemp;
   }
}
```

Codul prezentat mai sus ia în considerare cele două aspecte discutate la crearea unei liste simplu înlănțuite: lista este vidă şi se crează primul nod, sau lista are deja cel puțin un nod şi se adaugă noul nod la sfârşitul listei. Codul nu surprinde însă operațiile de completare a informațiilor suplimentare pentru nodul adăugat.

Reprezentarea grafică a acestei operații este dată în figura următoare:



1.4.2. Inserarea unui nod într-o listă circulară simplu înlănțuită

Inserarea unui nod într-o listă circulară poate fi făcută înaintea sau după un nod identificat printr-o *cheie*. În cadrul acestor operații de inserare trebuie avut în vedere faptul că ordinea efectuării operaților este critică. În caz contrar, se poate produce o distrugere a listei, spații din memoria heap alocate şi pierdute de către program, încercări de a accesa zone de memorie protejate, etc. Inserarea unui nod după un alt nod precizat a cărui adresă se află în *pNodCurent*, de exemplu, se face ca mai sus (rolul variabilei *pListaCirculara* este jucat de *pNodCurent*). În cazul inserării unui nod înaintea altui nod precizat printr-o cheie, în procesul de identificare al nodului înaintea căruia se face inserarea suntem obligați să memorăm şi adresa nodului anterior. Dacă am determinat această adresă (a nodului anterior), problema se transformă într-o *inserare nod după un nod cunoscut*.

1.4.3. Ştergerea unui nod dintr-o listă circulară simplu înlănţuită

Această problemă coincide cu problema ştergerii unui nod care are succesor şi este succesorul altui nod dintr-o listă simplu înlănţuită.

1.5. Liste dublu înlănţuite

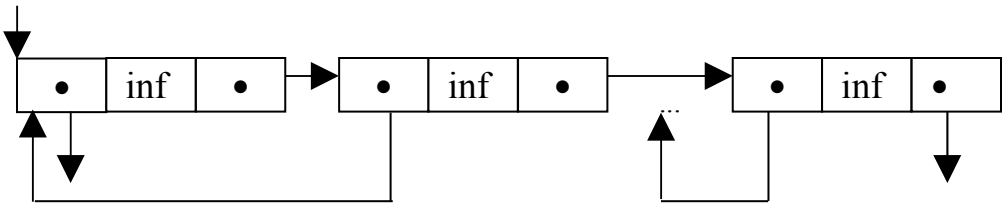
Listele simplu înlănţuite - cât şi cele circulare discutate până acum - au marele dezavantaj că relaţia de ordine dintre noduri este ori de precedenţă ori de succesiune. Cu alte cuvinte, parcurgerea acestor liste se face într-o singură direcţie, totdeauna putându-se identifica cel mult un vecin al unui nod. *Lista dublu înlănţuită* se defineşte în acelaşi mod ca o listă simplu înlănţuită, cu observaţia că pe mulţimea nodurilor definim *două* relaţii de ordine: precedent şi succesor (utilizate simultan). Pentru cele ce urmează vom presupune că nodurile listei dublu înlănţuite au tipul definit ca mai jos:

```
typedef struct _tagNodLD {
    declaraţii
    struct MPI_Nod_LDI      *pPrecedent;
    struct MPI_Nod_LDI      *pElementUrmator;
}MPI_Nod_LDI;
```

Pentru a gestiona o listă dublu înlănţuită vom utiliza variabilele `pInceputLista` şi `pSfarsitLista`, ca şi la listele simplu înlănţuite. Aceste variabile *punctează* spre capetele listei, care se caracterizează prin următoarele:

- primul nod al listei nu are precedent (*`pInceputLista->pPrecedent = NULL`*);
- ultimul nod al listei nu are succesor (*`pSfarsitLista->pSuccesor = NULL`*).

O reprezentare grafică a listelor dublu înlănţuite poate fi dată de figura de mai jos.



În legătură cu listele dublu înlănţuite se pot defini aceleaşi operaţii ca şi în cazul listelor simplu înlănţuite:

1. Crearea unei liste dublu înlănţuite.
2. Accesul la un nod al unei liste dublu înlănţuite.
3. Inserarea unui nod într-o listă dublu înlănţuită.
4. Ştergerea unui nod dintr-o listă dublu înlănţuită.
5. Ştergerea unei liste dublu înlănţuite.

1.5.1. Crearea unei liste dublu înlănţuite

În momentul creării unei liste dublu înlănţuite distingem două situaţii:

a) lista este vidă şi se adaugă primul nod la listă;
b) lista conţine noduri, ***adăugarea*** făcându-se ***după ultimul nod*** (la sfârşitul listei).

Pentru a), ordinea operaţiilor este următoarea (se reiau operaţiile de la liste simplu înlănţuite şi se modifică pentru a fi funcţionale pentru liste dublu înlănţuite):

C.1.1. Se inițializează pointerii *pInceputLista* şi *pSfarsitLista* cu valoarea *NULL*, deoarece la început lista este vidă.

```
pInceputLista = NULL;
pSfarsitLista = NULL;
```

C.1.2. Se rezervă zonă de memorie în memoria heap pentru nodul curent.

C.1.3. Se încarcă nodul curent cu informațiile suplimentare.

C.1.4. Se atribuie pointerului *pInceputLista* şi *pSfarsitLista* adresa din memoria heap a nodului curent (pointerii *pInceputLista* şi *pSfarsitLista* au aceeaşi valoare când lista este vidă (valoarea *NULL*) sau când lista are un singur nod.

C.1.5. Se atribuie valoarea *NULL* pointerului

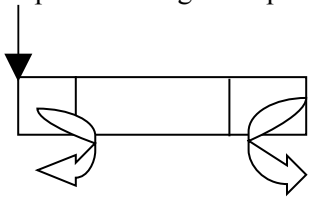
```
C.1.6.   pInceputLista->pElementUrmator
```

C.1.7. Se atribuie valoarea *NULL* pointerului *pInceputLista->pPrecedent*.

Codul ar putea arăta astfel:

```
...
// C1.1.
pInceputLista = NULL;
pSfarsitLista = NULL;

// C1.2.
pTemp = (MPI_Nod*)malloc(sizeof(MPI_Nod));
if (pTemp == NULL)
{
printf ("Memorie insuficienta la crearea listei\n");
exit(1);
}

// C1.3.
// acțiuni specifice de inițializare a datelor membru din nodul listei

// C1.4.
pInceputLista = pTemp;
pSfarsitLista = pTemp;

// C1.5.
pSfarsitLista->pElementUrmator = NULL;
pInceputLista = pSfarsitLista
pInceputLista->pPrecedent = NULL;
```

Dăm mai jos reprezentarea grafică pentru un nod al unei liste dublu înlănțuite:



```
pPrecedent = NULL              pElementUramator = NULL
```

Pentru a accesa un nod, ordinea operațiilor este următoarea:

C.2.1. Se rezervă zonă de memorie în memoria heap pentru nodul curent, *pTemp*.

C.2.2. Se încarcă nodul curent cu informațiile suplimentare.

C.2.3. Se atribuie pointerului *pSfarsitLista->pElementUrmator* adresa din memoria *heap* a nodului creat.

C.2.4. Se atribuie pointerului *pTemp->pPrecedent* valoarea lui *pSfarsitLista*.

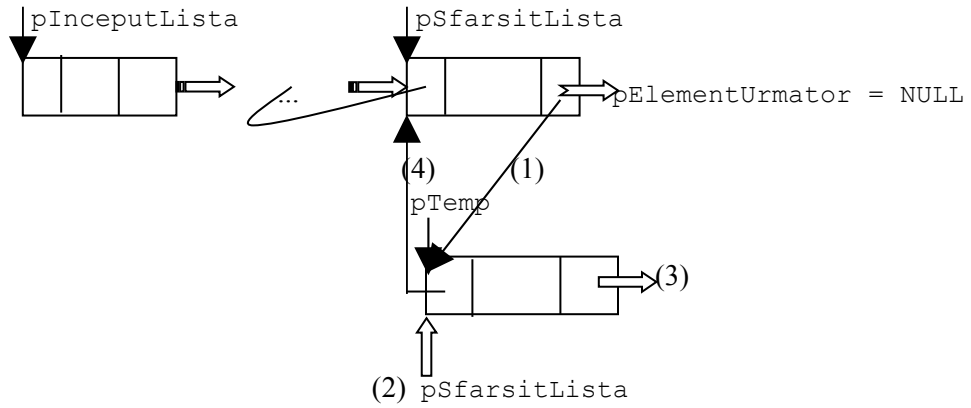
C.2.5. Se atribuie pointerului *pSfarsitLista* adresa din memoria *heap* a nodului creat.

C.2.6. Se atribuie valoarea *NULL* pointerului *pSfarsitLista->pElementUrmator*.

Codul ar putea fi următorul:

```
// C2.1.
pTemp =  (MPI_Nod*)malloc(sizeof(MPI_Nod));
if (pTemp == NULL) {
printf ("Memorie insuficienta la crearea listei\n");
exit(1);
}

// C2.2.
// operații specifice de inițializare a nodului

// C2.3. Se face legătura dintre ultimul nod al listei cu noul nod creat
pSfarsitLista->pElementUrmator = pTemp;

// C.2.4.
pTemp->pPrecedent = pSfarsitLista;

// C2.5.  Noul nod creat va deveni ultimul nod al listei
pSfarsitLista = pTemp;

//C2.6. Acum pSfarsitLista pointează spre noul nod creat care nu are //succesori
pSfaristLista->pElementUrmator = NULL;
```

Operația de adăugare a unui nod la o listă existentă poate fi schematizată ca mai jos:



Ordinea operațiilor care urmează este strictă:

1. (1) *pElementUrmator* ia valoarea lui *pTemp*;
2. (4) *pPrecedent* din nodul alocat ia valoarea variabilei *pSfarsitLista*;
3. (2) noul nod devine ultimul nod al listei, *pSfarsitLista* se schimbă corespunzător;
4. (3) *pElementUrmator* din ultimul nod adăugat ia valoarea *NULL*.

1.5.2. Accesul la un nod al unei liste dublu înlănțuite

Deoarece avem definite două relații de ordine, lista poate fi parcursă în două moduri: de la început spre sfârşit (se va folosi pointerul *pElementUrmator*) sau de la sfârşit spre început (se va folosi pointerul *pPrecedent*). Metoda a fost descrisă la liste simplu înlănțuite. Nu o reluăm.

1.5.3. Inserarea unui nod într-o listă dublu înlănțuită

Distingem următoarele situații:

a) inserare la începutul listei;

b) inserare după sau înaintea unui nod precizat, nod care nu este capăt al listei;

c) inserare la sfârşitul listei.

Observație. Inserarea la sfârşitul listei, c), coincide cu operația de adăugare a unui nod la sfărşitul listei, operație deja descrisă.

1.5.3.1. Inserare la începutul listei

Situația este foarte asemănătoare cu cea întâlnită la liste simplu înlănțuite. Operațiile care se execută în plus sunt cele referitoare la stabilirea corectă a informațiilor pentru pointerul ce implementează relația de precedență, *pPrecedent*.

Reluăm ceea ce am descris pentru listele simplu înlănțuite.

Adresa primului nod al listei (dacă nu este vidă) este păstrată în pointerul *pInceputLista*. Operațiile care trebuiesc efectuate, precum şi ordinea acestora, este descrisă în continuare:

1. alocăm memorie pentru noul nod; adresa se obține - de exemplu - în *pTemp*;
2. pointerul *pTemp->pElementUrmator* va păstra adresa următorului nod, care este în fapt fostul prim nod al listei, deci valoarea lui *pInceputLista*;

```
pTemp->pElementUrmator = pInceputLista;
```

3. pointerul *pInceputLista->pPrecedent* va păstra adresa noului nod creat, *pTemp*;
4. pointerul *pInceputLista* va primi ca valoare adresa noului nod creat

```
pInceputLista = pTemp;
```

6. pointerul *pInceputLista->pPrecedent* va primi drept valoare *NULL* (este noul nod de început al listei).

1.5.3.2. Inserare după sau înaintea unui nod precizat, nod care nu este capăt al listei

Vom descrie numai operația de inserare a unui nod înaintea unui nod precizat. Presupunem că dispunem de următoarele informații: adresa nodului precedent (*pNodAnterior*) şi de adresa nodului succesor (*pNodUrmator*) nodului ce va fi inserat. În acest caz, codul pentru determinarea celor două adrese este mai simplu, pentru că din nodul care satisface condiția cerută putem obține adresa nodului precedent (cu ajutorul pointerului *pPrecedent*). Adresa nodului anterior şi a celui următor se obțin printr-o procedură asemănătoare cu cea descrisă la liste simplu înlănțuite.

Înainte de a face inserarea, situația legăturilor (valorile pointerilor *pPrecedent* şi *pElementUrmator*) din cele două noduri sunt:

```
pNodAnterior->pElementUrmator = pNodUrmator; // succesorul
pNodUrmator->pPrecedent = pNodAnterior;      // precedentul
```

Ordinea operațiilor pentru inserare este:

1. alocarea memoriei pentru noul nod; adresa se păstrează în *pTemp* (presupunem că acțiunea de alocare s-a desfăşurat cu succes);
2. stabilirea precedenței pentru noul nod:

```
pTemp->pPrecedent = pNodAnterior;
```

3. stabilirea nodului succesor pentru noul nod:

```
pTemp->pElementUrmator = pNodSuccesor
```

4. stabilirea nodului succesor pentru nodul anterior:

```
pNodAnterior->pElementUrmator = pTemp;
```

5. stabilirea nodului precedent pentru nodul succesor:

```
pNodUrmator ->pPrecedent = pTemp;
```

Observație. Acest cod poate fi optimizat, varianta de față fiind preferată doar din motive didactice. Inserarea după un nod precizat se tratează exact la fel ca în cazul anterior, deoarece dispunem de adresele nodului anterior şi a celui succesor. Diferența apare din modul de determinare a celor două adrese. *Mai întâi* se obține adresa nodului anterior *şi apoi,* cu ajutorul pointerului *pElementUrmator* din nodul anterior, obținem adresa nodului următor.

1.5.4. Ştergerea unui nod dintr-o listă dublu înlănțuită

După modul de amplasare a nodului care trebuie şters, distingem următoarele cazuri:
(1) ştergerea primului nod al listei;
(2) ştergerea ultimului nod al listei;
(3) ştergerea unui nod care nu este *capăt* al listei.

1.5.4.1. Ştergerea primului nod al listei

Înainte de a efectua ştergerea acestui nod trebuie să ne asigurăm că am păstrat adresa nodului următor. Primul nod al listei este *dat* de valoarea pointerului *pInceputLista*.

Operația de ştergere în acest caz poate fi descrisă astfel:

(1) dacă lista este vidă, operația este terminată;
(2) păstrăm adresa nodului următor:

```
pTemp = pInceputLista->pElementUrmator;
```

(3) eliberăm memoria punctată de *pInceputLista*;
(4) actualizăm valoarea lui *pInceputLista* cu *pTemp* (nodul următor devine primul nod);
(5) noul nod de început al listei nu are precedenti:

```
pInceputLista->pPrecedent = NULL;
```

Observație. Ce se întâmplă dacă lista are exact un singur element? Funcționează corect etapele de mai sus? Analizând această situație constatăm că valoarea pointerului *pInceputLista* va fi *NULL* în etapa (*4*) pentru că valoarea pointerului *pTemp* este *NULL*. Dar valoarea pointerului *pSfarsitLista*, este corectă? Deoarece lista devine vidă, valoarea acestui pointer ar trebui să fie *NULL*. Conform operațiilor de mai sus așa ceva nu se întâmplă. Mai mult, în (*5*) vom obține o eroare datorită faptului că vom încerca să accesăm o zonă de memorie interzisă (adresa *0x00000000*).
Ce este de făcut? Modificăm (*2*) astfel:

> Dacă *pInceputLista->pElementUrmator = NULL*, atunci eliberăm zona de memorie pointată de *pInceputLista*, după care setăm pe *NULL* pointerii ce mențin informația despre capetele listei (*pInceputLista* şi *pSfarsitLista*). Operația se consideră terminată şi nu se mai execută celelalte etape.

1.5.4.2. Ştergerea ultimului nod al unei liste dublu înlănțuite

Dacă lista are un singur nod, această operație coincide cu cea a ştergerii primului nod al listei. Deci vom presupune că lista are cel puțin două noduri. În acest caz ordinea operațiilor poate fi:

(1) păstrăm adresa nodului precedent în *pNodPrecedent*:
```
pNodPrecedent = pSfarsitLista->pPrecedent;
```

(2) eliberăm zona de memorie punctată de *pSfarsitLista*;
(3) reactualizăm valoarea pointerului *pSfarsitLista* cu valoarea pointerului *pNodPrecedent*;
(4) ultimul nod al listei nu are succesor:
```
pSfarsitLista->pElementUrmator = NULL;
```

1.5.4.3. Ştergerea unui nod neterminal al listei

Datorită faptului că nodul nu este terminal (lista are cel puțin trei noduri) operațiile necesare ştergerii acestui nod, punctat de variabila *pTemp*, sunt:

(1) păstrarea adresei nodului precedent în *pNodPrecedent*:
```
pNodPrecedent = pTemp->pPrecedent;
```

(2) păstrarea adresei nodului următor în *pNodUrmător*:
```
pNodUrmator = pTemp->pElementUrmator;
```

(3) eliberarea zonei de memorie punctată de *pTemp*;
(4) refacere legături;
legătura cu nodul precedent:
```
pNodUrmator->pPrecedent = pNodPrecedent;
```

legătura cu nodul următor:

```
pNodPrecedent-pElementUrmator = pNodUrmator;
```

1.5.5. Ştergerea unei liste dublu înlănțuite

Pentru a şterge o listă dublu înlănțuită, se poate aplica în mod iterativ procedeul de ştergere a primului nod (ultimului nod) al listei până când lista devine vidă. Un cod simplu care realizează acelaşi lucru (nu mai reface legăturile după ştergerea unui nod) poate fi:

```
...
MPI_NodLD *pTemp1, *pTemp;
pTemp = pInceputLista;
while (pTemp != NULL)
{
    pTemp1 = pTemp->pElementUrmator;
    delete pTemp;     // eliberare memorie ocupată
    pTemp = pTemp1;
}
...
```

Exercițiu. Rescrieți codul de mai sus fără a folosi variabila temporară `pTemp1`.

Observație. Din punct de vedere metodic, profesorul trebuie să îndrume elevul într-un asemenea mod încât acesta să facă o distincție clară între *definiția formală a unei structuri de date*, *reprezentarea sa grafică (vizuală)* şi *diversele tipuri de implementare*.

2. Grafuri şi arbori

Deşi *arborii* sunt un caz particular de *grafuri*, vom începe cu tratarea arborilor. Vom lucra – în general – cu *grafuri orientate (digrafuri)*. Vom preciza în mod explicit referințele la *grafurile neorientate*. Conform (<16, 21, 30>) *arborii (orientați) sunt grafuri conexe şi fără circuite*. Ca reprezentare, arborii sunt structuri de date de natură recursivă şi dinamică. În acest sens, putem spune că *prin arbore înțelegem o mulțime finită şi nevidă de elemente numite noduri*, `A = {A1,A2,...,An}`, `n` număr natural pozitiv, care satisface proprietățile:

- *există un nod şi numai unul care se numeşte rădăcina arborelui;*
- *celelalte noduri formează submulțimi disjuncte ale lui* `A`*, care formează la rândul lor câte un arbore; arborii respectivi se numesc subarbori ai rădăcinii.*

Într-un arbore există noduri cărora nu le mai corespund subarbori. Un astfel de nod se numeşte *nod terminal* sau *nod frunză*.

O altă noțiune legată de arbori este cea de *nivel*. Rădăcina unui arbore (care se numeşte şi nod *tată*) are nivelul `1`. Dacă un nod are nivelul `n`, atunci descendenții lui (care se mai numesc şi *fii*) au nivelul `n+1`. Dacă pentru fiecare nod subarborii săi sunt ordonați (în sensul rădăcinilor), atunci arborele se numeşte *ordonat*.

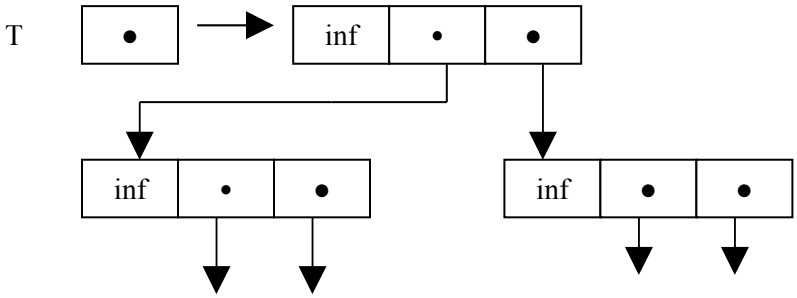
2.1. Arbori binari

Un arbore binar este o mulțime finită de elemente care este vidă sau conține un element numit rădăcină, iar celelalte elemente se împart în **două** submulțimi disjuncte, care fiecare la rândul ei, este un arbore binar. *Una dintre submulțimi se numeşte* subarborele stâng al rădăcinii, *iar cealaltă* subarborele drept. *Arborele binar este* ordonat, *deoarece în fiecare nod*, *subarborele stâng se consideră că* precede *subarborele drept*. *Deci un nod al unui arbore binar are cel mult*

doi fii (descendenți) numiți fiul stâng *şi* fiul drept. Structura ce defineşte un arbore binar este poate fi descrisă astfel:

```
typedef struct _tagArbore {
      declarații
      struct      _tagArbore* pStang;
      struct  _tagArbore*     pDrept;
} MPI_Arbore;
```

Practic, *tipul este acelaşi cu cel al unei liste liniare dublu înlănțuite* dar, pentru a evita unele confuzii, am *mutat* locul pointerilor dintr-un nod (şi am schimbat numele câmpurilor şi variabilelor folosite):



Situația menționată este un exemplu edificator pentru faptul că o structură de date trebuie văzută nu numai ca o colecție de informații organizată într-un anume mod (*simplă*, *compusă*, *array*, *record*, etc.) ci şi împreună cu mulțimea de operații admisă a se efectua asupra ei (asupra arborelui, altele sunt operațiile admise decât cele asupra listelor înlănțuite, stivă, coadă, etc.). În continuare vom insista asupra operațiilor cele mai des folosite asupra arborilor binari:

1. inserarea unui nod frunză;
2. accesul la un nod al unui arbore binar;
3. parcurgerea unui arbore binar;
4. ştergerea unui arbore binar.

Operațiile de inserare şi acces la un nod presupun - ca şi la liste - definirea unui *criteriu de identificare* al unui anumit nod. Gestiunea nodurilor unui arbore binar se realizează cu *ajutorul unei variabile ce punctează spre rădăcina (sub)arborelui.* Notăm această variabilă cu `pRadacina`, definită astfel:

```
MPI_Arbore*          pRadacina;
```

Această variabilă are ca valoare adresa de început a zonei de memorie în care se păstrează rădăcina arborelui. În cazul în care arborele este vid, `pRadacina` are valoarea `NULL`.

2.1.1. Inserarea unui nod frunză într-un arbore binar

Etapele ce trebuie parcurse pentru a realiza această operație sunt:

1. Se alocă zonă de memorie pentru nodul care urmează să se insereze în arbore. Notăm cu `pTemp` pointerul care are ca valoare adresa de început a zonei respective. Dacă alocarea se

face cu succes, se continuă cu etapa următoare. În caz contrar, inserarea nu poate fi efectuată. Operația de inserare se termină cu afişarea unui mesaj de eroare.

2. Se atribuie valori variabilelor ce formează acest nod. Dacă asignările se termină cu succes, se trece la etapa următoare.
3. Se atribuie valoarea *NULL* pointerilor *pStang* şi *pDrept* pentru noul nod punctat de *pTemp*.

```
pTemp->pStang = NULL;
pTemp->pDrept = NULL;
```

4. Unde se inserează noul nod? Dacă *pRadacina* este *NULL* (arbore fiind vid), atunci acest nod va fi primul nod al arborelui şi facem asignarea:

```
pRadacina = pTemp;
```

Procesul se opreşte. *În caz contrar, se determină poziția în care trebuie inserat noul nod.* Presupunem că această adresă este menținută în variabila *pNodTata*. De asemenea, criteriul folosit mai sus va indica dacă inserarea se va face în nodul stâng sau în nodul drept al nodului *pNodTata* sau operația nu poate fi efectuată. În cazul când operația de inserare nu poate efectuată, eliberăm zona de memorie a cărei adresă se află în *pTemp* şi procesul se termină.

5. Dacă inserarea se face în nodul stâng, atunci se face atribuirea (legătura nodului *pNodTata* cu noul nod):

```
pNodTata->pStang = pTemp;
```

şi procesul se termină.

6. Dacă inserarea se face în nodul drept, atunci se face atribuirea (legătura nodului *pNodTata* cu noul nod):

```
pNodTata->pDrept = pTemp;
```

şi procesul se termină.

Etapa 4. este cea mai importantă din cadrul acestui proces. Criteriul de identificare a nodului după care se face inserarea este specific pentru fiecare caz în parte. Informațiile care contribuie la identificarea nodului sunt cele din nodul care se vrea a se insera şi cele existente deja în nodurile alocate. Trebuie reținut că în acest caz se va începe cu cercetarea nodului rădăcină şi că operațiile care se efectuează sunt aceleaşi pentru fiecare nod. Deci acest criteriu de identificare poate fi implementat ca o funcție cu cel puțin doi parametri. Un parametru va conține adresa nodului supus testării – *parametru de intrare* – iar celălalt parametru va conține adresa nodului după care se face inserarea – *parametru de ieşire*. Prototipul funcției ar putea fi:

```
int   identificare(MPI_Arbore* pNodCurent, MPI_Arbore* pNodDeterminat);
```

cu următoarele convenții pentru valoarea de tip *int* returnată:

a) număr strict negativ: se face inserarea în nodul stâng;
b) numar strict pozitiv: se face inserarea în nodul drept;
c) valoarea zero: inserarea nu poate fi efectuată.

2.1.2. Accesul la un nod al unui arbore binar

Accesul la un nod al unui arbore binar presupune existența unui criteriu care să permită determinarea nodului respectiv. Acest lucru a fost discutat imediat anterior la **Etapa 4**.

2.1.3. Parcurgerea unui arbore binar

Sunt cunoscute trei metode (recursive) clasice de parcurgere a unui arbore binar:

(1) în preordine;
(2) în inordine;
(3) în postordine.

Parcurgerea unui arbore binar *în preodine* înseamnă accesul la rădăcina arborelui şi apoi parcurgerea celor doi subarbori, întâi a celui stâng şi apoi a celui drept. Subarborii, fiind la rândul lor arbori binari, se parcurg în acelaşi mod. Parcurgerea unui arbore binar *în inordine* înseamnă parcurgerea mai întâi a subarborelui stâng, apoi accesul la rădăcină şi în continuare parcurgerea subarborelui drept. Cei doi subarbori se parcurg în acelaşi mod. Parcurgerea unui arbore binar *în postordine* înseamnă parcurgerea mai întăi a subarborelui stâng, apoi a subarborelui drept şi în final accesul la rădăcina arborelui. Cei doi subarbori se parcurg în acelaşi mod. Pentru fiecare dintre cele trei metode construim funcţiile `Preordine`, `Inordine` şi `Postordine` care au următorul prototip:

```
void  Preordine(MPI_Arbore* pNod);
void  Inordine(MPI_Arbore* pNod);
void  Postordine(MPI_Arbore* pNod);
```

Pentru descrierea recursivă a lor vom folosi şi funcţia:

```
void Radacina(MPI_Arbore* pNod);
```

prin care vom descrie anumite operaţii specifice rădăcinilor subarborelui. Algoritmul pentru parcurgerea în preordine este descris în continuare. Dacă pointerul spre rădăcină nu este `NULL`, atunci se execută etapele:

a. se apelează funcţia `Radacina` cu valoarea parametrului *pointer spre rădăcină*;
b. fiul stâng devine noua rădăcină şi se apelează funcţia `Preordine` cu valoarea parametrului *pointer spre noua rădăcină* (astfel se parcurge în preordine subarborele stâng);
c. fiul drept devine noua rădăcină şi se apelează funcţia `Preordine` cu valoarea parametrului *pointer spre noua rădăcină*.

Codul funcţiei este:

```
void Preordine(MPI_Arbore* pNod)
{
if (pNod != NULL) {
      Radacina(pNod);
      Preordine(pNod->pStang);      //  parcurge  subarborele  stâng  în
preordine
      Preordine(pNod->pDrept);      //  parcurge  subarborele  drept  în
preordine
   }
}
```

Codurile pentru funcţiile *Inordine* şi *Postordine* se construiesc analog având în vedere definiţiile acestora.

2.1.4. Ştergerea unui arbore binar

Pentru a şterge un arbore binar este necesară parcurgerea lui şi ştergerea fiecărui nod. Arborele va fi parcurs în postordine (rădăcina arborelui trebuie ştearsă ultima). Codul pentru această funcţie este:

```
void StergArbore(MPI_Arbore* pNod)
{
      if (pNod != NULL) {
            StergArbore(pNod->pStang);
            StergArbore(pNod->pDrept);
            EliberezMemorie(pNod);
         }
}
```

unde:

```
void EliberezMemorie(MPI_Arbore* pNod)
{
      free(pNod->pStang);
      free(pNod->pDrept);
      free(pNod);
}
```

Observaţie. Nu au fost puse condiţii asupra valorii pointerilor înainte de a elibera memoria punctată de ei.

2.2. Grafuri

Principalele exemple se referă la parcurgerea grafurilor în *adâncime* sau în *lăţime*. În urma utilizării acestor algoritmi principali se pot rezolva numeroase probleme privind teoria generală a grafurilor (inclusiv reţele de calculatoare). A se consulta **Capitolul 4**, ***Secţiunea 8***.

3. Sortare şi căutare

Pentru o introducere în problematica vastă a acestui domeniu (inclusiv justificarea studiului său intensiv) , recomandăm cărţile <14, 23, 24, 33> . Deşi din punct de vedere practic *sortarea externă* este mult mai importantă, vom insista, din considerente didactice, asupra *sortării interne*. Aceasta înseamnă că algoritmii *în sine*, ideile importante de rezolvare a problemelor reale, complexitatea teoretică, primează asupra considerentelor legate de spaţiul (resurse hard) şi de timpul efectiv (măsurat în secunde) de rezolvare. Orice aplicaţie care presupune memorarea unor date şi regăsirea ulterioară a celor care satisfac un anumit criteriu, necesită mecanisme eficiente pentru *localizarea* lor. Acesta ar fi enunţul cel mai general al unei *probleme de căutare*. Rezolvarea problemei depinde în mod esenţial de modul în care sunt memorate datele. Dacă, în funcţie de criteriul după care se vor face căutările, datele sunt memorate într-o anumită ordine, algoritmul de căutare poate să fie implementat mai eficient. Dacă pentru o aceeaşi aplicaţie sunt necesare căutări după criterii diferite, lucrurile devin puţin mai complicate. De exemplu, dacă pentru un dicţionar de termeni este necesar să se facă atât regăsirea definiţiei unui termen după

numele acestuia, cât şi regăsirea tuturor termenilor care se referă la un anumit subiect, stabilirea ordinii utile nu mai este atât de simplă. Să presupunem însă pentru început că dorim să rezolvăm problema *ordonării (crescătoare*) a unui *vector* `A`. Pornind de la acest considerent vom prezenta diverse *metode de sortare*. În funcție de locul în care sunt păstrate elementele vectorului `A` în timpul prelucrării, distingem două tipuri de sortări:

a) *sortare internă*: elementele lui `A` sunt păstrate în memoria internă a calculatorului;

b) *sortare externă*: elementele lui `A` sunt păstrate pe un suport extern.

Metodele de sortare vor diferi în funcție de tipul sortării (internă sau externă). De asemenea, modul de soluționare a problemelor care presupun regăsirea datelor este puternic influențat de suportul de memorare al informației. În cazul sortării interne, există o multitudine de strategii de sortare, fiecare având avantajele şi dezavantajele sale care sunt analizate în funcție de diverse criterii:

- memoria ocupată;
- număr de comparații;
- număr de deplasare a elementelor;
- timp de execuție.

În cazul informațiilor aflate în memoria internă se pune problema dacă acestea sunt memorate în *structuri statice* sau *structuri dinamice.* Algoritmii (imperativi, ne-paraleli) de sortare internă pot fi împărțiți în două mari categorii : **algoritmi *banali*** (timp de lucru $O(n^2)$ sau chiar mai mare), care au însă marea calitate că sunt uşor de înțeles şi **algoritmi *performanți*** **(**$O(n\log n)$ sau mai mic)**,** care au însă defectul (din punctul de vedere al unui profesor de gimnaziu sau chiar de liceu) că fac apel la cunoştințe matematice, de specialitate şi chiar intelectuale mult prea complexe. În ceea ce priveşte *căutarea,* am adoptat aceeaşi tactică de natură didactică, fără a avea pretenția de a epuiza subiectul în sine (o tratare exhaustivă presupune familiarizarea cititorului, la nivel matematic şi informatic cu, de exemplu, domeniul *recunoaşterii formelor*).

3.1. Algoritmi clasici de sortare, de complexitate timp $O(n^2)$ şi mai mare

Indiferent dacă implementarea structurii de date aleasă pentru memorarea mulțimii `A` (ceea ce mai sus a fost numit *vector*) care va fi sortată este bazată pe ceva static (structura *array*) sau dinamic (structura *pointer*), descrierea în pseudocod a algoritmilor va fi orientată spre sublinierea ideii generale de realizare (a *metodei*) şi nu pe detaliile de implementare. Se poate consulta şi ***Capitolul 1***, în care sunt prezentate şi câteva detalii relative la corectitudinea şi terminarea algoritmilor.

3.1.1. Căutare

Problemă. Să se determine *apartenența* unui element la un şir ordonat crescător.

Soluție. A fost în întregime prezentată în **Capitol 1**.

3.1.2. Sortare clasică

Problemă. Fie „la intrare” o colecție de obiecte, nu neapărat distincte. Să se furnizeze „la iesire” aceeaşi colecție, eventual sub o altă formă, care să satisfacă anumite criterii (anterior precizate). Înainte de prezentarea *soluțiilor generale* (mai mult sau mai puțin performante), un exemplu poate fi util.

Exemplu. Un vector de dimensiune `n` (de ordinul milioanelor, să zicem), conține toate numerele naturale de la `0` la `n` (cu excepția unuia, bineînțeles). Să se determine numărul care lipseşte.

Iată o primă soluție:

- calculăm suma primelor n numere naturale cu formula `n(n + 1)/2`;
- printr-o parcurgere secvențială, calculăm suma elementelor vectorului;

- diferența celor două sume este numărul căutat.

Complexitatea este evident liniară pentru o implementare corectă.

O altă idee de rezolvare este dată în secvența de program **Pascal** care urmează.

```
m := 0;
for i:= 1 to n do
                m : = m + a [i] -i;
```

Numărul căutat va fi *m*. Vă invităm să găsiți (implementați) şi alți algoritmi, rezonabili ca ordin de complexitate.

3.1.2.1. Sortarea prin inserție directă

Vom începe această subsecțiune cu câteva exemple.

Exemplul 1. Un vector de dimensiune *n* (de ordinul milioanelor), conține numere naturale care se pot repeta. Să se specifice dacă există în şir un număr care se repetă de mai mult de *[n/2]* (partea întreagă inferioară) ori şi care este acesta. Dacă considerăm inițializate corespunzător (cu zero) variabilele *numar* şi *aparitii* atunci secvența de cod **Pascal** poate fi :

```
Pentru i de la 1 la n
     Dacă a [i] > numar atunci
           aparitii := aparitii + 1
     altfel
           Dacă aparitii >1 atunci
                 aparitii :=  aparitii -1
           altfel
                 numar :=  a [i] ;
```

La sfârşitul parcurgerii şirului vom obține în *numar* numărul căutat, dacă *ap >1*, sau *aparitii > 1* dacă numărul cerut nu se află în şir.

Exemplul 2. Să se implementeze în limbajul **Pascal** algoritmul de calcul a valorii minime dintr-un şir. Datele de intrare se vor citi dintr-un *fişier text*. Să presupunem, din motive metodice, că enunțul anterior reprezintă o *temă pentru acasă* şi că una dintre rezolvările posibile este (chiar dacă exemplul mai este discutat în lucrare):

```
program minim;
      Programul determina elementul minim dintr-un sir
var a : array [1..15]  of byte;
    min,n,i,k : byte;
    f : text;
begin
   assign(f,'sir.txt');
   reset(f);
   i := 0;
   repeat
      i := i + 1;
      read(f,a [i] );
   until eoln(f);
   close(f);
   n := i;
   write(' Sirul : ');
   for i = 1 to n do
       write(a i ,',');
```

```
   writeln(#8,'.');
   min := a [1] ;
   k := 1;
   for i := 2 to n do
       if a [i] < min then
          begin
             min := a [i] ;
             k := i;
          end;
   writeln(' Elementul minim este ',min,' si se afla pe pozitia ',k);
   readln;
end.
```

Un posibil dialog cu clasa – odată ce programul a fost scris pe tablă – şi care are drept scop verificarea temei, este:

Întrebările profesorului	**Răspunsurile (corecte ale) elevilor**
1. Ce reprezintă `a:array [1..15]` ?	1. Un tablou unidimensional cu maximum *15* elemente.
2. Ce înseamnă specificația **of byte**?	2. Elementele tabloului vor fi de tip byte, care este un tip întreg.
3. Ce alte tipuri întregi mai cunoaşteți?	3. Tipurile: *shortint*, *word*, *integer*, *longint*.
4. De unde se vor introduce valorile şirului în vectorul *a*?	4. Din fişierul text *sir.txt*.
5. Cât se foloseşte efectiv din tablou ?	5. Numărul de elemente ale şirului nu este cunoscut. El va fi determinat după terminarea citirii din fişier şi va fi memorat în variabila *n*.
6. Există o limită pentru prelucrarea propusă ?	6. Da. Limita este impusă de rezervarea memoriei făcută la declararea tabloului *a* şi *n* va putea lua valoarea maximum *15*.
7. Cum se vor plasa elementele "citite" în zona de memorie rezervată lor ?	7. Primele elemente ale tabloului vor fi ocupate de termenii şirului, iar restul vor rămâne nefolosite.
8. Cum ați descrie în cuvinte (limbaj natural) algoritmul anterior?	8. Am considerat că primul element este cel mai mic şi l-am memorat în variabila *min*; apoi începând cu al doilea element şi până la sfârşitul şirului, dacă întâlnim un element mai mic decât *min* îl reținem pe acesta ca element minim.

Revenind la problema sortării, *sortarea prin inserție directă* se poate descrie prin următoarea idee generală : pentru a sorta (crescător) un vector (şir) *a[1..n]*, se află mai întâi minimul subşirului *a[i..n]*, apoi se deplasează acesta pe poziția *i* (prima din subşir); ceea ce am descris mai sus se repetă pentru *i* luând valori între *1* şi *n – 1*.

3.1.2.2. Sortarea cu bule

Se poate consulta **Capitolul 1** (şi <14, 23, 24, 33>).

3.1.2.3. Sortarea prin selecție

Se poate consulta **Capitolul 1** (şi <14, 23, 24, 33>).

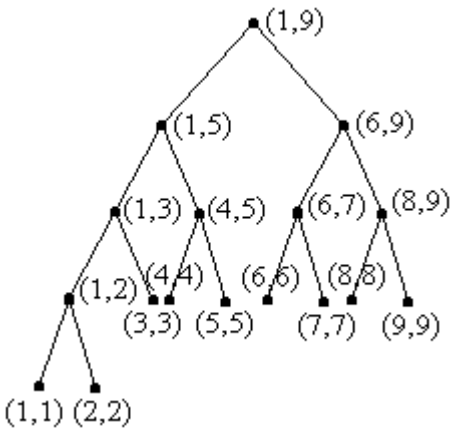
3.1.2.4. Sortarea prin interclasare

Fie secvența $a_1, a_2, \ldots, a_n$ $(n \geq 2)$. Aplicând metoda *divide et impera*, se împarte şirul în două subşiruri $a_1, a_2, \ldots, a_m$, respectiv $a_{m+1}, a_{m+2}, \ldots, a_n$ cu $m = (n+1)$ div 2. Procedeul se repetă pentru subşirurile $a_p, a_{p+1}, \ldots, a_q$ $(m = (p, q)$ div $2)$ până când se obțin subşiruri de lungime 1 (şir deja sortat). Dintre subşirurile sortate, se obțin prin *interclasare* alte subşiruri formate din elementele a două câte două subşiruri, până la obținerea şirului sortat, de lungime n. Vom începe cu un exemplu.

Exemplu. Fie şirul (secvența, vectorul, lista, etc.) 7, 1, 9, 4, 3, 1, 5, 2, 6, de lungime 9. El va fi împărțit în două subşiruri:

- 7, 1, 9, 4, 3, primul subşir, identificat prin capetele sale (1,5);
- 1, 5, 2, 6, al doilea subşir, identificat prin (6,9).

Subşirul (1,5) se împarte în subşirurile (1,3) şi (4,5), dintre care subşirul (1,3) se mai divide în (1,2) şi (3,3). Subşirurile (1,2) şi (4,5) se impart în subşiruri de lungime unu. Subşirul (6,9) se împarte în subşirurile (6,7) şi (8,9), care la rândul lor se împart în subşiruri de lungime unu. Acestei secvențe de divizări i se poate asocia un arbore binar care are rădăcina *marcată* cu (1,9) (capetele şirului inițial) şi pentru fiecare nod marcat cu (p,q), marcăm succesorii săi stâng şi drept cu (p,m) şi respectiv (m+1,q) până când q-p= 0. Mărcile nodurilor reprezintă capetele subşirurilor obținute în etape succesive de divizare. Nodurile terminale sunt marcate cu capetele subşirurilor de lungime unu. Reprezentarea grafică se numeşte *arbore de căutare* :



Subşirurile terminale sunt sortate. Se interclasează apoi şirurile terminale obținându-se noi şiruri (în locul şirurilor *părinte* ale acestora) care vor fi ulterior ordonate. Aplicarea succesivă a procedurii de interclasare se face printr-o parcurgere în *inordine* a arborelui binar asociat. Programul **Pascal** care implementează acest algoritm va fi prezentat în întregime în continuare, el reprezentînd un exemplu ilustrativ complet pentru metoda în cauză.

Fişierul de **Intrare** *sir.txt* va conține:

```
7 1 9 4 3 1 5 2 6
```

Ieşirea va fi de forma:

- şirul inițial : 7, 1, 9, 4, 3, 1, 5, 2, 6

- şirul sortat : *1, 1, 2, 3, 4, 5, 6, 7, 9*

Programul sursă (**Metoda**):

```
  program sortare_prin_interclasare;
  var a : array [1..20]  of byte;
      n,i : byte;
      f : text;
      procedure sortint(p,q:byte);       {   procedura de sortare prin
interclasare }
      var m : byte;
          procedure intercl(u,w,v:byte); {   procedura de interclasare
şiruri ordonate }
          var i,j,k : byte;
              b : array [1..20]  of byte;
          begin
             k := 0;
             i := u;
             j := w +1;
             repeat
                k := k +1;
                 if a [i] < a [j]  then      {  alege elementul cel mai
mic din cele două subşiruri }
                   begin
                      b [k] := a [i] ;
                      i := i +1;
                   end
                else
                   begin
                      b [k] := a [j] ;
                      j := j +1;
                   end
              until (i > w) or (j > v); {  până când unul dintre şiruri
se termină }
             if i > w then
                for i = j to v do
                    begin
                       k := k +1;
                       b [k] := a [i] ;     {      se   completează   cu
elementele din şirul neterminat }
                    end
             else
                for j = i to w do
                    begin
                       k := k +1;
                       b [k] := a [j] ;
                    end;
                  for  i  =  1  to  k  do{   se scrie şirul obţinut prin
interclasare peste subşirurile sursă }
                   a [u+i] -1 := b [i] ;
          end;
      begin      {  procedură de sortare prin interclasare  }
         if p < q then       {  se aplică metoda "divide et impera"  }
```

```
            begin
               m := (p + q) div 2; {  împărţire problemă  }
               sortint(p,m); {  sortare subproblema întâi }
              sortint(m +1,q);      {  sortare subproblema a doua }
               intercl(p,m,q);      {  combinare soluţii }
            end;
       end;
begin                    {  programul principal  }
      assign(f,'sir.txt');
      reset(f);
      n := 0;
      repeat             {  citire şir iniţial  }
         n := n +1;
         read(f,a [n])
      until eoln(f);
      close(f);
      write(' Sirul initial : ');   {  afişare şir iniţial  }
      for i = 1 to n do
          write(a i ,',');
          writeln(#8,'.');
      sortint(1,n);           {  apel procedură de sortare  }
      write(' Sirul sortat  : ');
      for i = 1 to n do       {  afişare şir sortat  }
          write(a i ,',');
        writeln(#8,'.');
end.
```

3.1.2.5. Sortarea *shell*

Pentru a mai alunga monotonia prezentărilor anterioare, metoda în cauză va fi descrisă (incomplet, fără cod sau discuţii precise legate de corectitudine, complexitate, etc.) sub forma unui *proiect didactic*.

PROIECT DE TEHNOLOGIE DIDACTICĂ (este bine să se consulte şi Capitolul 4)

Data:-.
Clasa: a X-a.
Profesor:-.
Disciplina: Algoritmică şi programare.
Subiectul lecţiei: Sortarea tablourilor prin **metoda *shell*.**
Scopul lecţiei: Introducerea unei noi metode de sortare a tablourilor cu ajutorul *arborilor de sortare şi căutare*.
Tipul de lecţie: Mixtă.
Obiective operaţionale:
Elevii trebuie să fie capabili :

- să deosebească această metodă de cele prezentate anterior;
- să implementeze corect metoda *shell*;
- să observe avantajele şi dezavantajele faţă de celelalte metode.

Metode folosite: expunerea, conversaţia, exerciţiul.
Mijloace de realizare: convenţionale.
DESFĂŞURAREA LECŢIEI
Punctul 1.

Etapa: Moment organizatoric.
Timp: 2'.
Activitatea desfăşurată de profesor:
Se face prezența şi se verifică existența celor necesare începerii orei.
Punctul 2.
Etapa: Verificarea cunoştințelor.
Timp: 15'.
Activitatea desfăşurată de profesor:
Verificarea temelor date elevilor pentru acasă.
Verificarea cunoştințelor din lecția precedentă cu tema: *Sortarea tablourilor prin metoda inserției directe* prin întrebări:

- În ce constă această metodă?
- Avantajele şi dezavantajele față de alte metode implementate anterior.
- Reluarea metodei cu ajutorul unui alt exemplu.

Metoda: Verificare orală.
Punctul 3.
Etapa: Trecerea la lecția nouă.
Timp: 2'.
Activitatea desfăşurată de profesor: Anunțarea şi scrierea pe tablă a titlului lecției: *Sortarea tabolourilor prin metoda shell*.
Punctul 4.
Etapa: Predarea noilor cunoştințe.
Timp: 2'.
Activitatea desfăşurată de profesor:
Pornind de la un exemplu de la metoda inserției directe, se poate observa că această metodă se poate îmbunătăți ajungându-se la shell-sort. Ideea este următoarea:

- se împarte la început tabloul în grupe de câte două elemente care se raportează separat: de exemplu, dacă dimensiunea este `16`, vom forma grupele `(1,9), (2,10)`…;
- se formează grupe de câte patru elemente din cele sortate anterior;
- procedeul continuă până când se ajunge la tabloul în cele din urmă ordonat crescător.

Metoda: Expunerea.
Punctul 5.
Etapa: Fixarea noilor cunoştințe.
Timp: 3'.
Activitatea desfăşurată de profesor:
Fiecare *grupă* fiind sortată separat, se observă ca elementele mari *se deplasează la dreapta*. Complexitatea timp în cazul cel mai nefavorabil al algoritmului shell-sort este n^3. Prin urmare, profesorul formulează întrebări şi exerciții în legătură cu:

- metoda de sortare *shell*;
- compararea cu celelalte metode.

Exemplifică noțiunile introduse, la tablă.
Metoda: Conversația.
Punctul 6.
Etapa: Precizarea temei pentru acasă.
Timp: 3'.
Activitatea desfăşurată de profesor:

- Exemplifică pas cu pas sortarea prin noua metodă pe un exemplu concret.
- Solicită implementarea algoritmului în **Pascal**.

3.1.3. Sortare rapidă

În acestă secțiune ne vom axa pe prezentarea succintă doar a două metode de complexitate `O(n log n)`. Enunțul problemei este acelaşi cu cel din secțiunea precedentă.

3.1.3.1. Sortarea *quicksort*

Ceea ce urmează este doar o variantă (datorată autorilor) a metodei descrise de E. W. Dijkstra.

PROIECT DE TEHNOLOGIE DIDACTICĂ

Disciplina: Informatică.
Clasa: a IX-a.
Profesor:-.
Data:-.
Tema: Căutare şi sortare. *Metoda quiksort*.
Tipul de lecție: Predare.
Metode didactice: Expunere, exerciții.
Mijloace de învățare: Manuale, culegeri.
Material bibliografic: (<33>).
Obiective operaționale.

- Să înțeleagă metoda predată.
- Să poată aplica metoda pentru un exemplu concret.
- Să poată face deosebire față de celelalte metode.
- Să poată identifica cu claritate metoda.

Nr. crt.	Etapa	Durata	Conținut	Metoda
1.	Organizarea clasei	2'-3'	Verificare prezență elevi. Verific dacă existămaterialele necesare.	
2.	Verificarea temei pentru acasă	5'-10'	Dacă au existat dificultăți la rezolvarea temei, scot la tablă pe cineva. Dacă ora anterioară s-a dat test, tema o va reprezenta testul.	Frontală.
3.	Verificarea cunoştințelor anterioare	10'-15'	Se pot scoate `2-3` elevi la tablă sau se poate da o scurtă lucrare.	Conversație. Exerciții.
4.	Actualizarea cunoştințelor	5'-10'	Facem legătura cu lecția anterioară. Se pot pune calificative.	Frontală.
5.	Anunțarea titlului lecției	1'	Titlul scris pe tablă. Se enunță obiectivele lecției.	

Metoda *quicksort* prezentată mai sus foloseşte informații memorate în structuri statice. Pentru a simplifica expunerea, să considerăm că avem un singur criteriu de căutare. Considerăm că datele sunt memorate în vectori de înregistrări (*array*) şi căutarea datelor se face pe baza valorii unui câmp. În mod tradițional acest câmp se numeşte *cheia înregistrării* (pe parcursul lucrării am mai folosit acest termen). Tipurile de date utilizate pentru câmpurile cheie sunt alese astfel încât asupra lor poate fi definită o relație de ordine. În cazul în care componentele vectorului sunt memorate în ordinea crescătoare sau descrescătoare a cheilor, atunci regăsirea unei înregistrări cu cheie dată se poate face mai rapid decât prin parcurgerea secvențială a tuturor elementelor vectorului. Mai complicat este să ordonăm (crescător) elementelor vectorului, pe baza valorilor

cheilor. Reformulând problemare de sortare, aceasta înseamnă să transformăm (*pe loc*) un vector *A* într-un vector pentru care:

$$A[i].cheie <= A[i+1].cheie, 1 <= 1 < n-1, \qquad (1)$$

utilizând o cantitate minimă de memorie suplimentară. După cum am mai precizat, *nu există un cel mai bun algoritm universal de sortare.* Alegerea algoritmului potrivit pentru o aplicație dată trebuie să țină seama de numărul de elemente ce trebuie sortate, de complexitatea operațiilor de schimbare între ele a valorilor a două înregistrări şi de cât de *neordonate* sunt elementele vectorului. În cazul *quicksort*-ului, folosim vectorul `A` cu elemente numere întregi. Prin parcurgerea vectorului pornind de la ambele capete (pe rând) şi interschimbarea elementelor care nu sunt în relația cerută, se împarte vectorul în două părți, nu neapărat de lungime egală, cu proprietatea că toate elementele din prima parte sunt mai mici (sau mai mari, în cazul sortării descrescătoare) decât toate elementele din cea de-a doua parte. Unul dintre subvectori este memorat (prin indicii de început şi sfârşit), iar cu cel rămas se procedează analog. Subşirurile memorate sunt prelucrate apoi pe rând în acelaşi mod (recursiv), în ordinea inversă a memorării lor.

Aplicând algoritmul, condiția de ordonare (1) poate fi rescrisă sub forma:

Pentruorice element $A[K]$ din vector (denumit *pivot*) este îndeplinită condiția:

$$i < K \Rightarrow A[i] \le A[K] \text{ şi}$$

$$K < j \Rightarrow A[K] \le A[j]$$

Sfpentru

Ca şi la *sortarea cu bule*, se verifică dacă la un moment dat este îndeplinită condiția corespunzătore, în caz contrar efectuându-se corecția necesară, printr-o inversare:

Dacă (găsim o pereche de valori $(i, i+1)$ pentru care)

$$A[i].cheie > A[i+1].cheie \qquad (2)$$

atunci

vom schimba între ele cele două elemente.

Sfdacă

Presupunem că a fost aleasă ca *poziție* `pivot (K),` cea din *mijlocul* vectorului. Algoritmul următor asigură îndeplinirea condiției anterioare pentru această poziție. Limitele între care variază indicii elementelor din vector sunt *primul* şi *ultimul*.

QuickSort (A, primul, ultimul)

$i \Leftarrow primul$

$j \Leftarrow ultimul$

$pivot = A[(primul + iltimul)/2]$

repetă

- poziționează I după elementele cu chei<pivot.cheie
- poziționează j înaintea elementelor cu chei>pivot.cheie

Dacă $(i < j)$ ***atunci***

```
  schimbă (A[i], A[j])
Dacă (i <= j) atunci
{
      i := i + 1
      j := j +1
Pânăcând (i >= j)
}
```

Exemplu. Fie şirul valorilor *9,2,4,10,3* şi considerăm că poziția inițială a pivotului este $K = 3$. Evoluția algoritmului produce următoarele transformări:

$$\text{primul} = 1 \Rightarrow i = 1$$

$$\Rightarrow K = 3$$

$$\text{ultimul} = 5 \Rightarrow j = 5$$

```
1)  9 2 4 10 3
2)  3 2 4 10 9    I=2, j=4, K=3
3)  3 2 4 10 9    I=3, j=3, K=3
```

Se observă că, deşi pentru $K = 3$ condiția este îndeplinită, şirul nu este încă ordonat. Pentru $K = 2$ şi $K = 4$ condiția nu mai este îndeplinită. Pentru a corecta aceste "greşeli" algoritmul trebuie să fie aplicat din nou, atât la stânga, cât şi la dreapta pivotului. Nu este necesar să se caute dincolo de vechea valoare pentru *K* (toate valorile aflate "la dreapta" acestei poziții sunt sigur mai mari decât valoarea aflată pe noua poziție de referință). Mai mult, este suficient să se caute numai până la ultima valoare pentru *j*.

$$\text{primul} = 1, \text{ultimul} = 3 \Rightarrow i = 1, j = 3, K = 2$$

```
1)  3 2 4
2)  2 3 4    i = 1, j = 2, K = 2
3)  2 3 4    i = 2, j = 2, K = 2
```

Toate cele de mai sus pot fi încadrate în *Punctul 6* al *Planului de lecție*. Am mai putea introduce şi *Punctul 7*, *Fixarea cunoştințelor,* destinat prezentării unei aplicații practice sau revenirii asupra unor cunoştințe teoretice esențiale, deja predate. Evaluarea procesului de învățare va rezulta prin supravegherea activității depuse şi constatarea dificultăților în asimilarea cunoştințelor şi rezolvarea acestor dificultăți. Cu cât se insistă mai mult pe punctele *problematice* cu atât rezultatul evaluării va fi mai bun. Iată şi un exemplu aproape complet de implementare **Pascal** a ordonării rapide:

```
procedure QuickSort (var px:vector; primul, ultimul:integer);
var
   I,j:integer;
   temp:element;
begin
   I:=primul;
   I:=ultimul;
   temp:=px[(primul+ultimul) div 2];
   repeat
     while (px[I].cheie<temp.cheie)do
      I:=I+1;
     while (px[j].cheie>temp.cheie)do
```

```
      j:=j-1;
     if (I<j) then
      schimbă (px[I],px[j]);
     if (I<=j) then
     begin
        j:=j-1;
    I:=I+1;
     end;
   until (I>=j);
   if (primul<j) then QuickSort (px,primul,j);
   if (I<ultimul) then QuickSort (px,I,ultimul);
end;
```

În implementarea considerată mai sus, elementul de refinţă a fost ales *mijlocul* vectorului. Se pot obţine soluţii de *accelerare* a algoritmului dacă se face o alegere mai inteligentă a elementelui de referinţă. Se poate arăta că pentru acest algoritm complexitatea medie este $O(n \log n)$. În cazul cel mai defavorabil şi acest algoritm este, totuşi, de complexitate $O(n^2)$. Dăm şi o variantă nerecursivă :

```
procedure QuickSort;
begin;
    verf:=0;
    push (1); push (N);
    repeat
      ultim:=pop; prim:=pop;
      repeat
         i:=prim, j:=ultim;
         mijloc:=A[(prim+ultim) div 2];
         repeat
            while A[i]<mijloc do i:=i+1;
            while A[j]>mijloc do j:=j-1;
            if i<=j then
            begin
                swap (i,j);
                i:=i+1;
                j:=j+1;
            end;
         until i>j;
         if i< ultim then
            begin
                                push (i);
                                push (ultim);
                    end;
         ultim:=j;
      until prim > ultim;
  until vîrf = 0;
end;
```

Un alt mod de implementare este cel bazat pe metoda *divide et impera.* Aplicând metoda *divide et impera* vom împărţi şirul în două subşiruri cărora le vom aplica acelaşi algoritm de

divizare până când subşirurile obținute vor avea lungimea *1* (şi vor fi ordonate). Soluțiile parțiale fiind memorate tot în acelaşi şir, operația de combinare a soluțiilor parțiale este deja efectuată. Subliniem că în acest algoritm **procedurile** *Prelucrare* şi *ObtinSolutieFinala* sunt vide. Procedura *Divide* are la bază ideea divizării subşirului *(p,q)* prin plasarea primului element din subşir pe poziția sa finală în subşirul sortat, în aşa fel încât toate elementele aflate în stânga sa să fie mai mici decât acesta, iar cele aflate la dreapta - mai mari sau egale cu acesta. Această divizare se obține astfel:

- memorăm capătul stâng într-o variabilă *i:= p*;
- memorăm capătul drept într-o variabilă *j: =q*;
- marcăm capătul drept al subşirului (prin valoarea *true* a unei variabile booleene *t*);
- dacă *a [i] >a [j]* schimbăm cele două elemente între ele, marcăm celălalt capăt al şirului *(t:=not t)* şi în funcție de capătul marcat incrementăm pe *i* (dacă *t* este adevărat) sau decrementăm pe *j* (în caz contrar);
- repetăm pasul anterior până când *i >= j*.

Procedura **Pascal** care implementează algoritmul de sortare rapidă *quicksort*, obținut prin metoda *divide et impera* este:

```
  procedure quicksort(p,q:byte);   { se aplică metoda "divide et
impera" }
       var i,j,k : byte;
           t : boolean;
       begin
         if p < q then   { test de ieşire din apelul recursiv }
             begin
              i := p;
              j := q;
              t := true; { marcarea capătului drept al şirului}
                repeat
                   if a [i] > a [j]  then
                      begin         {    interchimbare elemente de pe
poziţiile i şi j  }
                          k := a [i] ;
                          a [i] := a [j] ;
                          a [j] := k;
                      t := not t;    { schimbare marcă capăt şir }
                      end;
                   if t then
                      j := j -1     {  decrementare  indice capăt drept
}
                   else
                      i := i +1;    {  incrementare  indice capăt stâng
}
                until i = j;        {   până când cei doi indici sunt
egali  }
                if i =p then  {   dacă cei doi indici se întâlnesc în
capătul stâng }
                   quicksort(p +1,q)
                else
```

```
                    if i=q then{   dacă cei doi indici se întâlnesc în
capătul drept }
                       quicksort(p,q -1)
                    else
                       begin          {  dacă cei doi indici se întâlnesc
în interiorul şirului }
                          quicksort(p, i-1);
                          quicksort(i +1,q);
                       end;
              end;
  end;
```

Să facem şi câteva considerente de complexitate. În cazul cel mai defavorabil, când vectorul este inițial ordonat, se fac $n-1$ apeluri succesive ale procedurii *quicksort*, cu parametrii $(1,n)$, $(1,n-1),...,(1,2)$, dacă vectorul este inițial ordonat descrescător sau cu $(1,n)$, $(2,n),...,(n-1,n)$, dacă vectorul este inițial ordonat crescător. La fiecare apel al procedurii *quicksort* se efectuează $i-1$ (respectiv $n-i-1$) operatii elementare pentru divizarea intervalului. În total sunt $n-1+ n-2 + ... 1= n\cdot (n-1)/2$ operații elementare. În cazul cel mai defavorabil, complexitatea algoritmului este deci $O(n^2)$. Să analizăm comportarea algoritmului *în medie* (<14>):

- considerăm că orice permutare a elementelor vectorului are aceeaşi probabilitate de apariție şi notăm cu T_n numărul de operații elementare efectuate pentru a sorta n elemente;

- probabilitatea ca un element al vectorului să fie plasat pe poziția k în vectorul ordonat, este de $\frac{1}{n}$. Observăm că:

$$T_n = \begin{cases} 0 & \text{dacă} \quad n=0, \quad n=1 \\ \frac{1}{n}\left(\sum_{k=1}^{n}(T_{k-1}+T_{n-k})+n-1\right) & \text{dacă} \quad n>1 \end{cases}$$

Deci numărul de operații elementare necesare ordonării unui şir de lungime n notat T_n, se constituie din cele $n-1$ operații necesare determinării poziției k a primului element în vectorul ordonat şi din cele $T_{k-1} + T_{n-k}$ operații elementare necesare ordonării subşirului stâng, respectiv drept. Din relația de recurență rezultă:

$nT_n = T_0 + T_1 + ... + T_{n-1} = T_{n-1} + ... + T_1 + T_0 = n-1$

$$nT_n = n(n-1) + 2\sum_{k=1}^{n} T_{k-1}$$

Trecând în relația de mai sus pe n în $n-1$ şi scăzând cele două relații, obținem:

$$nT_n - (n-1)T_{n-1} = n(n-1) + 2\sum_{k=1}^{n} T_{k-1} - (n-1)(n-2) - 2\sum_{k=1}^{n-1} T_{k-1} = 2(n-1) + (n+1)T_{n-1}$$

Împărțind ambii membrii cu $n\cdot (n + 1)$ relația devine:

$$\frac{T_n}{n+1}=\frac{T_{n-1}}{n}+\frac{2(n-1)}{n(n+1)}$$

Apoi, trecem pe n în $n-1$:

$$\frac{T_{n-1}}{n}=\frac{T_{n-2}}{n-1}+\frac{2(n-2)}{(n-1)n}$$

.....

Pentru $n = 2$:

$$\frac{T_2}{3}=\frac{T_1}{2}+\frac{2\cdot 1}{2\cdot 3}$$

Sumând aceste relații obținem:

$$\frac{T_n}{n+1}=\sum_{k=2}^{n}\frac{2(k-1)}{k(k+1)}=2\sum_{k=2}^{n}\frac{k+1-2}{k(k+1)}=2(\sum_{k=2}^{n}\frac{1}{k}-\sum_{k=2}^{n}\frac{2}{k(k+1)}),$$

de unde rezultă că

$$\frac{T_n}{n+1}=2(\sum_{k=1}^{n}\frac{1}{k}-1-2\sum_{k=2}^{n}(\frac{1}{k}-\frac{1}{k+1})=2(\sum_{k=1}^{n}\frac{1}{k}-1-2(\frac{1}{2}-\frac{1}{n+1}))$$

sau,

$$\frac{T_n}{n+1}=2(\sum_{k=1}^{n}\frac{1}{k}-2+\frac{2}{n+1})=2\sum_{k=1}^{n}\frac{1}{k}-\frac{4n}{n+1}\cong 2\sum_{k=1}^{n}\frac{1}{k}=\frac{2}{n}\sum_{k=1}^{n}\frac{1}{\frac{k}{n}}=\int_1^n\frac{1}{x}dx=2\ln n.$$

În medie complexitatea algoritmului este astfel de *O(n ln n)*.

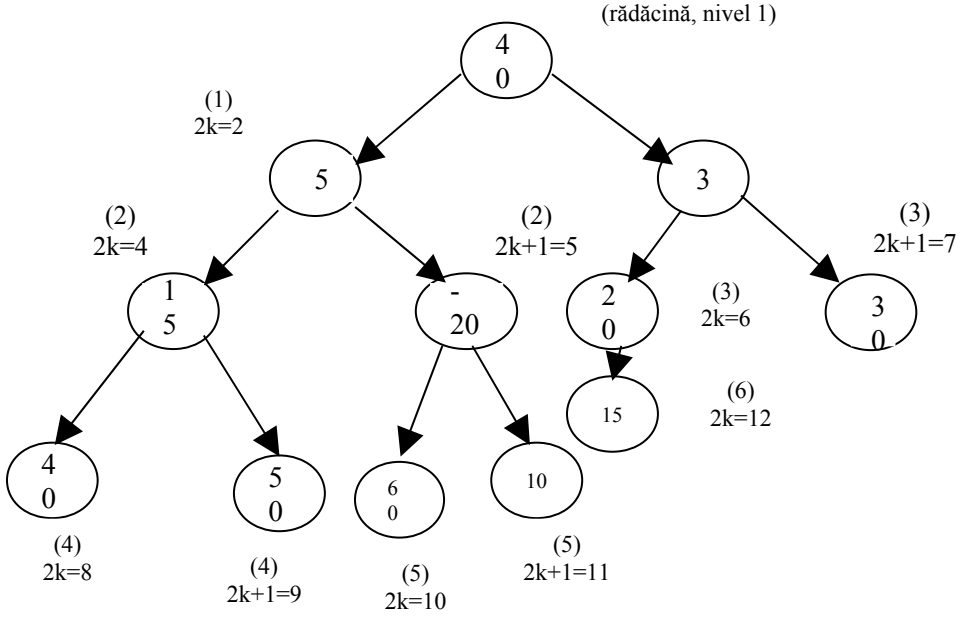
3.1.3.2. Sortarea cu grămezi, *heapsort*

Acest algoritm este prezentat şi într-un *proiect de lecție*, care va urma. Este necesar de amintit faptul că deşi informația de intrare poate fi conținută într-un vector sau o listă (utilizându-se pointeri), ea trebuie *văzută* ca alcătuind un *arbore binar*. Astfel, ca exemplu, să luăm vectorul *V* de mai jos, având *12* componente :

V

40	5	1	15	-20	20	30	40	50	60	10	15
1	2	3	4	5	6	7	8	9	10	11	12

Arborele asociat va conține valoarea `40` (corespunzătoare poziției `1` din `V`) în nodul rădăcină. Având valoarea `v` într-un nod, valoare corespunzătoare poziției `k` din `V`, cei (maxim `2`) succesori imediați ai nodului vor conține valorile situate pe pozițiile `2k` (fiul stâng), respectiv `2k + 1` (fiul drept) din `V`.



PROIECT DE TEHNOLOGIE DIDACTICĂ

Data:-.
Clasa: a XI-a .
Profesor: -.
Disciplina: Informatică aplicată.
Tipul lecţiei: Predare-învăţare.
Obiectiv fundamental: Formarea deprinderii de a ordona un şir utilizând *Heapsort-ul*.
Obiective operaţionale: La sfârşitul lecţiei elevii vor fi capabili :

- să definească un heap(grămadă, ansamblu);
- să creeze un heap;
- să aplice algoritmul de sortare Heapsort;
- să scrie programul pentru algoritmul Heapsort.

Strategii didactice: Conversaţia, explicaţia, metoda analitică, munca independentă, etc.
Mijloace de învăţământ: Manuale, culegeri de probleme.
Metode: Activitate frontală, individuală.
Resurse:

- pedagogice - *Metodica predării informaticii*, alte cursuri de informatică, ghiduri pentru profesori;:
- oficiale - programa şcolară;
- temporale - 50 minute;
- psihologice - cunoştinţe dobândite de către elevi până la această dată;
- colectiv eterogen (interesat de obiect);
- clasa împărţită pe grupe.

Un *heap(grămadă, ansamblu)* este o multimulţime (mulţime în care anumite componente se pot repeta). Multimulţimea poate fi reprezentată ca un arbore binar (în sensul celor spuse anterior, în exemplu). Atunci, un *max-heap* este un arbore binar complet (exceptând, eventual, lipsa unei *ultime* frunze/nod pendant) în care valoarea memorată în orice nod al său este mai mare sau egală decât valorile memorate în nodurile fii ai acestuia. Similar, *min-heap*-ul este un arbore binar complet în care valoarea memorată în orice nod al său este mai mică sau egală decât valorile memorate în nodurile fii ai acestuia. Deoarece, conform proprietăţii de *max-heap*, elementul maxim trebuie să se afle în rădăcina heap-ului, deci pe prima poziţie din vector, el poate fi plasat pe poziţia sa corectă, interschimbându-l cu elementul din poziţia `n`. Noul element din rădăcina *heap*-ului poate să nu respecte proprietatea de *max-heap*, dar subarborii rădăcinii rămân *heap*-uri. Prin urmare, trebuie *restaurat heap*-ul, apelând o funcţie de combinare a elementelor din poziţiile `1` şi `n-1`. Elementul de pe poziţia `n` fiind deja *la locul lui,* practic nu mai este nevoie să fie inclus (formal) în *heap*. Procedeul se repetă până când toate elementele vectorului sunt plasate pe poziţiile lor corecte.

Codul **C** este:

```
#include <stdio.h>
#include <alloc.h>
#include <conio.h>

void schimba(int *a, int *b)
 {
```

```
      int aux = *a;
  *a = *b;
  *b = aux;
 }

void urca(int *v, int i)
 {
  if (i > 0)
     {
            int j = (i - 1) / 2;
            if (v[i] > v[j])
     {
           schimba(&v[i], &v[j]);
  urca(v, j);
}

void coboara(int *v, int j, int i)
 {
    int k = 2 * (j + 1) - 1;
    if (k < i)
     {
      /* v[k] va fi fiul stang al lui v[j], iar v[k + 1] fiul drept. */
      if (v[j] < v[k])
     {
        if (k + 1 == i)
            schimba(&v[j], &v[k]);
        else
            if (v[k] > v[k + 1])
      {
              schimba(&v[j], &v[k]);
       coboara(v, k, i);
            }
      else
             {
       schimba(&v[j], &v[k + 1]);
             coboara(v, k + 1, i);
      }
     }
    else
      if (v[j] < v[k + 1] && k < i - 1)
       {
      schimba(&v[j], &v[k + 1]);
      coboara(v, k + 1, i);
       }
   }
 }

void afiseaza_lista(int *v, int n)
 {
  int i;
  printf("Lista sortata crescator este \n");
  for (i = 0; i < n; i++)
```

```
    printf("%d ", v[i]);
 }

void main()
 {
  clrscr();
  /* urmeaza citirea datelor; n este numarul de elemente;
     v este vectorul cu n elemente ce urmeaza a fi sortat */
  int n;
  printf("Dati numarul de elemente\n");
  scanf("%d", &n);
  int *v = (int *) malloc(n * sizeof(int));
  int i;
 if (v == NULL)
{
      printf("\n Alocare esuata.\n");
      exit(1);
}

 for (i = 0; i < n; i++)
   {
    printf("dati v[%d]=", i + 1);
    scanf("%d", &v[i]);
   }
  /* vom forma in continuare un ansamblu cu proprietatea de maxim;
     v[i] are fiul stang v[2*(i+1)-1], iar fiul drept v[2*(i+1)].
     Ideea este ca elementul v[i] va urca (eventual) pana la
     radacina. Se observa usor ca proprietatea de maxim a ansamblului
     se pastreaza. Urcarea se va face de la varfurile pendante catre
     radacina, deci se scade o unitate si se injumatateste indexul
     (conform codicarii). */
  for (i = 1; i < n; i++)
    urca(v, i);
/* Acum vom aplica o proprietate a max-ansamblelor, anume aceea   ca
radacina     are eticheta cea mai mare. Deci, vom
     interverti v[0] (adica radacina) cu v[i] (elementul curent) si
     vom crea imediat un ansamblu cu proprietatea de maxim pentru
     primele i - 1 elemente. Pentru aceasta, va trebui sa coboram
     noua radacina la locul ei (adica pe locul v[i]). */
  for (i = n - 1; i > 0; i--)
   {
    schimba(&v[0], &v[i]);
    coboara(v, 0, i);
   }
  afiseaza_lista(v, n);
  getch();
 }
```

Codul **PASCAL** este:

```
{sortare cu ansamble. Numim ansamblu (heap, gramada) - din punct de
vedere al executiei algoritmului! - un arbore in care inf(tata)>=inf(i),
unde i sint noduri ale subarborelui de radacina tata}
program heap_sort;
uses crt;
const nmax = 100;
type vector = array [1..nmax] of integer;
var  v : vector;
     i, {variabila de lucru}
     n, {numarul de elemente}
     m, {index de lucru, cu valori intre 1 si n}
     aux : integer; {variabila de lucru}

{procedura de interschimbare a doua elemente din vector}
procedure schimba (var i, j : integer );
var aux : integer;
begin
   aux := v[i];
  v[i] := v[j];
  v[j] := aux;
end;

{procedura determina cel mai mare element din arbore in urmatoarea
maniera: se ia elementul si se testeaza care informatie din fii este
mai mare. Eventual, eticheta tatalui se interschimba cu eticheta celui
mai mare fiu.}
procedure insereaza ( var i : integer );
{"i" reprezinta indexul nodului tata}
var j, aux : integer; {variabile de lucru}
begin
  {"2*i<=m" este conditia de terminare a recursiei}
  if (2 * i <= m) then
     begin
          {"j" reprezinta indexul fiului stang}
          j := 2 * i;
          {daca exista fiu drept, atunci comparam cei doi fii intre ei}
          if j + 1 <= m then
              {"j" va reprezenta in final indexul celui mai mare dintre
fii}
             if v[j] < v[j + 1] then j := j + 1;
          {comparam acum tatal cu fiul, eventual interschimbandu-i}
          if v[i] < v[j] then
             begin
                  schimba ( i, j );
                  {urmeaza apelul recursiv necesar pentru eventuala
                   "coborare" a nodului catre frunze.}
                  insereaza ( j );
             end;
     end;
end;

{programul principal}
```

```
begin
  {stergerea ecranului}
  clrscr;

  {citirea vectorului}
  write ( ' Dati dimensiunea vectorului= ' );
  readln ( n );
  writeln;
  writeln ( ' Dati acum sirul: ' );
  for i := 1 to n do
      begin
           write ( ' v[' ,i, ']=' );
           readln ( v[i] );
      end;
  {"m" reprezinta indexul elementului curent, care initial
   coincide cu numarul de elemente ale vectorului}
  m := n;

  {creearea ansamblului. Plecam de la penultimul nivel al arborelui
   deoarece trebuie sa fim siguri ca avem descendenti.}
  for i := trunc(m/2) downto 1 do
      insereaza ( i );

  {determinarea vectorului sortat}
  while m > 1 do
        begin
             {plecam din radacina, deci de la indexul 1}
             i := 1;
             {urmeaza apelul de "coborare" eventuala a radacinii}
             insereaza ( i );
             {deoarece eticheta radacinii este cea mai mare din vector
                o putem intershimba cu ultimul element. Astfel cel mai
mare
              element al vectorului se afla pe ultima pozitie.}
             schimba ( 1, m );
                {vom pastra proprietatea de ansamblu cu proprietatea de
maxim (max-heap)
              pentru vectorul cu n-1 elemente.}
             m := m-1;
         end;
  writeln;
  {urmeaza afisarea vectorului sortat crescator}
  write( ' sirul sortat este: ' );
  for i := 1 to n do write ( v[i], '  ' );
  writeln;
  readln;
end.
```

Anexele vor conţine şi alte exemple utile.