

COLLEGIUM

Metodică

© 2016 by Editura POLIROM

Această carte este protejată prin copyright. Reproducerea integrală sau parțială, multiplicarea prin orice mijloace și sub orice formă, cum ar fi xeroxarea, scanarea, transpunerea în format electronic sau audio, punerea la dispoziția publică, inclusiv prin internet sau prin rețele de calculatoare, stocarea permanentă sau temporară pe dispozitive sau sisteme cu posibilitatea recuperării informațiilor, cu scop comercial sau gratuit, precum și alte fapte similare săvârșite fără permisiunea scrisă a deținătorului copyrightului reprezintă o încălcare a legislației cu privire la protecția proprietății intelectuale și se pedepsesc penal și/sau civil în conformitate cu legile în vigoare.

Pe copertă: © Iconic Bestiary/Shutterstock.com

www.polirom.ro

Editura POLIROM

Iași, B-dul Carol I nr. 4; P.O. BOX 266, 700506

București, Splaiul Unirii nr. 6, bl. B3A, sc. 1, et. 1,
sector 4, 040031, O.P. 53

Descrierea CIP a Bibliotecii Naționale a României:

MASALAGIU, CRISTIAN-DUMITRU

Didactica predării informaticii / Cristian-Dumitru Masalagiu, Ioan Asiminoaei, Mirela-Anca Țibu. – Ed. a 2-a, rev. și adăug. – Iași: Polirom, 2016

Conține bibliografie

ISBN print: 978-973-46-3388-3

ISBN ePub: 978-973-46-6270-8

ISBN PDF: 978-973-46-6271-5

I. Asiminoaei, Ioan

II. Țibu, Mirela-Anca

371.3:004+007

Printed in ROMANIA

**Cristian Masalagiu, Ioan Asiminoaei
Mirela Țibu**

DIDACTICA PREDĂRII INFORMATICII

Ediția a II-a revăzută și adăugită

POLIROM
2016

CRISTIAN MASALAGIU este profesor universitar doctor la Facultatea de Informatică, Departamentul de Informatică, din cadrul Universității „Alexandru Ioan Cuza”, Iași, fiind angajat în învățământul superior din anul 1980. Informații suplimentare privind întreaga sa activitate didactică și științifică pot fi găsite la adresa web <http://www.info.uaic.ro/~masalagiu>.

IOAN ASIMINOAEI este administrator rețea calculatoare, CEC Bank SA, Sucursala Județeană Iași, cadru didactic asociat la Facultatea de Informatică a Universității „Alexandru Ioan Cuza”, Iași. Cursuri susținute în cadrul facultății : Metodica predării informaticii (anul IV, Secția Informatică), Programare Windows – MFC, COM/DCOM, dezvoltare de aplicații folosind platforma .NET, topici speciale. NET. Informații suplimentare privind întreaga sa activitate didactică și științifică pot fi găsite la adresa web <http://www.info.uaic.ro/~iasimin>.

MIRELA ȚIBU este profesor titular la Liceul Teoretic de Informatică „Grigore Moisil”, Iași, metodist al Inspectoratului Școlar Județean Iași, coordonator de practică pedagogică (disciplina Informatică, anul III) și colaborator la Facultatea de Informatică a Universității „Alexandru Ioan Cuza” din Iași, unde a susținut seminarele la disciplina Didactica informaticii (anul II, Secția Informatică).

Cuprins

<i>Cuvânt-înainte</i>	9
Introducere	11
Ce este informatica?	11
Societatea informațională și e-educația	15
Structura lucrării și contribuțiile personale ale autorilor	21
Capitolul 1. Curriculum, instruire, evaluare	23
1.1. Teoria curriculumului	23
1.1.1. Curriculumul național în informatică	23
1.1.2. Elaborarea ofertei curriculare	25
1.1.3. Planurile de învățământ	26
1.1.4. Programele școlare (analitice, curriculare)	30
1.2. Teoria instruirii	36
1.2.1. Manualele școlare	36
1.2.2. Structura lecției	42
1.2.3. Calitatea cunoștințelor asimilate	43
1.2.4. Formarea limbajului de specialitate	44
1.2.5. Caietele elevilor	45
1.3. Teoria evaluării	46
1.3.1. Repetare, recapitulare, evaluare	46
1.3.2. Practica evaluării activității didactice	51
1.3.3. Aprecierea cunoștințelor elevilor. Măsuri de prevenire a rămânerilor în urmă	53
1.3.4. Condiția profesorului	55
1.3.5. Planificarea activității didactice	57
Capitolul 2. Principii didactice și didactica formării de competențe	59
2.1. Clasificarea și caracteristicile principiilor didactice	59
2.1.1. Principiul intuiției	60
2.1.2. Principiul legării teoriei de practică	60
2.1.3. Principiul însușirii conștiente și active a cunoștințelor	61
2.1.4. Principiul sistematizării și continuității cunoștințelor	62

2.1.5. Principiul accesibilității cunoștințelor	62
2.1.6. Principiul însușirii temeinice a cunoștințelor	63
2.1.7. Principiul individualizării și diferențierii învățării	64
2.2. Didactica formării de competențe	69
2.2.1. Contextul trecerii de la obiective la competențe	69
2.2.2. Competențele-cheie în studiul informaticii	71
2.2.3. Proiectarea competențelor	73
2.2.4. Analiza resurselor	75
2.2.5. Elaborarea strategiei didactice	91
2.2.6. Clasificarea competențelor	91
2.2.7. Formularea competențelor	93
2.2.8. Momentele lecției	93
Capitolul 3. Metode, tehnici și procedee didactice	101
3.1. Metode generale de învățare	101
3.1.1. Expunerea sistematică a cunoștințelor	102
3.1.2. Metoda conversației	103
3.1.3. Problematizarea și învățarea prin descoperire	105
3.1.4. Modelarea	108
3.1.5. Exemplificarea sau demonstrarea materialului intuitiv	111
3.1.6. Metoda exercițiului	112
3.1.6.1. Exerciții și probleme de recunoaștere a unor noțiuni, formule, metode	113
3.1.6.2. Exerciții și probleme aplicative pentru formule sau algoritmi cunoscuți	113
3.1.6.3. Probleme reale care permit însușirea unor noțiuni	114
3.1.7. Metoda învățării în grupe mici	117
3.1.8. Metoda lucrului cu manualul și documentația	118
3.1.9. Metoda jocurilor didactice	120
3.1.10. Instruirea programată și învățarea asistată de calculator	120
3.2. Metode specifice de învățare	122
Capitolul 4. Noțiuni de bază	125
4.1. Noțiuni de bază în informatică	125
4.2. Paradigme de programare	130
4.3. Tehnici de programare. Proiectarea algoritmilor : sortare și căutare	133
4.4. Algoritmi clasici de sortare, de complexitate timp $O(n^2)$ și mai mare	136
4.4.1. Sortarea prin interschimbarea elementelor vecine	136
4.4.2. Sortarea prin selecție	137
4.4.3. Sortarea prin inserție directă	138
4.4.4. Sortarea Shell	139
4.4.5. Sortarea rapidă	141
4.4.5.1. Sortarea prin interclasare	142
4.4.5.2. Sortarea <i>quicksort</i>	143
4.4.5.3. Sortarea cu grămezi (<i>heapsort</i>)	146

4.5. Metode de elaborare (proiectare) a algoritmilor	150
4.5.1. Metoda <i>divide et impera</i>	151
4.5.2. Metoda <i>backtracking</i>	153
4.5.3. Metoda <i>greedy</i>	158
4.5.4. Metoda programării dinamice.....	161
4.6. Analiza complexității, corectitudinii și terminării algoritmilor/programelor	164
Capitolul 5. Structuri de date : liste, stive, cozi.....	177
5.1. Liste.....	177
5.1.1. Liste liniare simplu înlanțuite	178
5.1.1.1. Crearea unei liste simplu înlanțuite.....	181
5.1.1.2. Accesul la un nod al listei simplu înlanțuite	183
5.1.1.3. Inserarea unui nod într-o listă simplu înlanțuită.....	185
5.1.1.4. Ștergerea unui nod dintr-o listă simplu înlanțuită	188
5.1.2. Liste circulare simplu înlanțuite	191
5.1.2.1. Crearea unei liste circulare simplu înlanțuite	192
5.1.2.2. Inserarea unui nod într-o listă circulară simplu înlanțuită	193
5.1.2.3. Ștergerea unui nod dintr-o listă circulară simplu înlanțuită	193
5.1.2.4. Parcurgerea unei liste circulare simplu înlanțuite	193
5.1.3. Liste liniare dublu înlanțuite	194
5.1.3.1. Crearea unei liste dublu înlanțuite	195
5.1.3.2. Accesul la un nod al unei liste dublu înlanțuite	197
5.1.3.3. Inserarea unui nod într-o listă dublu înlanțuită	197
5.1.3.4. Ștergerea unui nod dintr-o listă dublu înlanțuită	199
5.1.3.5. Ștergerea unei liste dublu înlanțuite	200
5.2. Stive.....	201
5.3. Cozi.....	202
5.4. Baze de date	203
5.4.1. Furnizori ADO.NET.....	203
5.4.2. Conectarea la baza de date	205
5.4.3. Execuția comenzilor	205
5.4.3.1. Adăugarea înregistrărilor	205
5.4.3.2. Actualizarea înregistrărilor	206
5.4.4. Regăsirea informațiilor.....	207
Capitolul 6. Teoria grafurilor și a arborilor	209
6.1. Grafuri și arbori	209
6.2. Arbori binari	213
6.2.1. Inserarea unui nod frunză într-un arbore binar alocat dinamic	215
6.2.2. Parcurgerea unui arbore binar.....	217

6.2.3. Crearea arborilor binari plecând de la parcurgerile în preordine și inordine	219
6.2.4. Accesul la un nod al unui arbore binar.....	220
6.2.5. Ștergerea unui arbore binar	221
Concluzii.....	233
<i>Anexe</i>	235
Anexa 1. Proiecte de tehnologie didactică.....	237
Anexa 2. Subiecte date la concursuri de specialitate.....	263
Anexa 3. Adrese web utile	301
<i>Bibliografie</i>	303

Cuvânt-înainte

Informatica a început să fie considerată știință abia din deceniul nouă al secolului trecut, fiind denumită de atunci și *știința calculului*, *știința calculatoarelor* sau *știința tehnologiei informației și comunicării*. Fundamentele ei țin de științele exacte : matematică, fizică, chimie, biologie sau chiar medicină. Informatica este însă atât o știință abstractă, cât și una strâns legată de o realitate concretă, intermediată de o mașină (calculatorul). Viitorul omenirii este deja marcat de societatea informațională în care trăim, pe plan internațional războiul clasic și cel economic fiind înlocuite adesea prin războiul informațional. În România, informatica, industria IT (*Information Technology*) și-au căpătat actualul statut cumva cu întârziere, dificultățile fiind determinate de regimul comunist. Între 1970 și 1980 (poate chiar mai devreme) și-au făcut totuși loc calculatoarele analogice, mașinile de facturat și contabilizat, calculatoarele de tipul Felix sau IBM și mai apoi minicalculatoarele. În acea perioadă de relansare economică (și relaxare politică) s-au înființat centrele teritoriale de calcul electronic, oficiile de calcul din anumite întreprinderi, Institutul Central de Informatică (ICI), liceele cu profil de informatică, secțiile de informatică (mașini de calcul, automatizări ș.a.) din cadrul facultăților de matematică sau al facultăților de inginerie. După această etapă de oarecare efervescență internă și în ciuda unor succese recunoscute și pe plan internațional, sincopa a venit spre sfârșitul epocii Ceaușescu, exact în momentul în care, în lume, apariția calculatoarelor personale transforma informatica din știința unor aleși în bun public. A fost nevoie ca, după 1990, întreaga societate românească să depună un efort imens (sprijinită și din exterior, inclusiv prin contribuția generoasă a unor organizații nonguvernamentale) pentru ca țara noastră să nu rămână la periferia utilizării tehnologiilor informaționale moderne. Școala românească de informatică are astăzi un prestigiu internațional notabil, începând cu învățământul sau cercetarea și încheind cu specialiștii de înalt nivel care activează în marile companii naționale și internaționale.

Datorită noutății, importanței și dinamicii deosebite de schimbare a domeniului, există încă numeroase întrebări și dileme ale căror răspunsuri sau soluții se pot dovedi capitale atât pentru învățământ, cât și pentru ansamblul societății viitoare. Trecem în revistă, pe scurt, doar câteva dintre acestea. Din punct de vedere conceptual, este firesc să ne întrebăm mai întâi dacă informatica este o știință concretă/aplicativă sau abstractă. Este ea indisolubil legată de matematică sau mai mult de fizică, chimie ori biologie ? Terminologia sa actuală, încă incompletă și uneori ambiguă, trebuie să fie nouă în totalitate sau preluată și adaptată din științele fundamentale, de graniță ? Care este relația exactă dintre anumite concepte de bază de o natură (aparent) complet diferită, cum ar fi algoritm și cip ? Informatica înseamnă oare programare, algoritmică, structuri de informație sau comunicare (Internet, telefonie mobilă etc.), codificare/criptare, interfață om-mediul ? Care va fi în viitor relația exactă dintre informatică și celelalte științe ? Care este în acest moment adevărata substanță intelectuală a acestei științe ? Mai mult,

ne putem întreba cât de util poate fi procesul de predare-învățare a unei materii de tipul didactica (metodica) predării informaticii la nivel universitar? În multe dintre utilizările sale, informatica se prezintă mai degrabă ca o disciplină de tipul „fă ca mine” decât de tipul „citește și ai să înveți ce și cum să faci”. Trecerea pe primul plan a învățământului asistat, a educației permanente pare să nu lase loc pentru precepte metodico-pedagogice prestabilite. Autorii cred însă că tocmai datorită contextului amintit păstrarea și chiar impunerea unor linii directoare, a unor principii de natură metodică (și nu numai) în relația dintre cei care predau și cei care învață informatica, precum și în relația școală-companii IT reprezintă singura garanție pentru evitarea anumitor sincope în evoluția societății umane. La nivel național, din inițiativa unor inspectorate școlare județene și sub egida Ministerului Educației Naționale și Cercetării Științifice, care are o nouă structură organizatorică începând cu 3 februarie 2016, s-a definitivat deja o parte importantă a noilor programe curriculare obligatorii pentru învățământul primar și gimnazial (urmând, desigur, învățământul liceal), inclusiv în ceea ce privește informatica și tehnologia informației și a comunicațiilor. Ușurința aparentă a utilizării calculatorului, dezvoltarea microelectronicii, a comunicării *wireless* și, nu în ultimul rând, accesibilitatea din ce în ce mai mare a prețurilor au condus la apariția unui număr important de companii și școli particulare (unele chiar specializate) care au drept scop declarat organizarea de cursuri pentru instruirea în programare și în folosirea eficientă a unor dispozitive precum telefoane mobile, tablete, laptopuri etc. (dar și a unor softuri specializate, destinate, de exemplu, proiectării paginilor web sau creării de bloguri personalizate). Au apărut noi profesii importante legate de acest domeniu (vezi Nomenclatorul meseriilor sau Clasificarea ocupațiilor din România) și se pune întrebarea ce trebuie făcut neapărat în școală, respectiv ce poate fi lăsat în afara învățământului de stat. Înțelegerea necorespunzătoare a fenomenului informatic, precum și a profunzimii acestuia poate genera grave convulsii sociale.

Didactica predării informaticii studiază tehnicile și metodele de predare-învățare în întreaga lor complexitate, căutând să evidențieze problemele majore ale acestei discipline, să examineze și să dea sugestii cu privire la programele analitice (curriculare) curente, la funcționarea și organizarea sistemului de învățământ specific în ansamblu.

Lucrarea de față continuă și, în același timp, completează și aduce la zi volumele *Metodica predării informaticii* (Matrix-Rom, 2001) și *Didactica predării informaticii* (Polirom, 2004). O parte a materialului anterior a fost preluată ca atare (doar din a doua carte menționată, nu și din prima), dar s-au produs modificări importante în structura generală a volumului, legate în primul rând de transformările societății românești care au generat schimbări structurale, decizionale, curriculare la nivel de învățământ preuniversitar și nu numai.

Volumul se adresează în principal profesorilor din învățământul preuniversitar, dar și elevilor și studenților cu preocupări legate de informatică. Am dorit în primul rând să scoatem în evidență problemele importante de natură metodică și psihopedagogică ale predării informaticii în învățământul preuniversitar. Fără a avea pretenția de a trata exhaustiv un domeniu ale cărui frontiere (și impact social) nu pot fi clar delimitate, considerăm că materialul va acoperi un gol existent în literatura de specialitate din România.

Le mulțumim pe această cale tuturor celor care ne-au sprijinit și încurajat în conceperea și realizarea volumului. De asemenea, dorim să mulțumim membrilor familiilor noastre pentru răbdarea și înțelegerea acordate.

Autorii
mai 2016

Introducere

În primul subcapitol vom discuta despre informatică, mai exact despre disciplinele cunoscute în învățământul gimnazial și liceal sub denumirile generice de „informatică” și „tehnologia informației și comunicațiilor” (TIC). Apoi vom prezenta contextul general în care se plasează societatea românească la început de mileniu și, în particular, învățământul. Vom insista asupra învățământului preuniversitar, precizând și câțiva termeni intrați deja în vocabularul de bază al limbii române, dar a căror semnificație exactă este încă necunoscută sau interpretată în mod eronat ([Mas2], [*1]). Accentul este pus pe legătura cu mediul virtual cu care educația, mai exact educația pe cale electronică (*e-educația*), este astăzi interconectată de-a lungul întregii vieți (vorbit despre *lifelong learning*, LLL). La sfârșit prezentăm pe scurt conținutul fiecărui capitol în parte, precum și contribuția concretă a fiecăruia dintre cei trei autori.

Ce este informatica?

Începând cu 1980, termenul „informatică” a devenit sinonim cu „știința calculului”, „știința calculatoarelor”, „ingineria calculatoarelor”, „tehnologia informației și a comunicațiilor” ș.a.m.d. Neexhaustiv, alte definiții informale ar mai putea fi :

1. Informatica se ocupă cu studiul calculatoarelor și al fenomenelor majore din jurul acestora.
2. Informatica cuprinde totalitatea cunoștințelor asupra calculatorului și calculului. Ea are componente teoretice, experimentale și de proiectare și include :
 - a) Teorii pentru înțelegerea echipamentelor de calcul, a programelor și sistemelor.
 - b) Experimente pentru testarea și dezvoltarea conceptelor.
 - c) Metodologii (reguli, metode) de proiectare (algoritmi, *unelte* pentru aplicații practice particulare).
 - d) Metode de analiză pentru verificarea faptului că realizările îndeplinesc cerințele.
3. Informatica se ocupă cu studiul reprezentării cunoștințelor și implementării acestora.
4. Informatica se ocupă cu studiul modelelor și complexității cunoștințelor.

5. Informatica se ocupă cu studiul sistematic al proceselor algoritmice care descriu și transformă informația (teoria informației), precum și cu analiza, proiectarea, implementarea și aplicarea acestora.

Pe parcursul lucrării este posibil să întâlnim și alte descrieri ale acestui concept, dar considerăm util să facem pentru început câteva considerații asupra faptului că informatica este atât o știință inginerască, cât și una abstractă. O știință abstractă pune în general accent pe teorie (analiză și sinteză) și doar accidental și adiacent pe modelare, iar o știință inginerască sau experimentală pune accent în principal pe modelare și proiectare. Să evidențiem, așadar, câteva caracteristici ale teoriei, modelării și proiectării.

Acceptăm ideea că teoria își are rădăcinile în matematică. Pentru a fi validă și consistentă, orice teorie trebuie să urmărească parcurgerea a cel puțin patru etape :

1. Să caracterizeze prin definiții obiectele de studiu.
2. Să caracterizeze prin teoreme relațiile posibile dintre acestea.
3. Să determine dacă relațiile stabilite sunt adevărate, utilizând demonstrații, raționamente.
4. Să interpreteze rezultatele obținute, prin verificare.

Pașii anteriori pot fi reiteați dacă apar erori. Din punctul de vedere al teoriei, calculatorul ajută la demonstrarea teoremelor, poate controla consistența specificațiilor, furnizează contraexemple, verifică anumite calcule pe date de test, efectuează calcule simbolice.

Modelarea își are rădăcinile în științele experimentale, având drept scop investigarea unui fenomen, din nou prin câteva etape :

1. Formarea unei ipoteze de lucru.
2. Construirea unui model formal (abstract) și realizarea unei predicții.
3. Proiectarea sau realizarea unui experiment și colectarea datelor.
4. Analiza rezultatelor.

La fel ca mai înainte, aceste etape pot fi reiterate în vederea îmbunătățirii modelului și pentru înlăturarea erorilor. Modelarea este „datoare” calculatorului începând cu modul de prezentare a unui model și terminând cu evaluarea corectitudinii acestuia sau cu deducerea unor predicții asupra lumii modelate.

Proiectarea își are rădăcinile în inginerie, în construirea oricărui sistem fiind necesar să fie urmate etapele :

1. Elaborarea specificațiilor de stare și a relațiilor dintre stări.
2. Proiectarea efectivă și implementarea sistemului.
3. Testarea sistemului în condiții reale.

Proiectarea utilizează calculatorul în orice etapă a sa, căci fără calculator aceasta ar fi mare consumatoare de timp, energie, resurse umane și materiale, în anumite

situații, chiar imposibilă. Toate cele trei procese amintite interacționează: teoria apare în (aproape) fiecare etapă de modelare și, eventual, proiectare; modelarea are un rol esențial atât în teorie, cât și în proiectare; proiectarea este parte componentă a fiecărei etape de teorie și modelare.

Fiecare dintre cele trei procese principale descrise mai sus poate susține că informatica îi aparține și că dezvoltarea acesteia se face numai în cadrul unui anumit proces. În realitate, informatica se dezvoltă la granițele dintre aceste procese și, prin urmare, utilizează metodele lor de cercetare, la care se adaugă metodele proprii, specifice.

Scopul principal al acestui volum nu este însă acela de a demonstra că informatica este o știință de sine stătătoare (lucru de altfel confirmat de realitatea ultimelor decenii), ci mai curând de a identifica și studia actualitatea și adaptabilitatea metodelor folosite în predarea disciplinelor de informatică din ciclul preuniversitar (gimnazial și liceal). Vom admite astfel că scopurile introducerii informaticii ca disciplină de sine stătătoare (fie ea obligatorie, opțională sau facultativă) în planurile de învățământ sunt:

- Crearea unei atitudini pozitive privind importanța deosebită a informaticii în lumea contemporană și pătrunderea ei în toate domeniile vieții economico-sociale.
- Înțelegerea informaticii ca mijloc de modelare și simulare a fenomenelor realității înconjurătoare.
- Asigurarea nivelului de cultură generală în informatică prin parcurgerea principalelor etape din dezvoltarea informaticii ca știință.
- Dezvoltarea unei motivații intrinseci în studiul informaticii.
- Dezvoltarea capacității de utilizare a terminologiei, a unui limbaj informatic specific și a tehnicii de calcul în însușirea unor noi cunoștințe.
- Crearea unei atitudini favorabile activității de rezolvare a problemelor cu ajutorul calculatorului, prin deprinderea strategiilor de abordare a acestora și tratarea lor într-un mod riguros.
- Dezvoltarea competențelor digitale.
- Dezvoltarea spiritului inventiv și creator și a capacității de realizare a unor produse utilizabile.
- Dezvoltarea unor capacități de autoinstruire.

În România există o legislație clară privind organizarea învățământului preuniversitar, precum și curricula corespunzătoare. Până în 2016, planurile-cadru pentru învățământul primar și gimnazial s-au reconfigurat (apelându-se inclusiv la debateri publice), în curând același lucru urmând a fi făcut, sub egida Ministerului Educației Naționale și Cercetării Științifice (MENCS) și a Inspectoratului Școlar Județean (ISJ), și pentru învățământul liceal. În prezent, disciplinele informatică și tehnologia informației și a comunicațiilor, care abordează conținuturi de informatică în mod direct și declarat, se regăsesc (în gimnaziu și liceu) în trunchiul comun (disciplinele obligatorii), în curriculumul diferențiat (în funcție de specializare) și în curriculumul la decizia școlii. La gimnaziu, vorbind despre prezența disciplinelor de informatică, trebuie să spunem că nu există ore în trunchiul comun, dar sunt propuse tot mai des

discipline opționale de informatică în curriculumul la decizia școlii, care sunt alese de elevi. Câteva exemple care pun accentul pe algoritmică și programare :

- Introducere în algoritmică – clasa a V-a.
- Algoritmică și programare – clasele VI-VIII.
- Noțiuni de informatică – clasele V-VIII.
- Programarea în Scratch – clasele V-VII.
- Prietenul meu calculatorul – clasele V-VIII.
- Lumea digitală – clasele V-VIII.

Ca un exemplu, în urma derulării proiectului „Competențe-cheie TIC în curriculumul școlar : accesul la educație și formare profesională inițială de calitate”, cofinanțat din Fondul Social European (prin Programul Operațional Sectorial pentru Dezvoltarea Resurselor Umane 2007-2013 pe axa prioritară nr. 1 „Educație și formare profesională în sprijinul creșterii economice și dezvoltării societății bazate pe cunoaștere”, domeniul major de intervenție 1.1), au fost concepute programele școlare pentru disciplina tehnologia informației și a comunicațiilor, clasele V-VIII. Astfel, din 2013, majoritatea școlilor au introdus la clasele de gimnaziu această disciplină.

Conținuturi studiate astăzi la tehnologia informației și a comunicațiilor, în gimnaziu :

- Structura sistemului de calcul, componente hardware. Accesorii ale sistemului de operare Windows.
- Procesorul de texte Word ; prezentări multimedia ; comunicarea prin Internet ; mesageria electronică, e-mail.
- Calculul tabelar, documentare online.

Considerând disciplinele de informatică din curriculumul de liceu, să menționăm că la toate filierele (teoretică, tehnică și vocațională), profilurile și specializările, la primele două clase sunt prevăzute ore de tehnologia informației și a comunicațiilor (două ore pe săptămână la clasa a IX-a, o oră pe săptămână la clasa a X-a). În unele școli, acolo unde planul-cadru permite alocarea unor ore în curriculumul la decizia școlii pentru discipline din aria curriculară „Tehnologii”, din care fac parte și disciplinele informatică și tehnologia informației și a comunicațiilor (în special pentru filiera teoretică), sunt propuse discipline noi, care urmăresc să familiarizeze elevii cu tehnologii moderne, cum ar fi :

- Web design.
- Game development.
- Grafică 3D.
- Crearea jocurilor în limbajele Alice și Greenfoot.
- CISCO fundamentals (organizate în școlile care sunt academii CISCO).
- Publicitate în medii electronice.

Societatea informațională și e-educația

Să precizăm încă de la început faptul că științele implicate direct în demersul nostru sunt (conform [*2]) :

- a) *Pedagogia* – știința care se ocupă cu studiul metodelor de educație și de instruire a oamenilor, în special a persoanelor cu puțină experiență.
- b) *Psihologia* – știința care se ocupă cu studiul proceselor și particularităților psihice umane.
- c) *Metodica* – parte a didacticii generale care studiază principiile, metodele și formele de predare adaptate specificului fiecărui obiect de învățământ.
- d) *Didactica* – parte a pedagogiei care se ocupă cu principiile și metodele predării materiilor de învățământ, precum și cu organizarea învățământului.

Putem spune că didactica generală studiază problemele teoretice și instructiv-educative globale ale învățământului (inclusiv ale învățământului de informatică). Ele se referă la procesul însușirii de către elevi a cunoștințelor, priceperilor și deprinderilor, la sarcinile și conținutul învățământului, precum și la principiile, metodele și formele de organizare a învățământului. Metodica poate fi privită ca o didactică particulară. Obiectul de studiu al didacticii predării informaticii îl constituie astfel învățământul de informatică, precum și sarcinile care îi revin. Aceste sarcini decurg din obiectivele generale ale învățământului și educației în etapa actuală de dezvoltare a societății.

În ceea ce privește cadrele didactice, aceste științe sunt importante în egală măsură și trebuie studiate și stăpânite simultan. Cunoștințele științifice acumulate (oricât de vaste și profunde ar fi) nu sunt suficiente pentru desfășurarea în condiții corespunzătoare a procesului de instruire. Pentru ca activitatea profesorului să aibă rezultatul dorit este necesar ca aceasta să se desfășoare într-un mediu adecvat din punct de vedere legislativ, economic, administrativ etc. Asta pentru a nu mai aminti de talent și perseverență.

Ideile principale privind aspectul societății noi și rolul profesorului în procesul de schimbare conștientizată a acesteia datează aproximativ din deceniul nouă al secolului trecut. Iată câteva dintre ele, așa cum au fost enunțate în perioada amintită :

- Societatea viitorului (cea în care trăim acum) va fi o societate informațională (*Information Society*). Acest tip de societate va apărea datorită răspândirii vaste a noilor tehnologii privind informația și comunicațiile. Schimbări profunde se vor produce în toate domeniile, începând cu administrația (e-guvernare) și afacerile (e-commerce) și terminând cu educația/învățământul (e-education/e-learning), cultura (centre multimedia și biblioteci virtuale) și chiar maniera de a exercita o anumită profesiune.
- Astfel, nu vom putea evita (în viitorul foarte apropiat) educația permanentă, care va implica folosirea calculatorului în mod profesional. Ponderea învățământului deschis, la distanță și în timp real (și online), va fi covârșitoare. Existența

Internetului constituie cea mai importantă bază pentru promovarea unui asemenea tip de educație.

- Rolul profesorului va fi mult mai dificil și mai important, iar probleme ca instruirea profesorilor, reconversia profesională, multispecializarea vor căpăta importanță națională.
- Țara noastră trebuie să devină parte activă a efortului general dedicat reducerii costurilor economice și efectelor sociale negative generate de trecerea la această societate.
- Anumite condiții specifice României de azi, cum ar fi instabilitatea economică prelungită, lipsa sprijinului consistent (financiar sau de altă natură) din exterior, faptul că învățământul superior nu poate garanta o poziție socială sigură pentru absolvenți, existența unor regiuni defavorizate, legislația (uneori) ambiguă etc., ne forțează (poate surprinzător) să facem pași mai mulți și mai rapizi decât țările dezvoltate pentru a trece la societatea informațională.
- Folosirea generalizată a învățământului la distanță poate genera chiar economii. De exemplu, este posibil ca un singur calculator sau server performant (și câteva stații de lucru plasate în zone greu accesibile și cu puțini elevi) să coste mai puțin decât întreținerea și administrarea mai multor spații de învățământ tip școală.
- Tranziția la societatea informațională va genera însă probleme complexe la nivelul întregii umanități, astfel încât nu numai guvernele și administrațiile locale, dar chiar și organizațiile de anvergură mondială vor trebui să se implice activ în studierea și finanțarea unor proiecte de cercetare legate de rezolvarea acestor probleme. Factorii cei mai importanți pentru o asemenea tranziție sunt infrastructura de comunicații și aplicațiile informatice.

Ca o primă concluzie, sintagma „societate informațională” descrie o „economie și o societate în care accesul, achiziția, stocarea, prelucrarea, transmisia, răspândirea și utilizarea cunoștințelor și a informației joacă un rol decisiv” [*6]. Pe această linie, se observă că anumite consecințe pozitive ale tranziției spre societatea informațională pot fi lesne identificate și apreciate. În același timp, consecințele negative care ar fi putut să apară într-un mod aproape imprevizibil puteau genera un impact social negativ deloc de dorit. Societatea informațională trebuie să rămână o societate democratică, sporindu-și în același timp capabilitățile economice. Vorbind din punct de vedere cultural și științific, ea trebuie să rămână o societate a cunoașterii, și nu doar a acumulării de informații. Iată câteva dintre problemele viitoarei (adică... actualei!) realități, identificate tot în acea perioadă ([Sa], [WO], [*6]):

- Noile tehnologii legate de informație și comunicare (*Information and Communication Technologies*) vor elimina multe dintre locurile de muncă actuale. Va fi schimbată însăși natura principală a activității umane și chiar valorile eticii și moralei tradiționale derivate în principal din ideea „un loc de muncă egal un angajat” pot să fie distruse. Cadrele didactice vor constitui o categorie socială dintre cele mai afectate de schimbări, pregătirea profesorilor trebuind să fie din ce în ce mai

variata și profundă, în condițiile în care presiunea mediului asupra lor va fi tot mai mare. Din acest motiv și pentru ca școala românească să-și mențină în lume poziția de vârf deja dobândită, este necesar ca întregul sistem de învățământ să fie adaptat societății în transformare și să sprijine cu adevărat toate persoanele implicate în procesul de educație.

- Unul dintre lucrurile sigure în viitor va fi extinderea vastă a ceea ce se numește teleworking ([*4], [*5], [*10]). Vom fi nevoiți să lucrăm cu calculatorul și acasă, și la serviciu, cel puțin pentru a putea accesa la distanță diverși parteneri sau diverse informații.
- Contribuțiile financiare necesare pentru susținerea progresului trebuie să fie acceptate de fiecare membru al societății, indiferent de posibilitățile lui reale. Tendințele de a multiplica piața serviciilor interactive multimedia vor fi inutile fără acest sprijin financiar suplimentar.
- Creșterea numărului de produse soft dedicate publicului larg, în comparație cu cele dedicate lumii afacerilor, va fi semnificativă în viitor. Accentul va fi pus pe educația permanentă și pe posibilitatea de a accesa baze de date publice din ce în ce mai mari.
- Lumea afacerilor trebuie să devină conștientă de faptul că rolul educației va crește și de aceea va trebui să se implice activ și permanent în procesul educațional. Oamenii de afaceri ar trebui să se asigure că noile îndemânări, talente, cunoștințe pe care trebuie să le posedă angajații lor sunt cu adevărat predate și învățate în instituții specializate performante și acreditate.
- Soluția pentru a avea o siguranță materială cât de cât acceptabilă va fi pentru multe persoane ocuparea concomitentă a mai multor locuri de muncă, unele necesitând chiar calificări diferite.
- Unul dintre scopurile sociale cele mai importante în viitor va fi acela de a forța o balanță corectă între tendința naturală de dezvoltare a oricărei piețe și protecția intereselor publice sau individuale vitale. Prioritatea este de a crea în mod efectiv acel cadru legal regulator capabil să-i dea fiecăruia posibilitatea de a beneficia de avantajele societății informaționale, protejând în același timp comunitățile și minoritățile de orice fel.
- Administrațiile publice locale trebuie să se implice puternic în procesul de coordonare a tranziției către societatea informațională în menținerea încrederii publicului larg în siguranța socială.
- Educatorii trebuie să se schimbe în primul rând pe ei și să fie promotorii proceselor menționate.
- O amenințare majoră pentru fiecare individ într-o asemenea societate este posibilitatea pierderii intimității personale. Va fi dificil de menținut o balanță corectă între necesitatea publică de a ști cât mai mult despre tot și dreptul individual de a avea o viață privată protejată (să ne gândim că nu apăruseră încă site-urile de socializare!).

Putem concluziona că previziunile făcute au fost cu adevărat remarcabile, perioada care a trecut putând fi exploatată la maximum pentru optimizarea câștigurilor și minimizarea riscurilor de eșec legate de adaptarea la societatea informațională. Să subliniem faptul că printre obiectivele declarate ale tuturor guvernelor țărilor din cadrul Uniunii Europene s-a numărat și asigurarea unei tranziții cât mai lipsite de conflicte spre societatea informațională pentru toate statele membre. Acest lucru a fost dovedit de existența și activitatea unei Comisii Europene pentru Societatea Informațională (The European Commission for Information Society – ECIS), parte a Forumului pentru o Societate Informațională (Informational Society Forum – ISF, una dintre comisiile permanente ale Uniunii Europene), precum și a subcomisiei privind țările est- și central-europene care nu sunt membre ale Uniunii Europene (European Union Commission for Eastern and Central Europe Countries – EUCECC).

Să trecem pe scurt în revistă situația actuală a țării noastre, pornind de la un scurt istoric. În primul rând trebuie amintită Legea învățământului nr. 84/1995. Ca o consecință, Hotărârile de Guvern nr. 36/1997 și 102/1998 fixau condițiile prin care se puteau crea și finanța și puteau funcționa anumite rețele și instituții dedicate învățământului la distanță. Au urmat, în mod firesc, câteva ordine ale MENCȘ prin care s-au precizat condițiile concrete de aplicare a directivelor amintite. Printre acestea amintim :

- Ordinul MENCȘ 3289/1998 (<http://www.edu.ro>), de creare a centrelor zonale pentru învățământul la distanță, finanțate, în principal, din resurse locale, dar funcționând sub asistența proiectului european PHARE. Pentru a exista o coordonare eficientă, s-a creat și Oficiul Național pentru Învățământ la Distanță (ONIDD), iar centrele zonale au avut posibilitatea de a crea și controla subcentre departamentale, cum ar fi cele înființate în institutele de învățământ superior. Acestea din urmă, deși create sub egida unor instituții-mamă, aveau independență financiară.
- Ordinul MENCȘ 3495/1998, prin care s-a adoptat o nouă listă a specializărilor admise pentru învățământul superior. Învățământul la distanță a fost legal acceptat, în ansamblu, ca o nouă specializare.
- Ordinul MENCȘ 3354/1999, prin care s-a oficializat și crearea centrelor de învățământ la distanță în institutele (de stat) de învățământ superior. Acestea au avut drept scop primordial asigurarea educației permanente (sau chiar a celei de bază, în anumite situații) pentru acele persoane care nu au avut posibilitatea să urmeze un învățământ clasic. Conform legilor anterioare, finanțarea lor se realiza din fonduri europene, prin programul PHARE, precum și din resurse locale (inclusiv ale universităților sau facultăților implicate), cu o independență decizională corespunzătoare față de conducerea instituțiilor în cadrul cărora funcționau.

În 2016, întreaga activitate legată de societatea informațională este coordonată de Ministerul Comunicațiilor și pentru Societatea Informațională (MCSI ; fostul Minister al Comunicațiilor și Tehnologiei Informației ; <http://www.mcsi.ro>). MCSI este „organul de specialitate al administrației publice centrale în domeniul comunicațiilor

și tehnologiei informației, având scopul de a realiza politica Guvernului României în acest domeniu” ([http : //www.mcsi.ro](http://www.mcsi.ro)).

Misiunea principală este „crearea premiselor durabile trecerii la societatea informațională în România”. Instituții aflate în subordinea MCSI :

- Institutul Național de Studii și Cercetări pentru Comunicații ([http : //www.inscc.ro/](http://www.inscc.ro/)).
- Institutul Național de Cercetare-Dezvoltare în Informatică – ICI București ([http : //www.ici.ro](http://www.ici.ro)).
- Agenția pentru Serviciile Societății Informaționale.
- Societatea Națională de Radiocomunicații SA – RADIOCOM.
- Unitatea de Management de Proiect „Knowledge-Based Economy” ([http : //www.mcsi.ro](http://www.mcsi.ro)).

Menționăm că există și numeroase legături instituționale internaționale ale ministerului, prin care politica internă este permanent conectată la politica Uniunii Europene. În acest mod, se urmărește o permanentă re poziționare a țării noastre pe harta lumii, conferindu-i „un avantaj competitiv pentru o creștere economică durabilă, o convergență rapidă în spațiul european prin incluziune digitală și eliminarea decalajului digital și consolidarea unor sectoare-cheie prioritare pentru România” ([http : //www.mcsi.ro](http://www.mcsi.ro)). Programele derulate prin MCSI încearcă mereu să acopere problematica legată de „Internetul viitorului și perspectivele 5G, conținutul digital bazat pe tehnologii de tip *Big Data* și *cloud computing*, *cyber security* și *cyber intelligence*”. Există planuri strategice anuale și cu perspectivă mai largă (până în 2020), în cadrul cărora educației permanente și e-educației le revine un rol esențial.

În final, în privința e-educației și educației permanente, să spunem că nu cu mult timp în urmă școlile și liceele din România doar își imaginau ce ar putea face tehnologiile digitale pentru elevi și pentru întregul sistem de învățământ. Noțiuni precum teleprezența, interconectarea sau tabla interactivă nici nu existau în vocabularul curent. Conținutul lecțiilor poate fi înregistrat și încărcat pe o platformă online dedicată, pentru a fi ulterior accesat de elevi. Într-adevăr, doar prin intermediul tehnologiei putem face educația accesibilă oricui, oricând și oriunde. Se poate afirma cu certitudine că „multiplicarea și diseminarea bunelor practici, a rezultatelor, a experiențelor didactice și a resurselor educaționale create reprezintă o concluzie și o obligație logică de a continua un demers educațional complex, provocator și inovator” (Camelia Gavrilă, inspector general al IȘJ Iași). Schimbarea a venit, în mare, prin accesarea de fonduri europene și prin implicarea directă a unor mari companii. Să notăm și faptul că *setul de obiective și priorități naționale este bazat pe necesitățile actuale și realitățile societății românești, fiind în același timp în concordanță cu exigențele la nivel european, însoțite de mecanisme de finanțare și mecanisme de cooperare sustenabile*. În strânsă legătură cu cele afirmate, terminologia amintită la început – nici pe departe exhaustivă dacă ținem cont de îmbogățirea (exgerată, credem noi) a limbii române cu neologisme provenind din zona IT – este trecută aici în revistă cu scopul de a fixa un limbaj comun între cititori și autori (limbaj din ce în ce mai greu de

identificat cu precizie) și de a scoate la iveală alte concepte inventate pentru societatea informațională. Astfel, un calculator este o mașină programabilă. Cele două caracteristici principale ale sale sunt :

- „Răspunde” unei mulțimi fixate de instrucțiuni într-un mod bine precizat și bine definit.
- Execută o listă de instrucțiuni (program) încărcate anterior într-o memorie.

Știința calculatoarelor se ocupă cu studiul calculatoarelor și implică atât proiectarea și utilizarea de soft, cât și de hard. Ea este compusă din mai multe discipline, incluzând inteligența artificială sau ingineria soft. În societatea informațională, folosirea tehnicilor științei calculatoarelor este permanentă și profesionistă din partea (în mod ideal) a fiecărui membru al societății. Prin arhitectură se înțelege un domeniu structurat, format din entități (obiecte) și relații (legături între obiecte). Un sistem inteligent de învățare (*Intelligent Tutoring System*) este un concept care definește o nouă viziune asupra relației profesor/„antrenor”-elev/calculator. Arhitectura sa este compusă din patru subarhitecturi :

- baza de cunoștințe existentă inițial, împreună cu contextul acesteia ;
- baza de cunoștințe care va trebui să fie achiziționată pe parcursul utilizării sistemului ;
- informațiile colaterale presupuse a fi fost deja achiziționate ;
- interfața disponibilă dintre (de exemplu) elev și profesor.

Prin mediu inteligent de învățare (*Intelligent Learning Environment*) înțelegem o metodologie generală cu ajutorul căreia elevul este pus într-o situație de tip rezolvare de probleme. Această metodologie este total diferită de modul tradițional de predare, bazat în general pe o secvență de întrebări și răspunsuri (precedate eventual de expuneri și fiind necesare reveniri). Ideea este că elevul trebuie să rezolve singur anumite probleme specifice, primind în decursul acestui proces o asistență profesionistă din partea unui grup de persoane calificate, și anume profesori, cercetători, experți etc. Mediul inteligent de învățare reprezintă o extensie a sistemului inteligent de învățare (fiind sisteme mai bine organizate și consolidate). În cadrul lor mai pot fi înglobate și alte metode moderne de predare. Dintre acestea, nu putem să nu amintim trainingul sau coachingul. Foarte posibile devin clasele electronice, ce reprezintă o generalizare și, în același timp, o concretizare a conceptului de învățământ la distanță, precum și o generalizare a mediului inteligent de învățare. În acest caz, entitățile cunoscute ale învățământului clasic (profesor, elev, clasă) pot avea înțelesuri diferite și trebuie privite drept concepte virtuale. Astfel, profesorii pot fi agenți (cercetători, specialiști în domeniu sau chiar anumite aplicații), iar elevii pot fi și ei, de exemplu, anumite produse soft. Clasele, ca spații fizice de învățământ, s-ar putea să nu existe decât tot virtual, membrii unui asemenea grup sau subgrup putând fi identificați prin anumite codificări (semnături electronice, adrese de poștă electronică etc.).

Structura lucrării și contribuțiile personale ale autorilor

Capitolul 1 este destinat, după cum rezultă și din titlul său, prezentării teoriei curriculumului, instruirii și evaluării în învățământul preuniversitar, cu accent pe disciplina informatică. Se discută metodele de elaborare a ofertei curriculare, a planurilor de învățământ și a programelor școlare. Este subliniată importanța manualelor și a formării limbajului de specialitate al elevilor. În capitolul 2, obiectivele și principiile didactice sunt tratate din punct de vedere metodic, atât la nivel conceptual, cât și particularizat. Ca o continuare normală, în capitolul 3 se face o trecere în revistă a metodelor, tehnicilor și procedeele didactice folosite la nivel global în învățământul clasic, precum și în predarea informaticii moderne. Capitolele 4, 5 și 6 sunt dedicate practic exemplificării conceptelor tratate anterior, prin studiul detaliat al elementelor de bază ale folosirii calculatoarelor (în special la nivel de software): *paradigme de programare, algoritmi esențiali* (în special cei de sortare), metode generale de elaborare a algoritmilor (*divide et impera, backtracking, greedy*, programare dinamică), analiza complexității, corectitudinii și terminării programelor, structuri de date (liste, stive, cozi) și baze de date și, separat, grafuri și arbori. Anexele conțin exemple suplimentare de proiecte didactice, subiecte date în anii anteriori la bacalaureat și admiterea în învățământul superior de specialitate (împreună cu baremele aferente), precum și subiecte date la exemenle de grad și definitivativ (pentru profesorii de informatică ce predau la licee) etc. De asemenea, sunt prezentate resurse (*free*, online) utile pentru desfășurarea în condiții corespunzătoare a procesului didactic (platformele AeL, Cisco, Oracle, alte cărți și manuale), prin furnizarea unor adrese de site-uri pe care le considerăm importante. Bibliografia, deși vastă, este departe de a fi exhaustivă.

Să punctăm faptul că toți cei trei autori au contribuit la conceperea, elaborarea și redactarea textului, dacă nu într-un mod egal, oricum foarte greu de diferențiat explicit. Totuși, fiecare dintre ei are merite mai importante (uneori, chiar în întregime) la crearea anumitor capitole, secțiuni sau porțiuni de text, după cum urmează :

- Ioan Asiminoaei : capitolele 4 și 5, cu excepția secțiunilor privind paradigmele de programare și analiza complexității, corectitudinii și terminării algoritmilor/programelor, al căror autor este Cristian Masalagiu. Contribuții la anexe.
- Cristian Masalagiu : introducerea (secțiunea „Ce este informatica” e scrisă împreună cu Mirela Țibu), capitolul 3 (o bună parte – împreună cu Mirela Țibu), concluziile. Contribuții la anexe.
- Mirela Țibu : capitolul 1 (aproape în totalitate), capitolul 2 (ajutată de Cristian Masalagiu și Ioan Asiminoaei) și capitolul 6. Contribuții la anexe.

Să menționăm și că este foarte posibil ca textele de program/cod să mai conțină erori de redactare (neintenționate). Sperăm ca ele să nu fie atât de sofisticate încât să facă înțelegerea imposibilă (sau distorsionată).

Capitolul 1

Curriculum, instruire, evaluare

În acest capitol vom trata câteva probleme globale ale învățământului noului mileniu, punând accent pe actualitatea din România. Strategiile acceptate astăzi la nivel național pot influența ierarhiile și importanța utilizării *principiilor, metodelor și obiectivelor* (de fapt, *competențelor*) didactice.

1.1. Teoria curriculumului

Să începem prin a spune 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 cerințelor pedagogice, cât și celor ce implică transformarea societății. Una dintre cele mai importante părți ale acestui sistem este curriculumul școlar privind conținuturile învățării. În sensul politicii educaționale, *curriculumul definește sistemul de procese decizionale, manageriale și de monitorizare care precedă, acompaniază și urmează proiectarea, elaborarea, implementarea, evaluarea și revizuirea permanentă și dinamică a setului de experiențe de învățare oferite de școală* ([*7], [*8]).

1.1.1. Curriculumul național în informatică

Conținutul învățământului preuniversitar este asigurat prin *curriculumul național*. Acesta este ansamblul coerent al planurilor-cadru de învățământ, al programelor școlare și al manualelor școlare din învățământul preuniversitar (Legea educației naționale nr. 1/2011, actualizată, conform Anexei 3).

Curriculumul național (numit și *nucleu*) cuprinde sistemul documentelor de tip regulator și normativ în care se consemnează experiențele de învățare recomandate elevilor prin intermediul școlii. Conform acestei accepțiuni, curriculumul desemnează ansamblul experiențelor de învățare pe care școala îl oferă tinerilor, cu scopul de a-i asista în descoperirea și valorificarea maximă a propriilor disponibilități și interese și, în același timp, înseamnă ceea ce întreprind elevii în școală sub îndrumarea profesorilor

în materie de învățare și dezvoltare personală. Curriculumul național reprezintă ansamblul experiențelor de învățare prin care orice instituție școlară asigură realizarea idealului educațional și a finalităților învățământului. Acesta impune în primul rând fixarea *cadruului de referință* ca document regulator, înglobând anumiți indicatori care vor asigura coerența (în termeni de procese și produse) a întregului sistem curricular.

Planul-cadru de învățământ este un document regulator care delimitează ariile curriculare, obiectele de studiu și alocarea de timp minimă și maximă aferente acestora, pe niveluri de învățământ.

Programele școlare stabilesc programele analitice, insistându-se asupra conținutului particular (acestea fiind realizate pe clasele și disciplinele prevăzute în planul-cadru). În consecință, există metodologii de aplicare a lor, reprezentând ghiduri de implementare, reglementări suplimentare etc.

O discuție mai vastă asupra manualelor alternative și asupra curriculumului la alegerea școlii 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 reglatoare, cât și la nivelul programelor, ele fiind prezentate succint (neexhaustiv și, din motive obiective, poate nu în ultima formă) în cele ce urmează. Trebuie să vorbim mai întâi despre profilul de formare al unui absolvent. Acesta ar trebui să sintetizeze principalele cunoștințe, capacități și atitudini dezirabile obținute la capătul parcursului școlar obligatoriu, în concordanță cu așteptările societății față de el. În termeni operaționali, de la un absolvent de învățământ obligatoriu se așteaptă ([Cri], [*8]) :

- 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 și de integrare în medii diverse.
- Să rezolve probleme, să ia decizii și să-și exprime opiniile, folosind gândirea critică și creativă.
- Să folosească în mod eficient tehnologiile relevante pentru viața de toate zilele.
- Să înțeleagă fenomenele esențiale ale naturii înconjurătoare și ale mediului social imediat.
- Să contribuie la structurarea și ocrotirea unei vieți sociale de calitate.
- Să aplice și să-și valorifice propriile experiențe de învățare, în scopul dezvoltării unui ansamblu personal de atitudini și al identificării viitoarei orientări profesionale.
- Să-și formeze capacitățile și motivațiile pentru o învățare permanentă.

Prin *ciclu curricular* se exprimă un concept bazat pe stadiul de dezvoltare psihopedagogică al elevilor și care oferă un set coerent și clar de obiective de învățare, reflectate la nivelul programelor școlare. Specificitatea dominantelor curriculare ale fiecărui ciclu în parte este importantă atât pentru proiectarea curriculumului, cât și pentru profesori, elevi, părinți etc. Curriculumul nucleu și curriculumul la decizia școlii sunt cele două segmente principale care concură la formarea curriculumului național. *Curriculumul nucleu* este (general) obligatoriu pentru toate școlile și toți elevii, reprezentând segmentul prin care învățământul public speră să asigure egalitatea șanselor. Acesta reprezintă

unicul sistem de referință pentru examinarea externă admisă în sistem și constituie baza standardelor naționale de performanță și de evaluare. *Curriculumul la decizia școlii* vizează zona opțională a curriculumului național și se concretizează prin :

- segmentul opțional al disciplinelor obligatorii ;
- disciplinele propriu-zis opționale.

Avantajele acestui mod de abordare a organizării curriculare sunt :

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

În continuare, putem spune că ariile curriculare reprezintă grupaje de discipline, precum și de domenii și obiecte opționale, fiind neschimbate pe întreaga durată a școlii (segmentului școlar). Ponderea lor pe cicluri și clase variază în timp. În acest sens, obiectele de studiu sunt părți ale ariilor curriculare și pot fi obligatorii sau opționale. Programele școlare stabilesc obiectivele și conținuturile propriu-zise ale învățării la nivelul obiectelor de învățământ. Acestea reglează atât procesul de predare-învățare, cât și realizarea manualelor și a altor materiale suport destinate procesului de predare-învățare. Programa școlară 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 școală își poate decide propria schemă orară, în funcție de proiectul curricular pe care-l realizează. Se poate astfel contura personalitatea școlii, într-o societate care-și propune să respecte și să valorizeze diversitatea, în contextul respectării standardelor internaționale, al unei educații de calitate și acordării unor șanse cu adevărat egale tuturor tinerilor.

1.1.2. *Elaborarea ofertei curriculare*

Oferta curriculară generală este o componentă a culturii curriculare naționale, o parte integrantă a curriculumului național. Proiectarea și elaborarea curriculumului nu sunt apanajul unei elite, un sistem educațional puternic trebuind să fie preocupat de formarea inițială și continuă a unui contingent numeros și bine pregătit de cadre didactice capabile să proiecteze și să elaboreze un curriculum. O asemenea ofertă trebuie să urmărească compatibilitatea sistemului de învățământ (românesc) cu alte sisteme de

învățământ performante din lume, în același timp cu mărirea impactului sistemului de învățământ asupra reformei societății românești. Oferta curriculară a fiecărei unități școlare este un atu important pentru rezonanța școlii în microclimatul socio-economic în care s-a integrat. Pornind de la finalitățile fiecărui ciclu de pregătire, oferta curriculară a unei unități școlare trebuie să țină cont de următoarele elemente : nivelul de studiu, profilurile și specializările existente, baza didactico-materială, resursele umane, preferințele părinților și elevilor, specificul local, standardele ocupaționale, contextul sociocultural etc. Decizia privind modul în care va fi abordat acest segment de curriculum aparține deopotrivă ofertanților (școlii, adică profesori, manageri) și beneficiarilor (elevi, părinți, reprezentanți autorizați ai comunității locale). Curriculumul la decizia școlii poate fi astfel orientat spre :

- *Curriculumul nucleu aprofundat* – cuprinde numărul maxim permis de ore din planul-cadru și este conceput pentru o atingere integrală și efectivă a obiectivelor și conținuturilor din trunchiul comun (poate fi util claselor cu un nivel mediu spre slab al pregătirii de specialitate).
- *Curriculumul extins* – cuprinde numărul maxim de ore din plaja orară, dar în scopul extinderii obiectivelor și conținuturilor din trunchiul comun (poate fi util claselor cu performanțe bune la disciplina de specialitate).
- *Curriculumul elaborat în școală* – această componentă permite introducerea de opționale de tip nou, care abordează o tematică diferită, din afara programei obligatorii (util pentru a exploata resursele și tradițiile locale sau pentru a întâmpina cerințele și exigențele de instruire ale elevilor, părinților, comunității etc.).

Putem spune că aspectele formative și informative cuprinse în curriculumul la decizia școlii vor constitui obiectul evaluării interne.

Didactica pentru învățământul gimnazial și liceal se referă, în principal, la obiectivele formative (generale) ale studiului informaticii și legăturii acesteia cu alte discipline, la metodele și mijloacele didactice specifice disciplinei, principiile didactice clasice aplicate în predarea informaticii, planurile de învățământ, programele școlare, manualele școlare ș.a.m.d.

1.1.3. *Planurile de învățământ*

Planurile-cadru precizează disciplinele de învățământ în succesiunea lor pe ani de studiu și tipuri de școli sau niveluri. Pentru fiecare disciplină în parte sunt stabilite numărul de ore pe săptămână și ani de studiu, precum și numărul de ore de aplicații practice de laborator, acolo unde este cazul. Acestea au un caracter unic și obligatoriu pentru fiecare stadiu de pregătire sau tip de școală, cuprinzând obiectele de studiu din fiecare clasă (perioadă de studiu). Într-un moment de reorganizare a mecanismului de achiziții de cunoștințe, conceperea unui curriculum este o întreprindere greu de realizat dacă ținem cont de faptul că se impune renunțarea la lucruri depășite, dar și păstrarea unor soluții viabile. La baza elaborării planurilor-cadru stau următoarele principii (atenție, nu este vorba despre principiile didactice, care vor fi discutate în capitolul 2) :

- a) *Principiul selecției culturale* (alegerea domeniilor cunoașterii și gruparea lor în arii curriculare) vizează armonizarea dintre particularitățile personalității elevului, aptitudinile și interesele sale personale (exprimate prin opțiunea pentru o anumită filieră și specializare), diversitatea domeniilor cunoașterii sau perenitatea componentelor de bază ale personalității și a valorilor asociate acestora, proprii unei societăți democratice.
- b) *Principiul coerenței* vizează caracterul omogen și echilibrat al parcursului școlar, având în vedere integrarea verticală și orizontală a ariilor curriculare în cadrul fiecărei filiere, profil și specializare, această integrare fiind exprimată în raporturile procentuale dintre ariile curriculare și disciplinele de studiu.
- c) *Principiul funcționalității* presupune organizarea parcursului școlar pe cicluri curriculare care să respecte caracteristicile de vârstă, interesele și motivațiile elevilor. Acest principiu, coroborat cu strategiile de organizare internă a curriculumului, a determinat structurarea procesului de învățare pe cicluri curriculare (periodizări ale școlarității) care se suprapun structurii sistemului de învățământ, cu scopul de a focaliza obiectivele majore ale fiecărei etape școlare și de a regla prin modificări curriculare procesul de învățământ. Ciclurile curriculare asigură continuitatea în momentul trecerii de la o treaptă de școlarizare la alta prin conexiuni explicite la nivelul curriculumului, corelarea structurii curriculare cu vârsta psihologică, transferul de metode și procedee didactice.
- d) *Principiul egalizării șanselor* vizează oferta de oportunități echivalente de continuare a școlarizării în condițiile unor parcursuri școlare diferențiate. Acest principiu solicită un raport adecvat între trunchiul comun și disciplinele la decizia școlii.
- 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 capacităților de bază ale viitorului specialist în informatică, dintre care amintim :

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

Descentralizarea curriculară încurajează parcursurile individuale de învățare și spulberă mitul obligativității și uniformității parcurgerii conținuturilor, oferind cadrelor didactice o flexibilitate de decizie și acțiune. Dacă la liceu prezența disciplinelor de informatică în trunchiul comun le conferă un statut sigur, la gimnaziu, abia în 2016 s-a decis ca în Planul-cadru să apară o disciplină obligatorie, Informatică și TIC, prevăzută cu o oră pe săptămână la toate clasele. Școlile gimnaziale vor avea nevoie de resurse logistice și tehnice (precum și de cadre didactice specializate) pentru a putea susține această schimbare. Numai astfel elevii vor avea șanse egale la o educație corectă în domeniul IT, educație esențială într-o societate informatizată.

O prezentare succintă a disciplinelor de informatică ce se regăsesc în prezent în curriculumul național (la liceu) este furnizată în tabelul următor (www.edu.ro).

Tabelul 1.1. Disciplinele de informatică, conform curriculumului național

Filiera	Profilul	Specializarea	Disciplina	Numărul ore/ săptămână	Anii de studiu	Exemple de conținuturi studiate
Teoretică	Real	Matematică- informatică	Informatică	1 (IX-X) 3 (XI-XII) + 3 ore la intensiv	IX-XII	Elaborarea algoritmilor Implementarea algoritmilor în limbaj de programare (C/C++ , Pascal) : algoritmi elementari, structuri de date statice, tehnici și metode de programare, teoria grafurilor Structuri de date dinamice, programare orientată obiect C++ /C# (doar la intensiv)
		Matematică- informatică, Intensiv informatică				Sisteme de gestiune a bazelor de date Software utilizat : CodeBlocks, Free Pascal, Visual Studio, platforma Oracle SQL/Visual Fox
			TIC	1 (IX) 2 (X)	IX-X	Sisteme de calcul și sisteme de operare Comunicarea în medii electronice Elemente de realizare a website-urilor Editoare de texte Foi de calcul Prezentări multimedia Baze de date Access Software utilizat : OpenOffice, Microsoft Office (Word, Access, Power Point, Publisher), Photoshop, Movie Maker Aplicații online de prelucrare de imagini, utilitare free de editare text, grafică, video
	Uman	Științele naturii	Informatică	1	XI-XII	Elaborarea algoritmilor Implementarea algoritmilor în limbaj de programare (C/C++ , Pascal) : algoritmi elementari, structuri de date statice, tehnici de programare
		Științe sociale Filologie	TIC Tehnici de documentare asistată de calculator Tehnoredactare asistată de calculator	1 (IX) 2 (X) 1 (XI-XII) 1 (XI-XII)	IX-X	Analog profilul real XI-XIII – aplicații software dedicate, adaptate specializării (aplicații de birotică și documentare, aplicații multimedia, interfețe Web)

Vocațională	Militar	Matematică-informatică	Informatică	1 (IX-X) 3 (XI- XII)	IX-XII	Analog profilul real
			TIC	1 (IX) 2 (X)	IX-X	Analog profilul real
	Artistic	arhitectură, arte ambientale, design, arte plastice, arte decorative	TIC Procesarea computerizată a imaginii	1 (IX) 2 (X)	IX-XII	Analog profilul real
		Muzică, Arta actorului, Coregrafie	Tehnici de prelucrare audiovideo	1 (XI-XII)		XI-XIII – aplicații software dedicate, adaptate specializării Aplicații multimedia de procesare audiovideo, CorelDRAW, Photoshop, Movie Maker
Tehnologică	Teologic	Toate	TIC	1 (IX) 2 (X)	IX-XII	Analog profilul real
			Sisteme de gestiune a bazelor de date	1 (XI-XII)		XI-XIII – aplicații software de gestiune a bazelor de date, Visual Fox, Microsoft Access
	Tehnic	Toate	Informatică Tehnologii asistate de calculator	2 (IX-XI) 1 (XII-XIII)	IX-XIII	IX-X Analog profilul real
						XI-XIII – aplicații software dedicate, adaptate specializării

1.1.4. *Programele școlare (analitice, curriculare)*

Programele școlare au trecut printr-un proces complex de elaborare și revizuire în viziune curriculară, presupunând o reproiectare interactivă a obiectivelor, conținuturilor, activităților de învățare și a principiilor și metodelor de evaluare. Programele școlare (analitice) stabilesc conținutul disciplinelor (de informatică), pentru fiecare an de studiu și materie, pe niveluri, filiere, profiluri și specializări, precum și pe forme de învățământ. Acestea precizează ce cunoștințe, priceperi, deprinderi trebuie să-și însușească elevii în anul de studiu respectiv și care este succesiunea în care trebuie ele dobândite. Programele sunt elaborate de comisia de specialitate a MENCȘ, 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 la catedră, inspectori școlari din cadrul inspectoratelor județene și al ministerului, psihopedagogi și cercetători de la Institutul de Științe ale Educației (IȘE). Realizate în concordanță cu noile planuri-cadru, urmărind o descongestionare rațională a conținuturilor, actualele programe școlare reprezintă o adevărată revoluție didactică în ceea ce privește 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 școlară adecvată este rezultatul unui exercițiu colectiv, desfășurat 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 și 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 ministerului de resort ;
- prezentarea și argumentarea în fața comisiei de specialitate a ministerului a formei stabilite de 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 și i se aplică modificările necesare. Programele astfel finalizate sunt supuse aprobării CNC. Elaborarea programelor școlare, inclusiv a celor de la disciplinele din curriculumul la decizia școlii, trebuie să îndeplinească anumite cerințe de ordin științific, psihologic, didactic și metodic.

Programa școlară pentru un obiect de studiu trebuie să conțină :

- o notă informativă cu privire la scopurile și obiectivele predării, indicații relative la ordonarea materiei și repartizarea orelor pe capitole, subcapitole, teme ș.a.m.d. ;
- numărul de ore pe săptămână, numărul de ore de laborator (dacă e cazul), numărul de ore alocat recapitulărilor, ore la dispoziția profesorului ș.a.m.d. ;

- competențele-cheie, competențele generale și cele specifice (la clasele V-XII) ;
- valori și aptitudini pe care elevii le vor dobândi după parcurgerea disciplinei ;
- conținuturile de studiat prezentate în concordanță cu competențele vizate, numărul de ore alocate lucrărilor scrise, recapitulărilor, evaluării, ore la dispoziția profesorului ș.a.m.d. ;
- sugestii metodologice ;
- îndrumări cu privire la folosirea manualelor, materialului bibliografic etc.

Programa actuală nu prevede un număr fix de ore pentru fiecare temă în parte, aceasta fiind lăsată la aprecierea profesorului, în funcție de particularitățile claselor și de condițiile specifice de predare. Preocupările privind elaborarea programelor școlare pentru disciplinele de informatică sunt îndreptate spre îmbunătățirea programelor în sensul punerii de acord a conținuturilor cu cerințele sociale. Astfel, se dorește să se realizeze o pregătire a elevilor în direcția satisfacerii cerințelor necesare integrării rapide a absolvenților în activitatea economică. Dinamica conținuturilor este o cerință esențială pentru programa școlară a disciplinelor de informatică, necesară menținerii pasului cu progresele realizate în domeniu. Trebuie să remarcăm și faptul că există o tendință (cu efecte nu tocmai benefice, după opinia noastră) de realizare a unei programe la un nivel științific foarte ridicat și cu un volum de conținuturi foarte amplu. Nu trebuie scăpat din vedere nici un moment faptul că o programă școlară, odată concepută și aprobată, este obligatorie pentru toți elevii, dar nu toți sunt foarte dotați și motivați. Elaborarea curriculumului este însă un proces continuu, care marchează perioade de schimbări profunde și care țintește îndelungi perioade de stabilitate.

Noul curriculum își propune să realizeze stabilitatea printr-un echilibru între componenta națională (care vizează trunchiul comun) și componenta locală (care vizează oferta curriculară a școlii). Elaborarea curriculumului local devine astfel o componentă esențială a activității didactice, o rezultată a eforturilor reunite ale conducerii școlii, cadrelor didactice, elevilor, părinților, precum și ale altor parteneri sociali viabili.

Prezentăm în continuare o comparație între caracteristicile curriculumului oficial, imediat anterior și cele ale curriculumului actual.

Curriculumul anterior :

- A fost centrat pe conținuturi.
- Formularea obiectivelor viza în mod direct atestarea profesională a absolventului.
- Conținuturile învățării erau aceleași pentru toți elevii.
- Absența cooperării între elevi în realizarea unei aplicații era o regulă.
- Existau conținuturi didactice fixe, neadaptabile la resursele locale.

Curriculumul actual permite :

- Centrarea pe raționalizarea activităților de învățare, în funcție de competențele-cheie și de competențele specifice.

- Formularea obiectivelor, realizată în termeni de competențe și de capacități individuale.
- Oferirea unei palete largi de activități în cadrul curriculumului la decizia școlii, prin care elevul își poate acoperi propria sferă de interese.
- Încurajarea cooperării între elevi prin activități de grup cu asumarea de roluri individuale pentru realizarea unei aplicații.
- Adaptarea conținuturilor la resursele locale.

Prezentăm în continuare câteva programe analitice la disciplinele de informatică aflate în vigoare în prezent.

Disciplina informatică – clasa a IX-a
(Programa avizată de MENCS ; [http : //www.edu.ro](http://www.edu.ro))

Filiera teoretică, profil real, specializările : Matematică-informatică intensiv informatică

Filiera vocațională, profil militar, specializarea : Matematică-informatică intensiv informatică

Nota de prezentare

Prezentul document conține programa școlară pentru disciplina informatică, studiată în filiera teoretică, la profilul real, specializarea matematică-informatică, intensiv informatică, precum și la filiera vocațională, profil militar, specializarea matematică-informatică intensiv informatică, prevăzută săptămânal cu *o oră pentru activități teoretice și 3 ore pentru activități practice*, în conformitate cu art. 9. din OMECI 410/16.03.2009.

Studiul disciplinei informatică se va desfășura cu întregul colectiv de elevi ai clasei pentru activitățile teoretice și cu colectivul de elevi organizat pe grupe, obligatoriu în laboratorul de informatică, pentru activitățile practice.

Competențele-cheie europene vizate prin studiul disciplinei

Pe baza rezultatelor studiilor efectuate la nivelul Comisiei Europene au fost stabilite opt competențe-cheie, fiind precizate, pentru fiecare competență-cheie, cunoștințele, deprinderile și aptitudinile care trebuie dobândite, respectiv formate elevilor în procesul educațional.

Aceste competențe-cheie răspund obiectivelor asumate pentru dezvoltarea sistemelor educaționale și de formare profesională în Uniunea Europeană și, ca urmare, stau la baza stabilirii curriculumului pentru educația de bază.

Principalele competențe-cheie europene vizate prin studiul disciplinei sunt : competențe în matematică și competențe de bază în științe și tehnologie ; competențe digitale.

Competențe generale :

1. Identificarea conexiunilor dintre informatică și societate.
2. Identificarea datelor care intervin într-o problemă și a relațiilor dintre acestea.
3. Elaborarea algoritmilor de rezolvare a problemelor.
4. Aplicarea algoritmilor fundamentali în prelucrarea datelor.
5. Implementarea algoritmilor într-un limbaj de programare.

Valori și atitudini :

1. Exprimarea unui mod de gândire creativ în structurarea și rezolvarea problemelor.
2. Conștientizarea impactului social, economic și moral al informaticii.
3. Formarea obișnuințelor de a recurge la concepte și metode informatice de tip algoritmic specifice în abordarea unei varietăți de probleme.
4. Manifestarea unor atitudini favorabile față de știință și de cunoaștere în general.
5. Manifestarea inițiativei și disponibilității de a aborda sarcini variate.

Competențe specifice și conținuturi :

1. Identificarea conexiunilor dintre informatică și societate

Competențe specifice	Conținuturi
1.1. Identificarea aplicațiilor informaticii în viața socială	Definirea informaticii ca știință Rolul informaticii în societate
1.2. Recunoașterea situațiilor în care este necesară prelucrarea algoritmică a informațiilor	Studii de caz ale unor situații sociale, în abordare informatizată

2. Identificarea datelor care intervin într-o problemă și a relațiilor dintre acestea

Competențe specifice	Conținuturi
2.1. Descrierea unei succesiuni de operații prin care se obțin, din datele de intrare, datele de ieșire	Date cu care lucrează algoritmi (constante, variabile, expresii) Operații asupra datelor (aritmetice, logice, relaționale)

3. Elaborarea algoritmilor de rezolvare a problemelor

Competențe specifice	Conținuturi
3.1. Analizarea enunțului unei probleme și stabilirea pașilor de rezolvare a problemei	Etapele rezolvării problemelor. Exemple. Noțiunea de algoritm. Caracteristici. Exemple.
3.2. Reprezentarea algoritmilor în pseudocod	Reprezentarea algoritmilor. Pseudocod.
3.3. Respectarea principiilor programării structurate în procesul de elaborare a algoritmilor	Principiile programării structurate. Structuri de bază : structura liniară, structura alternativă, structura repetitivă.

	<p>Algoritmi elementari.</p> <p>1. Prelucrarea numerelor :</p> <ul style="list-style-type: none"> • prelucrarea cifrelor unui număr (de exemplu, suma cifrelor, testarea proprietății de palindrom etc.) • probleme de divizibilitate (de exemplu, determinarea divizorilor unui număr, determinarea c.m.m.d.c./c.m.m.m.c., testare primalitate etc.) • calculul unor expresii simple (sume, produse etc.) <p>2. Prelucrarea unor secvențe de valori :</p> <ul style="list-style-type: none"> • determinare minim/maxim • verificarea unei proprietăți (de exemplu, dacă toate elementele din secvență sunt numere perfecte etc.) • calculul unor expresii în care intervin valori din secvență (de exemplu : numărarea elementelor pare/impare etc.) • generarea șirurilor recurente (de exemplu : șirul Fibonacci)
--	---

4. Implementarea algoritmilor într-un limbaj de programare

Competențe specifice	Conținuturi
<p>4.1. Transcrierea algoritmilor din pseudocod într-un limbaj de programare</p> <p>4.2. Identificarea necesității structurării datelor în tablouri</p> <p>4.3. Prelucrarea datelor structurate</p> <p>4.4. Utilizarea fișierelor text pentru introducerea datelor și extragerea rezultatelor</p> <p>4.5. Utilizarea unui mediu de programare (pentru limbajul Pascal sau pentru limbajul C/C++)</p>	<p><i>Noțiuni introductive :</i></p> <ul style="list-style-type: none"> • Structura programelor • Vocabularul limbajului • Tipuri simple de date (standard) • Constante, variabile, expresii • Citirea/scrierea datelor <p><i>Structuri de control :</i></p> <ul style="list-style-type: none"> • Structura liniară • Structura alternativă • Structuri repetitive <p><i>Tipuri structurate de date. Tipul tablou :</i></p> <ul style="list-style-type: none"> • Tablouri unidimensionale • Tablouri bidimensionale • Algoritmi fundamentali de prelucrare a datelor structurate în tablouri • căutare secvențială, căutare binară • sortare • interclasare • parcurgerea tablourilor bidimensionale pe linii/coloane <p><i>Fișiere text. Definiție, operații specifice</i></p> <p><i>Mediul limbajului de programare studiat</i></p> <ul style="list-style-type: none"> • Prezentare generală • Editarea programelor-sursă • Compilare, rulare, depanare

5. Implementarea algoritmilor într-un limbaj de programare

Competențe specifice	Conținuturi
5.1. Elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specializării 5.2. Alegerea unui algoritm eficient de rezolvare a unei probleme	<p><i>Aplicații interdisciplinare (specifice profilului)</i> <i>Exemple orientative :</i></p> <ul style="list-style-type: none"> • Rezolvarea ecuației de gradul I și de gradul al II-lea • Simplificarea fracțiilor • Aplicații geometrice (distanța dintre două puncte, aria/perimetrul unui triunghi, volumul corpurilor regulate, centrul de greutate al unei mulțimi de puncte) • Determinarea primilor n termeni ai unei progresii aritmetice/geometrice • Determinarea punctului de intersecție a două mobile în mișcare rectilinie și uniformă • Determinarea masei moleculare a unui compus chimic <p><i>Analiza eficienței unui algoritm</i></p>

Sugestii metodologice

Predarea informaticii va fi orientată spre *rezolvarea de probleme*, utilizându-se preponderent metode activ-participative și punându-se accent pe *analiza problemei*. Pentru buna desfășurare a orelor și aplicarea programei se sugerează următoarele activități de învățare :

- discuții despre activități cotidiene și modelarea acestora sub forma unei secvențe bine definite de pași ;
- combinarea unor operații elementare (pași) pentru obținerea anumitor prelucrări complexe în funcție de scopul propus ;
- explicarea conceptului de algoritm și a caracteristicilor algoritmilor ;
- explicarea diferenței existente între informații care reprezintă date și cele care descriu calea de rezolvare a unei probleme ;
- descrierea unui algoritm în limbaj natural ;
- prezentarea obiectelor cu care operează algoritmii (constante, variabile, expresii) ;
- clasificarea datelor în date de intrare, de ieșire și date de manevră ;
- descrierea etapelor rezolvării unei probleme din punct de vedere algoritmic ;
- prezentarea schemelor logice ca modalitate de reprezentare a algoritmilor ;
- prezentarea structurilor de bază în pseudocod ;
- exersarea scrierii unor algoritmi simpli, folosind structuri liniare, alternative și repetitive ;
- identificarea unor situații în care alegerea unui algoritm prezintă avantaje în raport cu alegerea altuia ;
- exersarea creării și aplicării algoritmilor pentru rezolvarea unor probleme întâlnite de elevi în studiul altor discipline școlare ;

- evidențierea greșelilor tipice în elaborarea algoritmilor ;
- prezentarea unor exemple de implementare într-un limbaj de programare a unor algoritmi elaborați de elevi și executarea acestora pe calculator ;
- prezentarea mediului de programare (facilități de editare, de compilare și de rulare) ;
- prezentarea și exemplificarea elementelor de bază ale limbajului de programare ;
- activități de dezvoltare a deprinderilor de organizare a informațiilor în tablouri ;
- exersarea citirii datelor de la tastatură și a afișării datelor pe ecran ;
- exersarea scrierii unor programe simple ;
- codificarea structurilor de control învățate în limbaj de programare ;
- proiectarea unui algoritm și implementarea acestuia ;
- folosirea facilităților mediului în depanarea programelor ;
- implementarea structurilor de tip tablou ;
- exerciții de transfer al datelor din/în fișiere text ;
- evidențierea analogiilor și diferențelor între citire/scriere utilizând dispozitivele standard de intrare/ieșire și fișiere text ;
- testarea și analiza comportamentului programelor pentru diferite date de intrare ;
- încurajarea discuțiilor purtate între elevi, exprimarea și ascultarea părerilor fiecăruia.

1.2. Teoria instruirii

1.2.1. *Manualele școlare*

Crearea pieței libere a manualelor școlare și trecerea de la manualul unic la cel alternativ au însemnat un pas înainte prin înlăturarea unei politici de monopol. Dorința de a realiza (cel puțin) trei manuale alternative la fiecare disciplină este justificată de intenția de a atinge un standard minim de diversitate și calitate.

Manualul școlar reprezintă mijlocul didactic de bază folosit în procesul de învățământ, este profesorul la purtător al elevului și principalul material bibliografic al acestuia (deși în informatică lucrurile nu stau întotdeauna așa). Manualul exemplifică, printre altele, conținutul detaliat al programelor școlare, funcția lui principală fiind aceea de informare a elevului, mijlocul său principal de documentare. Autorii de manuale trebuie să țină seama că acestea ar trebui nu numai să-l ajute pe elev să învețe (informatică), ci și să-l obișnuiască cu munca/studiul individual. Manualul trebuie să îndrume elevul spre o gândire independentă și să-l îndemne la continuarea efortului creator. În același timp nu trebuie să inhibe sau să orienteze (voit sau nu) în anumite direcții.

O mare parte dintre funcțiile manualului pot fi preluate, în învățământul de informatică, de către calculator. Acesta poate fi privit și ca manual, și ca profesor, exercitându-și atât funcția de comunicare de cunoștințe, cât și cea de verificare a acestora, dar nu încă în mod exhaustiv, și cea de evaluare permanentă a gradului (relativ) de

acumulare a cunoștințelor. Dacă discutăm despre documentațiile de firmă, opinăm că acestea nu se ridică (și nici nu trebuie) la nivelul didactic al manualelor școlare, rolul lor fiind pur informativ și destinat nu neapărat specialiștilor sau viitorilor cunoscători. Tehnicile de învățare și evaluare a nivelului atins, corectarea deprinderilor și completarea cunoștințelor (simultan cu descoperirea lipsurilor) nu pot fi realizate decât de profesor. Manualul are încă un rol deosebit, și anume acela de *măsură a gradului de profunzime în abordarea noțiunilor, precum și a domeniului ca atare*. El ne poate indica până la ce grad de complexitate și detaliu trebuie întreprins demersul didactic. Apariția manualelor alternative scoate în evidență, o dată în plus, diversitatea punctelor de vedere în această privință. Rolul manualului ca mijloc de comunicare de cunoștințe se diminuează continuu în învățământul modern, locul lui fiind luat de alte mijloace didactice, mai eficiente și mai atractive: mijloacele audiovizuale specifice informaticii, calculatoarele cu echipamente periferice speciale, sistemele multimedia, utilizarea unor suporturi de mare capacitate și cu posibilități rapide de acces și de (re)găsire a informației (casete audiovideo, CD-uri, DVD-uri, teletext, Internet, telefonie mobilă etc.). Credem că profesorul și manualul, ca surse didactice consacrate, nu pot fi decât parțial înlocuiți. Toate mijloacele anterior enumerate sunt doar auxiliari mai mult sau mai puțin eficienți, în funcție de domeniul și disciplina abordate. Transformările societății românești din ultimii ani, dezvoltarea și răspândirea informaticii impun o pregătire diversificată a tinerilor în acest domeniu.

Disciplina informatică – din cadrul profilului matematică-informatică – trebuie astfel să asigure dobândirea unor cunoștințe de informatică la nivel de cultură generală, necesare continuării studiului, și a unor cunoștințe cu caracter aplicativ utile în societatea în care trăim. Pornind de la faptul că nu există domeniu de activitate unde să nu se prelucreze și să nu se transmită informații atât în domeniul respectiv, cât și spre exteriorul lui, informația este foarte prețioasă, ea trebuie stocată, prelucrată și transmisă în condiții care să asigure corectitudine și exactitate, adică la un nivel profesional. Indiferent de profesia pe care o va alege un tânăr, cu siguranță va avea nevoie de cunoașterea modului de utilizare a unui instrumentar informatic. Volumul cunoștințelor și deprinderilor necesare va depinde, desigur, de domeniu, de exigențele și cerințele concrete. Este însă o nevoie stringentă de inițiere a tinerilor din toate școlile în utilizarea calculatoarelor la un nivel profesional, pe care azi îl numim doar nivel de cultură generală.

Dezvoltarea gândirii algoritmice este un prim obiectiv la realizarea căruia informatica își aduce o contribuție esențială și eficientă. Asemenea matematicii, informatica *dezvoltă gândirea* (raționamentul), care în școală, dar și în viața de zi cu zi are un rol esențial în procesul de învățare, în formarea caracterului și a personalității. Aceasta nu se leagă doar de cunoștințele de programare, ci și (așa cum am menționat deja) de cunoștințele referitoare la gestionarea bazelor de date, la utilizarea editoarelor de texte etc. Prin specificul ei, informatica este esențial legată de lucrul individual cu un calculator și contribuie la *dezvoltarea deprinderii de a lucra individual*. Pe de altă parte, prin intermediul rețelelor de calculatoare este posibil un schimb de informații mult mai eficient decât prin orice altă metodă clasică. *Educarea elevilor*

în spiritul unei activități desfășurate în grup, în colaborare, se finalizează prin predarea *informaticii orientate pe proiecte*. Realizarea unor aplicații complexe impune lucrul în grup, modularizarea programului și păstrarea contactelor cu ceilalți membri ai grupului. În școală se pot crea condiții similare lucrului din viața reală, unde activitățile nu se desfășoară izolat. Aplicațiile, proiectele, dar și producția propriu-zisă sunt întrepătrunse cu o serie de faze de lucru în care calculatorul este un instrument de neînlocuit. Obișnuirea elevilor cu responsabilități privind finalizarea propriei munci și asigurarea înlănțuirii unor elemente realizate în paralel îi va pregăti pentru o activitate pe care cu siguranță o vor întâlni în viitor. Educarea elevilor pentru realizarea unor produse utilizabile, pentru dezvoltarea spiritului inventiv și creator apare ca un obiectiv impus de sistemul economic în care trăim. Indiferent de conținutul aplicației, ceea ce realizează elevul trebuie să funcționeze, trebuie să fie utilizabil. Altfel spus, trebuie să aibă toate calitățile unui produs comercial.

Datorită implicației pe care o are azi informatica în toate profesiunile, rezultă caracterul ei interdisciplinar. Informatica nu poate fi privită numai ca o disciplină independentă și nu poate fi ținută între bariere create artificial. În diverse domenii de activitate, rezolvarea problemelor concrete impune foarte des o fază de modelare. Informatica este printre puținele discipline care oferă un instrumentar adecvat pentru învățarea modelării. De asemenea, pune la dispoziție cele mai spectaculoase posibilități de simulare virtuală, care este o parte a modelării (neclasică și necostisitoare). Elevii trebuie să înțeleagă conexiunile dintre informatică și societate și să fie capabili să se adapteze dinamicii schimbărilor determinate de aceste conexiuni.

Manualele școlare avizate de Ministerul Educației și utilizate sau recomandate în prezent în funcție de profil și specializare

I. Filiera teoretică

1. Profilul real

a) Specializarea matematică-informatică

Manuale aprobate de minister :

- Mariana Miloșescu, *Manual pentru clasele IX-X*. București : EDP, 2004.
- Emanuela Cerchez, Marinela Paul Șerban, *Informatică – Manual pentru clasa a IX-a*. Iași : Polirom, 2006-2013.
- Sorin Tudor, *Manual de informatică pentru clasa a IX-a (Pascal și C++)*, *profilul real*. București : L&S Infomat, 2009.
- Sorin Tudor, Vlad Tudor (Huțanu), *Manual de informatică pentru clasa a X-a (Pascal și C++)*, *profilul real*. București : L&S Infomat, 2012.
- Sorin Tudor, Vlad Tudor (Huțanu), *Manual de informatică pentru clasa a XI-a (Pascal și C++)*, *profilul real*. București : L&S Infomat, 2006.

Auxiliare și culegeri :

- Carmen Popescu, Vlad Tudor (Huțanu), *Tehnologia Informației și a Comunicării (TIC). Competențe digitale* (vol. I-II). București : L&S Infomat, 2014.

Platforme de învățare online :

- Sisteme de gestiune a bazelor de date Oracle/SQL (clasa a XII-a) : <http://academy.oracle.com>.
- Software e-learning : AeL.

b) Specializarea matematică-informatică, intensiv informatică

Manuale aprobate de minister :

- Sorin Tudor, Vlad Tudor (Huțanu), *Manual de informatică pentru clasa a IX-a (Pascal și C++)*, profilul real (intensiv). București : L&S Infomat, 2009.
- Sorin Tudor, Vlad Tudor (Huțanu), *Manual de informatică pentru clasa a X-a (Pascal și C++)*, profilul real (intensiv). București : L&S Infomat, 2012.
- Sorin Tudor, Vlad Tudor (Huțanu), *Manual de informatică pentru clasa a XI-a (Pascal și C++)*, profilul real (intensiv). București : L&S Infomat, 2006.
- Mariana Miloșescu, *Informatică. Manual pentru clasa a IX-a, a X-a (intensiv)*. București : EDP, 2004.
- Sanda Junea, Adriana Simulescu, *Informatică C++. Manual pentru clasa a XI-a (intensiv)*. Târgoviște : Gimnasium, 2006.
- Carmen Popescu, *Manual de informatică pentru clasa a XII-a (Oracle)*. București : L&S Infomat, 2007.
- Carmen Popescu, Vlad Tudor (Huțanu), *Manual de informatică pentru clasa a XII-a*, profilul real intensiv. București : L&S Soft, 2007.
- Carmen Popescu, Vlad Tudor (Huțanu), *Tehnologia Informației și a Comunicării (TIC). Competențe digitale* (vol. I-II). București : L&S Infomat, 2014.
- Mariana Panțiru, *Tehnologia Informației și a Comunicării (TIC) Manual pentru clasele IX-X*. București : BIC ALL, 2006.
- Radu Boriga, Vlad Huțanu, Carmen Popescu, *Tehnologia Informației și a Comunicării (TIC). Manual pentru clasa a XII-a*. București : L&S Infomat, 2011.

Culegeri și auxiliare :

- Emanuela Cerchez, Marinel Paul Șerban, *Programarea în limbajul C/C++ pentru liceu* (vol. I-IV). Iași : Polirom, 2006-2013.
- Dana Lica, Mircea Pașoi, *Fundamentele programării, culegere de probleme* (clasele a IX-a, a X-a, a XI-a). București : L&S Soft, 2012.

Platforme de învățare online :

- Sisteme de gestiune a bazelor de date Oracle/SQL (clasa a XII-a) : <http://academy.oracle.com>.
- Software e-learning : AeL.

Cărți în format electronic disponibile pe <http://www.infobits.ro> :

- *Bacalaureat 2015. Subiecte rezolvate*.
- Cătălin Frâncu, *Psihologia concursurilor de informatică*.
- Victor Mitrană, *Bioinformatică*.
- Sorin Tudor, Vlad Tudor, *Bazele programării în Java*.

c) Specializarea științe ale naturii

Manuale aprobate de minister :

- Mariana Miloșescu, *Informatică. Manual pentru clasele IX-X*. București : Editura Didactică și Pedagogică, 2004.
- Emanuela Cerchez, Marinel Paul Șerban, *Informatică. Manual pentru clasa a IX-a*. Iași : Polirom, 2006-2013.
- Sorin Tudor, *Manual de informatică pentru clasa a IX-a (Pascal și C++), profilul real*. București : L&S Infomat, 2009.
- Sorin Tudor, Vlad Tudor (Huțanu), *Manual de informatică pentru clasa a X-a (Pascal și C++), profilul real*. București : L&S Infomat, 2012.
- Sorin Tudor, Vlad Tudor (Huțanu), *Manual de informatică pentru clasa a XI-a (Pascal și C++), profilul real*. București : L&S Infomat, 2006.
- Mariana Panțîru, *Tehnologia Informației și a Comunicațiilor. Manual pentru clasele IX-X, TIC3 (XI), TIC4 (XII)*. București : BIC ALL, 2006.

Auxiliare și culegeri :

- Carmen Popescu, Vlad Tudor (Huțanu), *Tehnologia Informației și a Comunicării (TIC). Competențe digitale* (vol. I-II). București : L&S Infomat, 2014.

Platforme de învățare online :

- Sisteme de gestiune a bazelor de date Oracle/SQL (clasa a XII-a) : <http://academy.oracle.com>.

Software e-learning : AeL.

2. Profilul uman. Specializări : științe sociale, filologie

Manuale aprobate de minister :

- Mihaela Garabet, Ion Neacșu, *Tehnologia Informației și a Comunicațiilor TIC2, tehnici de documentare, tehnoredactare asistată de calculator (clasa a XII-a)*. București : BIC ALL, 2007.
- Mihaela Garabet, Ion Neacșu, *Tehnologia Informației și a Comunicațiilor TIC3, tehnoredactare asistată de calculator (clasa a XII-a)*. București : BIC ALL, 2007.
- Mariana Panțîru, *Tehnologia Informației și a Comunicațiilor. Manual pentru clasele IX-X, TIC3 (XI), TIC4(XII)*. București : BIC ALL, 2006.
- Daniela Marcu, Ovidiu Marcu, *Tehnologia Informației și a Comunicațiilor TIC3, tehnoredactare asistată de calculator (clasa a XII-a)*. București : Editura GIL, 2007.
- Mariana Miloșescu, *Tehnologia Informației și a Comunicațiilor TIC4. Manual pentru clasele XI-XII*. București : Editura Didactică și Pedagogică, 2006.

Auxiliare și culegeri :

- Carmen Popescu, Vlad Tudor (Huțanu), *Tehnologia Informației și a Comunicării (TIC). Competențe digitale* (vol. I-II). București : L&S Infomat, 2014.

Platforme software e-learning : AeL

II. Filiera tehnologică, toate profilurile, toate specializările

Manuale aprobate de minister :

- Mariana Miloșescu, *Tehnologia Informației și a Comunicării. TIC4. Manual pentru clasele XI -XII*. București : Editura Didactică și Pedagogică, 2006.

- Mariana Panțiru, *Tehnologia Informației și a Comunicării (TIC)*. Manual pentru clasele IX-X, *TIC3 (XI)*, *TIC4 (XII)*. București : BIC ALL, 2006.

Auxiliare și culegeri :

- Carmen Popescu, Vlad Tudor (Huțanu), *Tehnologia Informației și a Comunicării (TIC)*. *Competențe digitale* (vol. I-II). București : L&S Infomat, 2014.

Platforme software e-learning : AeL.

III. Filiera vocațională, toate profilurile, toate specializările

Manuale aprobate de minister :

- Mariana Miloșescu, *Tehnologia Informației și a Comunicării (TIC4)*. *Manual pentru clasele XI-XII*. București : Editura Didactică și Pedagogică, 2006.
- Mariana Panțiru, *Tehnologia Informației și a Comunicării (TIC)*. *Manual pentru clasele IX-X, TIC3 (XI)*, *TIC4 (XII)*. București : BIC ALL, 2006.

Auxiliare și culegeri :

- Carmen Popescu, Vlad Tudor (Huțanu), *Tehnologia Informației și a Comunicării (TIC)*. *Competențe digitale* (vol. I-II). București : L&S Infomat, 2014.
- Utilizarea aplicațiilor free, disponibile online, pentru prelucrare audiovideo.

Pentru disciplina tehnologia informației și comunicării (TIC) din curriculumul nucleu (trunchiul comun), conținuturile corespunzătoare cerințelor probei de competențe digitale din cadrul examenului de bacalaureat sunt disponibile online la adresa [http : // competentedigitale.ro](http://competentedigitale.ro), o resursă accesibilă și amplă pentru activitatea de predare-învățare.

În urma implementării Programului SEI (Sistem Educațional Informatizat) în învățământul preuniversitar românesc, 13.000 de școli au fost dotate cu rețele de calculatoare și au fost distribuite pachete de lecții multimedia interactive (peste 500 de lecții la discipline variate, printre care și informatica). Lecțiile au fost avizate de comisii de specialitate din cadrul MENCȘ și integrate într-o platformă AeL (Advanced e-Learning) instalată pe rețele de calculatoare din toate liceele. Lecțiile sunt disponibile online pentru elevi și profesori pe portalul educațional SEI, [http : //portal.edu.ro/](http://portal.edu.ro/).

Metodologia de aplicare a programei pentru disciplina informatică trebuie să țină cont de faptul că studiul ei la profilul matematică-informatică are atât un caracter teoretic, cât și practic, fiind organizat după cum urmează :

- în trunchiul comun, din totalul de (două) ore aprobate, o oră cu caracter teoretic se poate desfășura în clasă sau în laborator, cu întregul colectiv al clasei, iar a doua oră, cu caracter practic, se va desfășura în laboratoarele de informatică, pe grupe de 10-15 elevi, fiecare grupă fiind asistată de câte un profesor ;
- în curriculumul la decizia școlii, orele se vor organiza în laborator cu întreaga clasă.

Profilul matematică-informatică poate funcționa în licee care dispun de cel puțin un laborator de informatică dotat corespunzător. Numărul de laboratoare trebuie să

asigure acoperirea orelor de laborator solicitate atât de trunchiul comun, cât și de curriculumul la decizia școlii. Programa pentru disciplina informatică, profilul matematică-informatică, este orientată pe competențe, profesorul având posibilitatea de a alege activitățile specifice atingerii acestora. Conținutul învățării pentru curriculumul obligatoriu este conceput astfel încât să asigure un bagaj minim de cunoștințe și deprinderi din domeniul informaticii, în timp ce curriculumul la decizia școlii 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.

1.2.2. Structura lecției

Lecția este 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 atât aplicarea metodelor alese de profesor, cât și competențele pe care acesta își propune să le formeze. Nu poate fi considerată lecție ceva care nu *leagă* ceea ce s-a studiat înainte, cunoștințele dobândite anterior, de cunoștințele care trebuie 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. Așa 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 cunoștințe abordate pe întreg cuprinsul lecției, și o *fixare*, prin care să se finalizeze activitatea întreprinsă. Ar trebui anticipate necesitatea introducerii unor noi noțiuni și planul de abordare a lecțiilor următoare. Considerăm că o asemenea *unitate de învățare ar trebui să dureze 90-100 de minute, fără întreruperi*. Lecția nu este numai o formă de organizare a activității de predare-învățare, ci și o succesiune de etape bine stabilite și (de dorit) realizate. Evenimentele imprevizibile, apariția unor particularități specifice care trebuie stăpânite sunt inevitabile. O cerință metodică este clasificarea lecțiilor : de comunicare/transmitere de cunoștințe, de studiu individual, de descoperire, de verificare, de recapitulare etc. Delimitările nu sunt însă stricte, fiecare lecție fiind o împletire (care se dorește armonioasă) de metode și tehnici ce concură la realizarea obiectivelor propuse, raportul în favoarea uneia sau alteia dintre metode fiind greu de stabilit în final și cu atât mai mult inițial. Vom puncta totuși câteva momente esențiale ale desfășurării unei lecții, subliniind relativitatea acestora (ca timp, importanță, ordine) :

- Momentul organizatoric impune, în primul rând, verificarea prezenței și a condițiilor de desfășurare (existența materialului didactic necesar, incluzând aici calculatoare, soft etc.). Ideal ar fi ca aceasta să se facă în pauza dintre ore și de către un personal specializat. Din acest motiv, *pauzele ar trebui să fie de minimum 20 de minute*. Ideală ar fi verificarea temei pentru acasă și 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 are loc fixarea unor cunoștințe anterioare).
- Se fixează cunoștințele (noi) prin (alte) exerciții.
- Se stabilește tema pentru acasă.

O lecție poate fi apreciată ca necorespunzătoare dacă, de exemplu, se „pierde timpul” cu momentul organizatoric, inclusiv cu verificarea temei și cu măsurile luate de profesor în legătură cu neefectuarea acesteia. Cel mai mult timp trebuie afectat comunicării cunoștințelor noi și fixării acestora prin exerciții. Tema pentru acasă nu trebuie dată în grabă (în pauză sau când se sună).

Observație

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 exerciții duce la lucru de mântuială, copieri, abandonarea întregii teme, refuz față de abordarea temei. Tema trebuie să fie pe măsura posibilităților elevilor și legată de însușirea și aplicarea cunoștințelor predate. Ea trebuie dată diferențiat, atunci când între elevii aceleiași clase există diferențe mari în ceea ce privește capacitatea sau pregătirea lor. Tema trebuie să fie însoțită de explicații ajutoare, de indicații potrivite. Când tema presupune artificii de calcul sau cere o pricepere deosebită, trebuie ca elevilor să li se atragă atenția asupra acestui aspect (de exemplu, prin exerciții marcate cu *, adică dificile). Mulți elevi învață pe de rost metodele de rezolvare a unor probleme și își formează șabloane pe care le aplică automat. Cu siguranță că și algoritmi importanți, rezultat al analizei și cercetării îndelungate, trebuie reținuți, dar uzându-se de logica internă a acestora. Profesorul are (și) obligația să-i învețe pe elevi cum să-și facă tema, nu să creeze un climat care impune angajarea unui mediator (mediator păgubos între elev și profesor și nu o prelungire a acestuia din urmă, în cazuri extreme). Temele pentru acasă își ating scopul doar dacă pot fi controlate în permanență de către profesor.

1.2.3. Calitatea cunoștințelor asimilate

Procesul de comunicare a cunoștințelor trebuie să aibă ca efect formarea de reprezentări corecte despre lucruri și fenomene reale, însușirea de noțiuni ce ajută la înțelegerea legilor care reglementează raporturile dintre fenomenele realității și care permit exprimarea acestor raporturi într-un mod clar. Formalismul excesiv este unul dintre pericolele care pândesc procesul instructiv-educativ și el se manifestă prin :

- *Lipsa* legăturii evident exprimate dintre formă și conținut.

- *Memorarea* mecanică a cunoștințelor și predominarea formei exterioare asupra esenței conținutului (schimbarea notației poate provoca uneori adevărate tragedii).
- *Predominarea* memorării asupra înțelegerii.
- *Supremația* șablonului asupra inventivității.
- *Ruperea* teoriei de practică.

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

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, depun un efort suplimentar învățând pur și simplu pe de rost programele făcute în clasă. Dacă acest viciu de tehnică de învățare nu este depistat la timp și înlăturat, în special prin scrierea de către elev sub supravegherea profesorului a unor programe simple (dar altele de fiecare dată), cu greu va mai putea fi corectat.

1.2.4. *Formarea limbajului de specialitate*

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

- Exprimarea corectă a profesorului însuși, care constituie (*ab initio*) un model pentru elevi. De aceea, profesorul trebuie să-și formuleze cu grijă afirmațiile.
- Supravegherea permanentă a exprimării elevilor și corectarea continuă a greșelilor lor.
- Încurajarea libertății de exprimare, cu argumentarea raționamentelor. Deseori, elevii răspund telegrafic sau numai „încep” să se exprime. Cum aceștia o fac mai greoi, ei sunt uneori întrerupți și profesorul continuă ideea formulând-o prin prisma înțelegerii și raționamentului său. Acest mod de abordare a dialogului elev-profesor are efecte negative în legătură cu formarea limbajului de specialitate și utilizarea

lui de către elev. În plus, prin intervenția prematură a profesorului, elevului i se întrerupe firul raționamentelor, el făcând cu greu față efortului de a urmări și înțelege raționamentul profesorului.

Este absolut necesară asigurarea unei anumite „libertăți individuale”, chiar cu riscul unor confuzii momentane. Putem vorbi astfel mai în amănunt despre *exprimarea fluentă în limbajul de specialitate și exercițiul oral*. Prezentarea orală a soluțiilor unor probleme înainte de abordarea lor strict științifică are menirea de a lămurii în totalitate aspectele neclare ale problemelor. Limbajul natural este o formă des uzitată de reprezentare a algoritmilor – prin urmare, o prezentare în limbaj natural a oricărei activități ce urmează a fi desfășurată clarifică și ușurează multe situații-limită. *Exercițiul oral* are o însemnătate deosebită din punct de vedere educativ, el educând atenția, capacitatea de concentrare, prezența de spirit, inițiativa creatoare. Exercițiile orale îmbunătățesc randamentul multor activități și contribuie în mod esențial la formarea limbajului de specialitate. Sunt necesare totuși anumite cerințe și precauții în folosirea exercițiilor orale. Astfel, acestea trebuie :

- să fie alese cu grijă, astfel încât să nu presupună un nivel ridicat de abstractizare sau acumularea unui volum mare de informații noi ;
- să fie prezentate gradat ;
- să nu presupună calcule mentale lungi și complicate ;
- să poată fi folosite cu un bogat material intuitiv/ilustrativ.

În final, profesorul trebuie să pună accent pe aspectele care riscă să devină ambigue.

1.2.5. *Caietele elevilor*

De ce notează elevii în caiete ? De regulă, elevii și profesorii acceptă că există un caiet care conține partea teoretică și aplicațiile ilustrative, iar un alt caiet este destinat exercițiilor individuale. Dacă există manual, la predare, elevii trebuie să noteze doar exemplele ilustrative, și nu partea teoretică. Ei notează doar concluzii și o schemă simplificată a lecției. 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.). Explicațiile orale lungi și complicate, chiar dacă trebuie să se evite a fi scrise în caiete, își au rolul lor. Astfel de notări sunt grele pentru elevii din clasele mici, iar efortul lor se canalizează în direcția notării, și nu a înțelegerii noțiunilor predate. Trebuie exclusă ideea copierii textelor din manuale pe caiete, exceptând situația în care se realizează o sinteză și o sistematizare a lecției din manual. O atenție specială trebuie acordată *Caietului de aplicații practice de laborator*. Datorită caracterului aplicativ al anumitor ore, există tendința de a se nota puțin și de cele mai multe ori

secvențe izolate și necorelate între ele. Un caiet de aplicații practice de laborator ar trebui să conțină la fiecare lecție :

- un rezumat al cunoștințelor teoretice necesare realizării aplicației 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 cunoștințelor nou-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 totuși biblioteca la purtător a elevului, cel mai rapid accesibilă, cu condiția ca programele să fie însoțite de explicații corespunzătoare. Sursele programelor fără enunțul problemelor și 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 și reprezentând o implementare verificată.

Este recomandată și constituirea unui portofoliu în format electronic care să conțină toate aplicațiile realizate în cadrul orelor de laborator, portofoliu bine sistematizat (și, eventual, comentat), la fiecare lecție.

1.3. Teoria evaluării

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

1.3.1. *Repetare, recapitulare, evaluare*

Repetarea materiei parcurse servește la îmborsăritarea cunoștințelor dobândite, ajută la formarea de noi corelații, la reluarea materiei predate într-un cadru mai general, uneori cu completarea unor cunoștințe. Repetarea lecției sau chiar a lecțiilor anterioare se poate face înainte de predarea unei lecții noi, la sfârșitul unui capitol, al unui semestru sau al anului școlar sau chiar premergător susținerii unui examen suplimentar. Recapitularea ar trebui să se realizeze după un plan dinainte stabilit.

Procesul de învățământ se desfășoară într-un cadru organizat și bine definit și cuprinde următoarele etape : *predare, învățare, evaluare*. Deși etapele se desfășoară separat, proiectarea lecțiilor nu poate fi făcută fără a avea în vedere toate aspectele legate de acestea, ele întrepătrundându-se. După cum știm, proiectarea unei lecții

începe cu stabilirea obiectivelor și are la bază programa școlară a disciplinei. Profesorul trebuie să se încadreze în numărul de ore stabilit prin programă.

Evaluarea este o componentă foarte importantă a procesului de învățământ. A evalua rezultatele școlare înseamnă a determina, a cuantifica măsura în care obiectivele programului de instruire au fost atinse, precum și eficiența metodelor de predare-învățare folosite – pe scurt, înseamnă a evalua randamentul școlar. Acțiunile efectuate în procesul de evaluare se referă la măsurare/cuantificare, interpretarea rezultatelor și, evident, la adoptarea unor decizii corespunzătoare. Activitatea de măsurare sau cuantificare se realizează cu ajutorul unor procedee specifice cunoscute sub numele de metode și instrumente de evaluare. Interpretarea și aprecierea rezultatelor evaluării sunt strâns legate de metodele și instrumentele de evaluare folosite, precum și de factori externi ce țin de vârsta elevilor, mediul de dezvoltare a acestora etc. În mod normal, aprecierea rezultatelor evaluării va conține două elemente importante: elevii evaluați, pe de o parte, disciplina și profesorul titular, pe de altă parte. De fapt, evaluarea, componentă esențială a procesului de învățământ, îndeplinește funcții bine definite:

- *Funcția de constatare și diagnosticare* a performanțelor obținute de elevi, explicate prin factorii și condițiile care au condus la succesul sau insuccesul școlar și care sunt de o mare diversitate (psihologică, pedagogică, socială etc.). Permite depistarea lacunelor și greșelilor elevilor, precum și înlăturarea acestora la timp.
- *Funcția de reglare și perfecționare* a metodologiei instruirii pe baza informațiilor obținute din explicarea factorilor și condițiilor care au determinat rezultatele la învățătură.
- *Funcția de predicție și decizie* vizează desfășurarea viitoarelor activități didactice și performanțele ulterioare ale elevilor.
- *Funcția de selecție și clasificare* a elevilor în raport cu rezultatele școlare obținute, aceasta permițând clasificarea și/sau ierarhizarea elevilor.
- *Funcția formativ-educativă*, de ameliorare a metodelor de învățare folosite de elevi, de stimulare și optimizare a învățării.
- *Funcția de perfecționare* a întregului sistem de învățământ.

Ca orice altă activitate didactică, evaluarea cunoștințelor elevilor trebuie să respecte normele impuse de minister. În general, aceste norme (directive, prevederi) stipulează o evaluare ritmică pe parcursul semestrelor. În funcție de vârsta și particularitățile psihointellectuale ale elevilor și de specificul fiecărei discipline, instrumentele de evaluare pot fi:

- a) probe (lucrări) scrise;
- b) probe (verificări) orale;
- c) activități practice;
- d) referate și proiecte;
- e) interviuri;
- f) alte instrumente stabilite de catedre/comisiile metodice și aprobate de director sau elaborate de minister sau de inspectorate.

Aplicarea uneia sau alteia dintre formele de evaluare depinde în mare măsură de forma răspunsului și de caracteristicile disciplinei respective. Cele mai răspândite forme de evaluare sunt cele orale și 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 și dezavantaje de fiecare parte (în cadrul acestei comparații intervin atât personalitatea cadrului didactic, cât și specificul disciplinei). O disciplină tehnică impune în general un mod direct de gândire și exprimare, înțeles uneori ca un *mod algoritmic* de prezentare a cunoștințelor, în timp ce o disciplină „netehnică” cere abilități suplimentare de exprimare a cunoștințelor. La nivel de discuții, aceste comparații (evaluare orală, evaluare scrisă) par să fie normale și corecte, dar practica demonstrează că nu putem renunța la nici una dintre ele în favoarea alteia.

Verificarea orală, cel mai frecvent folosită, are anumite avantaje care o impun. În primul rând, favorizează dialogul, elevul putând să-și argumenteze răspunsurile și să participe la o confruntare de idei cu întreaga clasă, iar profesorul poate detecta cu ușurință erorile și poate interveni și corecta „pe loc”. Verificarea orală are însă și numeroase limite: întrebările nu au toate același grad de dificultate; unii elevi sunt emotivi și se blochează (mai ales atunci când sunt ironizați de profesor); răspunsurile lor stârnesc ilaritate în clasă; timpul nu permite o verificare completă a conținutului predat. Mai mult, comportamentul și starea psihică a profesorului pot influența notarea. Majoritatea specialiștilor în domeniu consideră că verificarea orală are un grad înalt de subiectivism. Subiectivă sau nu, această metodă este una dintre puținele care dau cadrului didactic posibilitatea de a corecta deficiențele de limbaj și de abordare a cunoștințelor, elevul putând exersa și expune direct noțiunile învățate. Elevul nu are mult timp la dispoziție pentru a-și alege cuvintele potrivite și este oarecum obligat să redea cunoștințele în mod direct. De asemenea, această metodă oferă cadrului didactic posibilitatea de a face o incursiune în universul cunoștințelor elevului pentru a sesiza din timp noțiunile care creează dificultăți în înțelegerea lor corectă. Considerăm ca un mare avantaj al acestei metode crearea deprinderii de comunicare a elevului cu lumea exterioară. Dezavantajul este că profesorul nu poate testa decât un număr restrâns de elevi, iar în cazul unei programe aglomerate (materie multă, puține ore alocate), el se vede în situația de a o folosi foarte puțin, elevii putând percepe disciplina respectivă ca fiind ruptă de realitate. În fond, fiecare poate citi o carte și fără a i se povesti dinainte acțiunea, dar nu oricine este autodidact, nu oricine poate „puncta” esențialul.

Verificarea scrisă se utilizează sub forma unor *lucrări de scurtă durată*, lucrări *tip obiectiv*, lucrări de una sau două ore, *semestriale* (care sunt anunțate dinainte și pregătite și 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 și stimulativă. Probele scrise sunt preferate de elevi și 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 același grad de dificultate și favorizează realizarea comparării rezultatelor. Dezavantajele metodei sunt legate de faptul că profesorul nu poate interveni și corecta pe loc erorile descoperite, el urmând să o facă abia la discutarea lucrărilor. Elevii nu pot fi corecți pentru anumite confuzii sau când răspunsul nu este complet. Răspunsurile incomplete pot genera și diferențe de apreciere și notare. Metoda de evaluare prin verificare scrisă presupune în general un grad mai mare de obiectivitate din partea cadrului didactic în momentul aprecierii lucrării, dar are marele dezavantaj că rupe comunicarea dintre elev și profesor. Această metodă își dovedește eficacitatea în momentul în care este utilizată împreună cu verificarea orală sau cu metoda interviului. Verificările scrise pot fi din lecția curentă sau din materia unui capitol. Când verificarea scrisă vizează mai multe lecții (un capitol etc.), cadrul didactic trebuie să anunțe elevii în timp util (*lucrări anunțate*), eventual să puncteze ceea ce se urmărește în mod special în cadrul testului respectiv. La urma urmei, profesorul se bucură când elevii răspund bine cerințelor sale și îi poate aprecia cu note bune, iar elevii capătă încredere în forțele proprii și, de asemenea, în profesor. Profesorul nu va fi perceput astfel ca „un vânător” de elevi ce nu-și pregătesc lecțiile. Verificările scrise pot viza expunerea unei anumite tematici (în general, o lecție sau două) sau pot fi alcătuite sub formă de teste-grilă. Un rol important în reușita acestei metode îi revine cadrului didactic, responsabil cu alegerea subiectelor și 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; el trebuie să prezinte tematica în timpul alocat, deci activitatea de sintetizare aparține elevului. Verificarea scrisă cu ajutorul testului-grilă presupune alegerea judicioasă a întrebărilor și răspunsurilor posibile din partea profesorului, astfel încât să acopere materia anunțată pentru test, să dea elevilor posibilitatea să se încadreze în timpul alocat testului. Se recomandă ca întrebările cu răspunsuri multiple să fie separate de cele cu un singur răspuns și de asemenea semnalate verbal în cadrul testului. Strategia de notare pentru un test-grilă trebuie anunțată de către profesor (de exemplu, dacă se acceptă răspunsuri parțiale și în ce condiții nu se acceptă aceste răspunsuri). Se știe că elevii își redactează răspunsurile și în funcție de strategia profesorului. Dacă elevii știu că sunt acceptate răspunsuri parțiale la un test-grilă (test-grilă cu mai multe răspunsuri posibile), atunci ar putea încerca să completeze toate răspunsurile 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 și profesorul anunță că acceptă și răspunsuri parțiale, fără a mai specifica și altceva. În aceste condiții, o întrebare cu patru răspunsuri poate fi abordată de către elevi prin selectarea tuturor celor patru răspunsuri. Calculul elevului este simplu: *nu am fost anunțat că erorile se penalizează, deci ar trebui să obțin punctajul maxim*. Bineînțeles că este ipotetic acest lucru și că în realitate nu se poate admite așa ceva. Pe aceeași problemă, profesorul anunță următoarea strategie de notare: pentru fiecare răspuns incorect selectat se scade ponderea unui răspuns corect din valoarea testului. Calculul elevului poate fi: *dacă la o întrebare de 15 puncte cu patru răspunsuri posibile trei*

sunt corecte, atunci prin selectarea tuturor răspunsurilor obțin 10 puncte ; dacă sunt două corecte, obțin 0 (zero) puncte și atunci voi selecta doar trei întrebări la toate testele ce conțin patru răspunsuri posibile, asigurându-mi astfel un minim de 5 puncte pe întrebare. Și astfel de scenarii pot continua. Deși 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 a testelor-grilă) decât testele care au fost rezolvate corect. Primul test-grilă aplicat la o clasă va crea surprize mari atât pentru elevi, cât și pentru profesor, de aceea profesorul trebuie să fie conștient că este nevoie să-și pregătească elevii pentru un asemenea eveniment.

Testele-grilă prezintă următoarele *avantaje* imediate :

1. Obiectivitate și ușurință în notare.
2. Răspunsul se poate da într-un 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 așteptat etc.).

De asemenea, realizarea testelor-grilă impune 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 un pronunțat caracter aplicativ, iar informaticii cu atât mai mult. Ea se poate desfășura în forme variate, de la realizarea de programe simple sau editări de texte ori 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 și evaluate cunoștințele teoretice necesare realizării lucrării, precum și deprinderile și dexteritățile indispensabile executării ei. Este necesară și formarea la elevi a *capacității de autoevaluare*, prezentându-le criteriile de apreciere, ceea ce va mări încrederea elevului în propriile sale forțe și va înlătura orice urmă de suspiciune. Deși imperfect, sistemul actual de evaluare permite o ierarhizare a elevilor în „clase” după criterii reale de competență, oferă informații edificatoare asupra nivelului de cunoștințe al fiecărui elev, stimulează elevul să învețe. Putem face și o 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 cunoștințe cu care elevul „pornește la drum”. Trebuie să ne asigurăm asupra a ceea ce cunoaște elevul înainte de a-l învăța alte lucruri. Această formă de verificare creează și 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-și 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 și la încheierea unui capitol. Această formă de verificare oferă permanent informații cu privire la eficiența programului de instruire și permite profesorului să ia cele mai potrivite măsuri de prevenire a insuccesului școlar, ajutând totodată la ameliorarea metodelor de predare-învățare. Pe baza mecanismului de feedback continuu, verificarea ritmică oferă semnalele necesare atât elevului, cât și profesorului, fiind un veritabil metronom al activității didactice.
- c) Evaluarea *sumativă (cumulativă)* este forma tradițională de evaluare realizată la sfârșitul unui semestru sau an școlar și cuprinde întreaga materie conform programei școlare, 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 și o abordare globală a materiei parcurse, are efecte pozitive în direcția dezvoltării capacității de cuprindere și de sinteză a elevului. Superioară prin caracterul ei predictiv, evaluarea formativă trebuie totuși completată și cu celelalte forme. Rezultatele școlare sunt obiectivate în cunoștințele acumulate, în priceperi și deprinderi, capacități intelectuale, trăsături de personalitate și de conduită ale elevilor. Aprecierea cât mai obiectivă a rezultatelor la învățătură presupune și urmărirea anumitor criterii, cum ar fi :
- *Criteriul raportării rezultatelor la obiectivele generale și operaționale*, prevăzute în programa școlară. Prin aceasta se scot în evidență calitatea și eficiența programului de instruire. Pe lângă orientarea metodologică, obiectivele pedagogice permit și o verificare și o apreciere exactă a rezultatelor elevilor (astfel încât doi profesori care evaluează aceeași performanță să realizeze doar diferențe de notare foarte mici). În acest sens, obiectivele îndeplinesc funcția de criteriu de referință atunci când se formulează o judecată de valoare asupra rezultatelor școlare, dar ele sunt influențate și de factori perturbatori, uneori obiectivi, alteori subiectivi, cum ar fi dotarea materială, nivelul clasei, pretențiile profesorului etc.
 - *Criteriul raportării rezultatelor la nivelul general atins de populația școlară evaluată*, care se manifestă câteodată, din păcate, printr-o tendință de apreciere indulgentă a rezultatelor elevilor din clasele mai slabe și de exigență sporită pentru elevii din clasele considerate mai bune.
 - *Criteriul raportării rezultatelor la capacitățile fiecărui elev și la nivelul lui de cunoștințe de dinaintea încheierii programului de instruire*. Această formă de evaluare dă măsura progresului școlar realizat de elevi.

1.3.2. Practica evaluării activității didactice

Controlul cunoștințelor dobândite de elevi îi dă profesorului posibilitatea să dezvolte la aceștia simțul răspunderii, să sesizeze la timp lipsurile, să aprecieze cât mai just munca lor. Controlul trebuie făcut sistematic (dacă se poate, zilnic) și în mod echilibrat.

La fiecare lecție se verifică modul în care a fost înțeleasă și asimilată lecția nouă, iar dacă lecția are un caracter instructiv, trebuie verificat și gradul în care cele expuse au fost reținute. Verificarea gradului de asimilare se poate face :

- prin repetarea raționamentelor realizate 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* și o examinare *superficială*, cantitativă ;
- prin prezentarea rezolvării (ideea principală) de către un elev și 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 și dezvoltarea deprinderilor de a corela noțiunile teoretice între ele și 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 greșelile. 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 scoatere 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 pentru acasă. Când elevii răspund, este bine ca profesorul să se asigure că toată clasa este atentă și pregătită să intervină. Profesorul poate pune întrebări suplimentare sau ajutoare atât elevilor ascultați, cât și celor aflați în bănci. Prin întrebări se caută să se pună în evidență aspectele esențiale ale lecției. Profesorul trebuie să-și pregătească dinainte întrebările, să nu transforme verificarea într-o scoatere cu sila la tablă și să nu pună un 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 și la cunoștințele elevului. Lucrările de control scrise pot varia ca dimensiune :

- cele scurte (*10-15 minute*) se dau, de obicei, în a doua parte a lecției și urmăresc modul de asimilare a lecției noi sau a cunoștințelor predate anterior, dar în corelație cu lecția nouă ;
- cele de *1-2 ore* se dau numai după ce au fost anunțate din timp și pregătite eventual printr-o lecție de recapitulare, însă orice procedeu de verificare trebuie să îndeplinească anumite condiții, discutate în prealabil cu elevii.

În general, verificările 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 și de a da răspunsuri corecte, precise și scurte, dar complete ;
- să dezvolte la elevi grija pentru formulări exacte și exprimări corecte științifice și gramaticale ;
- să permită elevilor să aprecieze răspunsurile ;
- să fie operative.

1.3.3. *Aprecierea cunoștințelor elevilor. Măsuri de prevenire a rămânerilor în urmă*

Aprecierea se face, în principal, prin notă (calificativ). Ea trebuie să reflecte cât de bine și cât de conștient și-a însușit elevul materia parcursă și în ce măsură ar fi capabil să utilizeze în continuare cunoștințele dobândite. Există anumite criterii după care se realizează aprecierea și notarea. Greșelile pe care le comit elevii la verificare sunt diferențiate (*grave, mici, de neatenție, de înțelegere* etc.). Sunt calificate drept greșeli grave cele legate de necunoașterea sau neînțelegerea unei noțiuni elementare, nepriceperea în abordarea problemelor. Greșelile de genul celor legate de interpretarea eronată a unor enunțuri sau de neatenția de moment nu trebuie considerate ca fiind grave. Acestea se manifestă printr-o formă neîngrijită de prezentare, greșeli de exprimare, prescurtări ambigue în lucrările scrise. Profesorul se lovește deseori de dificultatea aprecierii răspunsurilor. De cele mai multe ori se cade în extreme. De aceea este indicat să se stabilească un *barem de notare* pentru fiecare subiect în parte și 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 *eșecului școlar* depinde în mare măsură de *metodica predării*, de buna organizare a muncii elevilor la clasă și în special la orele de aplicații practice de laborator. Interesul trezit de anumite aplicații este esențial. De aceea, trebuie alese probleme atractive, interesante, al căror rezultat (pozitiv) să fie ușor de constatat. Pentru prevenirea eșecului, este de asemenea important să se sesizeze la timp lipsurile și să se intervină prompt, înlăturându-se greșelile. Un profesor nu invocă în mod gratuit (uneori în glumă) greșeli *antologice* ale unor elevi. Astfel este posibilă evitarea repetării lor ; se creează în acest mod un *cont de greșeli personale* la care se face referire dacă e nevoie. În cazul rămânerilor în urmă, se recomandă reluarea unor noțiuni prin lecții suplimentare și ore de

consultație la care elevii întreabă și profesorii răspund. Se poate recurge și la teme suplimentare, individuale sau colective.

Promovarea succesului școlar nu se poate realiza decât printr-un ansamblu de măsuri și strategii la nivelul macrosistemului de învățământ, al unităților școlare, cu contribuția activă a profesorilor, părinților și elevilor. La nivelul macrosistemului, reforma învățământului trebuie să promoveze ideea perfecționării structurii sistemului de învățământ în raport cu cerințele sociale și cu dinamica mutațiilor economice și sociale, prin modernizarea obiectivelor pedagogice, a conținuturilor (planuri, programe, manuale), a metodologiei și mai cu seamă a bazei didactico-materiale a procesului de învățământ. Efortul devine singular dacă bunele intenții și inițiativele promovate la nivel macro nu sunt aplicate în unitățile școlare. Este necesar să se creeze un climat favorabil de muncă, să se stimuleze inițiativa și responsabilitatea corpului profesoral, să se accentueze perfecționarea profesională în raport cu noile cerințe. Aceasta va avea efecte benefice asupra strategiilor de proiectare, organizare și realizare a activității didactice și se va reflecta pozitiv în promovarea reușitei școlare. Ca un corolar, să punctăm și următoarele :

- În învățământul preuniversitar, evaluările se concretizează, de regulă, prin note de la 1 la 10.
- În clasele din învățământul primar, aprecierea rezultatelor elevilor se face și prin calificative sau alte forme de apreciere, conform reglementărilor ministerului.
- Numărul de note acordate fiecărui elev, la fiecare disciplină de studiu, exclusiv nota de la teză, trebuie să fie cel puțin egal cu numărul de ore săptămânale prevăzute în planul de învățământ, cu excepția disciplinelor cu o oră pe săptămână, la care numărul minim de note/calificative este de două.

Primul pas ar consta deci în a defini ceea ce încercăm să măsurăm/evaluăm, evaluarea fiind o componentă esențială a procesului de învățământ și îndeplinind funcții bine conturate :

- *Funcția de constatare și diagnosticare* a performanțelor obținute de elevi, explicate prin factorii și condițiile care au condus la succesul sau insuccesul școlar și care sunt de o mare diversitate (psihologică, pedagogică, socială etc.).
- *Funcția de reglare și perfecționare* a metodologiei instruirii pe baza informațiilor obținute din explicarea factorilor și condițiilor care au determinat rezultatele la învățătură.
- *Funcția de predicție și decizie*, care vizează desfășurarea în viitor a activității didactice.
- *Funcția de selecție și clasificare* a elevilor în raport cu rezultatele școlare obținute.
- *Funcția formativ-educativă*, de ameliorare a metodelor de învățare folosite de elevi, de stimulare și optimizare a învățării.
- *Funcția de perfecționare* a întregului sistem școlar.

Creșterea eficienței procesului de predare-învățare presupune o mai bună integrare a actului de evaluare în desfășurarea activității didactice prin verificarea și evaluarea sistematică a tuturor elevilor, pe cât posibil după fiecare capitol, prin raportarea la obiectivele generale și operaționale ale acesteia, prin verificarea procesului de instruire și corelarea notelor din catalog cu rezultatele obținute de elevi la probele externe (concursuri, olimpiade, examene de admitere etc.).

Metodele de verificare a randamentului școlar presupun observarea modului în care învață elevul (logic, mecanic, creativ, ritmic, continuu, în salturi etc.) și se realizează prin probe orale, scrise și practice, teste de cunoștințe și deprinderi, după cum am văzut.

1.3.4. *Condiția profesorului*

Analiza psihologiei *omului de la catedră* a constituit un obiect de studiu permanent pentru cercetători. De exemplu, în [Po] 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-și poate permite să manifeste dezinteres sau plictiseală față de subiectul pe care-l predă. El trebuie să-l considere și să-l facă interesant (chiar dacă este simplu, îl cunoaște și l-a mai abordat de zeci și 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 și încântat de subiectul abordat (numai așa va stârni și va menține treaz interesul elevilor).
- *Va cunoaște cu exactitate subiectul*. Este normal că orice metode am alege, orice mijloace am folosi în predare, nu-i putem face pe alții să înțeleagă ceva ce nici noi nu înțelegem cu exactitate. Celebra butadă „am explicat până am priceput și eu” vine să confirme regula. A explica o problemă sau a elucida un aspect pe care nu-l poate înțelege clasa presupune abordarea aspectului prin prisma puterii de înțelegere a elevului de nivel mediu din clasă și amplificarea în spirală, prin pași care presupun, pe lângă raționament, și 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 la 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 să realizeze conexiunile dintre conținuturi).
- *Să știe că înainte de a învăța de la altcineva, poți descoperi singur*. Recurgerea la metodele active (bazate pe activitatea proprie a elevului) în însușirea anumitor concepte, priceperi, deprinderi are un efect stimulator, elimină *șablonismul*, 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 citește 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 preconcepută că profesorul cunoaște cu exactitate modul de rezolvare și, 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 și argumentării ideilor sale; el trebuie lăsat să-și continue raționamentul (în anumite limite, chiar dacă acesta este greșit), până când se convinge de greșeală. Întreruperea brutală și fără argumentare transformă elevul din colaborator în adversar, acesta canalizându-și eforturile în contracarare, și nu în colaborare.
- În procesul instructiv, profesorul trebuie să *se coboare la nivelul de înțelegere și anticipare al elevului*, să se transpună în situația acestuia, relația profesor-elev fiind una de colaborare, în care, cu certitudine, profesorul este cel care știe și elevul cel care învață. Premisele colaborării pornesc de la cunoașterea reciprocă a exigențelor profesorului și a posibilităților elevului, iar împărțirea forțată și apriorică în profesori blânzi sau duri sau în elevi slabi și buni este dăunătoare. Profesorul are obligația să cultive elevului încrederea în posibilitățile sale, să-i depisteze punctele slabe și, 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 și 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 și de disciplină, introduc noțiuni și 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 cunoștințelor 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 făgașul descoperirii, corectându-i-se alegerile și sfătuindu-l să-și verifice singur pașii, învățându-l să facă analogii, să descopere diferențe, să intuiască situațiile-limită. Elevul trebuie învățat în același timp să abstractizeze, să aplice rezultatele teoretice care i-au fost prezentate, să aleagă dintr-o mulțime de soluții metoda cea mai adecvată de rezolvare. Elevul trebuie să fie conștient de faptul că nu este primul și nici singurul rezolvator al acelei probleme și că poate

să existe o metodă mai eficientă descoperită de alții. În acest fel, va fi preocupat mereu de optimizarea propriilor soluții, i se vor forma spiritul critic și autocritic și dorința de autodepășire.

- *Să învețe elevii să-și argumenteze și demonstreze corectitudinea soluției găsite.* Argumentele pro și contra unei soluții trebuie să însoțească fiecare pas al rezolvării. Elevul trebuie obișnuit să-și *suspecteze* corectitudinea soluției găsite prin analiza cazurilor-limită și să caute în permanență contraexemple. Analiza complexității algoritmilor este un concept care se deprinde și se aplică după o oarecare experiență.
- *Să formeze elevilor capacitatea de abstractizare și generalizare.* Posibilitatea adaptării și aplicării unui algoritm la o clasă de probleme de același tip a avut ca rezultat, printre altele, apariția metodelor de elaborare a algoritmilor, înțelegerea problematicei generale și 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 și pretinde totul așa 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 crește, 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 imaginează și improvizează 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 își va descoperi singur greșeala, vom suferi alături de el căutând-o pe cea corectă și ne vom bucura odată cu el descoperind-o. Profesorul nu poate fi supărat sau trist, nu poate fi melancolic, distrat, inexact. El trebuie să fie mereu bine dispus și atent. În plus, trebuie să-și soluționeze singur toate probleme cu clasa, să nu dea semne de slăbiciune. Cu cât se cunosc reciproc mai mult, cu cât colaborează și se ajută mai mult, cu cât se înțeleg și se apreciază mai mult, cu atât profesorul și elevii se vor apropia mai mult.

1.3.5. 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 eșalonarea conținuturilor disciplinei respective pe durata anului sau a semestrului, cu indicarea numărului de ore și a datei stabilite pentru studiul fiecărei teme. În paralel cu lecțiile de comunicare de cunoștințe sau mixte, este necesară planificarea *lecțiilor recapitulative*, iar la sfârșitul semestrului, a *lecțiilor de evaluare sumativă*. În planificarea calendaristică se vor face referiri la materialul

didactic și la lucrările practice care vor fi efectuate. Rubricația planificării calendaristice depinde de gradul de detaliu la care se dorește 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 și în care trebuie să răspundă la următoarele întrebări :

a) *Ce voi face ?*

Înainte de toate, se vor preciza cu claritate obiectivele educaționale ale activității viitoare.

b) *Cu ce voi face ?*

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

c) *Cum voi face ?*

Se va alege strategia educațională potrivită pentru realizarea obiectivelor stabilite.

d) *Cum voi ști dacă s-a realizat ceea ce trebuia ?*

De altfel, în orice activitate 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 totuși concretizarea și detalierea următoarelor etape (și vom încheia capitolul cu aceste considerații) :

a) *Proiectarea formării de competențe.* Presupune stabilirea în mod precis a *deprinderilor care se doresc a se forma* pe parcursul desfășurării activității didactice. Se va verifica dacă ceea ce s-a stabilit este ceea ce trebuia realizat în raport cu programa școlară. Se va verifica și dacă obiectivele stabilite sunt realizabile în timpul disponibil.

b) *Analiza resurselor.* Se va stabili conținutul activității. Se vor analiza calitatea materialului uman, dezvoltarea fizică și psihică a elevilor, particularitățile individuale, motivația învățării, mijloacele materiale. Se vor alege metodele didactice necesare.

c) *Elaborarea strategiei.* Se vor selecta mijloacele de instruire de care este nevoie, combinând metodele, materialele și mijloacele astfel încât să se amplifice eficiența lor didactică. Se va descrie în detaliu *scenariul* activității care urmează a fi desfășurată.

d) *Evaluarea.* Se vor analiza cu atenție standardele de performanță și se va elabora un sistem de metode și tehnici de evaluare adecvate atingerii scopului propus.

Capitolul 2

Principii didactice și didactica formării de competențe

În acest capitol sunt prezentate pe scurt câteva dintre principiile didactice generale și reperele didacticii formării de competențe, cu unele exemplificări ale aplicării acestora în domeniul informaticii. Trebuie să precizăm faptul că terminologia pe care o folosim în prezent (competențe în loc de obiective, competențe-cheie în loc de obiective-cadru, competențe specifice în loc de obiective de referință etc.) reflectă o schimbare conceptuală petrecută relativ recent la nivelul factorilor decizionali. Schimbarea se referă la o lecție și mută accentul de pe îndeplinirea obiectivelor unui profesor pe însușirea unor competențe de către elevi. Vom reveni pe parcurs.

2.1. Clasificarea și caracteristicile principiilor didactice

Un model al sistemului de învățământ trebuie să se încadreze în contextul legilor obiective care acționează în societate la momentul respectiv. Scopul, conținutul, sarcinile concrete ale predării informaticii pot fi deduse din planurile de învățământ, precum și din alte activități specifice (școlare sau extrașcolare), și corespund stadiilor (ciclurilor) de învățare fixate în conformitate cu dezvoltarea intelectuală a elevilor. O atenție prioritară trebuie direcționată spre adaptarea la nou, inclusiv în ceea ce privește dezvoltarea bazei materiale.

Principiile didactice reprezintă normele generale care orientează conceperea, organizarea și desfășurarea procesului de predare-învățare și conduc spre formarea eficientă a competențelor proiectate. Profesorul va decide când, cum și care dintre principiile didactice va fi aplicat în fiecare moment al lecției.

Așa cum este normal, începem prin a puncta câteva dintre caracteristicile generale ale principiilor, după care urmează o clasificare și o descriere mai detaliată a acestora.

Principiile didactice se caracterizează prin :

- caracter legic, ceea ce înseamnă că ele exprimă raporturile esențiale și globale care orientează conceperea și desfășurarea procesului de învățământ ;

- caracter obiectiv, adică se asigură o orientare a procesului de învățământ nefalsificată și detașată de impresii, tendințe și dorințe subiective ; procesul de învățământ este de dorit să fie orientat în concordanță cu legile dezvoltării psihice ale individului, precum și cu legile evoluției societății ;
- caracter algoritmic – se exprimă cerințe și soluții prin utilizarea unui sistem precis de reguli, care trebuie cunoscute și respectate cu exactitate dacă se dorește o orientare eficientă a procesului de învățământ ;
- caracter dinamic – principiile didactice sunt elemente legale, dar deschise înnoirilor și creativității. Ele trebuie să fie în pas cu schimbările și mutațiile care pot interveni în actul didactic ;
- caracter sistematic – fiecare principiu intră în relație cu celelalte principii, alcătuind un ansamblu unitar de legități ale cărui componente se condiționează reciproc.

Pentru o bună organizare și desfășurare a procesului de învățământ, profesorul trebuie să respecte mereu și să aplice corect cel puțin următoarele principii didactice clasice :

1. Principiul intuiției.
2. Principiul legării teoriei de practică.
3. Principiul însușirii conștiente și active a cunoștințelor.
4. Principiul sistematizării și continuității cunoștințelor.
5. Principiul accesibilității cunoștințelor.
6. Principiul însușirii temeinice a cunoștințelor.
7. Principiul individualizării și diferențierii învățării.

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

2.1.1. *Principiul intuiției*

Acest principiu exprimă necesitatea studierii obiectelor, fenomenelor, proceselor cu ajutorul simțurilor, ținându-se cont de importanța realizării unității dintre senzorial și rațional. A transmite cunoștințe 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 oferă soluții deosebit de eficiente. Totuși, folosind doar acest principiu, este posibil să nu putem descrie exact și complet o problemă, într-o singură fază. Putem deschide însă o cale spre înțelegerea acesteia, putem stabili un drum cât de cât sigur spre reveniri ulterioare.

2.1.2. *Principiul legării teoriei de practică*

Raportul dintre teorie și practică depinde, în ultimă instanță, de dificultatea noțiunilor implicate, de mijloacele tehnice avute la dispoziție, de cunoștințele anterioare, precum

și de capacitățile intelectuale ale clasei de elevi, de abilitatea și experiența cadrului didactic. În informatică, conștientizarea necesității utilizării performante a unor tehnici folosite frecvent astăzi în viața cotidiană (coduri de bare, telefonie mobilă, tehnici de localizare GPS, transmisie audiovideo prin satelit, poștă electronică, scanări etc.) este esențială. Mai mult, importanța verificării faptului că elevii sunt în stare să aplice în practică cunoștințele 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 și sălile de curs trebuie dotate la nivelul cerințelor moderne, anticipându-se condițiile ce ar putea fi întâlnite la viitoarele locuri de muncă (calculatoare performante necesare în lucrul independent, conexiune la Internet, sistem de videoproiecție, software variat, actualizat).
- Activitățile practice ale elevilor trebuie să aibă o finalitate și o aplicabilitate imediată, manifestate, de exemplu, prin lucrul în echipă la proiecte dezvoltate în parteneriat cu unități economice, gen *coaching*, prin elaborarea unui proiect complex și original pentru examenul de obținere a competențelor profesionale sau prin elaborarea unei lucrări cu contribuții personale, publicabilă în reviste școlare. Ar fi benefic ca atât recompensele, cât și pedepsele să fie similare cu cele aplicate într-o activitate reală, și nu doar reprezentate de note sau calificative, și să încurajeze autoevaluarea și evaluarea reciprocă.
- Activitățile serioase cer o fundamentare teoretică, conștientizându-se faptul că partea de teorie este efectiv utilă, chiar indispensabilă dacă se dorește o adaptare „din mers” la cerințe ulterioare.
- Asistența cadrelor didactice trebuie corelată cu apelarea la specialiști „lucrativi” din sfera producției directe, precum și cu o testare pe cât posibil individualizată și specifică a elevului.

2.1.3. *Principiul însușirii conștiente și active a cunoștințelor*

Acest principiu exprimă necesitatea ca procesul de instruire prin acumulare de cunoștințe să se realizeze organizat, prin fixarea unor scopuri, finalități și termene precise. Înțelegerea semnificațiilor și conexiunilor esențiale pentru studiul informaticii necesită un efort de gândire acțional. Profesorul trebuie să delimiteze încă de la începutul lecției scopul și utilitatea practică și teoretică a temei respective, folosind un bogat material exemplificativ. Se urmăresc trecerea de la intenție la gândirea abstractă, de la treapta senzorială la treapta rațională, precum și favorizarea formării de noi structuri informaționale. Pentru evitarea unei însușiri mecanice, se va pune accent pe metodele active de învățare, pe asigurarea participării permanente și conștiente a elevilor la desfășurarea lecțiilor, pe stimularea muncii creatoare și independente. Însușirea conștientă și activă a cunoștințelor determină formarea unor atitudini sau condiții favorizante pentru învățare cum ar fi :

- obținerea unei motivații favorabile și a satisfacției învățării ;
- asigurarea credibilității adevărilor și transformarea lor în convingeri și deprinderi științifice ;
- sporirea posibilităților de a utiliza în mod concret și profitabil informația asimilată, oferind potențialului intelectual individual șanse superioare de reușită, atât pe plan practic/constructiv, cât și pe plan creativ.

2.1.4. *Principiul sistematizării și continuității cunoștințelor*

Scopul oricărei activități de predare este de a dota elevii cu un sistem armonios și corect de cunoștințe. Logica internă a obiectului de predat și legile generale ale dezvoltării capacităților de cunoaștere individuale impun asigurarea continuității, dar și necesitatea sistematizării materiei. Noile informații relevante vor fi legate de cele deja introduse și vor prefigura informațiile ulterioare (respectându-se programa școlară). Principiul sistematizării se concretizează deci prin expuneri organizate asupra cunoștințelor 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 și generalizare a celor învățate, profesorul trebuie să-și folosească la maximum capacitățile creatoare și talentul pedagogic în pregătirea expunerilor. Activitatea individuală conștientă a elevului trebuie să fie esențială. Cunoștințele 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 și asigurarea unei pregătiri ritmice a elevilor, trebuie acceptat un control permanent și riguros al profesorului asupra modului și stadiului de însușire a cunoștințelor de către elevi. Recomandăm aplicarea câtorva reguli generale :

- secvențele de cunoștințe transmise trebuie să fie coerente și unitare, ordinea fiind determinată de conexiuni logice clare ;
- învățarea trebuie să aibă loc ritmic, la intervale optime, asigurându-se simultan restructurarea și reorganizarea pachetului de cunoștințe ;
- 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 și evaluarea periodică a calității receptării trebuie să fie o modalitate de reglaj, dar și de autoreglaj.

2.1.5. *Principiul accesibilității cunoștințelor*

Cunoștințele predate pot fi asimilate de elevi numai dacă sunt accesibile ca volum și conținut. O temă este accesibilă atunci când corespunde particularităților psihologice

de vârstă ale elevilor cărora le este adresată, reprezintă o continuare firească a celor acumulate anterior și corespunde capacității lor reale de muncă. Conform acestui principiu, respectarea programei școlare este esențială. De asemenea, demersul instructiv-educativ trebuie adaptat condițiilor concrete ale clasei, stabilindu-se un raport optim între efortul solicitat elevului și 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 aceluiași semestru. Respectarea particularităților psihologice de vârstă nu înseamnă scutirea elevilor 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 ușor la greu, de la particular la general, de la concret la abstract ;
- conștientizarea elevilor asupra faptului că efortul personal este absolut esențial pentru înțelegerea corectă și de durată a celor studiate ;
- asigurarea unui studiu ritmic pentru a evita golurile de cunoștințe și eforturile ulterioare de înțelegere și asimilare ;
- asigurarea unui control activ și a unei evaluări permanente, în scopul eficientizării maxime a actului didactic.

2.1.6. *Principiul însușirii temeinice a cunoștințelor*

Acest principiu reclamă cerința fixării materialului de specialitate studiat, astfel încât elevii să-l poată reproduce și utiliza în mod creativ atât în rezolvarea temelor școlare curente, cât și în activitatea practică viitoare. Expunerile trebuie făcute intuitiv, accentuându-se esențialul și evitându-se supraîncărcarea. Fixarea cunoștințelor nu trebuie realizată printr-o repetare succintă a celor expuse, ci trebuie să se bazeze pe o receptare logică, rațională, cu ajutorul căreia să se poată identifica esențialul. O asemenea însușire temeinică poate fi obținută prin diverse modalități de recapitulare : curentă, de sistematizare și sinteză, de preîntâmpinare a uitării celor deja învățate, de asigurare a fixării în memorie a sistemului de cunoștințe fundamentale.

Recomandăm respectarea câtorva reguli :

- predarea să fie intuitivă și accesibilă ;
- însușirea cunoștințelor trebuie direcționată spre o asimilare logică și conștientă, urmându-se un studiu sistematic ;
- elevii trebuie stimulați în ideea participării active și continue la lecții ;
- este de dorit să se asigure motivația învățării, în strânsă legătură cu anumite aspirații individuale.

2.1.7. *Principiul individualizării și diferențierii învățării*

Exprimă necesitatea adaptării strategiei instructiv-educative atât la particularitățile psihofiziologice ale fiecărui elev în parte, cât și la particularitățile unei grupe omogene de elevi, în vederea dezvoltării lor ca personalități distincte și a profesionalismului. Individualizarea învățării se referă la valorificarea cât mai bună a posibilităților și eforturilor individuale, atât pentru persoanele înzestrate, cât și pentru cele mai puțin înzestrate.

Se recomandă :

- Elaborarea de sarcini instructive (teme, lucrări etc.) individualizate pentru fiecare elev în parte (în funcție de aptitudinile, înclinațiile, opțiunile, nivelul de dezvoltare intelectuală, coeficientul de inteligență).
- Cerința ca oricare dintre sarcinile specificate anterior să fie identificată prin fișe de lucru individuale, cum ar fi :
 - fișe de recuperare (pentru cei rămași în urmă) ;
 - fișe de dezvoltare (pentru elevii foarte buni) ;
 - fișe de exerciții, destinate tuturor, în scopul formării unor priceperi și deprinderi aprofundate ;
 - fișe de autoinstruire, destinate în special însușirii unor tehnici de învățare individuală și independentă ;
 - fișe de evaluare generală, pentru constatarea nivelului general de pregătire.

Consultațiile speciale, individualizate, nu pot fi evitate. Diferențierea învățării exprimă însă necesitatea de a adapta conținutul strategiilor educaționale în funcție de particularitățile comportamentului individual sau de grup al elevilor (cum ar fi promovarea aptitudinilor specifice pentru anumite materii). Această diferențiere va răspunde atât a nevoilor destinate tratării unor particularități psihologice individuale, cât și satisfacerii unor cerințe sociale privind pregătirea și utilitatea existenței unor specialiști. Aici ar fi utile : crearea de școli și/sau profile specializate ; relaxarea învățământului prin introducerea mai multor discipline opționale și facultative ; intensificarea activităților de coordonare directă profesor-elev (consultații, discuții, mese rotunde, cercuri de profil etc.) ; cunoașterea cât mai completă a fiecărui elev, atât ca individualitate, cât și ca ființă socială ; îmbinarea judicioasă a tratării individuale și diferențiate cu cea globală, de grup, în care se rezolvă sarcini de echipă ; utilizarea învățământului asistat ; conștientizarea elevilor privind posibilitățile proprii de formare/dezvoltare intelectuală.

Exemplu

Pentru ilustrarea aplicării tuturor principiilor, vom încheia acest capitol cu un exemplu global. Problema turnurilor din Hanoi este, considerăm noi, un caz suficient de edificator și de complex, putând fi folosit și pentru :

- înțelegerea metodei *divide et impera* ;
- înțelegerea derecursivării automate în sens iterativ (parte a construcției compilatoarelor), precum și a necesității prezentării unui algoritm *doar* în forma sa recursivă ;

- înțelegerea unor tehnici de prelucrare a imaginilor ;
- introducerea câtorva considerații de corectitudine și complexitate a algoritmilor ;
- introducerea câtorva concepte de programare nestandard, cum ar fi programarea funcțională.

Enunțul problemei. În orașul Hanoi există trei turnuri de aur și un număr n de discuri de diamant. Fiecare disc are diametrul diferit de al celorlalte. Inițial, acestea sunt plasate pe un singur turn, de jos în sus, în ordinea descrescătoare a diametrelor, discul cu diametrul maxim găsiindu-se la bază. Se cere să se deplaseze cele n discuri de pe primul turn pe al doilea folosind, eventual, ca suport intermediar și al treilea turn.

Restricții :

- mutarea discurilor trebuie făcută într-un număr succesiv de pași independenți, la fiecare pas deplasându-se un singur disc de pe un turn pe altul ;
- se mută întotdeauna discul din vârf, adică cel cu diametrul minim de pe turnul respectiv ;
- nu se poate așeza 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, deși ideea unui algoritm iterativ general pentru această problemă nu este deloc transparentă. Propunem alegerea următoarelor notații, care vor simplifica exprimarea ulterioară a soluției :

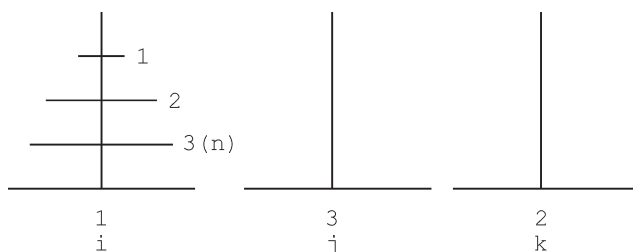
1. Pentru turnuri : $i, j, k \in \{1, 2, 3\}$, valori diferite între ele, unde i reprezintă turnul „de plecare”, j este turnul de sosire, iar k este „turnul intermediar”. În acest caz, putem observa că avem $k = 6 - i - j = \text{al_treilea}(i, j)$.
2. Discurile vor fi notate cu $1, 2, \dots, n$, în funcție de dimensiune (n este discul de dimensiune maximă).
3. Mutările vor fi desemnate prin triplete de tipul $\langle a, b, c \rangle$, 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 $c \in \{1, 2, \dots, n\}$.
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), atunci putem defini :

$$M(i, j, n) = M(i, k, n - 1) \cdot \langle i, j, n \rangle \cdot M(k, j, n - 1)$$

Intuitiv, pentru a deplasa n discuri de pe turnul i pe turnul j , se deplasează mai întâi $n - 1$ discuri de pe turnul i pe turnul k și 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 și la faptul că un limbaj de programare funcțional este un limbaj care lucrează cu liste și liste de cuvinte. Faptul că definiția recursivă este corectă rezultă imediat prin inducție. În final, se obține valoarea funcției ca o secvență finită de pași (cuvânt) de tipul $\langle i, j, n \rangle$. 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 obișnuit 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 :

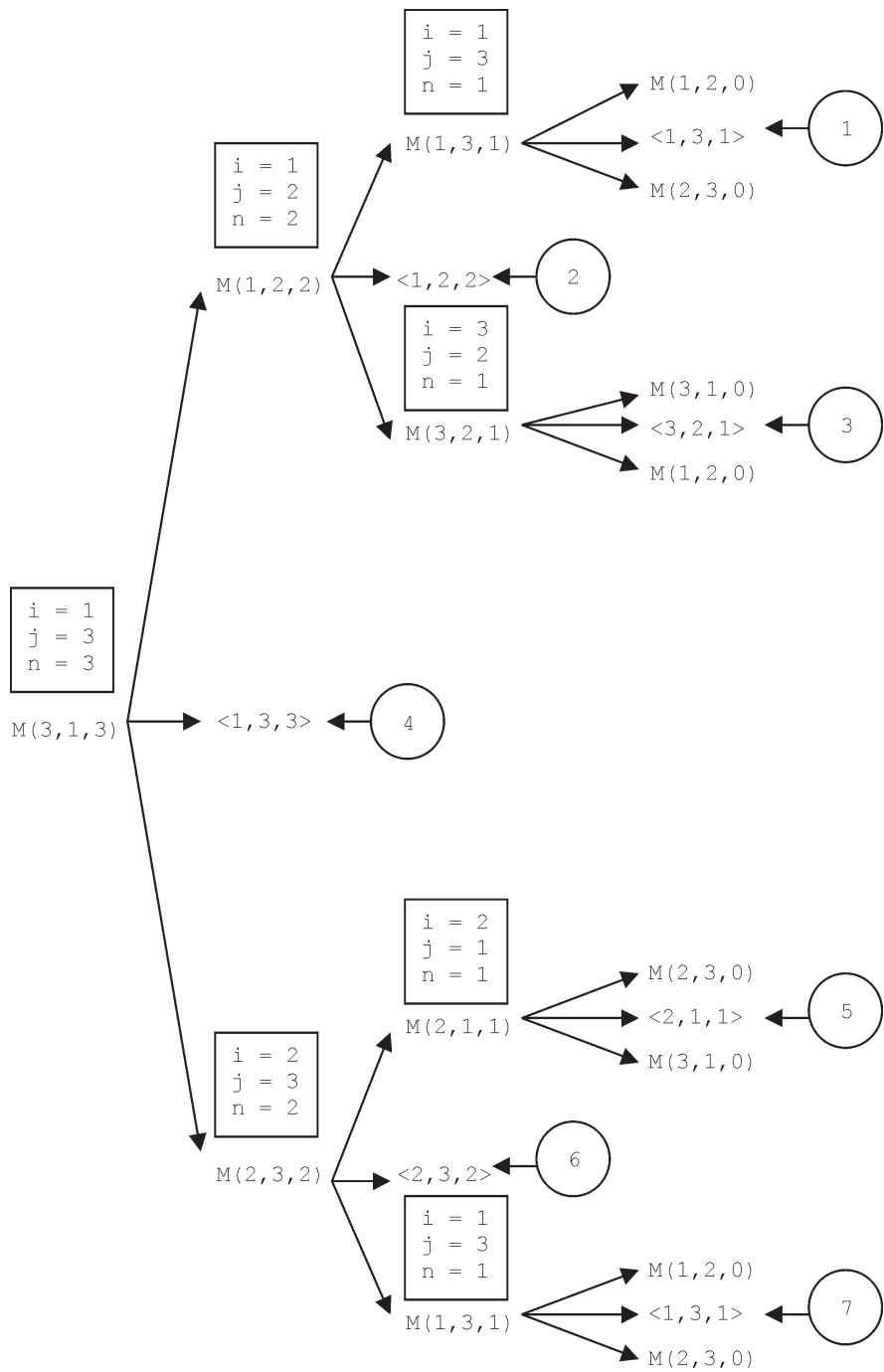
- a) $M(\dots)$ – apelul recursiv al funcției M .
- b) $M(\dots 0)$ – se ignoră aproape de orice acțiune (de fapt, acest simbol va fi șters ulterior).
- c) $\langle \dots \rangle$ – se efectuează o mutare normală.

Operațiile generale care se efectuează asupra stivei sunt :

- În cazul a). Dacă vorbim de un apel al funcției M cu ultima poziție diferită de zero, atunci conținutul vârfului se șterge și acesta se înlocuiește cu 3 celule noi. Restul conținutului stivei „coboară”.
- În cazul b). Conținutul vârfului stivei se șterge și restul conținutului urcă în stivă.
- În cazul c). Se execută efectiv mutarea indicată, se trece aceasta pe lista de ieșire (care va constitui în final soluția problemei) și apoi se procedează ca mai înainte.

Procesul se termină și se obține soluția finală doar în momentul când stiva redevine goală. În exemplul detaliat de mai jos, considerăm $i = 1, j = 3, k = 2, n = 3$. Ceea ce se găsește în final, ca succesiune de mutări, este :

$\langle 1, 3, 1 \rangle \cdot \langle 1, 2, 2 \rangle \cdot \langle 3, 2, 1 \rangle \cdot \langle 1, 3, 3 \rangle \cdot \langle 2, 1, 1 \rangle \cdot \langle 2, 3, 2 \rangle \cdot \langle 1, 3, 1 \rangle$.



2.2. Didactica formării de competențe

2.2.1. Contextul trecerii de la obiective la competențe

Datorită progresului tehnologic accelerat de la începutul acestui secol, s-au produs schimbări majore pe piața muncii și, implicit, în domeniul educației. A devenit necesară trecerea de la învățarea centrată pe cunoștințe aplicate în situații tipice la învățarea centrată pe elev, pe formarea competențelor profesionale și de dezvoltare personală ale acestuia. Astfel, la începutul anilor 2000, atât în curriculumul școlar, cât și în variantele Legii educației naționale, Legea educației naționale nr. 1/2011, actualizată (lege5.ro/Gratuit/gezsobvgy/legea-educatiei-nationale-nr-1-2011), a avut loc această transformare conceptuală, prin care competențele au luat locul obiectivelor ca finalități educaționale.

Din anul 2009, noile programe școlare pentru clasele V-XII au prezentat un curriculum centrat pe competențe, cu referiri explicite la competențele-cheie europene.

Sunt multe definiții ale conceptului de *competență*, dar în această lucrare vom considera competența ca fiind capacitatea intelectuală ce generează multiple posibilități de transfer sau de aplicabilitate în operarea cu conținuturile învățării ([MC], [MCBA]).

Prin includerea competențelor în Legea educației naționale nr. 1/2011, actualizată a fost întărit rolul lor normativ.

Competențele-cheie în Legea educației naționale :

Art. 4 – *Finalitatea principală a educării o reprezintă formarea competențelor* (definite ca un ansamblu multifuncțional și transferabil de cunoștințe, deprinderi/abilități și aptitudini necesare în situații diferite).

Art. 13 – subliniază că *învățarea pe tot parcursul vieții* este un drept garantat ;

Art. 68 – (1) Curriculumul național pentru învățământul primar și gimnazial se axează pe opt domenii de competențe-cheie care determină profilul de formare a elevului :

- a) competențe de comunicare în limba română și în limba maternă, în cazul minorităților naționale ;
- b) competențe de comunicare în limbi străine ;
- c) competențe de bază de matematică, științe și tehnologie ;
- d) competențe digitale de utilizare a tehnologiei informației ca instrument de învățare și cunoaștere ;
- e) competențe sociale și civice ;
- f) competențe antreprenoriale ;
- g) competențe de sensibilizare și de expresie culturală ;
- h) competența de a învăța să înveți.

Art. 68 (5) – subliniază ideea conform căreia învățământul liceal este centrat pe dezvoltarea și diversificarea competențelor-cheie și formarea competențelor specifice, în funcție de filieră, profil, specializare sau calificare.

Art. 70 (2) – biblioteca școlară virtuală și platforma e-learning includ programe, exemple de lecții, ghiduri, exemple de probe și reprezintă o concretizare a competenței-cheie TSI – tehnologia societății informației.

Art. 72 – face precizarea că evaluarea se centrează pe competențe.

Art. 329 – „învățarea pe tot parcursul vieții se centrează pe formarea și dezvoltarea competențelor-cheie și a competențelor specifice unui domeniu de activitate sau unei calificări”.

Competențele-cheie conform Comisiei Europene

Definiția competențelor-cheie promovată de Comisia Europeană : „Competențele-cheie reprezintă un pachet transferabil și multifuncțional de cunoștințe, deprinderi (abilități) și atitudini de care au nevoie toți indivizii pentru împlinire și dezvoltare personală, pentru incluziune socială și inserție profesională. Acestea trebuie dezvoltate până la finalizarea educației obligatorii și trebuie să acționeze ca un fundament pentru învățarea în continuare, ca parte a învățării pe parcursul întregii vieți”.

Toate ariile curriculare, deci și aria „Tehnologii”, din care face parte informatica, sunt compatibile cu cele opt domenii de competențe-cheie stabilite la nivel european :

- 1) Comunicarea în limba maternă.
- 2) Comunicarea în limbi străine.
- 3) Competențe matematice și competențe de bază în științe și tehnologii.
- 4) Competența digitală (TSI – tehnologia societății informației).
- 5) Competența socială și competențe civice.
- 6) A învăța să înveți.
- 7) Inițiativă și antreprenoriat.
- 8) Sensibilizare și exprimare culturală.

Competențele-cheie nu se formează prin studiul unei singure discipline, ci sunt *transversale* și au un caracter teoretic, cu un grad înalt de generalitate, cuprinzând cunoștințe, abilități (aptitudini, deprinderi), atitudini.

Ținând cont de faptul că prezenta lucrare se adresează profesorilor și viitorilor profesori de informatică, am încercat, fără a avea pretenția că am atins toate aspectele, să formulăm o ierarhie a competențelor generale și a competențelor specifice ce trebuie atinse prin studiul disciplinelor de informatică în liceu, jalonând astfel etapele de pregătire a elevilor. Pentru detalii, se mai pot consulta referințele bibliografice [MC], [MCBA], [Ba], [Bi], [Fr],

Programa școlară este expresia cea mai simplă a modelului curricular și este structurată astfel : notă de prezentare, sistem de valori și atitudini, competențe-cheie, competențe generale (CG), competențe specifice (CS), conținuturi, sugestii metodologice.

Competențele generale exprimate în programa școlară se definesc pentru fiecare disciplină de studiu, au un grad ridicat de generalitate și complexitate și se formează pe durata unui ciclu de învățământ, deși abordează niveluri de formare diferite de la un an la altul. Competențele generale exprimă rezultate durabile ale învățării, condiționează nivelul la care elevul învață noi sarcini și pot fi transferate la o mare varietate de sarcini specifice. Pot fi focalizate pe cunoaștere, pe anumite abilități și priceperi sau pe atitudini.

Competențele specifice se formează pe parcursul unui an de studiu, sunt deduse din competențele generale și sunt etape intermediare în formarea acestora.

Conținuturile învățării sunt mijloace prin care se urmărește formarea competențelor specifice și implicit a competențelor generale propuse. Unitățile de conținut sunt organizate tematic.

Profesorul de informatică trebuie să subscrie întregul său demers didactic, de la proiectare și până la evaluare, formării competențelor elevilor.

2.2.2. *Competențele-cheie în studiul informaticii*

Transformările care au loc în societate, dezvoltarea și răspândirea informaticii, pătrunderea rapidă în viața economică, socială și în învățământ a celor mai noi realizări în domeniul hardware-ului și software-ului impun o diversificare a pregătirii elevilor de liceu în acest domeniu. Învățământul preuniversitar trebuie să asigure în primul rând dobândirea unor cunoștințe de informatică la nivel de cultură generală. Totuși, cunoștințele de tehnologia informației, utilizarea calculatoarelor în rezolvarea problemelor din diversele domenii ale vieții economice și sociale reprezintă o cerință a integrării în toate domeniile profesionale ale pieței muncii. Din acest motiv, este posibil să admitem și introducerea/predarea în liceu, eventual în cadrul unor discipline opționale, a unor aplicații complexe precum : Unity, pentru dezvoltarea aplicațiilor vizuale, cu grafică și animație 2D/3D, Java, aplicații pentru cursuri Cisco, software profesional pentru dezvoltare de aplicații dedicate telefoanelor mobile sau prelucrării audiovideo etc. Astfel, în funcție de filieră și specializare, elevii trebuie să-și formeze și să-și dezvolte, până la un anumit nivel de aprofundare, un sistem solid de competențe legate de prelucrarea informației cu ajutorul calculatoarelor personale. Pentru dezvoltarea acestor competențe considerăm că este necesar ca elevii :

- a) Să dobândească cunoștințele necesare înțelegerii principalelor aspecte legate de noțiunea de informație (culegere, prelucrare, stocare, transmitere).
- b) Să-și formeze și să-și modeleze modul de gândire și abordare a problemelor. Asemenea tuturor ramurilor științei, informatica dezvoltă gândirea, având un rol esențial în procesul de învățare, în formarea caracterului și a personalității. În plus, informatica formează și dezvoltă o manieră sistemică de abordare, provoacă o analiză progresivă a detaliilor, o rezolvare în context general a problemelor particulare. Aceasta este gândirea algoritmică, practică, diferită cumva de gândirea teoretică și abstractă. O astfel de manieră de abordare a problemelor leagă cunoștințele de programare de contextul bazei de date pe care o prelucrează și de cel al soluțiilor pe care le va obține. Formarea unor competențe cognitive bazate pe gândirea algoritmică, analitică și sistematică, precum și a unui mod de lucru ordonat are consecințe deosebite în evoluția viitoare a elevului și este un obiectiv esențial al studiului informaticii în învățământul preuniversitar.
- c) Să-și formeze și să-și dezvolte deprinderi de a munci individual și 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 și realizarea bazelor de date mari și a produselor software de dimensiuni medii sau mari. Se impune formarea la elevi a acelor deprinderi elementare de lucru cu calculatorul care oferă șansa unei învățări în ritmul propriu al fiecăruia, dar și posibilitatea asimilării lucrului în echipă. Acesta va fi un element esențial de integrare socială și va conduce la formarea unor trăsături de caracter ce pot oferi individualismului o alternativă civilizată. În viața reală, activitățile nu se desfășoară izolat, astfel încât se impune realizarea unor aplicații complexe care necesită lucrul în grup, modularizarea programului și păstrarea contactelor cu ceilalți membri ai grupului. Se realizează astfel asumarea responsabilității cu privire la finalizarea propriei munci și asigurarea condițiilor de finalizare a activității celorlalți membri ai colectivului. Conducerea rațională a activității de proiectare și programare, dezvoltarea intuiției determină elevul să capete încredere în propriile-i forțe.

- d) Să capete deprinderi care-l vor ajuta să devină un utilizator profesionist, adică să dobândească cunoștințele necesare exploatarea resurselor hardware și software puse la dispoziție de tehnologia informatică actuală. Pentru aceasta, elevul trebuie să-și formeze o cultură generală informatică, ce presupune identificarea și înțelegerea funcționării principalelor componente ale calculatorului și rețelelor de calculatoare, precum și dobândirea deprinderilor necesare utilizării noilor produse software. Punctăm din nou faptul că, pentru dezvoltarea acestor competențe-cheie, trebuie urmărită dezvoltarea în mod diferențiat a următoarelor competențe cognitive și practice :
- cunoașterea și utilizarea până la un anumit nivel de detaliu a sistemelor de operare, a mediilor de programare cel mai des folosite (Windows, Unix, Linux etc.) ;
 - cunoașterea structurii și arhitecturii sistemelor de calcul și a noțiunilor elementare de hardware, care să le permită să ia decizii corecte și eficiente în situații care au legătură cu caracteristicile tehnice ale unui dispozitiv electronic de tip calculator personal, laptop, tabletă, smartphone etc. ;
 - cunoașterea și utilizarea unui limbaj de programare de nivel înalt și a unui limbaj de gestiune a bazelor de date (C/C++, Pascal, SQL, Prolog, Java, Python etc.) și a noțiunilor elementare despre limbajele de asamblare (măcar în liceele de specialitate), a unui mediu de programare vizual (de exemplu, C#) sau, de ce nu, Scratch (pentru gimnaziu sau chiar pentru clasele I-IV) ;
 - cunoașterea și utilizarea tehnicilor de proiectare a produselor-program cu caracter științific, a metodelor de elaborare a algoritmilor, a algoritmilor fundamentali, a tehnicilor de optimizare a algoritmilor (elevii ar trebui să aibă și o bună capacitate de apreciere a complexității algoritmilor) ;
 - cunoașterea unor noțiuni privind analiza și proiectarea aplicațiilor de gestiune economică ;
 - cunoașterea și utilizarea unui sistem de gestiune a bazelor de date, a unor procesoare de calcul tabelar etc. ;
 - cunoașterea și utilizarea celor mai uzitate programe utilitare, editoare de texte și editoare grafice, pachete de programe de compresie (arhivare), programe antivirus, noțiuni primare de inginerie de sistem etc. :

- cunoașterea principalelor modalități de exploatare a facilităților oferite de rețelele locale și interconectate, serviciile de Internet, documentele html, facilitățile multimedia etc.
- e) Formarea unei conduite și a unei moralități profesionale reprezintă o competență de dezvoltare personală esențială. În informatică, respectarea strictă a eticii profesionale este o necesitate impusă de respectarea legii copyright-ului. Elevii trebuie să conștientizeze impactul social al dezvoltării informaticii, care poate chiar să modifice societatea, de aici rezultând necesitatea înțelegerii rolului pe care ea îl are în schimbările din viața socială, economică, a aspectelor etice care derivă din aceste schimbări, a avantajelor și riscurilor impuse de utilizarea calculatoarelor. Elevii trebuie să cunoască prevederile legale cu privire la dreptul de autor, confidențialitatea informațiilor, protecția 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 cunoaștere a istoricului dezvoltării informaticii ca domeniu al culturii, a realității și a perspectivelor, fără impunerea respectului față de valorile materiale și umane, precum și față de efortul colectivului din care elevul face sau va face parte. Acest aspect trebuie avut în vedere pe toată durata școlarizării elevului și 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 și ilustrate prin înseși produsele lor informatice, face din imaginea unui text-sursă, din modul de organizare a instrucțiunilor în program, o oglindă fidelă a personalității intelectuale și sociale a elevului.

2.2.3. Proiectarea competențelor

Competența școlară este un ansamblu de cunoștințe, capacități, deprinderi și atitudini dobândite de elev în activitatea de învățare și utilizate în contexte specifice, adaptate vârstei elevului și nivelului cognitiv al acestuia, în vederea rezolvării unor probleme cu care acesta se poate confrunta în viața reală.

Succesul oricărei activități didactice este condiționat de claritatea alegerii competențelor pe care aceasta urmărește să le formeze sau să le dezvolte. Mai mult decât în oricare alt domeniu, învățământului de informatică îi este caracteristică intenționalitatea – orientarea către dezvoltarea unor competențe care determină unele schimbări și transformări ce pot fi controlate și dirijate. În acest spirit, cea mai importantă condiție pentru reușita predării informaticii face referire la structurarea, conștientizarea și ierarhizarea unor competențe generale și specifice, adaptate particularităților de vârstă ale elevilor, conținutului cunoștințelor și pregătirii lor științifice și metodice.

Poate fi aplicat următorul algoritm pentru formarea competențelor :

Cunoștințe aplicabile → Capacități, Abilități, Priceperi → Conștientizare → Atitudini, Comportamente pozitive → *Competență*

Competențele generale ale predării informaticii au anumite determinări care trebuie să pună în evidență :

- importanța informaticii în lumea contemporană, în știință, în tehnică sau economie ;
- necesitatea învățământului de informatică și rolul acestuia în formarea culturii generale și nu numai ;
- necesitatea dezvoltării capacității intelectuale și a gândirii algoritmice ;
- necesitatea formării elevului pentru activitățile viitoare, ca utilizator de calculator, la diferite niveluri.

Fixarea competențelor generale ale informaticii trebuie să răspundă cel puțin la următoarele două întrebări :

- De ce se predă informatica în școală ?
- Ce se urmărește prin includerea ei în planul de învățământ ?

Predarea științelor informatice în școală include cu siguranță :

- trezirea interesului pentru studiul acestora ;
- formarea priceperilor și deprinderilor de bază în utilizarea și exploatarea calculatoarelor ;
- stimularea creativității ;
- integrarea utilizării informaticii în modul de gândire și de viață al elevului.

În afara competențelor generale pe care le vizează, informatica participă – prin mijloace ce-i sunt proprii – la modelarea personalității, nu numai sub aspect intelectual, ci și sub aspect estetic și moral (partea estetică vizează programarea ca artă, iar personalitatea autorului se manifestă prin opera sa ; partea morală se referă la faptul că activitatea în domeniul informaticii nu se poate desfășura în afara unei etici profesionale sănătoase, dacă ne gândim doar la pericolul hackerilor și la relația defec-tuoasă a acestora cu cyberspațiul). Din competențele generale, putem desprinde anumite competențe specifice care pot fi la rândul lor structurate pe trei niveluri :

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

La nivelul elevului, competențele specifice sunt :

- integrarea și asimilarea cunoștințelor cuprinse în programă ;
- memorarea activă a acestor cunoștințe ;
- dezvoltarea judecății deductive și inductive ;
- conștientizarea procedeele ce stau la baza raționamentelor ;
- formarea capacității de analiză și sinteză ;
- formarea capacității de structurare și planificare ;
- formarea capacității de abordare a unei probleme complexe.

La nivelul profesorului, competențele specifice se referă la capacitatea de apreciere a fenomenelor și rezultatelor.

Nivelul acțiunii are în vedere asimilarea de către elev a noțiunilor și aplicarea lor în practică. Pe baza acestor considerații, se pot formula și delimita competențele

specifice ale fiecărui capitol, lecție etc., cu detalierea tuturor componentelor. Formularea competențelor specifice trebuie făcută în termeni comportamentali cât mai preciși, care să excludă formulările vagi, iar acest lucru presupune :

- identificarea performanței finale care trebuie realizată ;
- descrierea în detaliu a condiției esențiale în care se poate produce comportamentul respectiv ;
- precizarea nivelului de performanță la care trebuie să se ajungă pentru a fi acceptată ca atare.

În același 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 ?

2.2.4. *Analiza resurselor*

În acest moment trebuie să răspundem la întrebarea : cum pot dezvolta competențele propuse ? Sunt necesare :

- o analiză a resurselor psihologice, care necesită cunoștințe de psihologia copilului, a capacității de învățare, a particularităților de vârstă și natură psihică, a motivației învățării etc. ;
- o analiză a resurselor materiale ;
- o analiză a conținutului învățării.

Programa școlară determină conținutul învățării, dar acest conținut este prelucrat după două categorii de competențe :

- cognitive (ce va ști elevul ?) ;
- comportamentale, acționale și atitudinale (ce va putea face elevul ?).

Clasificarea anterioară trebuie să stea la baza întocmirii planificării calendaristice, care se poate realiza după următoarea procedură :

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

Conform recomandărilor existente pe site-ul ministerului (www.edu.ro), planificarea calendaristică anuală la disciplina informatică, profilul matematică-informatică, intensiv informatică poate avea următorul format (conținuturile sunt corespunzătoare programei școlare în vigoare pentru anul școlar 2015-2016).

Unitatea școlară :
 Disciplina INFORMATICĂ
 Număr ore pe săptămână : o oră – teorie

Profesor :
 Clasa a IX-a

Planificare calendaristică

Programa aprobată cu O.M. nr. 5099/09.09.2009
 Filiera teoretică
 Profil matematică-informatică/intensiv informatică

<i>Unități de învățare</i>	<i>Competențe specifice</i>	<i>Conținuturi</i>	<i>Nr. ore alocate</i>	<i>Săpt.</i>	<i>Obs.</i>
Informatică și societate	1.1 1.2	Definirea informaticii ca știință. Rolul informaticii în societate. Studii de caz ale unor situații sociale, în abordare informatizată.	1	S1	
Identificarea datelor care intervin într-o problemă și a tipurilor acestora	2.1	Date cu care lucrează algoritmi (constante, variabile, expresii). Clasificarea datelor. Tipuri de date. Operații asupra datelor. Operatori. Clasificarea operatorilor. Precedența operatorilor. Expresii. <i>Evaluare sumativă.</i>	3	S2-S4	
Elaborarea algoritmilor de rezolvare a problemelor și implementarea lor într-un limbaj de programare	2.1 3.1 3.2 3.3 4.1 4.5	Etapele rezolvării problemelor. Noțiunea de algoritm. Caracteristici. Reprezentarea algoritmilor în pseudocod. Principiile programării structurate. Structuri de bază : structura liniară, structura alternativă, structura repetitivă. <i>Evaluare sumativă.</i> Algoritmi elementari 1. Prelucrarea numerelor : <ul style="list-style-type: none"> • prelucrarea cifrelor unui număr (de exemplu, suma cifrelor, inversul unui număr, testarea proprietății de palindrom etc.) ; • probleme de divizibilitate (de exemplu, determinarea divizorilor unui număr, determinarea c.m.m.d.c./c.m.m.m.c., testarea primalității, descompunere în factori primi etc.) ; • calculul unor expresii simple (sume, produse etc.) . <i>Evaluare sumativă.</i> 2. Prelucrarea unor secvențe de valori : <ul style="list-style-type: none"> • determinare minim/maxim ; 	8		
				S5-S12	

		<ul style="list-style-type: none"> • verificarea unei proprietăți (de exemplu, dacă toate elementele din secvență sunt numere perfecte etc.) ; • calculul unor expresii în care intervin valori din secvență (de exemplu : numărarea elementelor pare/impare etc.) ; • generarea șirurilor recurente (de exemplu : șirul Fibonacci, progresii aritmetice și geometrice). <i>Evaluare sumativă.</i>			
		<p>Elementele de bază ale limbajului de programare. Noțiuni introductive.</p> <p>Structura programelor.</p> <p>Vocabularul limbajului.</p> <p>Tipuri simple de date (standard).</p> <p>Constante, variabile, expresii.</p> <p>Citirea/scrierea datelor.</p> <p>Reprezentarea algoritmilor într-un limbaj de programare.</p> <p>Structuri de control implementate în limbajul de programare.</p> <i>Evaluare sumativă.</i>	5	S13-S17	
Fișiere text	4.4 4.5	<p>Definire, operații specifice :</p> <ul style="list-style-type: none"> • citirea și afișarea datelor folosind fișiere text 	1	S18	
Tablouri unidimensionale	2.1 3.1 3.3 4.1 4.2.	<p>Algoritmi fundamentali de prelucrare a datelor structurate în tablouri :</p> <ul style="list-style-type: none"> • parcurgerea tablourilor unidimensionale ; • interschimbarea, deplasarea, ștergerea și inserarea de elemente ; • operații cu mulțimi ; • căutare secvențială, căutare binară ; • sortare ; • interclasare ; • secvențe și subșiruri. <i>Evaluare sumativă.</i>	11	S19-S29	
Tablouri bidimensionale	2.1 3.1 3.3 4.1 4.2 4.3 4.5	<ul style="list-style-type: none"> • parcurgerea tablourilor bidimensionale pe linii/coloane ; • tablouri bidimensionale pătrate, diagonale. <i>Evaluare sumativă.</i>	5	S30-S34	
Aplicarea algoritmilor în prelucrarea datelor	5.1 5.2	<p>Aplicații interdisciplinare (specifice profilului).</p> <p>Analiza eficienței unui algoritm.</p>	2	S35-S36	

Competențe generale și specifice :

CG1. *Identificarea conexiunilor dintre informatică și societate.*

CS1.1. Identificarea aplicațiilor informaticii în viața socială.

CS1.2. Recunoașterea situațiilor în care este necesară prelucrarea algoritmică a informațiilor.

CG2. *Identificarea datelor care intervin într-o problemă și a relațiilor dintre acestea.*

CS2.1. Descrierea unei succesiuni de operații prin care se obțin, din datele de intrare, datele de ieșire.

CG3. *Elaborarea algoritmilor de rezolvare a problemelor.*

CS3.1. Analizarea enunțului unei probleme și stabilirea pașilor de rezolvare a problemei.

CS3.2. Reprezentarea algoritmilor în pseudocod.

CS3.3. Respectarea principiilor programării structurate în procesul de elaborare a algoritmilor.

CG4. *Implementarea algoritmilor într-un limbaj de programare.*

CS4.1. Transcrierea algoritmilor din pseudocod într-un limbaj de programare.

CS4.2. Identificarea necesității structurării datelor în tablouri.

CS4.3. Prelucrarea datelor structurate.

CS4.4. Utilizarea fișierelor text pentru introducerea datelor și extragerea rezultatelor.

CS4.5. Utilizarea unui mediu de programare (pentru limbajul Pascal sau pentru limbajul C/C++).

CG5. *Aplicarea algoritmilor fundamentali în prelucrarea datelor.*

CS5.1. Elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specializării.

CS5.2. Alegerea unui algoritm eficient de rezolvare a unei probleme.

Unitatea școlară:

Profesor:

Disciplina INFORMATICĂ

Clasa a IX-a

Număr ore pe săptămână : 3 ore – aplicații practice de laborator

Planificare calendaristică

Programa aprobată cu O.M. nr. 5099/09.09.2009

Filiera teoretică

Profil matematică-informatică/intensiv informatică

<i>Unități de învățare</i>	<i>Competențe specifice</i>	<i>Conținuturi</i>	<i>Nr. ore alocate</i>	<i>Săpt.</i>	<i>Obs.</i>
Identificarea datelor care intervin într-o problemă și a tipurilor acestora	2.1 4.1 4.5	<ul style="list-style-type: none"> • Date cu care lucrează algoritmi (constante, variabile). Clasificarea datelor. Tipuri de date standard. • Citirea și scrierea datelor de la/la consolă. • Operații asupra datelor. Expresii. 	12	S1-S4	
Elaborarea algoritmilor de rezolvare a problemelor și implementarea lor într-un limbaj de programare	2.1 3.1 3.2 3.3 4.1 4.5	<p>Aplicații cu structuri de bază :</p> <ul style="list-style-type: none"> • structura liniară (exemple : calculul ariei și al perimetrului unor figuri geometrice) ; • structura alternativă (exemple : rezolvarea ecuației de gradul 2, verificarea dacă trei numere pot fi laturile unui triunghi) ; • structura repetitivă (exemple : parcurgerea numerelor naturale dintr-un interval în ordine crescătoare sau descrescătoare, sau cu un pas precizat, folosind toate cele trei tipuri de structuri repetitive). <p>Algoritmi elementari.</p> <p>1. Prelucrarea numerelor :</p> <ul style="list-style-type: none"> • prelucrarea cifrelor unui număr (exemple : suma cifrelor unui număr, inversul unui număr, testarea proprietății de palindrom, numărul de cifre dintr-un număr etc.) ; • probleme de divizibilitate (exemple : determinarea divizorilor unui număr, determinarea c.m.m.d.c./c.m.m.m.c., testarea primalității, descompunerea în factori primi etc.) ; • calculul unor expresii simple (sume, produse etc.). 	21	S5-S11	

			<p><i>Evaluare sumativă.</i></p> <p>2. Prelucrarea unor secvențe de valori :</p> <ul style="list-style-type: none"> • determinare minim/maxim ; • verificarea unei proprietăți (exemplu : dacă toate elementele din secvență sunt numere prime etc.) ; • calculul unor expresii în care intervin valori din secvență (exemplu : numărarea elementelor pare/impare, însumarea sau înmulțirea numerelor dintr-o secvență etc.) ; • generarea șirurilor recurente (de exemplu : șirul Fibonacci, progresii aritmetice și geometrice). <p><i>Evaluare sumativă.</i></p> <ul style="list-style-type: none"> • Mediul limbajului de programare studiat. Prezentare generală. • Editarea programelor-sursă. Compilare, rulare, depanare. • Structura programelor. • Elemente de vocabular al limbajului. • Scrierea pe ecran. 	3	S12	
Elementele de bază ale limbajului de programare	2.1 4.1 4.5		<p>Elementele de bază ale limbajului de programare. Noțiuni introductive :</p> <ul style="list-style-type: none"> • Structura programelor. • Vocabularul limbajului. • Tipuri simple de date (standard). • Constante, variabile, expresii. • Citirea/scrierea datelor. • Reprezentarea algoritmilor într-un limbaj de programare. • Structuri de control implementate în limbajul de programare. <p><i>Evaluare sumativă.</i></p>	9	S13-S15	
Fișiere text	4.4 4.5		<p>Definire, operații specifice :</p> <ul style="list-style-type: none"> • citirea și afișarea datelor folosind fișiere text, aplicații cu fișiere text (exemplu : ordonarea numerelor dintr-un fișier, interclasarea conținutului a două fișiere care memorează numere ordonate crescător etc.). 	3	S16	
Tablouri unidimensionale	4.1 4.2 4.3 4.5		<p><i>Algoritmi fundamentali de prelucrare a datelor structurate în tablouri :</i></p> <ul style="list-style-type: none"> • aplicații cu parcurgerea tablourilor unidimensionale (exemple : citirea și afișarea elementelor tablourilor) ; • aplicații cu interschimbarea, deplasarea, ștergerea și inserarea de elemente (exemple : inversarea ordinii elementelor unui tablou, ștergerea elementelor cu o anumită proprietate, permutări circulare) ; 	36	S17-S28	

		<ul style="list-style-type: none"> • elemente distincte (exemplu : transformarea unui vector în mulțime prin eliminarea elementelor nedistincte) ; • operații cu mulțimi (reuniune, intersecție, diferență, apartenență, incluziune) ; • căutare secvențială, căutare binară ; <i>Evaluare sumativă.</i> <ul style="list-style-type: none"> • sortare (selecție, inserție, <i>bubble sort</i>, numărare) ; • aplicații cu vectori de frecvență (exemplu : frecvența cifrelor unui număr, ordonarea de cifre) ; • interclasare (exemplu : operațiile cu mulțimi ordonate) ; • secvențe și subșiruri, generarea submulțimilor unei mulțimi ; • aplicații cu conversii între diferite sisteme de numerație (cu șiruri de cifre) ; <i>Evaluare sumativă.</i>			
Tablouri bidimensionale	4.1 4.2 4.3 4.5	<ul style="list-style-type: none"> • parcurgerea tablourilor bidimensionale pe linii/coloane (exemple : elemente minime/maxime, vecinii unui element din matrice, sume pe linii sau coloane, ștergerea sau inserarea de linii și coloane etc.) ; • tablouri bidimensionale pătratice, diagonale, împărțirea matricii în zone în funcție de diagonale, generarea unei matrici după o regulă etc.) . <i>Evaluare sumativă.</i>	15	S29-S33	
Aplicarea algoritmilor în prelucrarea datelor	5.1 5.2	Aplicații interdisciplinare (specifice profilului) : <ul style="list-style-type: none"> • Operații cu fracții și numere raționale (simplificarea fracțiilor, adunare, scădere, înmulțire, împărțire, comparare). • Generarea primilor n termeni ai unei progresii. • Aplicații geometrice (distanța dintre două puncte, volumul corpurilor regulate, centrul de greutate al unei mulțimi de puncte etc.) . • Determinarea punctului de intersecție a două mobile în mișcare rectilinie și uniformă. • Determinarea masei moleculare a unui compus chimic. Analiza eficienței unui algoritm. Analiza eficienței a doi sau mai mulți algoritmi care rezolvă aceeași problemă. <i>Evaluare sumativă.</i>	9	S34-S36	

Competențe generale și specifice :

CG1. *Identificarea conexiunilor dintre informatică și societate.*

CS1.1. Identificarea aplicațiilor informaticii în viața socială.

CS1.2. Recunoașterea situațiilor în care este necesară prelucrarea algoritmică a informațiilor.

CG2. *Identificarea datelor care intervin într-o problemă și a relațiilor dintre acestea.*

CS2.1. Descrierea unei succesiuni de operații prin care se obțin, din datele de intrare, datele de ieșire.

CG3. *Elaborarea algoritmilor de rezolvare a problemelor.*

CS3.1. Analizarea enunțului unei probleme și stabilirea pașilor de rezolvare a problemei.

CS3.2. Reprezentarea algoritmilor în pseudocod.

CS3.3. Respectarea principiilor programării structurate în procesul de elaborare a algoritmilor.

CG4. *Implementarea algoritmilor într-un limbaj de programare.*

CS4.1. Transcrierea algoritmilor din pseudocod într-un limbaj de programare.

CS4.2. Identificarea necesității structurării datelor în tablouri.

CS4.3. Prelucrarea datelor structurate.

CS4.4. Utilizarea fișierelor text pentru introducerea datelor și extragerea rezultatelor.

CS4.5. Utilizarea unui mediu de programare (pentru limbajul Pascal sau pentru C/C++).

CG5. *Aplicarea algoritmilor fundamentali în prelucrarea datelor.*

CS5.1. Elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specializării.

CS5.2. Alegerea unui algoritm eficient de rezolvare a unei probleme.

După realizarea planificării anuale este necesară proiectarea unităților de învățare, detaliate la nivel de lecție. Unitățile de învățare :

- sunt teme stabilite de profesor, care constituie capitole sau subcapitole coerente din punctul de vedere al conținutului și care pot fi evaluate sumativ ;
- în rubrica „Competențe specifice” se trec simbolurile competențelor specifice din programa școlară ;
- conținuturile selectate sunt extrase din lista de conținuturi din programă ;
- numărul de ore alocate se stabilește de către profesor în funcție de experiența acestuia și de nivelul de achiziții ale elevilor.

Planificarea este, desigur, orientativă, iar eventualele modificări determinate de aplicarea efectivă la clasă se pot consemna în rubrica „Observații”. O planificare anuală corect întocmită trebuie să acopere integral programa școlară la nivel de competențe specifice și conținuturi.

Vom da un exemplu de planificare a unităților de învățare din planificările anuale prezentate anterior.

Unitatea școlară :
Disciplina INFORMATICĂ

Profesor :
Clasa a IX-a

Planificare unități de învățare

Unitatea de învățare : Informatică și societate

Număr de ore : o oră

Forma de organizare :

- teorie : o oră

Conținuturi	Competențe specifice	Activități de învățare	Resurse	Evaluare
Definirea informaticii ca știință Rolul informaticii în societate Studii de caz ale unor situații sociale, în abordare informatizată	1.1 1.2	Exemple de aplicații informatice din viața socială Exemple de situații în care este necesară prelucrarea algoritmică a informațiilor	în clasă : – conversația – exemplificarea	evaluare curentă

Unitatea de învățare : Identificarea datelor care intervin într-o problemă și a tipurilor acestora.

Număr de ore : 2 + 6 ore.

Forma de organizare :

- teorie : 2 ore
- activitate practică : 6 ore

Conținuturi	Competențe specifice	Activități de învățare	Resurse	Evaluare
Date cu care lucrează algoritmi (constante, variabile, expresii). Clasificarea datelor. Tipuri de date. Operații asupra datelor. Operatori. Clasificarea operatorilor. Precedența operatorilor. Expresii. Citirea și scrierea datelor de la/la consolă.	2.1 4.1 4.5	Studii de caz cu exemplificarea succesiunilor de operații prin care se obțin, din datele de intrare, datele de ieșire Prezentarea mediului de programare Exerciții de citire și scriere a datelor de la/la consolă utilizând mediul de programare	în clasă : – conversația – exemplificarea – exercițiul în laborator : conversația ; observația ; exercițiul	evaluare curentă evaluare sumativă la sfârșitul unității de învățare

Unitatea de învățare : Elaborarea algoritmilor de rezolvare a problemelor și implementarea lor într-un limbaj de programare

Număr de ore : 5 + 15 ore

Forma de organizare :

- teorie : 5 ore
- activitate practică : 15 ore

<i>Conținuturi</i>	<i>Competențe specifice</i>	<i>Activități de învățare</i>	<i>Resurse</i>	<i>Evaluare</i>
<p>Etapela rezolvării problemelor.</p> <p>Noțiunea de algoritm. Caracteristici.</p> <p>Reprezentarea algoritmilor în pseudocod.</p> <p>Principiile programării structurate.</p> <p>Structuri de bază : structura liniară, structura alternativă, structura repetitivă.</p>	<p>2.1</p> <p>3.1</p> <p>3.2</p> <p>3.3</p> <p>4.1</p>	<p>Exemple de algoritmi din diferite domenii de activitate</p> <p>Exerciții de descriere a unor succesiuni de operații prin care se obțin, din datele de intrare, datele de ieșire</p>	<p>în clasă :</p> <ul style="list-style-type: none"> – conversația – exemplificarea – exercițiul 	<p>evaluare curentă</p>
<p>Aplicații cu structuri de bază :</p> <ul style="list-style-type: none"> • structura liniară (exemple : calculul ariei și al perimetrului unor figuri geometrice) ; • structura alternativă (exemple : rezolvarea ecuației de gradul 2, se verifică dacă trei numere pot fi laturile unui triunghi) ; • structura repetitivă (exemple : parcurgerea numerelor naturale dintr-un interval în ordine crescătoare sau descrescătoare, sau cu un pas precizat, folosind toate cele trei tipuri de structuri repetitive). <p>Algoritmi elementari</p> <p>1. Prelucrarea numerelor :</p> <ul style="list-style-type: none"> • prelucrarea cifrelor unui număr (de exemplu, suma cifrelor, inversul unui număr, testarea proprietății de palindrom etc.) ; 	<p>4.5</p>	<p>Studii de caz ce implică analizarea enunțului unei probleme și stabilirea pașilor de rezolvare a problemei</p> <p>Exerciții de utilizare a structurilor de bază</p> <p>Exerciții de reprezentare a algoritmilor în pseudocod</p> <p>Elaborare de algoritmi cu respectarea principiilor programării structurate</p> <p>Exerciții de transcriere a algoritmilor din pseudocod într-un limbaj de programare.</p> <p>Lucru în mediul de programare pentru familiarizarea elevului cu interfața și opțiunile acestuia</p>	<p>în laborator :</p> <ul style="list-style-type: none"> – exercițiul – exemplificarea – problematizarea – algoritimizarea 	<p>evaluare sumativă la sfârșitul unității de învățare</p>

<ul style="list-style-type: none">• probleme de divizibilitate (de exemplu, determinarea divizorilor unui număr, determinarea c.m.m.d.c./c.m.m.m.c., testare primalitate, descompunere în factori primi etc.);• calculul unor expresii simple (sume, produse etc.). <p>2. Prelucrarea unor secvențe de valori :</p> <ul style="list-style-type: none">• determinare minim/maxim ;• verificarea unei proprietăți (de exemplu, dacă toate elementele din secvență sunt numere perfecte etc.) ;• calculul unor expresii în care intervin valori din secvență (de exemplu : numărarea elementelor pare/impare etc.) ;• generarea șirurilor recurente (de exemplu : șirul Fibonacci, progresii aritmetice și geometrice).				
--	--	--	--	--

Unitatea de învățare : Elementele de bază ale limbajului de programare

Număr de ore : 5 + 15 ore

Forma de organizare :

- teorie : 5 ore
- activitate practică : 15 ore

Conținuturi	Competențe specifice	Activități de învățare	Resurse	Evaluare
<p>Noțiuni introductive.</p> <p>Structura programelor.</p> <p>Vocabularul limbajului.</p> <p>Tipuri simple de date (standard).</p> <p>Constante, variabile, expresii.</p> <p>Citirea/scrutarea datelor.</p> <p>Reprezentarea algoritmilor într-un limbaj de programare.</p> <p>Structuri de control implementate în limbajul de programare.</p> <p>Mediul limbajului de programare studiat.</p> <p>Prezentare generală. Editarea programelor-sursă. Compilare, rulare, depanare.</p>	<p>2.1</p> <p>4.1</p> <p>4.5</p>	<p>Exerciții de scriere a programelor C++ cuprinzând :</p> <ul style="list-style-type: none"> • tipuri de date standard ; • constante, variabile ; • operatori ; • expresii C++ ; • citiri/scrutări în C++. <p>Lucrul cu mediul Codeblocks :</p> <ul style="list-style-type: none"> • lansare ; • meniuri ; • editarea unui program ; • compilarea unui program ; • lansarea în execuție a unui program. 	<p>în clasă :</p> <ul style="list-style-type: none"> - conversația - exercițiul - exemplificarea - problematizarea în laborator ; exercițiul ; ob- - exemplificarea ; ob- - servarea 	<p>evaluare curentă / evaluare sumativă la sfârșitul unității de învățare</p>

Unitatea de învățare : Fișiere text

Număr de ore : 2 + 6 ore

Forma de organizare :

- teorie : 2 ore
- activitate practică : 6 ore

Conținuturi	Competențe specifice	Activități de învățare	Resurse	Evaluare
<p>Definire, operații specifice : citirea și afișarea datelor folosind fișiere text, aplicații cu fișiere text (exemplu : ordonarea numerelor dintr-un fișier, interclasarea conținutului a două fișiere care memorează numere ordonate crescător etc.).</p>	<p>4.4</p> <p>4.5</p>	<p>Exerciții de utilizare a fișierelor text pentru introducerea datelor și extragerea rezultatelor</p> <p>Lucrul în mediul de programare Codeblocks pentru limbajul C++</p> <p>Exerciții de citire/scrutări cu/fără format</p>	<p>în clasă :</p> <ul style="list-style-type: none"> - conversația - exercițiul - problematizarea în laborator ; exercițiul ; observarea ; implementarea 	<p>evaluare curentă/evaluare sumativă la sfârșitul unității de învățare</p>

Unitatea de învățare: Tablouri unidimensionale

Număr de ore : 11 + 33 ore

Forma de organizare :

- teorie : 11 ore
- activitate practică : 33 ore

<i>Conținuturi</i>	<i>Competențe specifice</i>	<i>Activități de învățare</i>	<i>Resurse</i>	<i>Evaluare</i>
<p>Algoritmi fundamentali de prelucrare a datelor structurate în tablouri</p> <ul style="list-style-type: none"> • Aplicații cu parcurgerea tablourilor unidimensionale (exemple : citirea și afișarea elementelor tablourilor). • Aplicații cu interschimbarea, deplasarea, ștergerea și inserarea de elemente (exemple : inversarea ordinii elementelor unui tablou, ștergerea elementelor cu o anumită proprietate, permutări circulare). • Elemente distincte (exemplu : transformarea unui vector în mulțime prin eliminarea elementelor nedistincte). • Operații cu mulțimi (reuniune, intersecție, diferență, apartenență, incluziune). • Căutare secvențială, căutare binară. • Sortare (selecție, inserție, <i>bubble sort</i>, numărare). • Aplicații cu vectori de frecvență (exemplu : frecvența cifrelor unui număr, ordonare de cifre). • Interclasare (exemplu : operațiile cu mulțimi ordonate). • Secvențe și subșiruri, generarea submulțimilor unei mulțimi. • Aplicații cu conversii între diferite sisteme de numerație (cu șiruri de cifre). 	<p>2.1 3.1 3.3 4.1 4.2 4.3 4.5</p>	<p>Exerciții de transcriere a algoritmilor ce utilizează tipul tablou din pseudocod într-un limbaj de programare</p> <p>Exerciții de identificare a necesității structurării datelor în tablouri</p> <p>Exerciții de prelucrare a datelor structurate în tablouri</p> <p>Lucrul în mediul de programare Codeblocks pentru implementarea algoritmilor ce prelucrează datele structurate în tablouri</p>	<p>în clasă :</p> <ul style="list-style-type: none"> – conversația – exercițiul – exemplificarea – problematizarea – algoritimizarea <p>în laborator :</p> <ul style="list-style-type: none"> – exercițiul – exemplificarea – problematizarea – algoritimizarea – implementarea 	<p>evaluare curentă/ evaluare sumativă la sfârșitul unității de învățare</p>

Unitatea de învățare: Tablouri bidimensionale

Număr de ore : 5 + 15 ore

Forma de organizare :

- teorie : 5 ore
- activitate practică : 15 ore

<i>Conținuturi</i>	<i>Competențe specifice</i>	<i>Activități de învățare</i>	<i>Resurse</i>	<i>Evaluare</i>
<p>Algoritmi fundamentali de prelucrare a datelor structurate în tablouri bidimensionale</p> <ul style="list-style-type: none"> • parcurgerea tablourilor bidimensionale pe linii/coloane (exemple : elemente minime/maxime, vecinii unui element din matrice, sume pe linii sau coloane, ștergerea sau inserarea de linii și coloane etc.) ; • tablouri bidimensionale pătratice, diagonale, împărțirea matricii în zone în funcție de diagonale, generarea unei matrici după o regulă etc.). 	<p>2.1 3.1 3.3 4.1 4.2 4.3 4.5</p>	<p>Exerciții de transcriere a algoritmilor ce utilizează tipul matrice din pseudocod într-un limbaj de programare</p> <p>Exerciții de identificare a necesității structurării datelor în tablouri bidimensionale</p> <p>Exerciții de prelucrare a datelor structurate în matrice</p> <p>Lucrul în mediul de programare Codeblocks pentru implementarea algoritmilor ce prelucreză datele structurate în tablouri bidimensionale</p>	<p>în clasă :</p> <ul style="list-style-type: none"> – conversația – exercițiul – exemplificarea – problematizarea – algoritimizarea <p>în laborator : exercițiul ; exemplificarea problematizarea ; algoritimizarea ; implementarea</p>	<p>evaluare curentă/evaluare sumativă la sfârșitul unității de învățare</p>

Unitatea de învățare: Aplicarea algoritmilor în prelucrarea datelor

Număr de ore : 4 + 12 ore

Forma de organizare :

- teorie : 4 ore
- activitate practică : 12 ore

<i>Conținuturi</i>	<i>Competențe specifice</i>	<i>Activități de învățare</i>	<i>Resurse</i>	<i>Evaluare</i>
<p>Aplicații interdisciplinare (specifice profilului)</p> <ul style="list-style-type: none"> • Operații cu fracții și numere raționale (simplificarea fracțiilor, adunare, scădere, înmulțire, împărțire, comparare). • Generarea primilor n termeni ai unei progresii. • Aplicații geometrice (distanța dintre două puncte, volumul corpurilor regulate, centrul de greutate al unei mulțimi de puncte etc.). • Determinarea punctului de intersecție a două mobile în mișcare rectilinie și uni-formă. • Determinarea masei moleculare a unui compus chimic. <p>Analiza eficienței unui algoritm. Analiza eficienței a doi sau mai mulți algoritmi care rezolvă aceeași problemă.</p>	<p>5.1</p> <p>5.2</p>	<p>Exerciții de elaborare a algoritmilor de rezolvare a unor probleme din aria curriculară a specializării</p> <p>Studii de caz în vederea alegerii unui algoritm eficient de rezolvare a unei probleme</p>	<p>în clasă :</p> <ul style="list-style-type: none"> – conversația – exercițiul – exemplificarea – problematizarea – algoritimizarea <p>în laborator : exercițiul ; problematizarea ; algoritimizarea ; implementarea</p>	<p>evaluare curentă/ evaluare sumativă la sfârșitul unității de învățare</p>

Resurse bibliografice :

Emanuela Cerchez, *Informatica. Culegere de probleme pentru liceu*, Editura Polirom, Iași, 2002.

Emanuela Cerchez, Marinel Șerban, *Informatica pentru gimnaziu*, Editura Polirom, Iași, 2002.

Emanuela Cerchez, Marinel Șerban, *Informatica. Manual pentru clasa a IX-a*, Editura Didactică și Pedagogică, București, 2004.

Emanuela Cerchez, Marinel Șerban, *PC. Pas cu pas*, ed. a II-a, Editura Polirom, Iași, 2005.

Emanuela Cerchez, Marinel Șerban, *Programarea în limbajul C/C++ pentru liceu*, vol. I, Editura Polirom, Iași, 2005.

Emanuela Cerchez, Marinel Șerban, Alexandru Periețanu, Dragoș Răducanu, *Elemente de bază ale limbajului C/C++*, soft educațional, 2006.

Mariana Miloșescu, *Informatica. Manual pentru clasa a IX-a*, Editura Didactică și Pedagogică, București, 2006.

Competențe generale și specifice :

CG1. *Identificarea conexiunilor dintre informatică și societate.*

CS1.1. Identificarea aplicațiilor informaticii în viața socială.

CS1.2. Recunoașterea situațiilor în care este necesară prelucrarea algoritmică a informațiilor.

CG2. *Identificarea datelor care intervin într-o problemă și a relațiilor dintre acestea.*

CS2.1. Descrierea unei succesiuni de operații prin care se obțin, din datele de intrare, datele de ieșire.

CG3. *Elaborarea algoritmilor de rezolvare a problemelor.*

CS3.1. Analizarea enunțului unei probleme și stabilirea pașilor de rezolvare a problemei.

CS3.2. Reprezentarea algoritmilor în pseudocod.

CS3.3. Respectarea principiilor programării structurate în procesul de elaborare a algoritmilor.

CG4. *Implementarea algoritmilor într-un limbaj de programare.*

CS4.1. Transcrierea algoritmilor din pseudocod într-un limbaj de programare.

CS4.2. Identificarea necesității structurării datelor în tablouri.

CS4.3. Prelucrarea datelor structurate.

CS4.4. Utilizarea fișierelor text pentru introducerea datelor și extragerea rezultatelor.

CS4.5. Utilizarea unui mediu de programare (pentru limbajul Pascal sau C/C++).

CG5. *Aplicarea algoritmilor fundamentali în prelucrarea datelor.*

CS5.1. Elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specializării.

CS5.2. Alegerea unui algoritm eficient de rezolvare a unei probleme.

2.2.5. *Elaborarea strategiei didactice*

Elaborarea strategiei presupune alegerea unui sistem de forme, metode, materiale și mijloace. De selectarea și combinarea acestora depinde reușita activității didactice. Selectarea tehnicilor de învățare se face în funcție de materialele didactice, care variază în funcție de metodele utilizate, iar metodele sunt determinate de competențele de dezvoltat, de conținut și de nivelul colectivului de elevi. Succesul lecției depinde și de îmbinarea judicioasă a celor „3M” (Metode, Materiale, Mijloace). Această corelare este gândită din momentul întocmirii scenariului didactic, prin care se înțelege o descriere anticipată a desfășurării pas cu pas a unei lecții. Gradul de detaliu vizează aspectele esențiale ale condiției elevului și schimbările pe care dorim să le realizăm.

2.2.6. *Clasificarea competențelor*

În momentul proiectării unei lecții, vom avea în vedere formarea de :

- competențe sub raport stadial ;
- competențe sub raport psihopedagogic.

a) Competențe sub raport stadial

Sunt competențe pe care le putem împărți, la rândul lor, în :

- *Competențe-cheie*, care definesc elementele și sarcinile rezultate din delimitarea scopului final al educației, cum ar fi cele legate de formarea unei personalități puternice, complexe, cu o mare dispoziție spre inițiativă și creativitate. Avem în vedere :
 - formarea capacității de asimilare a cunoștințelor de către elevi ;
 - formarea capacității de transfer a cunoștințelor și a experienței deja dobândite la rezolvarea unor sarcini necunoscute, apărută pe parcursul derulării procesului didactic ;
 - formarea limbajului științific de specialitate ;
 - formarea unei atitudini științifice ; trebuie insuflat un respect al elevului pentru știință și importanța acesteia în evoluția sa ulterioară ; elevul trebuie să înțeleagă că procesul de cunoaștere nu se încheie într-o perioadă determinată de timp, că procesul de cercetare trebuie să prelucreză orice informație în mod critic pentru a fi eficient, fără a se face afirmații categorice/definitive.

Competențele-cheie sunt orientate spre anumite laturi ale procesului de educație : intelectuală, tehnologică, profesională, morală, estetică, fizică etc.

- *Competențe generale* ale disciplinei, care sunt formulate în planul-cadru al procesului de învățământ (privit ca un sistem complex și într-o permanentă evoluție). În primul rând, se urmărește dobândirea unei culturi generale de bază (în învățământul preuniversitar), a unei culturi de specialitate (în învățământul superior) sau chiar a unei meserii (școli de profil).

- *Competențe specifice*, care se definesc pe obiect de studiu și se formează pe parcursul unui an școlar. Ele sunt derivate din competențele generale, fiind etape în dobândirea acestora. Competențelor specifice li se asociază prin programă unități de conținut.
- *Competențele derivate* privesc îndeplinirea concretă a unor activități curente, cum ar fi cele legate de predarea unei lecții sau de exemplificarea unor teme de laborator.

b) *Competențe sub raport psihopedagogic*

Sunt competențe necesare formării de capacități intelectuale (teoretice, practice) și/sau afective. Și acestea pot fi clasificate pe mai multe categorii :

- *Competențe cognitive*. Prin acestea se urmărește formarea/dezvoltarea următoarelor capacități intelectuale :
 - cunoașterea : posibilitatea, în principal, a îndeplinirii sarcinilor legate de memorarea, reproducerea și recunoașterea materiei de asimilat ;
 - înțelegerea : se referă la transpunere, interpretare și extrapolare.

Transpunerea înseamnă reformularea unei definiții/noțiuni sau a unui rezultat cu propriile cuvinte ; de exemplu, traducerea unui algoritm dintr-o reprezentare oarecare într-un limbaj implementat.

Interpretarea înseamnă înțelegerea comportării/evoluției unui obiect/sistem dat într-un context/mediu clar precizat.

- *Competențe de extrapolare*. Au drept consecință căpătarea îndemnării de a evidenția consecințe 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 și a cunoștințelor asimilate, în vederea realizării unor lucrări cu caracter personal ;
 - evaluarea : implică posibilitatea formulării de către elevi a unor judecăți de valoare, originale (de natură științifică, socială, culturală), raportate, desigur, la cantitatea de informații acumulate până în acel moment.
- *Competențe psihomotorii*. Asemenea obiective includ formarea de percepere, capacități, deprinderi motorii/practice legate de utilizarea corectă a întregii aparatură de laborator (tastatură, mouse, joystick etc.). Totul trebuie însușit într-un mod profesional și utilizat rapid, precis, cu o bună coordonare a mișcărilor și implicând agilitate și suplețe.
- *Obiective afective (conative)*. Au scopul de a dezvolta emoții și sentimente superioare, contribuind la formarea conștiinței și conduitei morale – vizează deci, în mare, formarea intereselor, atitudinilor și valorilor etico-morale, a personalității elevului. Personalitatea poate fi formată începând cu o vârstă foarte fragedă, etapizat și utilizând idei, norme, practici și valori deja recunoscute. Deși 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 informațională și de faptul că, practic, informatica poate deveni un mediu de lucru pentru toate celelalte discipline.

2.2.7. *Formularea competențelor*

Nu considerăm că scopul principal al acestui volum este acela de a intra în detaliile elaborării unui plan sau a unei programe analitice pentru o disciplină specifică, fie ea știința calculatoarelor sau tehnologia informației și comunicării. Acestea fac parte din strategiile (pe termen scurt sau lung) de dezvoltare/promovare a disciplinei și sunt de competența consiliilor profesionale, inspectoratelor școlare, senatelor universitare, ministerului de resort etc. În momentul în care un plan de învățământ și o programă analitică sunt însă fixate, alegerea competențelor specifice din programă și stabilirea competențelor derivate pentru o lecție reprezintă obligația profesorului și constituie o parte indispensabilă a oricărei proiectări didactice. Formarea acestora presupune în plus faptul că un cadru didactic are o orientare globală și coerentă asupra întregului proces de învățământ, că el cunoaște și aplică în mod curent elementele de metodă, că procesul în sine de coordonare a învățării în clasă nu mai are secrete. În urma oricărei lecții, elevii trebuie să dobândească anumite cunoștințe, să aibă abilitatea de a le structura (analiza, sintetiza) în mod creator. Aceștia trebuie să poată avea și posibilitatea de a se manifesta direct, intervenția profesorului trebuind să fie mai degrabă discretă. Prin urmare, scopurile urmărite sunt transpuse în competențele formulate în termenii unor operații, acțiuni sau manifestări observabile și aflate în concordanță cu cerințele generale. Formarea competențelor derivate „este imediată”, putând însă avea în anumite situații și o finalitate pe un termen mai lung ; aceasta în ideea că deprinderile și cunoștințele dobândite anterior vor trebui să fie completate prin acțiuni viitoare care să contribuie decisiv la includerea lor în sistemul individual de cunoștințe, deprinderi și priceperi. Formarea de competențe trebuie să implice, eventual gradat, etape diferite de dificultate care să precizeze :

- competențe în termeni comportamentali observabili ;
- sarcini concrete de învățare, precum și contextul de realizare ;
- criteriul de succes și modul de evaluare.

2.2.8. *Momentele lecției*

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

- captarea atenției ;
- enunțarea competențelor de dezvoltat ;
- reactualizarea cunoștințelor învățate anterior ;
- prezentarea conținutului noii lecții ;

- dirijarea învățării ;
- asigurarea feedbackului ;
- intensificarea atenției ;
- asigurarea retenției și transferului de cunoștințe.

Sucesiunea și importanța lor variază de la un tip de lecție la altul. Principalele tipuri de lecții sunt :

- lecția de comunicare/însușire de noi cunoștințe ;
- lecția de formare de priceperi și deprinderi ;
- lecția de fixare și sistematizare ;
- lecția de verificare și apreciere a rezultatelor școlare ;
- lecția mixtă.

Subliniem încă o dată că lecția este, în concepția noastră, un act de creație care nu se poate încadra în șabloane. 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

Școala : –

Disciplina : Informatică

Clasa : a IX-a

Profilul : Matematică-informatică, intensiv informatică

Data : –

Profesor : –

Unitatea de învățare : Tablouri bidimensionale

Tema lecției : Parcurgerea tablourilor bidimensionale pe linii și coloane

Tipul lecției : Mixtă

Durata : 50 de minute

Competențe generale :

(CG2) Identificarea datelor care intervin într-o problemă și a relațiilor dintre acestea.

(CG3) Elaborarea algoritmilor de rezolvare a problemelor.

(CG4) Aplicarea algoritmilor fundamentali în prelucrarea datelor.

Competențe specifice :

(CS4.2.) Identificarea necesității structurării datelor în tablouri.

(CS4.3.) Prelucrarea datelor structurate.

(CS5.1.) Elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specializării.

(CS5.2.) Alegerea unui algoritm eficient de rezolvare a unei probleme.

Competențe derivate :

La sfârșitul activității didactice elevii vor fi capabili :

(CD1) Să analizeze problema propusă și să identifice necesitatea reprezentării eficiente a datelor sub formă de tablou bidimensional/unidimensional.

(CD2) Să descrie parcurgerile pe linii/coloane ale elementelor unui tablou bidimensional care intervin în rezolvarea problemelor propuse.

(CD3) Să elaboreze algoritmi eficienți de rezolvare a problemelor propuse folosind parcurgerile pe linii/coloane ale unui tablou bidimensional.

(CD4) Să implementeze secvențe de cod C++ pentru rezolvarea cerințelor problemelor propuse.

C1. Competențe cognitive

La sfârșitul lecției, elevii vor fi capabili :

C1.1 : să analizeze o problemă, să identifice necesitatea utilizării structurilor de tip tablou bidimensional și/sau unidimensional.

C1.2 : să descrie structurile de date identificate.

C1.3 : să conceapă algoritmul de rezolvare a aplicației propuse.

C2. Competențe afective

La sfârșitul lecției, elevii vor fi capabili :

C2.1 : să argumenteze corect soluția prezentată.

C2.2 : să se autoevalueze corect.

C2.3 : să dovedească curiozitate și interes pentru noțiunile prezentate.

C3. Competențe atitudinale/comportamentale

La sfârșitul lecției, elevii vor fi capabili :

C3.1 : să conștientizeze importanța alegerii unor structuri de date adecvate în rezolvarea problemelor cu ajutorul calculatorului.

C3.2 : să argumenteze eficiența structurilor alese din punctul de vedere al spațiului de memorie.

C4. Competențe acționale

La sfârșitul lecției, elevii vor fi capabili :

C4.1 : să utilizeze corect în aplicații noțiunile teoretice însușite.

C4.2 : să implementeze corect în C++ algoritmi elaborați.

Strategii didactice

Principii didactice :

- principiul sistematizării și continuității cunoștințelor ;
- principiul accesibilității ;
- principiul individualizării și diferențierii învățării.

CD2	40'	<ul style="list-style-type: none"> • Cum calculăm punctajul unui candidat ? • Ce tip de date este potrivit pentru reprezentarea răspunsurilor candidaților și a răspunsurilor corecte, astfel încât structura de date aleasă să utilizeze eficient memoria ? Dar pentru reprezentarea punctajelor ? • Care sunt candidații cu punctaj maxim ? 	
CD3		<p>Răspunsurile așteptate ale elevilor :</p> <ul style="list-style-type: none"> • Pentru memorarea răspunsurilor a n candidați la un test cu m întrebări se poate folosi un tablou bidimensional, în care liniile sunt candidații, iar coloanele sunt întrebările. • Răspunsurile corecte pot fi memorate într-un vector cu m elemente sau pe prima/ultima linie a tabloului bidimensional. • Întrebările la care a răspuns corect un candidat se găsesc pe linia candidatului și au proprietatea că valoarea coincide cu valoarea răspunsului corect aflat pe aceeași coloană și pe linia 0 sau pe ultima linie (în funcție de locul în care au fost memorate răspunsurile corecte). • Candidații care au răspuns greșit la întrebarea k se determină prin parcurgerea coloanei k și numărarea elementelor care sunt diferite de cele de pe linia 0/linia $n + 1$ și coloana k (răspunsul corect). 	E
C2.1.			
C2.2.		<ul style="list-style-type: none"> • Actualizarea răspunsului candidatului x la întrebarea y presupune modificarea valorii din matrice aflate pe linia x și coloana y. • Punctajul unui candidat este numărul răspunsurilor corecte la cele m întrebări și se determină parcurgând pe linii candidații I, calculând pentru fiecare linie numărul de valori egale cu cele de pe linia răspunsurilor corecte ; punctajul fiecărui candidat se poate memora într-un vector cu n componente sau pe coloana 0 sau $M + 1$ în matrice ; 	A
C3.2.		<ul style="list-style-type: none"> • Fiind test grilă cu răspuns unic, răspunsurile corecte pot fi memorate într-o matrice cu componente de tip char, care ocupă 1 octet în loc de valori de tip int, care ocupă 4 octeți fiecare. • Punctajele candidaților nu mai pot fi memorate în matrice pentru că vor fi de tip int și se va opta pentru reprezentarea lor într-un tablou unidimensional, la fel și pentru răspunsurile corecte. • Pentru determinarea candidaților cu punctaj maxim este necesară determinarea punctajelor, a punctajului maxim, apoi, prin parcurgerea candidaților, se vor determina cei care au punctajul maxim. 	M
C1.3.		<p>Intensificarea atenției</p> <p>Pot veni elevi la tablă care să exemplifice funcționalitatea algoritmului pe date concrete, realizând eventual o reprezentare grafică a matricii numerice, încercuind pentru fiecare cerință linia/coloana/elementul, pentru o mai bună înțelegere.</p>	
C4.1.		Elevi diferiți vor scrie la tablă secvențe de cod C++ care implementează algoritmi discutați pentru rezolvarea cerințelor problemei propuse. Clasei i se cere să urmărească, să corecteze sau să îmbunătățească varianta scrisă.	
C4.2.		<p><i>Profesorul încurajează conversația, acordă feedback răspunsurilor elevilor, intervenind cu explicații suplimentare acolo unde observă nelămuriri sau erori de logică ori de exprimare.</i></p> <p><i>Profesorul încurajează elevii cu performanțe superioare să rezolve cerințele suplimentare.</i></p>	

CD4		Asigurarea reținerii și transferului de cunoștințe Profesorul pune câteva întrebări pentru a puncta competențele vizate : – de ce a fost aleasă reprezentarea datelor prin matrici sau/și vectori ? – când este necesară reprezentarea datelor prin tablouri uni- sau bidimensionale ? – cum se referă o valoare dintr-un astfel de tip structurat de date ? Evaluarea : – pe parcurs, prin observarea sistematică a reacțiilor elevilor și a muncii lor independente, prin aprobare sau dezaprobare verbală în urma răspunsurilor date de elevi la întrebări ; – la finalul orei, cu note argumentate de rezultatele prestațiilor lor (cei care au ieșit la tablă, cei care au rezolvat independent și corect sarcinile distribuite de profesor).	
	2'	Aprecierea activității Profesorul face aprecieri privind performanțele elevilor, recomandări de recuperare celor care nu au reușit să rezolve sarcinile obligatorii și îi notează pe cei care au fost activi. Profesorul poate cere elevilor să se autoevalueze.	
	3'	Tema pentru acasă Profesorul propune tema pentru acasă, elevii notează în caiete. Tema pentru acasă ar putea fi o aplicație asemănătoare cu cea rezolvată și pe care elevii o au pe fișa de probleme.	

Clasa : a IX-a

Tema lecției : Tablouri bidimensionale – parcurgerea pe linii/coloane a unei matrici

Fișă de probleme

Un test cu M întrebări de tip grilă cu răspuns unic este aplicat unui grup de N candidați ($1 \leq N \leq 100$, $3 \leq M \leq 300$). Răspunsurile corecte și răspunsurile candidaților sunt numere din mulțimea $\{1, 2, 3, 4\}$. Fiecare răspuns corect valorează un punct.

Scrieți un program C/C++ care citește din fișierul *test.in*, N M , apoi de pe următoarele N linii câte M numere reprezentând răspunsurile elevilor la cele M întrebări. De pe ultima linie se citesc cele M răspunsuri corecte.

Se cere :

- Numărul candidaților care au greșit la întrebarea 3.
- Întrebările la care nimeni nu a răspuns corect.
- Punctajul maxim obținut de un candidat.
- Pentru un candidat K citit de la tastatură, să se afișeze punctajul obținut la fiecare întrebare, în ordine.
- Să se modifice răspunsul candidatului x la întrebarea y (x , y și noul răspuns se citesc de la tastatură).

Cerințe suplimentare :

- Să se determine candidații care au obținut punctajul maxim.
- Să se afișeze lista candidaților în ordinea descrescătoare a punctajelor.

Temă : Adaptați algoritmul propus astfel încât să rezolve următoarele cerințe : considerăm că se dă punctajul asociat fiecărei întrebări, iar punctajul total al unui candidat se calculează adunând punctajele răspunsurilor corecte și scăzându-le pe cele ale răspunsurilor greșite. Se cere :

- Întrebările la care au greșit cei mai mulți candidați.
- Procentul de întrebări la care a răspuns corect candidatul k .
- Candidații cu punctajul maxim.
- Lista candidaților cu punctaj pozitiv, în ordine descrescătoare.

Capitolul 3

Metode, tehnici și procedee didactice

Sarcinile didactice se realizează cu ajutorul *metodelor, tehnicilor și procedeeelor didactice*. Folosirea judicioasă a acestora are o deosebită importanță pentru reușita activității la catedră. Pe de altă parte, conținuturile fiecărei discipline și obiectivele propuse impun metode specifice. *Adoptarea* – și nu *adaptarea* – metodelor de predare a unor discipline la alte discipline poate conduce la rezultate contradictorii. Aplicarea metodelor, tehnicilor și procedeeelor didactice generează activități de învățare specifice.

3.1. Metode generale de învățare

Trebuie să avem în vedere care dintre competențe de tip operațional rezultate în urma studierii competențelor-cheie și a celor specifice sunt urmărite prin studiul disciplinelor de informatică, ce cunoștințe noi vor asimila elevii și ce cunoștințe deja dobândite în cadrul altor discipline vor fi utilizate. Cert este că informatica poate *adopta* și *adapta* metode de predare de la alte discipline, dar acest lucru trebuie să se facă ținându-se cont de :

- Dinamica conținuturilor și particularitățile metodice ale predării disciplinei.
- Individualizarea învățării informaticii ca disciplină deschisă și dinamică.
- Constructivism, care pretinde o participare prioritară conștientă a elevului la procesul de autoinstruire.
- Studiul informaticii, atât ca disciplină autonomă, cât și ca instrument operațional al altor discipline.

Dintre metodele de predare specifice matematicii, de exemplu, amintim :

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

Aceste metode nu fac, în principiu, obiectul cărții de față. Cititorul interesat poate consulta însă [An], [CMS], [RV].

În cele ce urmează se vor analiza metodele generale, clasice, utilizate (și) în predarea informaticii :

- Expunerea sistematică a cunoștințelor.
- Conversația.
- Problematizarea.
- Modelarea.
- Demonstrarea folosind materialul intuitiv.
- Exercițiul.
- Învățarea pe grupe mici.
- Lucrul cu manualul.
- Jocurile didactice.
- Instruirea programată.

În tratarea acestor metode se vor urmări cu predilecție particularitățile specifice predării disciplinelor de informatică și, în special, aplicațiile practice de laborator, eventual, contribuția informaticii la însușirea competențelor didactice ale altor discipline din învățământul preuniversitar. În activitățile didactice specifice informaticii este necesară utilizarea combinată a metodelor de mai sus.

3.1.1. *Expunerea sistematică a cunoștințelor*

Dintre formele pe care le îmbracă expunerea sistematică a cunoștințelor (*povestirea, prelegerea, descrierea, explicația, conversația* etc.), opinăm că informatica utilizează cu precădere explicația. Elementele explicative domină procesul de instruire informatică, acestea fiind caracteristice atingerii unor competențe specifice care cuprind formarea de deprinderi și abilități practice de utilizare a unor produse soft uneori complexe, deși suficient de intuitive prin interfață. Ceea ce conferă o notă accentuată de adaptabilitate este operativitatea impusă de aplicarea acestei metode prin alternarea expunerii cu demonstrația practică, elevii fiind astfel scoși din pasivitatea posturii de simpli receptori. Analogiile cu situații cunoscute fac din receptorul pasiv un participant activ la expunere. Expunerea nu se desfășoară în condiții perfect univoce, adică fără alternative și reveniri, nici la disciplinele cărora metoda le este caracteristică. La informatică, aceasta se întâmplă cu atât mai puțin. Elevul primește în condiții univoce doar ceea ce i se comunică în funcție de nivelul de cunoștințe dobândit, de propriile presupuneri, de experiența sa practică, de nivelul său de gândire, de înțelegerea codului de comunicare, ca să nu mai vorbim de oscilațiile de atenție. Profesorul trebuie să reprojeteze lecția prin prisma posibilităților elevilor și cu mijloacele lor de gândire. Accentul trebuie pus pe raționament, prin argumentări temeinice, prin scoaterea în evidență a modului în care trebuie să gândească. Expunerea

trebuie să fie însoțită de un control permanent al gradului de receptivitate al clasei, urmărindu-se *mimica* elevilor (edificatoare în special la elevii mici), *satisfacția înțelegerii* lecției sau *îngrijorarea și neliniștea* în cazul în care elevul a pierdut firul explicației. Întrebările, repetiția, explicațiile suplimentare, analogiile cu alte noțiuni cunoscute permit realizarea unui control permanent al receptivității la expunere. În informatică recurgem neapărat la metoda expunerii (explicației) atunci când tema este complet nouă și printr-o *metodă activă* nu se poate descoperi noutatea, sau metoda activă este inefficientă din punctul de vedere al operativității. Astfel este necesară această metodă pentru a înțelege noțiunile de algoritm (inclusiv exemplificările clasice), de structură de date (inclusiv modalitățile de reprezentare), de comandă, funcție sau procedură standard (în legătură cu sistemul de operare sau mediul de programare ales), de raționament (într-un spațiu închis ales) și chiar modalitatea de prezentare și introducere a unor programe utilitare, softuri de aplicație etc. În acest context, pentru prezentarea comenzilor unui sistem de operare, a unui editor de texte sau editor grafic, a altor softuri mai complicate prevăzute de programa școlară, se poate recurge la următoarele metode :

- Expunerea la tablă în paralel cu secvențe de prezentări multimedia.
- Explicarea meniurilor diverselor aplicații simultan cu exersarea utilizării acestora în cadrul orelor de aplicații practice de laborator.

Fiecare dintre cele două variante de mai sus prezintă atât avantaje, cât și dezavantaje. Prima este folosită atunci când profesorul nu are la dispoziție un laborator special pentru predare, iar aceasta se face cu întreaga clasă. A doua variantă este, desigur, mai eficientă, deoarece elevul are posibilitatea de a testa practic funcționalitatea cunoștințelor proaspăt explicate. Totuși, unii elevi își formează mai repede deprinderea utilizării, iar alții mai greu, primii fiind tentați să încerce între timp alte opțiuni (chiar neprezentate încă de profesor), ceea ce creează disfuncționalități în desfășurarea lecției, aprecierea gradului de asimilare și chiar formarea unor idei greșite de utilizare (datorate încercărilor individuale, necoordonate). Pe lângă acestea, se pierde uneori din vedere realizarea unui rezumat sistematic al modului de utilizare, elevul fiind tentat să exerseze imediat și uită să-și noteze (în stil propriu) modul de utilizare a acesteia.

3.1.2. *Metoda conversației*

Metoda conversației se referă la dialogul dintre profesor și elev, în care profesorul nu trebuie să apară în rolul examinatorului permanent, ci al unui colaborator care nu numai întreabă, ci și răspunde la întrebările elevilor. Prin metoda conversației se stimulează gândirea elevilor în vederea însușirii, fixării și sistematizării cunoștințelor și deprinderilor, a dezvoltării spiritului de colaborare și 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 și profesor (și nu numai), respectându-se anumite reguli elementare de colaborare constructivă care să nu determine diminuarea demersului didactic, ci să-l amplifice și să-l consolideze. Conversația didactică poate îmbrăca forme diferite, în funcție de anumite criterii.

Raportată la numărul de persoane, conversația poate fi :

- *Individuală*. Se poartă între un elev și 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 :

- *Introdactivă*. Aceasta este folosită în momentul captării atenției și reactualizării cunoștințelor asimilate anterior, pentru a trezi interesul față de lecția care urmează.
- *Expozitivă*. În timpul prezentării unei noi lecții, ea poate trezi interesul pentru fixarea noilor cunoștințe.
- *Recapitulativă*. Este utilizată atunci când se urmăresc recapitularea și generalizarea unor rezultate prezentate anterior.
- *Evaluativă*. Este indicată, desigur, pe parcursul procesului de verificare și evaluare.
- *Dezvoltată*. Este destinată prezentării unui subiect nou, discutat anterior.

Caracteristicile principale ale întrebărilor, indiferent de forma de conversație, impun precizie și vizarea unui singur răspuns. De multe ori se adresează întrebări vagi care încep cu „Ce puteți spune despre...” sau „Ce știți despre...”, care plasează elevul într-un dubiu total în legătură cu conținutul răspunsului. Din aceeași gamă face parte și celebrul îndemn de evaluare „Prezintă subiectul pe care-l cunoști cel mai bine”. Nu este normal nici ca întrebarea să conțină răspunsul sau să ceară un răspuns prin „da” sau „nu”. Ea trebuie să 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 și rolul de a corecta greșelile din răspuns. Identificarea cauzei, eliminarea greșelii, precum și posibilitatea reapariției ei sunt foarte importante. Conversația are un rol primordial prin faptul că ajută la formarea limbajului informatic, la dezvoltarea raționamentului logic și a gândirii elevului.

Dificultățile pe care elevul le întâmpină în formarea limbajului de specialitate pot lăsa urme în plan afectiv, repercutându-se asupra dezvoltării lui intelectuale. De aceea se impune o analiză amănunțită a cauzelor acestor dificultăți, iar scoaterea lor în evidență trebuie relevată prin examinări scrise sau orale. A fi la curent cu dificultățile de limbaj pe care le au elevii la anumite vârste școlare și la un anumit stadiu de însușire a disciplinei înseamnă, în primul rând, să nu se abuzeze de termeni de specialitate. Se recomandă înlocuirea unor astfel de termeni cu sinonime din vocabularul curent sau explicarea sensului acestora în limbaj uzual, dacă un alt înțeles este accesibil. Dificultatea formării vocabularului de specialitate constă și în faptul că aceste

cuvinte noi sunt introduse în același timp cu noțiunile noi, ceea ce face ca îmbogățirea limbajului informatic să se realizeze simultan cu dezvoltarea și formarea gândirii informatice. Stăpânirea limbajului se reflectă în înțelegerea textelor și documentațiilor de specialitate și, în final, în rezolvarea problemelor. Nestăpânirea acestuia provoacă inhibiție, imposibilitatea comunicării sau chiar o comunicare și o înțelegere defectuoase, făcându-l pe elev timid, incoerent sau chiar ridicol în exprimare.

Această metodă mai are și următoarele subdirecții :

- *Euristică*. Nu există reguli precise, se bazează doar pe întrebare/răspuns, în funcție de evoluția concretă a dialogului.
- *Tip dezbateri*. Se realizează un schimb de păreri în care este implicat un anumit colectiv. Ar fi bine să fie trase și niște concluzii care să nu aibă doar un rol istoric.
- *Catehetică*. Impune efectuarea unor teste care implică memoria.

Este clar că o conversație se face prin întrebări care trebuie să satisfacă următoarele condiții (unele dintre ele rezultând din ceea ce am amintit mai înainte) :

- Să fie precise (vizând un singur răspuns).
- Să nu conțină răspunsul și să aibă un rol instructiv.
- Să stimuleze gândirea și capacitatea de creativitate a elevilor („De ce ?”, „Din ce cauză ?”, „În ce caz ?” etc.).
- Să fie formulate prin enunțuri variate și 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 și exprimate în termeni preciși și să oglindască un anumit nivel de înțelegere. Răspunsurile eronate se corectează imediat, prin discuții individuale. Cadrul didactic va dirija conversația astfel încât ideile să fie bine conturate înainte de a trece la altele, în timp ce lecția își menține caracterul unitar. În ceea ce privește informatica, recomandăm și utilizarea unor instrumente ajutătoare, cum ar fi introducerea/exprimarea noțiunilor printr-un *limbaj algoritmic* (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.

3.1.3. *Problematizarea și învățarea prin descoperire*

Predarea și învățarea prin problematizare și descoperire presupun utilizarea unor tehnici care să producă elevului conștientizarea conflictului dintre informația dobândită și o nouă informație, determinându-l să acționeze în direcția rezolvării acestuia prin descoperirea unor noi proprietăți ale fenomenului studiat.

Începem cu descrierea metodei problematizării. Pedagogic vorbind, conflictele se mai numesc și *situații-problemă*, putând fi de cel puțin două tipuri :

- *Contradicții* între posibilitățile existente ale elevului (nivelul intelectual și de pregătire) și cerințe sau situațiile în care este pus de noua problemă. Aceste conflicte se datorează imposibilității elevului de a le selecta dintre cunoștințele sale anterioare pe cele potrivite cu valoarea operațională de aplicabilitate a viitorului.
- *Incapacitatea elevului* de a integra noțiunile selectate într-un sistem, în același timp cu conștientizarea faptului că sistemul este pe moment inefficient 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 însuși 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 și să fie enunțate într-un moment optim al lecției. Ele trebuie să înglobeze cunoștințe deja însușite 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.
- Definirea problemei de către elev în sensul distingerei caracteristicilor esențiale ale situației, însușirii 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 și a altora noi, și găsirea efectivă a unei soluții (sau a tuturor soluțiilor).

Desigur că, în contextul de mai sus, sintagmele „situație conflictuală”, „problemă” și „rezolvare de problemă” se referă la probleme și soluții noi, necunoscute încă de elev, și nu la ceva de tipul substituirii de valori numerice în expresii date, execuția unui program dat pentru niște valori fixate de intrare etc. Utilizarea în predare a acestei metode este întotdeauna utilă în momentul în care se și găsește rezolvarea conflictului.

Învățarea prin descoperire apare ca o întregire a problematizării. După tipul de raționament folosit, se pot pune în evidență trei modalități principale de învățare prin problematizare și descoperire :

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

În primul caz este vorba de generalizări. Elevul trebuie încurajat să-și dezvolte propria cale de învățare, care să nu contrazică lucrurile în care crede, prin folosirea unor mijloace tehnice și resurse informaționale personale. În al doilea caz se folosește logica sau, mai exact, sistemele deductive ca metodă de raționament. Putem deriva (obține) cunoștințe noi din cunoștințe vechi cu ajutorul unor reguli de inferență specifice ([Mas3]). În ultimul caz, se încurajează folosirea unei experiențe anterioare nu numai dintr-un domeniu conex, ci chiar din domenii total diferite.

Problematizarea are astfel interferențe cu conversația, întrebările individuale sau frontale care se adresează gândirii, raționamentului născând situații conflictuale. Generarea situațiilor-problemă trebuie produsă astfel încât întrebările să apară în mintea elevului fără ca acestea să fie adresate de către profesor. După cum am mai precizat, ca disciplină cu caracter formativ, informatica își propune dezvoltarea unei gândiri algoritmice, sistematice și riguroase, care să promoveze creativitatea, să stimuleze imaginația și să combată rutina. Chiar dacă, aparent, travaliul informatic se sprijină pe anumite șabloane, ele reprezintă numai tendințe utile de standardizare. Procesele care izvorăsc din situații reale, care implică folosirea calculatorului în rezolvarea unor probleme aparținând diferitelor sfere ale vieții de zi cu zi, analiza acestor probleme, alegerea structurilor de date pe care se mulează informația oferită de mediul înconjurător, pașii algoritmilor și programarea în sine determină folosirea metodei problematizării, iar aplicarea ei necesită formarea unor deprinderi ce nu se obțin decât printr-un exercițiu îndelungat. Rezolvarea de probleme, ceva curent în învățarea informaticii, poate fi privită ca un proces prin care elevul descoperă că o altă combinație de reguli învățate anterior conduce la rezolvarea unei noi situații problematice. Formularea de probleme de către elevi constituie forme ale creativității și presupune că elevii și-au format deprinderi intelectuale eficiente din punctul de vedere al generalizării și aplicabilității (orice soluție generează o nouă problemă). Problemele propuse pot fi inspirate din viața cotidiană, din cunoștințele dobândite prin studiul altor discipline, din generalizarea unor probleme de informatică rezolvate anterior, probleme de perspicacitate, jocuri etc.

Problematizarea și descoperirea fac parte dintre metodele *formativ-participative*, care solicită gândirea creatoare a elevului, îi pun la încercare voința, îi dezvoltă imaginația, îi îmbogățesc experiența. În lecțiile la care se aplică aceste metode, profesorul alege problemele, le formulează, dirijează învățarea și controlează munca depusă de elev în toate etapele activității sale. Aceste metode sunt caracteristice, de exemplu, unor lecții de aplicații practice de laborator, metoda învățării prin descoperire fiind frecvent aplicată în momentul în care este necesară folosirea unor softuri de aplicație. Concentrarea atenției va fi dirijată spre rezolvarea problemei, și nu asupra analizei facilităților și lipsurilor produsului software. Cu siguranță, aici sunt deosebit de importante experiența dobândită, cunoștințele și deprinderile formate în alte situații similare de învățare. Cunoașterea facilităților produsului soft se face în momentul ivirii necesității exploatării acestuia, și nu printr-o prezentare a lui ca o înșiruire mai mult sau mai puțin sistematică și completă de funcții sau facilități. În contextul unor produse similare, trebuie concepută o viziune de ansamblu din care

să se desprindă caracteristicile dominante ale aplicațiilor din clasa respectivă și să se prezinte particularitățile specifice produsului, cu îmbunătățiri față de versiunile anterioare și perspective de dezvoltare pentru cele viitoare.

Ca informaticieni, în acest context ne interesează ceea ce numim rezolvarea problemelor (*problem solving*). Competențele dobândite în legătură cu acest subiect depind în primul rând de cunoștințele specifice acumulate, dar din punctul de vedere al psihologiei, ca știință, există consensul că se pot dobândi și competențe generale. Procesul cognitiv în ansamblu este foarte complicat și vom sublinia doar câteva elemente-cheie și direcții principale pentru abordarea rezolvării unor probleme. Astfel, când ni se solicită rezolvarea cu ajutorul calculatorului a unei probleme reale (complexe), presupunând că enunțul formal este „acceptat”, trebuie să răspundem la câteva întrebări, cum ar fi :

- Ce știm în legătură cu domeniul implicat ?
- Cum sunt apreciate rezultatele ?
- Ce strategii generale sunt aplicabile ?
- Care sunt motivațiile suplimentare ?

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

- Pot să rezolv problema (am cunoștințele necesare).
- Descriu algoritmul de rezolvare în mod (semi)formal.
- Caut informațiile suplimentare astfel încât să am o definiție formală concretă (eventual, într-un limbaj de programare cunoscut).
- Concep planul de implementare și îl execut : scriu *programele* și le *rulez pe date de test*.
- Verific dacă ceea ce am făcut este *corect*.
- Identific posibilități de generalizare la alte probleme (se poate urmări și subcapitolul 4.4).

3.1.4. Modelarea

Modelarea, ca metodă pedagogică, poate fi descrisă ca un mod de lucru prin care gândirea elevului este condusă spre descoperirea adevărului, folosind un așa-numit model și utilizând raționamentul prin analogie. Modelul și metoda în sine nu presupun o asemănare perfectă cu cazurile reale specificate inițial, ci numai o analogie rezonabilă. Ea constă în construirea unui sistem S_1 a cărui descriere coincide cu descrierea sistemului original S până la un anumit punct. S_1 poate avea o natură

diferită și este în general mai simplificat și formalizat. Ideea este că, investigând sistemul S_1 prin metode specifice legate de o anumită temă de lecție, se pot găsi noi soluții, care apoi pot fi translatate în concluzii asupra evoluției sistemului de bază S .

Modelarea are o mare valoare euristică colaterală, prin utilizarea ei putându-se dezvolta spiritul de observație, capacitatea de analiză și sinteză, creativitatea. Ideea ar fi să putem determina elevii să descopere singuri modelul. Astfel, elevul se obișnuiește să creeze noi probleme ce trebuie rezolvate, să adapteze algoritmi cunoscuți la situații noi etc. Realitatea înconjurătoare este percepută și înțeleasă pe baza unor modele deja cunoscute. Dezvoltarea deprinderilor de modelare, obișnuirea elevilor cu gândirea logică se realizează prin prezentarea exactă și clară a modelelor și prin transparența particularizărilor. Un exemplu edificator îl constituie învățarea metodelor de elaborare a algoritmilor. Necesitatea unor formalizări se impune prin rigoarea modului de abordare a problemei, prin sistematizarea organizării informației de intrare, a exactității proiectării prelucrării și prin standardizarea ieșirii. Formalizarea necesită cunoștințe dobândite în studiul altor discipline, fundamentate teoretic, iar accesibilitatea formalizării este condiționată de factori specifici nivelului de cunoștințe dobândit anterior, de categoria de vârstă, de capacitatea de asimilare (la nivelul clasei, de exemplu). Abordarea ponderată a acestor aspecte conduce la dezvoltarea deprinderilor de abstractizare, a gândirii algoritmice și sistemice. Utilizarea modelelor în realizarea algoritmilor presupune stabilirea unor analogii și în organizarea datelor de intrare.

„Învățarea” algoritmilor este legată de cunoașterea modului de organizare a datelor, de cunoașterea profundă a structurilor de date ce pot fi prelucrate ușor de către calculator. Etapa cea mai importantă este cea a descoperirii algoritmului, urmată de stabilirea modului de organizare a datelor, iar importanța acestui ultim aspect este esențială în determinarea performanțelor 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, grație raționamentului prin analogie :

- *Modelarea similară* constă în realizarea unui sistem de aceeași natură cu originalul și care să permită evidențierea trăsăturilor esențiale ale originalului. De exemplu, o gamă variată de probleme pot fi 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 regine*, și, prin anumite adaptări, el poate implementa algoritmi ce rezolvă alte probleme clasice, cum ar fi : generarea aranjamentelor/combinărilor, generarea funcțiilor injective/bijective, a partițiilor unei mulțimi, problema celor opt turnuri etc. Similar se procedează pentru rezolvarea problemelor care necesită utilizarea structurilor de tip stivă sau coadă, folosind operațiile specifice acestor structuri dinamice elementare. Pentru alte detalii, vezi anexe, capitolul 5 și capitolul 6.
- *Modelarea analogică* nu presupune o asemănare perfectă cu originalul, ci numai folosirea unor analogii.

Momentele cunoașterii în procesul modelării sunt :

- Trecerea de la original la model.
- Transformarea modelului sau experimentarea pe model.
- Transferul pe original al 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, și că noile informații obținute pe baza modelului vor fi transferate cu grijă asupra originalului, având în vedere diferența dintre model și 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 le împarte în :

- *Modele materiale*, care au suport concret : folosirea unui table de șah în rezolvarea problemei celor opt dame determină o rapidă înțelegere a mecanismului metodei *backtracking* ; utilizarea unei stive de monede de dimensiuni diferite este optimă pentru înțelegerea rezolvării problemei turnurilor din Hanoi ; calculatorul însuși este potrivit pentru studiul structurii și arhitecturii sistemelor de calcul.
- *Modele ideale (virtuale)*, care se exprimă prin imagini, sisteme de simboluri sau semne convenționale.

Învățarea informaticii prin modelare presupune două etape :

- Învățarea se va face pe baza modelelor construite de profesori. În această etapă se vor analiza trăsăturile modelului, apoi acesta va fi comparat cu originalul. Pentru a reliefa condițiile pe care trebuie să le îndeplinească modelul, se vor da și contraexemple.
- Elevii vor fi deprinși să construiască singuri modele. Importanța descoperirii modelului de către elev constă în faptul că el este obișnuit să reprezinte într-o formă standard condițiile impuse de problemă și-și adâncește convingerea 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 și aplicate orbește. Se va pune accentul pe înțelegerea pașilor unui algoritm și se va încuraja prezentarea oricăror metode care exclud modelul și care se impun prin eleganță și eficiență. Elevii vor fi încurajați să-și dezvolte și să-și prezinte ideile proprii, contribuind astfel la creșterea încrederii în posibilitățile lor, în valoarea ideilor lor. Ei nu trebuie să fie obligați să reproducă ideile altora, să aștepte ca totul să fie prezentat de profesor, să asimileze rețete, ci să descopere metode noi, să le prezinte, să le analizeze și să le perfecționeze printr-o comunicare continuă și constructivă.

Folosirea modelelor în învățare deschide pentru informatică o arie de aplicabilitate inter- și transdisciplinară impresionantă, de la artele plastice la cele mai diverse domenii ale tehnicii.

3.1.5. Exemplificarea sau demonstrarea materialului intuitiv

Prin exemplificare sau demonstrație, în acest caz, înțelegem prezentarea sistematizată și organizată a unor obiecte, procese, experimente, cu scopul de a ușura înțelegerea intuitivă și executarea corectă a unor activități programate. Cuvântul „intuitiv” din subtitlu se referă la utilizarea oricărui raționament inductiv, în contextul temei și bagajului de cunoștințe ale elevului. Utilizarea intuiției împreună cu exemplificarea necesară poate implica folosirea unor modalități și tehnici didactice diverse, datorită varietății materialului de studiu. Exemplificarea sau demonstrarea materialului intuitiv presupune utilizarea unei diversități de materiale multimedia, de softuri de învățare și de alte resurse online. În acest context putem spune că: „Prin demonstrarea materialului intuitiv se înțelege prezentarea sistematică și organizată a unor obiecte, procese etc. sau producerea unor experiențe, fenomene în fața elevilor, cu scopul de a ușura înțelegerea și executarea corectă a unor activități” ([RV]). Un rol deosebit îl joacă astfel intuiția (o experiență mentală; înseamnă o simplă observare și notare a unor fapte; poate fi asimilată cu un raționament de tip inductiv [RV]). Intuiția realizează corelația dintre imagine și cuvânt, fiind atât sursă de cunoștințe, cât și mijloc de verificare. Informatica nu poate fi desprinsă de bazele ei intuitive și de extinderea ei în realitatea cotidiană. Convertirea principiului intuiției în metoda demonstrației se realizează în funcție de materialul intuitiv, folosit în numeroase lecții, cum ar fi:

- Învățarea algoritmilor de sortare – prin diferite moduri de reprezentare sunt urmărite grafic valorile care se compară și se schimbă între ele, conducând la ordonarea șirului (sau... imagini cu o formație de dansuri populare!).
- Învățarea metodei *backtracking* – folosind materialul avut, într-un mod natural, se urmărește formarea soluției prin avansări și întoarceri repetate.
- Vizualizarea ocupării și eliberării zonelor de memorie prin alocarea dinamică a variabilelor.
- Ilustrarea modului de lucru cu elementele listelor simplu și dublu înlanțuite, a stivelor și a cozilor.
- Echilibrarea arborilor binari (arbori AVL).

Ținând cont de eficiența transmiterii informației prin mijloace vizuale (preluate inclusiv de pe Internet) și de orientarea cu predilecție spre mijloacele de informare rapidă care solicită atât memoria vizuală, cât și cea auditivă și formarea involuntară a unui public consumator de informație audio/video, o orientare a metodelor și procedeelor didactice în vederea exploatării acestei stări de lucruri creează un avantaj aparte procesului instructiv-educativ. Crearea unor secvențe video cu scop didactic

care să urmărească cu exactitate programa școlară generează facilități de predare pentru multe discipline și permite elevului să reia vizualizarea. Aceasta ar putea elimina ambiguitățile sau golurile create de momentele de neatenție din timpul predării și ar constitui un veritabil „profesor la purtător” al elevului. Este evident că un astfel de mijloc nu poate suplini exercițiul individual și nici prezența efectivă a cadrului didactic. Efortul profesorului este cu totul special. Elevul trebuie învățat să „vadă”, nefiind suficient ca el doar să urmărească materialul propus.

În acest moment trebuie să aducem în discuție euristicele și încurajarea creativității. Conform [WO], se pot pune în evidență euristici pentru dezvoltarea creativității :

- Încercați să aveți cât mai multe idei. Cu cât sunt mai multe, cu atât cresc șansele să puteți selecta câteva utile.
- Inversați problema (reformulați, reiteați, puneți într-un alt context etc.).
- Ghiciți o soluție la întâmplare (chiar urmărind un dicționar...).
- Gândiți-vă la ceva distractiv, apropo de utilizările posibile ale rezolvării.
- Gândiți-vă la probleme similare și la soluțiile acestora, chiar în contexte diferite.
- Concepeți o listă generală explicativă de cuvinte-cheie, proprietăți utile, stimulente ș.a.m.d. care au cât de cât legătură cu tema în cauză.

3.1.6. *Metoda exercițiului*

La modul cel mai general, exercițiile pot fi privite ca acțiuni concrete efectuate conștient și repetat în scopul dobândirii unor priceperi și deprinderi (mai rar cunoștințe) noi, pentru a ușura anumite activități și a contribui la dezvoltarea unor aptitudini. Avantajele metodei exercițiului sunt :

- Formarea unei gândiri productive, creatoare, cu posibile implicații financiare.
- Câștigarea unei anumite independențe individuale.
- Inițierea unui dialog cu obiective precise asupra unor metode și soluții.
- Activarea unei atitudini critice și creșterea discernământului elevilor în privința celor mai bune metode de lucru.
- Profesorul poate analiza și evalua mai direct activitatea sau performanțele generale ale unui elev.

Condiția primordială de reușită este dată în principal de selectarea corespunzătoare a problemelor sau exercițiilor, precum și de activitatea de îndrumare-proiectare. Prin urmare, exercițiile sunt acțiuni efectuate în mod conștient și repetat de către elev, cu scopul dobândirii unor priceperi și deprinderi și chiar a unor cunoștințe noi, pentru a ușura alte activități și a contribui la dezvoltarea altor aptitudini. Însușirea cunoștințelor de informatică este legată de exersarea utilizării unor softuri de aplicație, de rezolvarea unor probleme de programare etc. Nu există lecție în care să nu se aplice această metodă. Alte avantaje sunt concretizate în rezultatele aplicării ei : formează

o gândire productivă, oferă posibilitatea muncii independente, a analizei diverselor metode și soluții de rezolvare a problemelor, activează simțul critic și autocritic și îi învață pe elevi să-și aprecieze rezultatele și metodele de lucru, oferă posibilitatea depistării și eliminării erorilor.

Este clar că metoda nu contribuie numai la formarea priceperilor și deprinderilor de lucru cu calculatorul, ci contribuie substanțial și la dezvoltarea unui raționament flexibil și operant. Pentru profesor, alegerea, formularea și rezolvarea problemelor și apoi exploatarea rezultatelor obținute constituie o sarcină de o importanță deosebită. Alegerea problemelor este condiționată de programa analitică, de succesiunea prezentării noțiunilor în manuale, de metodele de rezolvare ce pot fi folosite și de nivelul elevilor cărora li se adresează. Formularea problemelor trebuie să țină cont de noțiunile cunoscute de elevi, să fie clară (neambiguă), concisă și să folosească limbajul de specialitate numai în măsura în care este cunoscut elevilor. Rezolvarea trebuie să aibă în vedere obținerea rezultatelor pe căi cât mai simple și ușor 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 cunoașterea noțiunilor învățate și adâncirea semnificației lor, asimilarea metodelor de rezolvare și aplicarea lor la rezolvarea altor probleme. Utilizarea pe scară largă a acestei metode a condus la o clasificare a exercițiilor și problemelor în funcție de aportul capacităților intelectuale necesare rezolvării lor.

În subsecțiunile care urmează ne axăm asupra anumitor particularizări, pe subiecte posibil abordate deja.

3.1.6.1. Exerciții și probleme de recunoaștere a unor noțiuni, formule, metode

De exemplu, utilizând metoda *backtracking*, se pot descrie algoritmi care generează permutările, aranjamentele, combinările, apoi li se poate cere elevilor să genereze toate funcțiile injective/surjective/bijective definite pe o mulțime cu m elemente, cu valori într-o mulțime cu n elemente.

3.1.6.2. Exerciții și probleme aplicative pentru formule sau algoritmi cunoscuți

Cunoscând modul de lucru cu elementele structurilor de date tip stivă sau coadă, elevilor li se poate propune să rezolve problema parcurgerii în lățime sau în adâncime a unui graf oarecare ([Cro], [CP], [LG]). Exercițiile aplicative trebuie utilizate atât timp cât trezesc interesul. Repetarea lor nejustificată poate conduce la efecte contrarii. Sunt astfel necesare contraexemple însoțite de o analiză amănunțită a erorilor.

3.1.6.3. Probleme reale care permit însușirea unor noțiuni

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

- *Exerciții de recunoaștere* a unor noțiuni : funcții recursive, grafuri, structuri de date (liste, arbori) etc.
- *Exerciții aplicative*.

Aceste două clase de exerciții sunt recomandate în special pentru fixarea unor cunoștințe 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 rezultate eronate.

- *Exerciții grafice*.
- *Exerciții complexe*, care presupun o analiză mult mai detaliată a problemei în ansamblu și implică descompunerea ei în subprobleme, succesiv, până în momentul în care rezolvarea subproblemelor elementare este cunoscută.

Pentru formarea unor priceperi sau abilități legate de munca independentă se poate utiliza și așa-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, metoda devine activ-participativă). Se pot evidenția permanent avantajele și dezavantajele rezolvărilor propuse, alte metode posibile de rezolvare, idei privind utilizarea acestor rezolvări sau particularizări ale lor din lecțiile anterioare. Specifice informaticii sunt problemele al căror grad de dificultate crește treptat, odată cu formarea și asimilarea noțiunii, fiecare nouă problemă aducând un plus de dificultate. În rezolvarea unei probleme este necesar să se țină seama de următoarele etape :

- *Analiza* inițială a problemei, prin care se stabilesc formatul, natura și intervalele de variație ale datelor de intrare/ieșire și ale variabilelor de lucru (date intermediare). Tot în această etapă se va stabili un algoritm de rezolvare, exprimat inițial în limbaj natural, ceea ce 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 (implementarea). 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, este indicată alegerea căilor optime în funcție de anumite criterii fixate anterior (paradigma de programare, complexitatea algoritmului, simplitatea structurilor de date etc.).

- *Verificarea* soluțiilor obținute va permite elevului să-și dea seama de corectitudinea lor. În această etapă, profesorul intervine cu seturi de date de test care să cuprindă majoritatea sau chiar toate cazurile posibile, în special cazurile critice, *la limită*, pentru datele de intrare. Se recomandă introducerea cât mai devreme posibil a noțiunilor privind verificarea formală apriorică a corectitudinii algoritmului (conform subcapitolului 4.4.2).

Etapele anterioare se pot modifica sau dezvolta după natura problemelor. Acolo unde problema permite mai multe căi de rezolvare, profesorul analizează cu ajutorul elevilor toate aceste căi, le selectează pe cele mai importante și le propune spre rezolvare pe grupe. Elevii vor compara apoi rezultatele, avantajele și dezavantajele fiecărei metode în parte și vor putea evidenția astfel cea mai bună soluție.

Exemplu (conținutul unei lecții în care se utilizează mai multe dintre metodele amintite).

Pentru înțelegerea completă a algoritmului de determinare a arborelui parțial de cost minim, profesorul poate propune spre rezolvare elevilor probleme din viața reală, care să le dezvolte simultan capacitatea de analiză, de recunoaștere a modelelor teoretice, de abstractizare a datelor și de optimizare a obținerii soluției/soluțiilor. Astfel, s-ar putea cere rezolvarea cu ajutorul calculatorului a problemei care are următorul enunț :

O localitate complet înzăpezită are n puncte vitale ($2 \leq n \leq 100$), legate prin străzi pe care se poate circula în ambele sensuri, străzi ale căror lungimi (în km) se cunosc. Deoarece nu dispune de suficiente rezerve de combustibil, Primăria este obligată să deszăpezească un număr de străzi, astfel încât toate punctele vitale ale localității să fie accesibile direct sau indirect din fiecare alt punct, iar operațiunea să se realizeze cu un consum minim de carburant. Să se determine străzile care vor fi deszăpezite și cantitatea minimă totală de carburant necesar știind că orice consum de carburant este direct proporțional cu lungimea drumului deszăpezit și costul deszăpezirii unui km de stradă este k .

Propunerea de rezolvare va avea cel puțin trei etape :

- a) *În prima etapă* se verifică dacă elevii cunosc noțiunile teoretice de arbore și de arbore parțial de cost minim. Prin conversație, se solicită elevilor să analizeze problema, să reprezinte graful pe baza exemplului numeric oferit de profesor și anticipează pe exemplu datele de ieșire. Profesorul conduce discuția prin întrebări, astfel încât elevii să tragă singuri concluzia că problema se reduce la determinarea arborelui parțial de cost minim într-un graf neorientat ponderat și că se va utiliza algoritmul lui Kruskal. Se stabilesc apoi datele de intrare, formatul, tipul și semnificația acestora (deja se gândește în direcția implementării algoritmului într-un limbaj de programare). În acest caz, datele de intrare se vor putea citi dintr-un fișier text cu structura, să zicem :

```

n //numărul de puncte vitale
m //numărul de străzi
k //costul de deszăpezire pentru 1 km de stradă
x1 y1 c1 // xi yi ci au semnificația: ci este lungimea străzii
care unește punctele vitale xi și yi
x2 y2 c2 //
.....
Xm ym cm //

```

Având contextul precedent, se poate fixa și formatul fișierului de ieșire :

x₁ y₁ // x_i y_i cu semnificația : se deszăpezește strada care unește punctele vitale x_i și y_i

```

x2 y2 //
.....
Xn-1 yn-1 //
cmin // consumul total minim de carburant necesar desză-
pezirii celor n - 1 străzi alese

```

- b) În a doua etapă se va stabili modalitatea de memorare a datelor de intrare. Dacă elevii vor fi tentați să reprezinte graful printr-o matrice de cost (simetrică), profesorul va sublinia risipa de memorie realizată prin acest tip de memorare și va propune sau va încerca să obțină de la elevi o memorare mai eficientă (printr-un vector, de exemplu) doar a muchiilor și costurilor acestora. Se va defini atunci un tip de dată numit muchie :

```

typedef struct{
    int punctX, punctY;
    double lungime;
}STRADA;

```

cu semnificația că *punctX*, *punctY* sunt nodurile adiacente, iar *lungime* reprezintă costul muchiei. În consecință, se va alocă un vector *STRADA*, a cărui dimensiune maximă se va stabili împreună cu elevii. Elevii vor explica aplicarea *Algoritmului lui Kruskal* pe enunțul dat al problemei. În continuare, se sugerează aplicarea unei strategii de tip *greedy* ; se vor sorta muchiile crescător după cost ; se va parcurge graful în inordine, plecând inițial de la un arbore parțial vid ; la fiecare pas se va selecta muchia de cost minim neselectată anterior și care nu formează un ciclu cu muchiile deja selectate ; procedeul se oprește după selectarea a $n-1$ muchii ([Cro], [CP], [MM]). Se insistă asupra importanței criteriului de oprire, profesorul având două posibilități : să prezinte el însuși criteriul și să verifice cu clasa de ce acesta este cel corect sau să încerce să obțină de la clasă un criteriu de oprire.

- c) *În a treia etapă* (posibil, ea nu va constitui finalul lecției) li se va propune elevilor implementarea algoritmului (fie memorând datele de intrare într-o matrice, fie într-un vector „de muchii”) pe grupe de lucru. Li se poate cere, de asemenea, folosirea de algoritmi de sortare diferiți, pentru a constata faptul că soluția nu este unică și, în plus, li se va cere să determine *cauza* obținerii de soluții diferite, dar tot optime. Profesorul va supraveghea implementarea solicitând elevilor verificarea etapă cu etapă a scrierii programului.
- d) *În (eventual) ultima etapă*, elevii vor proba corectitudinea programului prin folosirea de date de test construite de ei și prin noi teste propuse de profesor, aceste teste putând fi prezentate (de exemplu) chiar sub diverse forme grafice, pentru ca ei să fie în stare să-și construiască singuri fișierul de intrare.

La sfârșitul lecției ar fi indicat să se propună elevilor probleme care să utilizeze rezultatul obținut și să folosească tehnici asemănătoare pentru rezolvarea altora.

3.1.7. *Metoda învățării în grupe mici*

Activitatea de învățare pe grupe mici se definește ca o metodă în care sarcinile sunt executate de grupuri de elevi, grupuri care sunt câteodată autoconstituite și care se autodirijează. În informatică, activitatea se desfășoară în general în echipă, travaliul individual fiind o componentă a muncii corelate din cadrul unui grup de lucru. În aceste condiții este normal ca și activitatea didactică să recurgă la metode de învățare colectivă, fără a neglija însă munca individuală, ci doar privind-o pe aceasta ca pe o componentă a muncii în echipă ([Cri], [St]). Profesorii recunosc eficacitatea unei asemenea organizări a activității didactice și o integrează în arsenalul metodic al predării disciplinei. Alegerea criteriilor de formare a grupelor depinde de obiectivele urmărite, cum ar fi însușirea de noi cunoștințe, rezolvarea de probleme etc. Se pot forma astfel : grupuri *omogene* (elevi cu același nivel de cunoștințe) ; grupuri *eterogene* (elevi de toate categoriile – foarte buni, buni și 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 3-5 elevi. Grupurile alcătuite din mai mult de 2 elevi la un calculator se dovedesc a fi neproductive. Este bine ca la întocmirea grupurilor să se stabilească o serie de criterii clare de formare, iar elevii să fie lăsați să se grupeze singuri. Pentru grupurile omogene, sarcinile pot diferi în funcție de scopul propus. Pentru grupurile eterogene sau create pe criterii afective, sarcinile vor fi aceleași la fiecare grup, dar profesorul poate distribui sarcini suplimentare elevilor mai buni din fiecare grup.

Etapele identificate în metoda învățării în grupe mici 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 :

- Etapa *proiectivă*, în care se pregătește materialul de repartizat pe grupe și materialul în plus pentru elevii buni.
- Etapa de *îndrumare/supraveghere* și de animare a activității grupelor de lucru.

Ajutorul acordat grupelor de lucru trebuie să fie dat numai la cerere și în așa fel încât profesorul să se situeze pe poziția de colaborator, nu pe cea de autoritate care își impune părerile și soluțiile personale. Profesorul va interveni cu autoritate numai atunci când activitatea grupului se îndreaptă într-o direcție greșită. Când unul sau mai multe grupuri descoperă o soluție, propunerile lor vor fi discutate și analizate succesiv sau în paralel. Scopul acestei discuții este de a reliefa corectitudinea rezolvării, determinarea celei mai eficiente și mai elegante soluții și de a descoperi eventualele erori. Importanța dezbaterilor pentru dezvoltarea raționamentului este foarte mare, iar rolul profesorului este acela de a incita și coordona discuțiile în direcția obținerii concluziilor care se impun. Se impută, pe bună dreptate, acestei munci în grup o intensitate și o productivitate scăzute. Diversificarea sarcinilor grupurilor și împărțirea sarcinilor între membrii grupurilor atenuează această deficiență. Dacă prin activitatea în grup se intenționează dobândirea de noi cunoștințe prin lucrul cu manualul, documentația sau prin testarea unor produse soft, profesorul este obligat să organizeze dezbaterile finale care să stabilească dacă elevii și-au însușit corect noțiunile și și-au format deprinderi corecte. Este de asemenea greșit să se lucreze mereu cu grupuri constituite după aceleași criterii, pentru că în acest caz fie sunt suprasolicitați elevii buni din grupurile eterogene, iar elevii slabi se bazează exclusiv pe aportul liderilor de grup, fie, în grupurile omogene, elevii slabi se complac în postura în care se află și 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 ajutătoare* mult mai individualizate (ele nu țin neapărat de conținutul în sine al lecției). Ce *întrebări ajutătoare* se adresează și modul în care se adresează pot fi mai importante decât întrebarea în sine. Ulterior, atât în timpul lecției, cât și după aceea, comunicarea cu elevii este mai simplă. Ca prioritate și soluție la anumite probleme locale de învățământ, susținem aducerea unor specialiști din afara sistemului de învățământ pentru a preda lecții de sinteză, lecții speciale etc.

3.1.8. *Metoda lucrului cu manualul și documentația*

Manualele școlare, purtătoare ale valențelor formative prin deosebitul lor conținut metodic și didactic, reprezintă o limită impusă de programa școlară din punctul de vedere al conținutului informativ. În informatică, mai mult decât în alte domenii, manualul este supus perisabilității conținuturilor prin frecvența cu care disciplina este receptivă la noutățile care apar. Realitatea didactică reliefează faptul că elevul folosește pentru învățarea teoriei doar notițele luate în clasă la predare (sau resursele multimedia recomandate/accesibile online) și, din considerente de comoditate sau din

obișnuință, foarte puțin (sau deloc) manualele. Acestea sunt consultate în cel mai fericit caz doar pentru citirea enunțurilor problemelor. Atitudinea de reținere sau de respingere față de manual (fie el și în format electronic) are consecințe negative atât asupra caracterului formativ, cât și asupra celui informativ al învățării. Capacitatea de raționament a unui elev nu se formează numai după modele de raționament oferite de profesor, ci și prin eforturi proprii, prin activitatea individuală de căutare și comparare cu alte scheme de raționament. Valoarea acestei metode nu constă numai într-o însușire temeinică a cunoștințelor, ci și în formarea unor deprinderi de activitate intelectuală. Mulți elevi încheie ciclul liceal fără a avea formate deprinderi de lucru cu manualul și documentația, ceea ce le creează serioase probleme de adaptare și explică eșecurile din primul an de studenție și greutatea de adaptare la cerințele studiului universitar. Metoda muncii cu manualul este un aspect al studiului individual și se utilizează sub directă îndrumare și supraveghere a profesorului. Înainte de a aborda această metodă, profesorul trebuie să atragă atenția elevului asupra aspectelor importante ale lecției, care trebuie urmărite în mod special, cerându-i să realizeze un rezumat cu principalele idei de reținut. Rolul profesorului nu se limitează numai la a indica lecția din manual sau documentația care trebuie studiată. În timpul studierii de către elevi a noului material, profesorul are un rol activ. El urmărește cum își întocmește conspectul fiecare elev, dă îndrumări elevilor care-l solicită, verifică planurile întocmite de aceștia, corectând acolo unde este cazul. Profesorul poate să descopere în acest fel anumite lacune în cunoștințele dobândite anterior de elevi și să intervină ulterior pentru remedierea lor. El se ocupă deopotrivă de elevii slabi și de cei buni, cărora le dă sarcini suplimentare, reușind astfel să-și facă o imagine despre stilul de lucru și ritmul fiecărui elev. După studierea individuală din manual sau documentație, urmează discuții asupra celor însușite de elevi. Aceste discuții au scopul de a preciza problemele esențiale ale lecției, de a le sistematiza, de a înlătura posibilitatea unor omisiuni din partea elevilor sau chiar a însușirii eronate a unor noțiuni. Profesorul trebuie să selecteze și să pregătească minuțios materialul necesar, pentru a fi în măsură să răspundă prompt la orice întrebare pusă de elevi. Nu orice lecție poate fi însușită din manual. Metoda se aplică numai lecțiilor care au în manual o redactare sistematică și accesibilă nivelurilor de vârstă și de cunoștințe ale elevilor. Acestora li se recomandă studiul temei stabilite pentru acomodarea cu noțiunile, apoi profesorul reia prezentarea cu sublinierea aspectelor esențiale ([Max1]). Având o asemenea bază, profesorul se poate concentra asupra discursului său. Avantajele unui discurs bine organizat sunt ([WO]) :

- Urmărirea atentă a audienței : fiecărui „ascultător” (elev) îi poate fi sugerată ideea că este personajul principal, că el este cel vizat în primul rând.
- Noi porțiuni de text pot fi ușor introduse suplimentar, prin referirea la manual.
- Se prezintă lucruri deja verificate. Nimic nu poate merge rău.
- Stresul fiecărui elev în parte poate fi micșorat, el știind că nu este destinatarul special.
- Există posibilitatea unui feedback rapid și anumite principii de învățare pot fi folosite imediat.

- Există posibilitatea pregătirii prealabile a materialului, cu durată determinată, inclusiv cea a expunerii.
- Posibilitatea de a controla ceea ce s-a transmis/recepționat, cui, când, sub ce formă, precum și a modului de reacție este foarte mare.

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

3.1.9. *Metoda jocurilor didactice*

Jocurile didactice au valențele lor educative și în cazul informaticii. Ca metodă de învățare, acestea dau rezultate deosebite în special la clasele mici. Marele pericol în utilizarea acestei metode de instruire îl constituie folosirea abuzivă a unor aplicații soft care au o încărcătură educativă redusă. Datorită atractivității, ele captează și rețin atenția elevului, uneori chiar ore în șir, fără însă ca el să dobândească cunoștințe sau deprinderi pe măsura efortului depus. Un rol aparte se atribuie jocurilor manipulative, prin care elevul devine conștient de proprietățile obiectului studiat și își formează deprinderi și dexterități de utilizare a acestuia prin simularea pe calculator a utilajului sau dispozitivului respectiv. Aceste jocuri, numite uneori și simulatoare, necesită în cele mai frecvente cazuri echipamente periferice suplimentare, unele specializate pe lângă cele clasice. Amintim în acest sens utilizarea imprimantelor 3D, a unor căști speciale pentru obținerea efectului de realitate virtuală sau a echipamentelor care simulează condiții de zbor (pentru pilotaj) etc. Alte tipuri de jocuri (numite reprezentative), prin simbolizarea sau abstractizarea unor elemente reale, conduc la descoperirea unor reguli de lucru cu aceste elemente, dezvoltând astfel imaginația elevului. Ce altceva reprezintă un produs soft, atunci când înveți să-l utilizezi, decât un joc mult mai serios? Chiar dacă metoda nu este caracteristică studiului informaticii, la limita dintre jocul didactic și învățarea asistată de calculator se situează o bună parte dintre software-urile de învățare ([RV]).

3.1.10. *Instruirea programată și învățarea 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 și produsele demonstrative, pe care ar trebui să le descriem elevilor. Trebuie avut în vedere ca numărul de ore afectat acestei instruirii programate să nu fie foarte mare. Ea trebuie să beneficieze de un număr suficient de ore de verificare a cunoștințelor acumulate, evitându-se însă monotonia și instaurarea plictiselii (se recomandă utilizarea alternativă a altor metode). Trebuie evitată

și folosirea metodei un timp îndelungat, lucru care poate conduce în anumite situații la o izolare socială a elevului. O idee pentru contracararea acestor efecte ar fi creșterea numărului de ore sau organizarea activităților pe grupuri sau în echipă. *Instruirea asistată de calculator* este un concept diferit de instruirea programată doar prin modalitatea de utilizare. Există aceleași premise și moduri de utilizare, cu excepția faptului că un sistem de calcul devine principala interfață dintre un profesor și 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 îndeplinite, dirijări, verificări cu ajutorul calculatorului. În informatică (de fapt, în majoritatea disciplinelor și majoritatea școlilor) se utilizează intens platforma de învățare AeL, accesibilă online (<http://advancedelearning.com/>).

Procesele de predare-învățare și 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 și o informare operativă a rezultatelor învățării, necesară atât elevului, pentru autocorectare, cât și profesorului. În elaborarea programelor de învățare se au în vedere următoarele operații ([RV]):

- precizarea obiectivelor operaționale în funcție de conținut și posibilitățile elevilor ;
- structurarea logică a conținutului după principiul pașilor mici și al învățării gradate ;
- fracționarea conținutului în secvențe de învățare (unități didactice) inteligibile și înlănțuite logic ;
- fixarea după fiecare secvență a întrebărilor, exercițiilor sau problemelor ce pot fi rezolvate pe baza secvenței informaționale însușite ;
- stabilirea corectitudinii răspunsurilor sau soluțiilor elaborate.

Ca orice inovație, instruirea programată a trecut prin câteva faze contradictorii. La început s-a lovit de rezerva tenace a tradiției și de dificultățile materiale (tehnice), apoi, după ce a câștigat teren în conștiința teoreticienilor și practicienilor, s-au exagerat într-o oarecare măsură valențele ei aplicative, creându-se iluzia descoperirii pietrei filosofale în domeniul pedagogic. În final, după o analiză lucidă, s-a admis că există părți pozitive și părți negative. Criticile aduse instruirii programate sunt atât de ordin psihologic, cât și de ordin pedagogic și metodic. Psihologic, instruirii programate i se impută faptul că nu ține seama de principiile psihologice ale învățării, vizând învățarea ca o simplă succesiune și înmagazinare de fapte. De asemenea, se știe că motivația învățării nu poate fi analizată numai prin prisma reținerii și învățării imediate, făcând abstracție de interesul elevului față de conținut. În plus, lucrând singur sau cu calculatorul, elevul se simte izolat. Pedagogic vorbind, fărâmițarea conținuturilor este în detrimentul formării unei viziuni globale, iar valoarea cunoașterii imediate de către elev a rezultatului obținut are valențe contestabile. Metodic, decupajul analitico-sintetic al conținuturilor îi îngustează elevului posibilitatea formării aptitudinilor de analiză și sinteză. Aceste critici au determinat mutații serioase

în concepția de aplicare a metodei, dar practica didactică dovedește că, atunci când se cunosc și se evită cauzele care generează efecte negative, metoda produce rezultate bune. Tendințele de îmbunătățire a aplicării metodei se îndreaptă spre alternarea utilizării metodei cu celelalte metode clasice. Inserarea într-o lecție programată a unor metode clasice schimbă determinarea muncii școlare, repunându-l pe elev în directă dependență de activitatea profesorului și dându-i acestuia posibilitatea să verifice gradul de însușire a cunoștințelor conținute în program. O altă tendință este aceea de a modifica modul de redactare a programului, în special prin mărirea volumului de informație din unitățile logice și 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 un 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.

Perspectiva învățării asistate de calculator, inclusiv prin intermediul Internetului, este certă. Ea oferă posibilitatea prezentării programului, verificării rezultatelor și corectării erorilor, modificând programul după cunoștințele și conduita elevului. Calculatorul nu numai că transmite un mesaj informațional, dar el poate mijloci formarea și consolidarea unor metode de lucru, de învățare. Prin aplicarea acestei metode de învățare nu se întrevide diminuarea rolului profesorului. Dimpotrivă, sarcinile lui se amplifică prin faptul că va trebui să elaboreze programe și să le adapteze la cerințele procesului educativ. Oricât de complete ar fi programele de învățare asistată de calculator, profesorul rămâne cea mai perfecționată mașină de învățat. Pentru informații suplimentare, se poate consulta site-ul ministerului (www.edu.ro).

3.2. Metode specifice de învățare

Acestea se referă la ramuri/subramuri particulare ale informaticii (cum ar fi teoria algoritmilor, logica etc.). Fără a intra în prea multe detalii (fiecare subramură având metodele ei specifice), invităm cititorul să consulte atât bibliografia, cât și capitolul 4. Să mai menționăm faptul că, în realitate, metodele (generale sau specifice) nu sunt complet independente, precum obiectivele sau principiile didactice. Ele se combină în majoritatea cazurilor, iar dacă luăm în calcul și varietatea de obiective și/sau metode/metodologii specifice, ajungem la un număr impresionant de variante educaționale pe oră și temă. Credem că am prezentat în carte suficiente exemple și mai ales edificatoare (în principal în anexe și în capitolele 4-6). Alegerea problemelor reale poate fi condiționată de planul de învățământ, manualele alternative, contextul local, nivelul clasei, materialul didactic disponibil sau criteriile de valoare receptate. În plus, formularea problemelor trebuie să țină cont, la rândul ei, de conținutul manualelor, de noțiunile anterioare pe care le posedă elevii, poate și de caracterul fundamental sau legislativ al problemelor.

Să ne oprim totuși asupra câtorva metode, specifice poate nu atât unor subramuri, cât contextului actualei societăți informaționale (se poate consulta site-ul <http://www.cursuriazautorizate.eu>, de unde ne-am și inspirat uneori), care impune, printre altele, și educația adulților (ele ar putea fi numite și metode/tehnici de instruire):

- Metoda brainstorming, corelată cu discuțiile în cadrul unei mese rotunde.
- Maieutica.
- Asa-numita metodă *role play*.

Brainstormingul este un procedeu în care un grup de participanți (sigur că aceștia pot fi chiar elevi) se concentrează (prin discuții) asupra unei probleme specifice și caută soluții prin intermediul procesului colectiv de adunare de propuneri. Nu sunt admise formulări descurajatoare de tipul: „Am încercat acest lucru mai înainte”; „Nu va da rezultate niciodată”; „Cine are timp pentru așa ceva?”. După ce se epuizează lista sugestiilor, aceasta poate fi redusă la un număr de opțiuni aplicabile sau poate fi o listă a soluțiilor în ordinea priorităților. Unele sugestii pot fi eliminate dacă nu se întrunește consensul grupului. Profesorul trebuie să aibă grijă ca discuția să continue, prevenind gândirea negativă. Ca un avantaj de luat în seamă, această metodă permite enunțarea unor opinii și/sau aprecieri creative, fără restricții, referitoare la tema în discuție. Masa rotundă se referă la ideea de a desfășura discuțiile mai întâi în cadrul unor subgrupuri, pentru ca ulterior acestea să fie extinse între subgrupuri.

Maieutica este o metodă prin care se urmărește ajungerea la adevăr tot pe calea unor discuții/dialoguri, în care întrebările ipotetice sunt folosite ca tehnică de predare. Mai exact, predarea de tip „întrebare-răspuns” este câteodată mult mai interesantă și mai activă decât prelegerea uzuală. O discuție care începe cu o întrebare (de preferat, dificilă) necesită un angajament activ al participanților de a răspunde și de a găsi soluții, fiind mai productivă decât receptarea pasivă.

În sfârșit, *interpretarea de roluri* (autoexplicativă ca sens, credem) are ca scop primordial să contribuie la conștientizarea de către participanți a multiplelor perspective, valori, stiluri de comunicare și norme culturale care se pot manifesta în cursul unor discuții (de asemenea, dorința este și de a-i învăța pe aceștia cum să le facă față cu succes). În acest mod, s-ar asigura aplicarea în practică a unei game variate de aptitudini evident necesare lucrului în echipă, cum ar fi soluționarea conflictelor, luarea unor decizii echilibrate, rezolvarea situațiilor neprevăzute, evidențierea diverselor opțiuni pentru soluționarea unor situații dificile.

Ca o concluzie (parțială și nicidecum exhaustivă) a acestui capitol, precizăm că tratarea rezolvărilor trebuie să aibă în vedere obținerea rezultatelor pe căi clare (și, pe cât posibil, verificabile printr-o altă metodă), analiza metodelor utilizate, reținerea tipurilor de raționamente folosite, deschiderea unor noi perspective pentru probleme similare sau mai complexe. Se urmăresc cunoașterea activă a noțiunilor învățate, adâncirea semnificațiilor, asimilarea metodelor de rezolvare, aplicarea lor în soluționarea altor tipuri de probleme.

Capitolul 4

Noțiuni de bază

Acest capitol este destinat în principal prezentării unor elemente introductive, absolut necesare pentru păstrarea caracterului de sine stătător al lucrării. În învățământul preuniversitar, anumite noțiuni deosebit de importante sunt predate și tratate diferit.

4.1. Noțiuni de bază în informatică

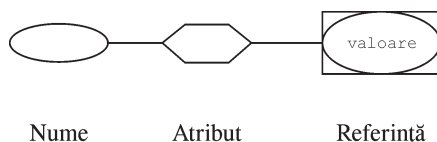
Începem cu o scurtă trecere în revistă a câtorva concepte care vor fi utilizate intensiv în carte, în special ca suport teoretic pentru exemplele alese. Pentru detalii, se mai pot consulta [An], [At], [CMS], [Hor], [HS], [Max1], [Po], [RV].

Temele și domeniile abordate în tratarea disciplinelor de informatică sunt stabilite prin competențele-cheie și competențele specifice. În concepția noastră, *algoritmul* reprezintă o noțiune fundamentală, un concept de bază pentru informatică. Prin *algoritm* (imperativ) se înțelege ansamblul de transformări (metode) ce se aplică asupra unui set de date de intrare și care determină obținerea, într-un timp finit și după o succesiune precisă de pași, a unui set de date de ieșire ([CLR], [K1], [K2]). Aceasta nu este o definiție, ci o descriere a unui concept. 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* și *finitudinea*, *eficacitatea* ([K1], [Max2]). Introducerea oricărei noțiuni (chiar nefundamentală) ar trebui să parcurgă următoarele etape:

- *Etapa de elaborare și motivație* (inițială). Fundamentată și 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 și, de dorit, demonstrată matematic, o noțiune se constituie ca un util și puternic mijloc de producție în domeniul pentru care a fost elaborată. Didactic, acest aspect cuprinde argumentarea științifică a noțiunii introduse și reliefaarea unor noi posibile domenii de aplicabilitate.
- *Etapa de consolidare* se realizează prin operare consecventă cu noțiunea. O noțiune poate fi considerată asimilată dacă devine și instrument de dobândire a altor cunoștințe și dacă elevii pot opera cu ea în situații noi.

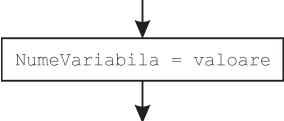
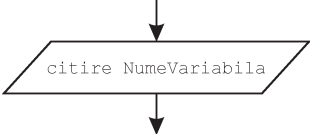
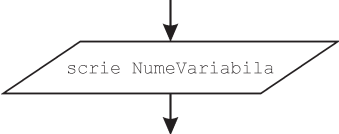
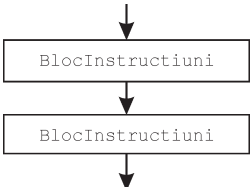
De exemplu, în privința reprezentării algoritmilor, optăm pentru folosirea pseudocodului sau a altor tipuri de „diagrame”. Noțiunile introduse ulterior vor apărea în mod firesc, căpătând caracteristicile unor înălțuiri cauzale. Unele teme de predare pot fi organizate în *spirală* (ceea ce presupune o reîntoarcere la același conținut, dar pe un nivel superior). Acest mod de planificare corespunde sistemului *concentric propriu-zis* (*concentric calitativ*) și sistemului *concentric cantitativ* (*concentric liniar*). *Sistemul concentric calitativ* desemnează modul de organizare a cunoștințelor în programele de învățământ, manuale și lecții, în așa fel încât noțiunile (cunoștințele) se însușesc prin reluări, restructurări, reinterpreări, până la formarea lor completă. *Sistemul concentric cantitativ* este modul de organizare a cunoștințelor în programele școlare, manuale și lecții (inclusiv pe Internet), constând în reluarea adăugită și detaliată a materiei parcurse anterior, reluare reclamată nu atât de dificultatea înțelegerii noțiunilor, cât mai ales de nevoia lărgirii cunoștințelor în succesiunea claselor și treptelor școlare. Trebuie astfel făcută diferența între noțiunea de *variabilă*, așa cum este ea cunoscută din matematica clasică, și cea de *variabilă* în sensul limbajelor de programare imperative ([Barr]), noțiune care poate fi reprezentată astfel (de unde poate rezulta și interpretarea corectă a atribuirii) :

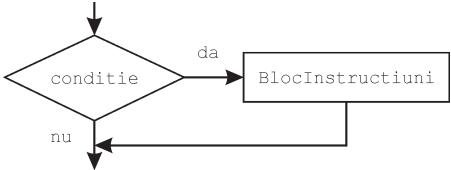
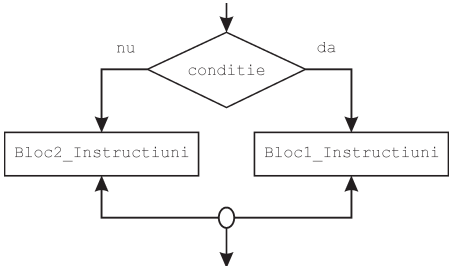
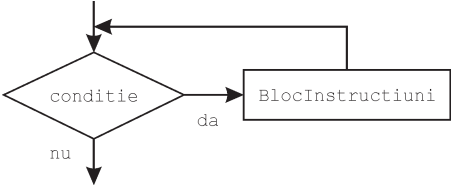
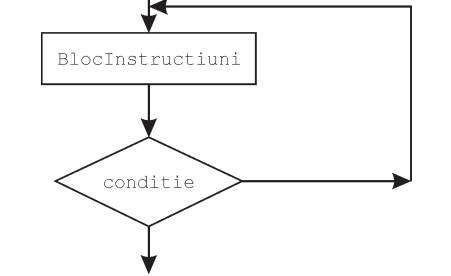
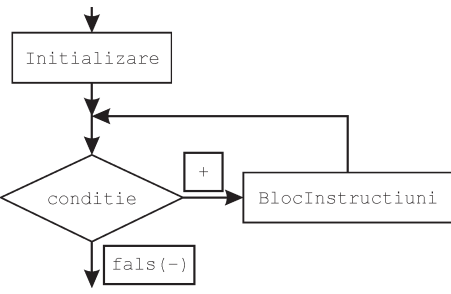


Intuitiv vorbind, pentru a parcurge drumul de la realitatea de modelat la implementarea pe calculator, trebuie înțelese, cel puțin la nivel descriptiv, și alte noțiuni, cum ar fi cele de *problemă*, *complexitate*, *corectitudine/verificare* etc. După cum am mai precizat, pentru că noțiunea de algoritm este dată printr-o *descriere*, și nu prin utilizarea *genului proxim* și a *diferenței specifice* (în sensul logicii aristotelice clasice, ca subdisciplină a *filosofiei* [Mas3]), 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, spre deosebire de limbajul natural, este o formă de reprezentare mai exactă, permițându-ne în plus orice nivel de detaliere. Anumite operații/instrucțiuni și structuri de informație ([Bac1], [Barr]) nu lipsesc dintr-un astfel de 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 ;
 - structura de tip repetitiv.
- Clase de structuri de date :
 - numere ;
 - șiruri de caractere ;
 - tablouri ;
 - arbori ;
 - liste etc.
- Blocul de instrucțiuni poate conține una sau mai multe instrucțiuni. Ordinea de execuție a instrucțiunilor este secvențială, dacă nu se specifică altfel (vezi instrucțiuni de control, instrucțiuni de salt etc.).

Tabelul 4.1. Forma pseudocodului și reprezentarea grafică

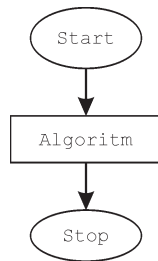
Operație	Pseudocod ; C/C++/C#	Reprezentare grafică
Atribuire	NumeVariabila = valoare; In C/C# semnul pentru atribuire este =	
Citare	citire NumeVariabila; <i>Observație</i> Variabila <i>NumeVariabila</i> va avea atribuită valoarea furnizată în cadrul acestei operații (de exemplu, ceea ce tastăm sau citim într-un fișier).	
Scriere	scrie NumeVariabila; <i>Observație</i> Valoarea variabilei va fi afișată pe ecran sau scrisă într-un fișier. Ecranul este văzut tot ca un fișier.	
Structura secvențială (bloc de instrucțiuni)	BlocInstrucțiuni; <i>Observație</i> În C/C++/C# BlocInstrucțiuni este delimitat de acolade	

<p>Structura de control alternativă – incompletă</p>	<p>Daca (conditie) atunci BlocInstructiuni SfDaca</p> <p>// C/C++/C# if (conditie) BlocInstructiuni</p>	 <pre> graph TD Start(()) --> Cond{conditie} Cond -- da --> Bloc[BlocInstructiuni] Bloc --> Join(()) Cond -- nu --> Join Join --> Exit(()) </pre>
<p>Structura de control alternativă – completă</p>	<p>Daca (conditie) atunci Bloc1_Instructiuni altfel Bloc2_Instructiuni Sfda</p> <p>// C/C++/C# if (conditie) Bloc1_Instructiuni else Bloc2_Instructiuni</p>	 <pre> graph TD Start(()) --> Cond{conditie} Cond -- da --> Bloc1[Bloc1_Instructiuni] Cond -- nu --> Bloc2[Bloc2_Instructiuni] Bloc1 --> Join(()) Bloc2 --> Join Join --> Exit(()) </pre>
<p>Structura de control repetitivă cu test la intrarea în ciclu</p>	<p>Cattimp (conditie) executa BlocInstructiuni Sfcattimp</p> <p>// C/C++/C# while (conditie) BlocInstructiuni</p>	 <pre> graph TD Start(()) --> Cond{conditie} Cond -- da --> Bloc[BlocInstructiuni] Bloc --> Cond Cond -- nu --> Exit(()) </pre>
<p>Structura de control repetitivă cu test la ieșirea din ciclu</p>	<p>Repeta BlocInstructiuni Panacand (conditie)</p> <p>// C/C++/C# Do BlocInstructiuni while(conditie);</p>	 <pre> graph TD Start(()) --> Bloc[BlocInstructiuni] Bloc --> Cond{conditie} Cond -- da --> Bloc Cond -- nu --> Exit(()) </pre>
<p>Structura de control repetitivă cu număr finit de pași</p>	<p>for(initializare; conditie; actualizare) { BlocInstructiuni; Actualizare; } // C/C++/C# for(initializare; conditie; actualizare) { BlocInstructiuni; Actualizare; }</p>	 <pre> graph TD Init[Initializare] --> Cond{conditie} Cond -- "+" --> Bloc[BlocInstructiuni] Bloc --> Cond Cond -- "fals(-)" --> Exit(()) </pre>

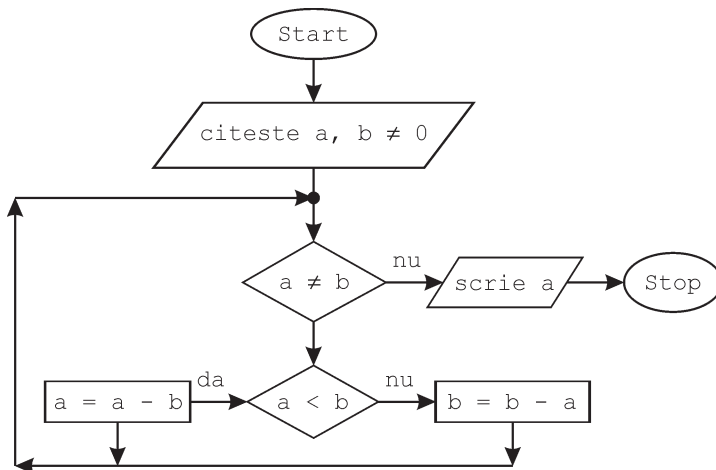
Observație

Pseudocodul, schema logică etc., folosite în mod real, pot diferi de la profesor la profesor, de la școală la școală. Sperăm ca abordarea sugerată de noi să nu conțină diferențe majore, care să genereze neînțelegeri insurmontabile. Nu am indicat și o formă de pseudocod similară cu un cod Pascal deoarece acest limbaj este din ce în ce mai puțin folosit.

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



În 1966 ([BJ]) s-a demonstrat că *orice algoritm (imperativ) poate fi reprezentat folosind numai structurile de control: secvențială, alternativă și repetitivă*. Rezultatul obținut a condus în acel moment la apariția unor noi viziuni de proiectare a algoritmilor, cum ar fi cele modulară și structurată. Din același motiv vom folosi pe parcursul lucrării, în caz că anumite confuzii pot fi evitate, și 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) :

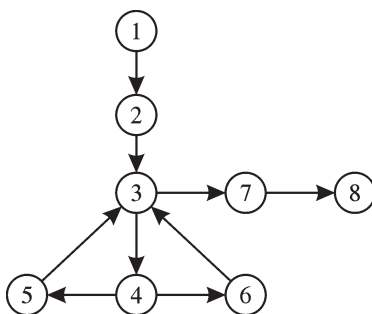


Observație

Înainte de a prezenta schema logică sau codul pentru un algoritm este necesar să realizăm o descriere a acestuia. O abordare directă, similară cu cea oferită mai sus, nu este productivă și va forța elevul să memoreze ceva ce nu a înțeles.

Se observă că în orice algoritm rezultatul final este condiționat de datele inițiale și, mai mult, că succesiunea în care se execută operațiile elementare depinde de datele de intrare și de rezultatele intermediare obținute în urma execuțiilor anterioare. Datele inițiale, rezultatele intermediare și deciziile luate în structurile de control alternative și repetitive determină astfel o *traiectorie* ([Max1], [Max2]) a execuției operațiilor (prelucrărilor), aceasta putând fi reprezentată printr-un graf orientat (*digraf*).

Algoritmului anterior i se poate asocia digraful :



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-Sfârșit).

4.2. Paradigme de programare

Începem această secțiune prin a puncta că, din punctul nostru de vedere, orice algoritm, indiferent de forma sa de prezentare sau de descriere (mai mult sau mai puțin apropiată de un limbaj natural, mai mult sau mai puțin formalizată), trebuie, în cele din urmă, să fie implementat într-un limbaj de programare și executat pe un calculator real. Totul depinde de modalitatea de a defini operațional un algoritm, adică fără a folosi direct (de exemplu, mașina Turing) sau indirect (prin genul proximal și diferența specifică) un concept formal. Sunt astfel necesare câteva precizări legate de *paradigmele* (sau *stilurile fundamentale*) de programare. Suportul bibliografic principal este dat de linkurile din Anexa 3.

Odată cu intrarea noțiunilor de algoritm și calculator în limbajul curent, au apărut diverse idei privind modalitatea în care gândim rezolvarea unei probleme. Din punct de vedere general filosofic, paradigmele formează totalitatea practicilor care definesc o disciplină științifică la un moment dat. Ele dictează „ce se studiază”, „ce fel de probleme importante se pun”, „cum sunt interpretate rezultatele”. Putem astfel accepta ceea ce afirmă Aristotel (știința „normală” se bazează pe acumularea de noi informații peste ceea ce se cunoaște deja, totul alcătuind și îmbogățind paradigma curentă) și Platon (știința „revoluționară” pune la îndoială însăși paradigma curentă, punând-se problema cum s-ar putea cunoaște ceva nou „altfel”). Atunci vom accepta și necesitatea trecerii, câteodată, de la o paradigmă la alta (*schimbarea de paradigmă*).

Paradigma de programare ar reprezenta astfel un set de reguli prin care se precizează modul în care se construiește un program într-un limbaj de programare și chiar modul în care se proiectează limbajul însuși. O paradigmă de programare dictează astfel : cum se reprezintă informația (variabile, funcții, obiecte, fapte, constrângeri etc.) și cum se prelucrează aceste reprezentări (prin atribuirii, evaluări, fire de execuție, fluxuri de lucru etc.).

Menționăm doar câteva tipuri de (paradigme de) programare :

- Programarea imperativă/procedurală/von Neumann (concretizată de limbaje procedurale ca : ALGOL, FORTRAN, COBOL, BASIC, Pascal, C etc. sau de limbaje de asamblare și codurile mașină, specifice fiecărei generații de calculatoare, începând cu ENIAC și terminând cu PC-urile, tabletele și telefoanele mobile de astăzi).
- Programarea declarativă/nonprocedurală, subdivizată de obicei în :
 - programarea funcțională (câteva limbaje cunoscute : Lisp, APL, Erlang, Haskell, ML, OCaml, Scheme etc.) ;
 - programarea logică (limbaje de tip Prolog) ;
 - programarea asociativă (CLIPS).
- Programarea orientată pe obiecte (Smalltalk, C++ , Java, C#, PHP etc.).

Mai putem adăuga ca fiind reprezentative : programarea paralelă/concurentă, programarea bazată pe reguli sau constrângeri, programarea textuală (de tip .html). O paradigmă complet nouă, care încă nu s-a structurat și individualizat complet, ar fi programarea probabilistă. Să punctăm de pe acum că nu putem clasifica un limbaj real, implementabil, ca aparținând unei paradigme doar după sintaxă, că există limbaje multiparadigmă sau integrate (precum Python) și că exemplele de limbaje date anterior sunt de nivel înalt și comerciale. Clasificarea menționată nu este nici pe departe exhaustivă.

Să detaliem, începând cu paradigma despre care s-a mai discutat, adică cu programarea imperativă (= date + operații). Pe scurt, aceasta desemnează faptul că un cod (algoritm) implică efectuarea unor operații asupra datelor. Starea programului asociat variază în timp, execuția operațiilor/instrucțiunilor este secvențială (conform ordinii de scriere textuală a lor) și variabilele denumesc/reprezintă zone/locatii de

memorie ale calculatorului. Conținutul zonelor se modifică în timp prin atribuire. Abstracțiunea specifică programării imperative ar fi conceptul de procedură. Modelul (formal) de calcul/ execuție este mașina Turing/*Turing Machine*/TM (un model similar ca putere de calcul este mașina cu memorie cu acces aleator/*Random-Access Memory machine*/RAM). Mai sunt și alte aspecte care ar putea fi caracteristice pentru încadrarea unui limbaj într-o anumită paradigmă: de exemplu, în limbajele imperative se permite existența efectelor secundare, a tipurilor pentru valori și variabile etc. În cadrul acestei paradigme nu este important *ce* este o soluție (obținută în urma execuției codului), ci *cum* se obține aceasta.

Programarea funcțională (= funcții + control) este bazată pe matematică, mai exact pe teoria generală a funcțiilor (λ -calculul). Este, practic, atemporală, valorile fiind imuabile: nu există atribuire, nu există efecte secundare propriu-zise. Toate prelucrările se bazează pe aplicarea unei funcții unui argument și pe compunerea de funcții (de un argument; funcțiile cu mai multe argumente sau valori se pot simula simplu cu funcții de un argument și având o singură valoare). Abstracțiunea specifică este conceptul matematic de funcție (reprezentată în notația lambda). Se permite tipizarea (aceasta fiind însă greu de manipulat computațional).

Programarea logică (= relații/logică + control) este bazată tot pe matematică, dar nu pe ideea de funcție, ci pe cea de relație. În loc de λ -calcul, se folosește ca suport teoretic calculul cu predicate de ordinul I/*first order predicate logic*/(FOL). Deși orice relație poate fi introdusă ca fiind o funcție particulară și reciproc (orice funcție poate fi prezentată ca o relație), conceptele generale (la nivelul fundamentelor axiomatice ale matematicii) sunt diferite (de aceea și limbajele „pilon” sunt diferite). În programarea logică se permite tipizarea, cu aceeași observație de la programarea funcțională. Abstracțiunea specifică este dată de conceptul de predicat (așa cum este el folosit în logica matematică binară, clasică). Pornind de la informații (fapte/axiome și reguli de inferență/relații între fapte), soluția (teoremă, ca răspuns la o întrebare) se construiește prin raționamente/demonstrații/deducții. Se descrie practic soluția (se spune ce/care este aceasta) și nu cum se ajunge la ea (mai exact, se ajunge la aceasta printr-o execuție standard, soluția în sine fiind practic doar un efect secundar al acestei execuții).

Programarea asociativă este de multe ori asimilată cu programarea logică, deși există câteva deosebiri importante (atât în sintaxă, cât și în semantică/ execuție). De exemplu, faptele sunt doar de tipul „Dacă... atunci...”, și toată „execuția” este bazată pe substituții textuale și tehnici de *pattern matching*, abstracțiunea specifică reprezentând-o conceptul de mașină Markov/*Markov Machine*/MM.

În sfârșit, programarea orientată pe obiecte (= clase + metode) este bazată pe teoria conceptelor, legată de permanenta interacțiune umană cu mediul. Conceptele reale se reprezintă prin clase, iar instanțele/întrepunerea fizică a conceptelor sunt obiectele. Obiectul este, de fapt, și abstracțiunea specifică acestui tip de programare. Există o ierarhie generală între clase/obiecte, bazată pe noțiunea de moștenire. Execuția implică folosirea metodelor și comunicarea între clase/obiecte (prin transmiterea de mesaje/*message passing*).

Nu vom detalia mai mult. În momentul necesității învățării sau chiar al proiectării unui anumit limbaj concret (de nivel înalt sau nu) pentru a fi folosit practic, într-un mod constant și regulat, sperăm să fie suficiente considerațiile de mai sus.

4.3. Tehnici de programare. Proiectarea algoritmilor – sortare și căutare

Pentru o introducere în problematica vastă a acestui domeniu (inclusiv justificarea studiului său intensiv), recomandăm volumele [CLR], [K1], [K2], [Me]. Deși din punct de vedere practic *sortarea externă* este mult mai importantă, din considerente didactice vom insista asupra *sortării interne*. Aceasta înseamnă că algoritmi în sine, ideile importante de rezolvare a problemelor reale, complexitatea teoretică primează asupra considerentelor legate de spațiul (resurse hard) și timpul efectiv (măsurat în secunde) de rezolvare.

Enunțul problemei de sortare

Fie „la intrare” o colecție de obiecte nu neapărat distincte, peste care este definită o relație de ordine. Să se furnizeze „la ieșire” aceeași colecție, eventual sub o altă formă, care să satisfacă anumite criterii, precizate anterior și legate de ordine.

Exemplu

Considerăm colecția :

```
{ (Popescu, 9, 1.78),  
  (Ionescu, 10, 1.76),  
  (Eva, 10, 1.65),  
  (Adam, 8, 1.79),  
  (Ionescu, 9, 1.72) }.
```

Această colecție o putem ordona crescător/descrescător după primul element al perechii, *nume*, sau după al doilea element al perechii, *nota*, sau după al treilea element al perechii, ce ar putea reprezenta *înălțimea*.

Enunțul problemei de căutare

Se consideră dată o mulțime A , finită, peste care este definită o relație binară. Să se determine dacă un element x , de tipul elementelor mulțimii A , se găsește sau nu în mulțimea A .

Observații

1. Nu ne interesează dacă elementul x se află de mai multe ori în mulțimea A .
 2. Unele probleme de căutare definesc criterii complexe pentru elementul x ce se caută, nu se rezumă numai la existența/inexistența acestuia în mulțimea A .
 3. Deoarece mulțimea A este finită, rezultatul căutării poate fi adevărat sau fals, adică elementul există sau nu în mulțimea A și întotdeauna va exista un rezultat.
 4. Dacă elementele mulțimii A sunt aranjate într-o anumită ordine, este posibil ca algoritmul de căutare a elementului x să aibă o formă particulară.
 5. Algoritmul normal de căutare începe prin a compara rând pe rând elementele mulțimii A cu elementul x . La prima apariție a elementului x , algoritmul va semnaliza acest lucru și se poate opri. Dacă s-a parcurs toată mulțimea A și elementul x nu a fost găsit, atunci se va termina algoritmul prin afișarea unui mesaj corespunzător.
-

Referitor la problema căutării considerăm un exemplu clasic, dar care nu se încadrează perfect în enunțul de mai sus.

Exemplu

Un vector notat cu A , de dimensiune n , conține toate numerele naturale și distincte de la 1 la $n + 1$ (cu excepția unuia, bineînțeles). Să se determine numărul care lipsește.

O idee pentru rezolvare poate fi exprimată astfel :

- a) Calculăm suma primelor $n + 1$ numere naturale cu formula $(n + 1)(n + 2)/2$.
- b) Printr-o parcurgere secvențială, calculăm suma elementelor vectorului A .
- c) Diferența celor două sume este numărul căutat.

Complexitatea este evident liniară pentru o implementare corectă.

O altă idee de rezolvare este dată în următoarea secvență de cod :

Code1:

```
int m = 0;
for (int i = 1; i < n + 1; i++)
    m = m + i - A[i];
m = n + 1 + m;
```

Observații

1. Repetăm, autorii nu recomandă descrierea unui algoritm prin furnizarea unui cod și apoi discutarea lui. Am putea descrie metoda de mai sus, folosindu-ne de prima idee, în felul următor: odată cu calculul sumei elementelor vectorului vom calcula și suma primelor n numere naturale și vom face direct diferența dintre acestea (vezi *c*). Avem nevoie de suma primelor $(n + 1)$ numere naturale (vezi *a*) și, în concluzie, trebuie să adunăm $(n + 1)$ la rezultatul obținut.
 2. Codul de mai sus poate fi scris detaliat după cum urmează. Acum se observă clar că este în fapt prima idee, dar exprimată altfel.
-

Code2:

```
int m = 0;
    int suma = 0;
    for (int i = 1; i < n + 1; i++)
    {
        m = m + A[i];
        suma = suma + i; // suma primelor n numere naturale
    }
    // suma primelor (n + 1) numere naturale
    suma = suma + n + 1;
    m = suma - m; // Acest număr lipsește
```

Numărul căutat va fi m . Vă invităm să găsiți (implementați) și alți algoritmi, rezonabili ca ordin de complexitate.

Dacă vectorul ar fi sortat crescător, atunci putem formula un algoritm astfel: numărul lipsă din vector coincide cu primul indice k pentru care $A[k] \neq k$, dacă există, altfel este $(n + 1)$, iar k ia valori de la 1 la n . Ce concluzie am putea extrage din enunțul de mai sus? Modul de prezentare a informației contribuie la elaborarea algoritmilor.

Revenim la problema ordonării elementelor unui vector, notat A . În majoritatea problemelor analizate, elementele vectorului A vor fi numere naturale, întregi sau reale. Să presupunem însă, pentru început, că dorim să rezolvăm problema *ordonării (crescătoare)* a unui vector A . Peste elementele acestui vector există definită o relație de ordine totală. 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:

- a) *Sortarea internă*: elementele lui A sunt păstrate în memoria internă a calculatorului.
- b) *Sortarea externă*: elementele lui A sunt păstrate pe un suport extern; se utilizează neapărat în cazul unui volum foarte mare de date.

Metodele de sortare vor fi diferite, 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 a informației. În cazul sortării interne, există o multitudine de strategii de sortare, fiecare cu avantajele și dezavantajele sale, strategii care sunt analizate în funcție de diverse criterii :

- Memoria ocupată.
- Numărul de comparații.
- Numărul de deplasări ale elementelor.
- Timpul de execuție.

În cazul informațiilor aflate în memoria internă se pune problema dacă acestea sunt memorate în *structuri statice* sau în *structuri dinamice*. Algoritmii de sortare internă pot fi împărțiți în : algoritmi *banali* – timp de lucru $O(n^2)$ sau chiar mai mare –, care au însă marea calitate că sunt ușor de înțeles, și algoritmi *performanți* – $O(n \log n)$ sau mai mic –, care au însă dezavantajul că fac apel la cunoștințe matematice de specialitate care nu se predau în liceu. În ceea ce privește căutarea, am adoptat aceeași tactică de natură didactică, fără a avea pretenția de a epuiza subiectul în sine (o tratare exhaustivă presupune, de exemplu, familiarizarea cititorului, la nivel matematic și informatic, cu domeniul *recunoașterii formelor*).

4.4. Algoritmi clasici de sortare, de complexitate timp $O(n^2)$ și mai mare

Indiferent dacă implementarea structurii de date alese pentru memorarea mulțimii A (ceea ce mai sus s-a 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 concretizare (a metodei), și nu pe detaliile de implementare.

4.4.1. Sortarea prin interschimbarea elementelor vecine

Ideea algoritmului este următoarea :

- Se parcurge vectorul de sortat și se compară elementele vecine. Dacă acestea nu sunt în ordinea corectă, se schimbă elementele între ele. Procesul se reia pentru întregul vector până când nu se mai face nici o interschimbare.
- Notăm cu x vectorul ce conține n numere reale și pe care dorim să-l sortăm.
- Algoritmul poate fi ușor implementat și verificat. Întrebările utile în acest caz ar trebui să se refere la cazul cel mai favorabil și la cel mai defavorabil. Algoritmul nu necesită alocarea unui nou tablou.

Codul în C/C# poate fi :

```
do
{ // vectorul ce se sortează este notat cu x
  bool ok = true;
  for(i=1; i<n-1; i++)
  {
    if (x[i] > x[i+1]) // crescător/descrescător [*]
    {
      ok = false;
      temp = x[i]; // temp are același tip ca x[i] și se
presupune că a fost declarat
      x[i] = x[i+1];
      x[i+1] = temp;
    }
  }
} while (ok);
```

Observații

1. Condiția $[*]$ din *if* determină modul de sortare, crescător sau descrescător.
 2. Un șir sortat crescător parcurs de la sfârșit spre început este descrescător.
-

4.4.2. Sortarea prin selecție

Notăm cu x vectorul ce conține n numere reale și pe care dorim să-l sortăm.

Sortarea prin selecție poate fi descrisă astfel :

Se determină elementul minim din șirul dat și se trece pe poziția 1. În continuare determinăm elementul minim din subșirul $\{x[2], x[3], \dots, x[n]\}$ și-l trecem pe poziția primului element din subșir etc.

Observație

Am considerat indexul primului element din vector ca având valoarea 1. În acest caz, vectorul în C/C# trebuie declarat ca având dimensiunea $n+1$. Reamintim că în C/C# indexul primului element al unui *array* are valoarea 0.

Presupunând că am *citit* elementele vectorului, numărul de elemente, am făcut toate declarările de variabile și inițializările necesare, codul în C/C# este următorul :

```

for(i = 1; i < n; i++)
{
    minim = x[i];
    indexMinimInitial = i;
    for (int j = i + 1; j<=n; j++)
    {
        if (minim > x[j])
        {
            minim = x[j];
            indexMinimActual = j;
        }
    }
    if (indexMinimInitial != indexMinimActual)
    {
        // Interschimbăm elementele x[indexMinimInitial] și
        x[indexMinimActual]
        int temp = x[indexMinimInitial];
        // se presupune că elementele lui x
        // sunt de tip int
        x[indexMinimInitial] = minim; // adică x[indexMinimActual]
        x[indexMinimActual] = temp;
    }
}

```

Observații

1. Am considerat indexul primului element ca fiind 1. Dimensiunea declarată a tabloului trebuie să fie în acest caz $n+1$.
 2. Secvența de cod arată și modalitatea de a obține elementul cu valoarea minimă sau maximă dintr-un șir.
-

4.4.3. Sortarea prin inserție directă

Notăm cu x vectorul ce conține n numere reale și pe care dorim să-l sortăm. Primul element are indexul egal cu 1.

Descrierea algoritmului

Considerăm că primele $j-1$ elemente ale vectorului sunt sortate crescător, adică : $[1] < x[2] < x[3] < \dots < x[j-1]$. Următorul element, $x[j]$, îl vom compara cu $x[j-1]$, $x[j-2]$ etc., până când este îndeplinită condiția $x[i] < x[j]$ și apoi îl vom insera pe poziția $i+1$. Inserarea pe poziția $i+1$ presupune salvarea lui $x[j]$ într-o variabilă locală și apoi deplasarea, în cadrul vectorului x , a elementelor

$x[j-1], x[j-2], \dots, x[i+1]$ cu o poziție spre dreapta. După deplasare, memorăm valoarea salvată a lui $x[j]$ pe poziția $i+1$ din vectorul x . Codul în C/C# poate fi scris ca mai jos. Se presupune că vectorul este notat cu x , iar în n sunt precizate numărul de elemente din vector. Tipul elementelor din vector este întreg. Din nou, primul element din vector are indexul egal cu 1.

```
int temp = 0;
int i = 0;
for (int j = 2; j<=n; j++)
{
    temp = x[j];
    i = j - 1;
    while ((i>0) && (temp < x[i]))
    {
        x[i + 1] = x[i];
        i = i - 1;
    }
    x[i + 1] = temp;
}
```

Observații

1. Determinarea locului unde se va insera elementul $x[j]$ se face aici prin parcurgerea secvențială a subșirului deja ordonat crescător. Acest algoritm de căutare poate fi înlocuit cu cel de căutare binară. Subșirul ordonat este împărțit în două subșiruri. Se examinează relația de ordine dintre elementul de la mijlocul subșirului și elementul $x[j]$ și se stabilește în care subșir se va face căutarea. Operația de divizare continuă până se găsește poziția în care urmează să fie inserat elementul $x[j]$.
 2. Deoarece se ia în considerare următorul element al șirului față de subșirul deja ordonat, această metodă poate fi numită „sortare prin inserție directă cu pas unu”.
-

4.4.4. Sortarea Shell

Vom începe prezentarea cu o observație pe care o considerăm utilă.

Metoda de sortare Shell face parte din categoria metodelor de sortare cu inserție directă în care pasul de inserare este mai mare ca 1. Pentru a mai alunga monotonia prezentărilor anterioare, metoda în cauză va fi descrisă (incomplet, fără cod sau discuții precise legate de corectitudine, complexitate etc.) sub forma unui proiect didactic. Pentru că pe parcursul subcapitolului privind sortarea vom mai introduce câteva proiecte de tehnologie didactică, rugăm cititorul să consulte Anexa 1, pentru a identifica forma standard (recomandată astăzi) a acestora. În acest loc, forma unui proiect este doar aproximativă.

Proiect de tehnologie didactică

Data :

Clasa : a X-a

Profesor :

Disciplina : Algoritmică și programare

Subiectul lecției : Sortarea tablourilor prin *metoda Shell*

Scopul lecției : Introducerea unei noi metode de sortare a tablourilor cu ajutorul arborilor de sortare și căutare

Tipul de lecție : Mixtă

Competențe de natură operațională

Elevii trebuie să fie capabili :

- Să deosebească această metodă de cele prezentate anterior.
- Să implementeze corect metoda Shell.
- Să observe avantajele și dezavantajele față de celelalte metode.

Metode folosite : expunerea, conversația, exercițiul.

Mijloace de realizare : convenționale.

Desfășurarea lecției :

Punctul 1

Etapă : Moment organizatoric.

Timp : 2 min.

Activitatea desfășurată de profesor :

- Se face prezența și se verifică existența celor necesare începerii orei.

Punctul 2

Etapă : Verificarea cunoștințelor.

Timp : 12 min.

Activitatea desfășurată de profesor :

- Verificarea temelor date elevilor pentru acasă.
- Verificarea cunoștințelor din lecția precedentă, cu tema *Sortarea tablourilor prin metoda inserției directe*, prin întrebări :
- În ce constă această metodă ?
- Avantajele și dezavantajele față de alte metode implementate anterior.
- Reluarea metodei cu ajutorul unui alt exemplu.

Metoda : Verificare orală.

Punctul 3

Etapă : Trecerea la lecția nouă.

Timp : 2 min.

Activitatea desfășurată de profesor : Anunțarea și scrierea pe tablă a titlului lecției : *Sortarea tablourilor prin metoda Shell*.

Punctul 4

Etapa : Predarea noilor cunoștințe.

Timp : 5 min.

Activitatea desfășurată de profesor :

Pornind de la metoda inserției directe, observăm că această metodă se poate îmbunătăți, ajungându-se la sortarea Shell. Ideea este următoarea :

- Se împarte la început tabloul în grupe de câte două elemente care se raportează separat : de exemplu, dacă dimensiunea este 16, putem forma grupele $(1, 9)$, $(2, 10) \dots$; $(1, 9)$ înseamnă subșirul $\{x[1], x[9]\}$.
- Se formează grupe de câte patru elemente din cele sortate anterior.
- Procedul continuă până se ajunge la tabloul în cele din urmă ordonat crescător.

Metoda : Expunerea.

Punctul 5

Etapa : Fixarea noilor cunoștințe.

Timp : 3 min.

Activitatea desfășurată de profesor :

Fiecare grupă fiind sortată separat, se observă că 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 și exerciții în legătură cu :

- Metoda de sortare Shell.
- Compararea cu celelalte metode.
- Exemplifică la tablă noțiunile introduse.

Metoda : Conversația.

Punctul 6

Etapa : Precizarea temei pentru acasă.

Timp : 3 min.

Activitatea desfășurată de profesor :

- Exemplifică pas cu pas sortarea prin noua metodă pe un exemplu concret.
- Solicită implementarea algoritmului în limbajul de programare studiat.

4.4.5. Sortarea rapidă

În această secțiune ne vom axa pe prezentarea succintă a doar două metode de complexitate $O(n \log n)$. Enunțul problemei este același cu cel din secțiunea precedentă.

Ideea celor doi algoritmi ce vor fi prezentați în continuare se bazează pe două acțiuni importante :

- Divizarea șirului în subșiruri care vor fi supuse procesului de sortare.
- Combinarea subșirurilor pentru a obține rezultatul final.

Tabelul 4.2. Realizarea celor două etape prezentate mai sus, pentru algoritmi de sortare prin interclasare și Quicksort

Algoritm	Divizare	Combinare
Sortare prin interclasare	În funcție de poziție	Interclasare
Quicksort	În funcție de valoare	Concatenare

4.4.5.1. Sortarea prin interclasare

Fie secvența a_1, a_2, \dots, a_n ($n \geq 2$). Aplicând metoda *divide et impera*, se împarte șirul în două subșiruri a_1, a_2, \dots, a_m , respectiv $a_{m+1}, a_{m+2}, \dots, a_n$, cu $m = (n+1) \div 2$. Procedul se repetă pentru subșirurile a_p, a_{p+1}, \dots, a_q ($m = (p, q) \div 2$) până când se obțin subșiruri de lungime 1 (șir deja sortat). Dintre subșirurile sortate, se obțin prin *interclasare* alte subșiruri formate din elementele a două câte două subșiruri, până la obținerea șirului sortat, de lungime n .

Observație

Acest algoritm presupune crearea unui spațiu temporar de memorie care în final va conține șirul sortat. Spațiul suplimentar de memorie are aceeași dimensiune cu cel necesar șirului ce urmează a fi sortat.

Vom începe cu un exemplu. Fie șirul (secvența, vectorul, lista etc.) $A = \{7, 1, 9, 4, 3, 1, 5, 2, 6\}$ de lungime 9. El va fi împărțit în două subșiruri:

- $\{7, 1, 9, 4, 3\}$ primul subșir, identificat prin capetele (index) ale sale $[1, 5]$; Deci $A[1] = 7$, iar $A[5] = 3$.
- $\{1, 5, 2, 6\}$ al doilea subșir, identificat prin $[6, 9]$.

Subșirul $[1, 5]$ se împarte în subșirurile $[1, 3]$ și $[4, 5]$, dintre care subșirul $[1, 3]$ se mai divide în $[1, 2]$ și $[3, 3]$. Subșirurile $[1, 2]$ și $[4, 5]$ se împart în subșiruri de lungime 1. Subșirul $[6, 9]$ se împarte în subșirurile $[6, 7]$ și $[8, 9]$, care la rândul lor se împart în subșiruri de lungime 1. Acestei secvențe de divizări i se poate asocia un arbore binar care are rădăcina *marcată* cu $[1, 9]$ (capetele șirului inițial) și pentru fiecare nod marcat cu $[s, d]$ marcăm succesorii săi stâng și drept cu $[s, m]$, respectiv $[m+1, d]$, până când $s-d = 0$ (s = stânga, d = dreapta). Mărcile nodurilor reprezintă capetele subșirurilor obținute în etape succesive de divizare. Nodurile terminale sunt marcate cu capetele subșirurilor de lungime 1. Reprezentarea grafică obținută se numește *arbore de căutare*. Nu dăm această reprezentare aici.

Subșirurile terminale sunt sortate. Se interclasează apoi șirurile terminale, obținându-se noi șiruri (în locul șirurilor *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.

Codul pentru sortare poate fi următorul :

```
public void SortarePrinInterclasare(ArrayList<int> x, int
left, int right)
{
    ArrayList<int> leftArray = new ArrayList<int>();
    ArrayList<int> rightArray = new ArrayList<int>();
    if (left<right)
    {
        // Divizare
        int middle = (left + right) / 2;
        // Rezolvare
        SortarePrinInterclasare(x, left, middle);
        // Rezolvare
        SortarePrinInterclasare(x, middle + 1, right);
        // Combinare
        Interclasare(x, left, middle, middle + 1, right);
        // (left, middle): index inițial și final pentru primul
        subtablou
        // (middle + 1, right): index inițial și final pentru al
        doilea subtablou.    }
    }
}
```

Codul pentru metoda *Interclasare* va executa următoarele acțiuni :

1. Se parcurg subtablourile în paralel și la fiecare pas se transferă cel mai mic element.
2. Se transferă eventualele elemente rămase în primul subtablou.
3. Se transferă eventualele elemente rămase în al doilea subtablou.

Observație

Există variante ale acestui algoritm, dar ideea de bază rămâne aceeași.

4.4.5.2. Sortarea *quicksort*

Ceea ce urmează este doar o variantă (datorată autorilor) a metodei lui E.W. Dijkstra.

Observație

Reamintim că operația de divizare cerută de acest algoritm se face în funcție de valoarea elementelor.

Proiect de tehnologie didactică

Disciplina : Informatică

Clasa : a IX-a

Profesor :

Data :

Tema : Căutare și sortare. Metoda *quicksort*

Tipul de lecție : Predare

Metode didactice : Expunere, exerciții

Mijloace de învățare : Manuale, culegeri

Material bibliografic : ([Me])

Desfășurarea lecției este asemănătoare cu cea de la proiectul anterior, de aceea am renunțat să o prezentăm și aici.

Competențe de natură operațională :

Elevii trebuie să fie capabili :

- 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.

Pentru a simplifica expunerea, să presupunem că avem un singur criteriu de căutare. Considerăm că datele sunt memorate în vectori de înregistrări (*array*) și căutarea datelor se face pe baza valorii unui câmp. În mod tradițional, acest câmp se numește *cheia înregistrării*. 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) elementele vectorului, pe baza valorilor cheilor. Reformulând problema de sortare, aceasta înseamnă să transformăm (*pe loc*) un vector *A* într-un vector pentru care :

$$A[i].cheie \leq A[i+1].cheie, 1 \leq i < n-1, \quad (1)$$

utilizând o cantitate minimă de memorie suplimentară. După cum am mai precizat, *nu există un cel mai bun algoritm universal de sortare*. Alegerea algoritmului potri-vit 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 înre-gistrări și de cât de *neordonate* sunt elementele vectorului. În cazul *quicksort*, folosim vectorul *A* cu elemente numere întregi. Prin parcurgerea vectorului pornind de la ambele capete (pe rând) și interschimbarea elementelor care nu sunt în relația cerută, se împarte vectorul în două părți, nu neapărat de lungimi egale, cu proprietatea că

toate elementele din prima parte sunt mai mici (sau mai mari, în cazul sortării descrescătoare) decât toate elementele din a doua parte. Unul dintre subvectori este memorat (prin indicii de început și sfârșit), iar cu cel rămas se procedează analog. Subsirurile memorate sunt prelucrate apoi pe rând în același mod (recursiv), în ordinea inversă a memorării lor.

Aplicând algoritmul, condiția de ordonare (1) poate fi rescrisă sub forma :

Pentru orice element $A[k]$ din vector (denumit *pivot*) este îndeplinită condiția:

$i < k \Rightarrow A[i] \leq A[k]$ **si**

$k < j \Rightarrow A[k] \leq A[j]$

Sfpentru

La fel ca la *sortarea cu bule*, se verifică dacă la un moment dat este îndeplinită condiția corespunzătoare, în caz contrar efectuându-se corecția necesară, printr-o inversare :

Daca (gasim o pereche de valori $(i, i + 1)$ pentru care)
 $A[i].cheie > A[i+1].cheie$ (2)

atunci

vom schimba între ele cele doua elemente.

Sfdaca

Presupunem că s-a ales ca *poziție pivot* (k), poziția din *mijlocul* vectorului. Algoritmul următor asigură îndeplinirea condiției anterioare pentru această poziție. Limitele între care variază indicii elementelor din vector sunt *prim* și *ultim*.

QuickSort (A , *primul*, *ultimul*)

$i \leq \text{primul}$

$\leq \text{ultimul}$

$\text{pivot} = A[(\text{prim} + \text{ultim})/2]$

repetă

- poziționează i după elementele cu $\text{chei} < \text{pivot.cheie}$
- poziționează j înaintea elementelor cu $\text{chei} > \text{pivot.cheie}$

Daca $(i < j)$ **atunci**

schimba $(A[i], A[j])$

Dacă $(i \leq j)$ **atunci**

{

$i := i + 1$

$j := j + 1$

}

Panacand $(i \geq j)$

Exemplu.

Fie șirul următor $\{9, 2, 4, 10, 3\}$. Considerăm poziția inițială a pivotului ca fiind $k = 3$. Evoluția algoritmului produce următoarele transformări :

prim = 1 \Rightarrow i = 1

\Rightarrow k = 3

ultim = 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, deși pentru $k = 3$ condiția este îndeplinită, șirul nu este încă ordonat. Pentru $k = 2$ și $k = 4$, condiția nu mai este îndeplinită. Pentru a corecta aceste „greșeli”, algoritmul trebuie aplicat din nou, atât la stânga, cât și 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 .

prim = 1, ultim = 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 *planul de lecție* al proiectului. Evaluarea procesului de învățare va rezulta din supravegherea activității depuse, constatarea dificultăților în asimilarea cunoștințelor și rezolvarea acestor dificultăți. Cu cât se insistă mai mult pe punctele *problematic*, cu atât rezultatul evaluării va fi mai bun.

Se poate arăta că pentru acest algoritm complexitatea medie este $O(n \log n)$. În cazul cel mai defavorabil, și acest algoritm este totuși de complexitate $O(n^2)$.

Un alt mod de implementare este cel bazat pe metoda *divide et impera*. Utilizând metoda *divide et impera*, vom împărți șirul în două subșiruri cărora le vom aplica același algoritm de divizare până când subșirurile obținute vor avea lungimea 1 (și vor fi ordonate). Soluțiile parțiale fiind memorate în același șir, operația de combinare a soluțiilor parțiale este deja efectuată.

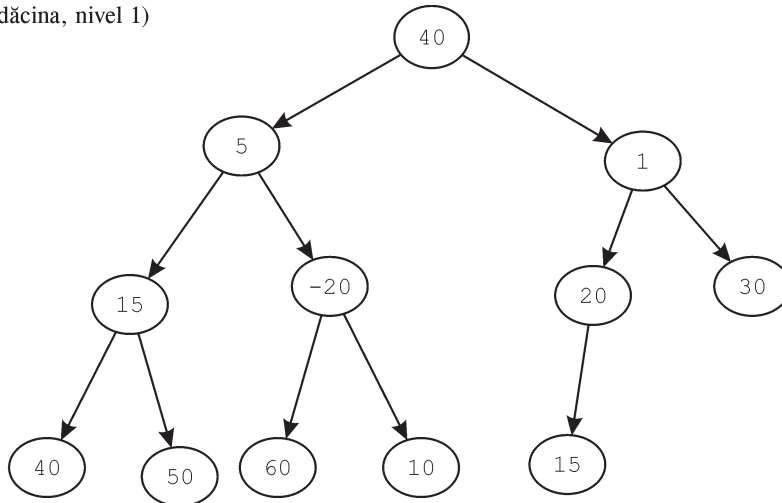
4.4.5.3. Sortarea cu grămezi (*heapsort*)

Acest algoritm este prezentat și într-un *proiect de lecție* care va fi discutat ulterior. Trebuie amintit faptul că, deși informația de intrare poate fi conținută într-un vector sau o listă (utilizându-se pointeri), ea trebuie văzută ca alcătuind un *arbore binar*. Considerăm că vectorul v de mai jos are 12 componente :

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 .

(rădăcina, nivel 1)



Proiect de tehnologie didactică

Data :

Clasa : a XI-a

Profesor :

Disciplina : Informatică aplicată

Tipul lecției : Predare-învățare

*Competențe-cheie : Formarea deprinderii de a ordona un șir utilizând *heapsort**

Competențe de natură operațională

La sfârșitul lecției, elevii vor fi capabili :

- Să definească un *heap* („grămadă”, „ansamblu”).
- Să creeze un *heap*.
- Să aplice algoritmul *heapsort*.
- Să scrie programul pentru algoritmul *heapsort*.

Strategii didactice : Conversația, explicația, metoda analitică, munca independentă etc.

Mijloace de învățământ : Manuale, culegeri de probleme

Metode : Activitate frontală, individuală

Resurse :

- pedagogice : *Metodica predării informaticii*, alte cursuri de informatică, ghiduri pentru profesori ;
- oficiale – programa școlară ;

- temporale – 50 de minute ;
- psihologice – cunoștințe dobândite de elevi până la această dată ;
- colectiv eterogen (interesat de obiect) ;
- clasa împărțită pe grupe.

Un *heap* este o multimulțime (mulțime în care anumite componente se pot repeta). Multimulțimea poate fi reprezentată ca un arbore binar (în sensul celor spuse anterior). În aceste condiții, 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 cu valorile memorate în nodurile-fii ai acestuia. Similar, *min-heap* este un arbore binar complet în care valoarea memorată în orice nod al său este mai mică sau egală cu valorile memorate în nodurile-fii ai acestuia. Deoarece, conform proprietății de *max-heap*, elementul maxim trebuie să se afle în rădăcina *heap*-ului, deci pe prima poziție din vector, el poate fi plasat pe poziția sa corectă, interschimbându-l cu elementul din poziția 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 funcție de combinare a elementelor din pozițiile 1 și $n-1$. Elementul de pe poziția n fiind deja *la locul lui*, practic nu mai este nevoie să fie inclus (formal) în *heap*. Procedeu se repetă până când toate elementele vectorului sunt plasate pe pozițiile 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()
{
    /* 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: ");
    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("Tastati valoarea pentru 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 codificarii). */
    for (i = 1; i < n; i++)
        urca(v, i);
    /* Acum vom aplica o proprietate a max-ansamblurilor, 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();
    ...}

```

4.5. 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 și analiză, a bagajului de cunoștințe și experiență ale celui care îl elaborează, punându-se în evidență caracterul de creativitate, de artă chiar al acestei activități. Reușitei 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 și 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 a etalat avantaje și 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 [D1], [DM], [DCTPCN], [Hor], [HS], [CP].

4.5.1. Metoda divide et impera

Metoda „împarte și stăpânește” a fost sugerată de ideea firească de rezolvare a unei probleme complexe prin divizarea acesteia în două sau mai multe subprobleme de același tip cu cea inițială, mai simple, prin rezolvarea cărora (folosind soluțiile deja obținute) va rezulta soluția 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. Metoda *divide et impera* admite o implementare recursivă.

Etapale necesare pentru implementarea ei pot fi descrise astfel :

1. *Divide* : împarte problema în una sau mai multe probleme similare, probleme de dimensiuni mai mici.
2. *Impera* (stăpânește) : rezolvă subprobleme recursiv ; dacă dimensiunea subproblemelor este mică, se rezolvă iterativ.
3. *Combină* : combină soluțiile subproblemelor pentru a obține soluția problemei inițiale.

Complexitatea algoritmului este dată de suma complexităților etapelor descrise mai sus.

Observație

Nu orice problemă poate fi rezolvată (eficient sau simplu) cu această tehnică.

Problemă.

Să considerăm $n \geq 1$ elemente a_1, a_2, \dots, a_n și un subșir al acestuia a_p, a_{p+1}, \dots, a_q , cu $1 < p < q < n$, asupra căruia avem de efectuat o prelucrare oarecare. Presupunem că această problemă poate fi rezolvată folosind metoda *divide et impera*.

Soluție. Metoda *divide et impera* de rezolvare a acestei probleme presupune împărțirea șirului determinat de capetele acestuia (funcția *Divide*), (p, q) , în două subșiruri (p, m) și $(m + 1, q)$, $p \leq m < q$ sau $(p, m - 1)$ și (m, q) ,

$p < m \leq q$, asupra cărora să se poată efectua mai ușor prelucrarea. Prin prelucrarea celor două subșiruri se vor obține rezultatele β și γ , care, *combine* (funcția *Combina*), vor conduce la soluția, notată cu a , a problemei inițiale. Împărțirea în subșiruri poate continua până la gradul de detaliu care permite obținerea imediată a soluției prelucrării unui subșir.

Reluând exemplul de la sortarea prin interclasare și având în vedere metoda *divide et impera*, putem rescrie codul astfel :

```
public void DiveEtImpera(ArrayList<int> x, int left, int
right)
{
    ArrayList<int> leftArray = new ArrayList<int>();
    ArrayList<int> rightArray = new ArrayList<int>();
    if (left<right)
    {
        int middle = (left + right) / 2; // Divide
        DivideEtImpera(x, left, middle); // Impera
        DivideEtImpera(x, middle + 1, right); // Impera
        Combina(x, left, middle, middle + 1, right); // Combina
        // (left, middle): index inițial și final pentru primul
        subtablou
        // (middle + 1, right): index inițial și final pentru al
        doilea subtablou.
    }
}
```

Observație

În codul preluat de la sortarea prin interclasare s-au modificat doar denumirile metodelor, pentru a pune în evidență etapele descrise în vadrul metodei *divide et impera*.

În cele mai frecvente cazuri, funcțiile *Divide* și *Combina* sunt compuse dintr-un număr redus de instrucțiuni, nemotivându-se descrierea și apelul lor separat ca funcții în cadrul funcției *DivideEtImpera*.

Exemplu

Să se testeze apartenența unui element la un șir ordonat crescător. Vezi și observația de la sortare prin inserție directă. Nu vom prezenta codul pentru această problemă, care este foarte bine tratată în majoritatea lucrărilor de specialitate.

4.5.2. Metoda backtracking

Backtracking-ul constituie una dintre metodele cel mai des folosite pentru căutarea soluției optime la 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 evitarea generării (și verificarea) tuturor soluțiilor posibile.

Soluția se construiește pas cu pas (în mod incremental). La fiecare pas se adaugă un candidat la soluție și, dacă acesta satisface restricțiile problemei, se trece la următorul pas, în caz contrar se caută alt candidat. Dacă nu există un asemenea candidat, se revine la pasul anterior, unde se caută alt candidat pentru soluție. Procesul continuă până când se găsește (sau) nu o soluție.

Algoritmul *backtracking* enumeră o mulțime de candidați parțiali care, în principiu, ar putea fi completați în diverse moduri pentru a obține toate soluțiile posibile la problema dată.

Fiecare candidat are la dispoziție o mulțime de resurse, o mulțime de valori posibile, din care la un moment dat poate utiliza una dintre acestea, în funcție de restricțiile problemei.

Înainte de prezentarea teoretică a acestei metode considerăm util exemplul următor, referitor la problema colorării hărților.

Enunțul problemei

Se consideră dată o hartă ce conține n țări, $T = \{T_1, T_2, T_3, \dots, T_n\}$ și o mulțime de culori $C = \{C_1, C_2, C_3, C_4\}$. Să se coloreze fiecare țară cu una dintre culorile date, cu condiția ca două țări vecine (cu frontieră comună) să nu fie colorate cu aceeași culoare.

Discuție

Soluțiile posibile se regăsesc în mulțimea $S = C \times C \times C \times \dots \times C$, iar S are n componente. Un vector soluție poate arată astfel: $\{C_1, C_3, C_4, \dots, C_2\}$, unde componenta de rang k semnifică țara T_k și are valoarea C_i , adică țara T_k este colorată în culoarea C_i . Înainte de a descrie algoritmul este necesar să specificăm faptul că putem reprezenta această hartă ca un graf orientat. Prima țară, T_1 , aleasă în mod arbitrar, se învecinează cu țări din mulțimea $\{T_2, T_3, \dots, T_n\}$. T_2 se învecinează cu țări din mulțimea $\{T_3, T_4, \dots, T_n\}$, țara T_k se învecinează cu țări din mulțimea $\{T(k+1), \dots, T_n\}$.

Observații

Fiecare țară are la dispoziție mulțimea de culori (resurse) $\{C_1, C_2, C_3, C_4\}$. Înainte de a alocă o resursă pentru o anumită țară, se va verifica dacă există resurse disponibile, adică țara respectivă mai are culori cu care să se deseneze.

Folosind acest exemplu, putem descrie algoritmul astfel :

1. În primul pas, țara $T1$ alege prima culoare disponibilă, $C1$. În primul pas, țara $T1$ are disponibile toate cele patru culori. Vectorul soluție devine $x = \{C1, \dots\}$.
2. Se va elimina această culoare de la țările cu care se învecinează. Resursele pentru colorarea țărilor vecine scad ; vor exista mai puține culori.
3. Se caută candidați pentru elementul următor al vectorului soluție, adică culoarea pentru țara $T2$. Presupunem că țările $T2$ și $T4$ sunt vecine cu $T1$. În acest caz, țările $T2$ și $T4$ vor avea la dispoziție culorile $\{C2, C3, C4\}$. În pasul următor, țara $T2$ va verifica dacă are culori pe care să le poată folosi și, în caz afirmativ, va alege prima culoare pe care o are la dispoziție, adică $C2$, și va proceda la eliminarea acestei culori din țările vecine (din țara $T1$ nu va elimina această culoare), al căror indice în vectorul de reprezentare al țărilor este mai mare ca 2. Dacă $T2$ nu are culori pe care să le poată folosi, ne întoarcem la țara $T1$, verificăm dacă putem alege alte culori față de cea curentă și vom repune culoarea eliminată în țările vecine.
4. Acțiunile pentru un pas generic, notat k , sunt :
 - a) Dacă nu există candidați posibili (mulțimea resurselor este vidă), ne reîntoarcem la pasul anterior, $k - 1$, cu condiția ca acesta să existe.
 - b) Dacă există candidați posibili, selectăm drept candidat posibil prima culoare disponibilă pentru Tk și eliminăm acea culoare din țările vecine care se regăsesc în $\{T(k + 1), T(k + 2), \dots, Tn\}$, apoi trecem la pasul $(k + 1)$. Reîntoarcerea la pasul anterior și selectarea altei valori au următoarele efecte :
 - repunerea vechii valori în resursele țărilor vecine ce urmează după acea țară.
 - se verifică dacă există resurse (culori) disponibile ; în caz afirmativ, se trece la b), în caz contrar se revine la pasul anterior și la eliminarea noii valori selectate din resursele acestor țări.

Problemă

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

Soluție

Mulțimea finită $A = A_1 \times A_2 \times \dots \times A_n$ se numește 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 numește *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 . La un moment dat se va alege un element dintr-o mulțime – pe care convenim să o numim *mulțimea curentă* – și, presupunând că elementele fiecărei mulțimi A_i ($1 \leq i \leq n$) sunt ordonate, elementul care se adaugă la vectorul x va fi numit *elementul curent*. Următorul algoritm (prezentat în limbaj natural) descrie metoda *backtracking* la nivel conceptual :

Pasul1. Considerăm prima mulțime, A_1 , ca fiind *mulțimea curentă*.
Pasul2. 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).
Pasul3. Verificăm dacă un asemenea element există (adică nu s-au epuizat elementele *mulțimii curente*).
a. Dacă nu există un asemenea element, atunci mulțimea curentă devine mulțimea anterioară celei curente; când o asemenea mulțime nu există, algoritmul se **oprește** (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 stabilește dacă sunt îndeplinite *condițiile de continuare* a construirii soluției optime):
i. Dacă „Da” (condițiile de continuare sunt îndeplinite), următoarea mulțime devine mulțime curentă, și se continuă cu **Pasul2**;
ii. altfel se continuă cu **Pasul3**.

Etapele în detaliu ale acestui algoritm pot fi următoarele :

B1. Definesc mulțimile A_i , $i = 1, 2, \dots, n$. Fiecare mulțime are m_i elemente, $i = 1, 2, \dots, n$, iar modul de memorare a acestor mulțimi îl alegem ca fiind coloanele matricei $A[m,n]$ (coloana i din această matrice reprezintă mulțimea A_i , iar m este cel mai mare număr dintre m_1, m_2, \dots, m_n).
B2. Completez cu informațiile necesare lipsă matricea A .
B3. Memorez numărul maxim de elemente pentru fiecare mulțime A_i , $i = 1, 2, \dots, n$ 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 și semnificația lui `null` sau `nil` din limbajele de programare).

B6. Definesc vectorul indecșilor, notat *index* (de exemplu, *index[1]* va păstra indexul elementului selectat din mulțimea *A1* și care se găsește în vectorul soluție), pentru fiecare mulțime și 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 a soluției (variabila *i* păstrează indexul mulțimii curente și ia valori de la 1 la *n*):

B7.1. *i* = 1; // luăm prima mulțime, *A1*, 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. **Cât timp** (mai am mulțimi de selectat) **execută**

```
{
// atât timp cât mai există elemente în A[.,i]
Cât timp (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!
        afișare_soluție();
    altfel
    {
        i++; // trecem la următoarea mulțime
        index[i] = 1; // în anumite cazuri se poate
                    // și 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,
// și 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];
} cât timp (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 mulțimile A_i , $i = 1, 2, \dots, n$ au același număr de elemente care sunt în ordine crescătoare și sunt numere naturale: $\{1, 2, 3, \dots, n\}$. Se pleacă inițial cu vectorul soluție $x = \{0, 0, \dots, 0\}$. Pentru componenta $x[i]$, trecerea la următorul element înseamnă $x[i]++$, iar la elementul anterior $x[i]--$. Testul de existență a elementelor pentru $x[i]$ este $1 \leq x[i] \leq n$ (similar se poate proceda și în cazul codului pentru problemele permutărilor, aranjamentelor etc.). În codul anterior, funcția `valid()` trebuie detaliată și este dependentă de enunțul problemei. Este evident că între *condițiile interne* (de optim) și *condițiile de continuare* există o strânsă legătură, sincronizarea acestora având ca efect o importantă reducere a numărului de operații.

O sinteză a metodei *backtracking* scoate în evidență patru etape principale :

- etapa în care unei componente a vectorului soluție i se atribuie o valoare din mulțimea corespunzătoare acesteia, urmată de trecerea la mulțimea (componenta) următoare ;
- etapa în care atribuirea unei valori pentru o componentă a vectorului soluție se soldează cu un eșec, situație care se încearcă a fi depășită prin trecerea la următorul element din mulțimea (curentă) corespunzătoare componentei ;
- etapa în care elementele mulțimii curente au fost epuizate, situație generată de o alegere anterioară nepotrivită, caz în care se impune o revenire la mulțimea anterioară, revenire care poate încheia nefericit – fără găsirea unei soluții – întregul proces de căutare a soluțiilor ;
- etapa revenirii în procesul de căutare a unei noi soluții după obținerea unei soluții, etapă care se realizează prin trecerea la elementul următor din ultima mulțime.

Algoritmul prezentat mai sus conduce la obținerea unei soluții (dacă există măcar o soluție). De fiecare dată, pornind de la ultima soluție obținută, pot fi determinate următoarele eventuale soluții optime.

Procedura pseudocod de mai jos realizează acest lucru pornind de la premisa că cele n mulțimi sunt cunoscute.

Vom nota cu a_{ik} al k -lea element din mulțimea A_i și 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 semnificatia  $k_1 = 0$ }
  Repeta
Repeta
  k:= k + 1
  Daca (k > mk) atunci
    k = 0      {k = 0 are semnificatia  $k_1 = 0$ }
    i = i - 1 {se realizeaza „intoarcerea”}
  altfel
     $x_i = a_{ik}$ 
    Daca ( $x_1, x_2, \dots, x_i$  conduce la optim) atunci
      i = i + 1 se verifica conditia de continuare
    Sfdaca
  Sfdaca
Panacand (i > n sau i = 0)
Daca (i > n) atunci
  „afisare solutie”
  i = n
Sfdaca
Panacand (i = 0)
Sfarsit

```

Problemă. Să se genereze toate permutările unei mulțimi având n elemente.

Să considerăm mulțimea $A = \{1, 2, \dots, n\}$, $n > 0$. Să se determine toate n -uplele de elemente distincte din A .

Soluție

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

4.5.3. Metoda greedy

Spre deosebire de metoda *backtracking*, metoda *greedy* permite determinarea unei singure soluții care corespunde unui anumit *criteriu de optim*, în cazul problemelor în care soluția se construiește ca 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 construiește soluția optimă, cu toate că în practică, înainte de aplicarea metodei, se fac prelucrări ale acestei mulțimi care măresc ordinul de complexitate.

Problemă

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

Soluție

Familia părților mulțimii finite A , notată $P(A)$, se numește *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 numește *soluție posibilă*. Pot exista mai multe soluții care satisfac condițiile de optim, dar se dorește obținerea măcar a uneia dintre ele.

Construirea unei soluții optime constă în 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, și mulțimea vidă poate fi considerată o *soluție posibilă*.

Descrierea metodei :

1. Considerăm submulțimea B , mulțimea vidă.
2. Se alege un element a din A , neales la un pas anterior.
3. Verificăm dacă submulțimea $B \cup \{a\}$ conduce la o soluție posibilă.
4. În caz afirmativ, adăugăm elementul a la mulțimea B . Se continuă cu 2 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 și căutând în fiecare pas să îmbunătățim soluția deja obținută. Această tehnică de obținere a soluției, care a dat și denumirea, oarecum ironică, a metodei (*greedy* = „lacom”), conduce în cele mai frecvente cazuri la îndepărtarea involuntară de optim, cunoscut fiind faptul (exprimat plastic prin *lăcomia pierde optimalitatea*) că optimul local nu atrage optimul global.

Acest aspect al metodei *greedy* a condus la disocierea algoritmilor elaborați cu ajutorul ei 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 și soluții *foarte proaste* în alte cazuri.

Disocierea în cele două categorii se realizează prin modalitatea de alegere a elementelor din mulțimea A . De aceea, se folosește frecvent 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 din A
    dacă ( $B \cup \{a\}$  este soluție posibilă) atunci
      k:= k + 1
      B:=  $B \cup \{a\}$ 
    sfdacă
  pana când (nu se mai pot alege elemente din A)
  sfarsit

```

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

Exemplu. Funcția maxim

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

Rezolvare

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 :

- a) Se va ordona crescător mulțimea A ;
- b) Pornind de la $B := \emptyset$, 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 corespunzător), executăm : celui mai mic coeficient neasociat unui element din A îi atașăm 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) și 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 $f(5, 7, -5, -8, -7, 3, 8) = 270$.

4.5.4. Metoda programării dinamice

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

Problemă

Noțiunea de algoritm, așa cum a fost prezentată în această lucrare, presupune ca entități distincte existența unui set de date de intrare și a unei metode de transformare succesivă a acestora, în vederea obținerii unui set coerent de date de ieșire ca rezultat al tuturor prelucrărilor. Abordarea celor trei elemente ca un sistem presupune

existența unor intercondiționări între acestea. Ca metodă de elaborare a algoritmilor pentru rezolvarea unor clase de probleme, programarea dinamică presupune identificarea acestor corelații, privind problema inițială ca un sistem de miniprobleme ce se condiționează reciproc.

Soluție

Pentru o problemă dată, fie S_0 starea sistemului format din datele de intrare și de lucru (intermediare), precum și din corelațiile existente 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 ei într-o nouă stare, S_1 . Suntem în acest moment puși în fața uneia sau mai multor probleme similare cu cea inițială și care, printr-o nouă decizie (comună) de prelucrare, conduc la o nouă stare. Schimbarea stării sistemului va continua până la obținerea unei stări finale din care se deduce o soluție optimă a problemei inițiale.

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

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

O modalitate naturală de abordare a problemei constă în luarea succesivă de decizii optime de prelucrare în ordinea d_1, d_2, \dots, d_{i-1} , pornind de la starea inițială S_0 . Decizia următoare, d_i , depinde de șirul de decizii optime deja luate d_1, d_2, \dots, d_{i-1} . Spunem în acest caz că se aplică metoda spre *înapoi* (sfârșitul șirului de decizii).

Dacă se poate stabili starea sistemului S_n din care s-ar deduce soluția optimă a problemei, este de dorit să se determine o decizie d_n , precum și 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 decizii* și starea sistemului anterioară luării acestei decizii. Fie secvența de decizii optime $d_{i+1}, d_{i+2}, \dots, d_n$ 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 șirul de decizii $d_{i+1}, d_{i+2}, \dots, d_n$. Spunem în acest caz că se aplică metoda spre *înainte* (începutul șirului de decizii).

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

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

Cele trei modalități de abordare au la bază *principiul optimalității*. Dacă d_1, d_2, \dots, d_n este un șir optim de decizii ce determină trecerea sistemului din starea inițială S_0 în starea finală S_n , atunci sunt adevărate următoarele afirmații :

- $d_{i+1}, d_{i+2}, \dots, d_n$ este un șir optim de decizii care determină trecerea sistemului din starea S_i în starea finală S_n , $\forall i, 0 \leq i \leq n-1$;
- d_1, d_2, \dots, d_i este un șir optim de decizii care determină trecerea sistemului din starea inițială S_0 în starea S_i , $\forall i, 1 \leq i \leq n$;
- $d_{i+1}, d_{i+2}, \dots, d_n$ și d_1, d_2, \dots, d_i sunt șiruri optime de decizii care determină trecerea sistemului din starea S_i în starea finală S_n și, respectiv, din starea inițială S_0 în starea S_i , $\forall i, 1 \leq i \leq n$.

Principiul optimalității sugerează stabilirea unor relații de recurență. În concluzie, rezolvarea unei probleme prin metoda programării dinamice presupune identificarea unor caracteristici ale problemei care o fac rezolvabilă cu ajutorul acestei metode :

- problema se poate descompune în subprobleme de același tip cu aceasta ;
- subproblemele nu sunt distincte, ci se intercondiționează reciproc (altfel s-ar putea aplica tehnica *divide et impera*, mult mai eficientă din punctul de vedere al consumului de memorie) ;
- necesitatea satisfacerii principiului optimalității, care implică stabilirea relației de recurență prin care se exprimă intercondiționarea subproblemelor.

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

Problemă

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

Intrare : k , de la tastatură.

Ieșire : pe ecran, *termenul de rang k din șirul lui Fibonacci* este v .

Soluție (metodă)

În șirul lui Fibonacci, primii doi termeni sunt $a_0 = 1$ și $a_1 = 1$. Relația de recurență $a_k = a_{k-1} + a_{k-2}$, $\forall k > 2$ arată că un termen se obține ca suma ultimilor doi termeni anteriori lui.

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

Decizia de trecere la o nouă stare determină următoarele prelucrări :

- aplicarea relației de recurență (calculul sumei $s = u + v$), care respectă principiul optimalității ;
- obținerea noii stări prin atribuirea valorilor $u = v$ și $v = s$.

Se obține starea $u = 1$, $v = 2$.

Aceasta este starea nou-obținută (succesoare).

Sunt respectate caracteristicile problemelor 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 $k - 1$ și $k - 2$);
- este satisfăcut principiul optimalității (*o soluție optimă pentru problema anterioară conduce la soluția optimă a problemei curente*).

4.6. Analiza complexității, corectitudinii și terminării algoritmilor/programelor

Algoritmii corespunzători rezolvării unei probleme pot diferi, chiar dacă se folosește aceeași metodă de proiectare, de către aceeași persoană, dar la date diferite. Asta ca să nu mai vorbim de folosirea unor metode diverse sau de implicarea mai multor persoane. Existența unor repere general valabile, a unor unități de măsură standard, care să poată fi folosite pentru compararea calității performanțelor algoritmilor, reprezintă clar o necesitate. După o (scurtă) introducere conceptuală ([Cro]), vom începe prin a exemplifica analiza complexității (timp/spațiu) a algoritmilor care rezolvă anumite probleme (acest lucru fiind tratat, la nivel practic, într-adevăr incomplet, și în subcapitolul anterior).

Menționăm de la început că spațiul de memorie real utilizat de un program care implementează un algoritm este format și dintr-o parte constantă, independentă de datele de intrare (în care se află memorat, de exemplu, codul executabil), a cărei mărime/dimensiune este de obicei ignorată. De asemenea, într-o analiză formală, timpul necesar introducerii valorilor de intrare și extragerii rezultatului execuției este (de obicei) ignorat. Am văzut deja că un algoritm (imperativ) reprezintă o secvență finită de pași (instrucțiuni) care descriu operații precise asupra unor informații (date) inițiale (de intrare) sau intermediare (de lucru, temporare), în vederea obținerii unor informații (rezultate) finale (de ieșire). Pașii se execută (adică operațiile precizate se efectuează în mod concret) în ordinea scrierii lor în secvență. Un algoritm calculează o funcție sau rezolvă o problemă. Intuitiv, datele de intrare reprezintă elemente din domeniul de definiție al funcției de calculat (sau informațiile inițiale din realitatea în care își are originea problema pe care vrem să o rezolvăm), iar datele de ieșire sunt elemente din codomeniul funcției (respectiv, soluțiile problemei). Un algoritm, la modul ideal, se termină pentru toate intrările admise, prin urmare există întotdeauna un ultim pas, a cărui execuție marchează de obicei și obținerea rezultatelor de ieșire. Din motive tehnice, vom lua uneori în considerare și algoritmi care nu se termină pentru toate intrările, pe care-i vom numi *semialgoritmi*.

Astfel, o *problemă algoritmică* P poate fi definită ca fiind o funcție totală $f: A \rightarrow B$. În acest caz, A constituie mulțimea informațiilor inițiale ale problemei (intrărilor posibile), iar B – mulțimea informațiilor finale (ieșirilor/rezultatelor/răspunsurilor). Dacă B are exact două elemente (DA, NU), problema se numește *problemă de decizie*. Elementul a aparținând domeniului funcției se mai numește și instanță a problemei (prin abuz de notație, vom scrie și $a \in f/a \in P$). Un *algoritm* (secvențial) care rezolvă problema f va porni cu o codificare a oricărei instanțe $a \in f$ și va calcula o codificare a rezultatului $f(a)$. Un algoritm (pseudocod, program într-un limbaj de programare etc.) va fi privit în această subsecțiune în sensul paradigmei imperative. Presupunem că fiecărei instanțe $a \in f$ i se poate asocia un număr natural $g_a(f)$, numit dimensiunea problemei, pentru instanța a . *Dimensiunea* poate fi gândită, de exemplu, ca lungimea (în număr de simboluri) a unei codificări (să zicem, binare) pentru instanța considerată. De asemenea, $g_a(f)$ poate reprezenta uneori o dimensiune structurală a informației inițiale a , în ideea că lungimea codificării va fi mărginită (superior) de un polinom având ca argument pe $g_a(f)$. Lungimea/dimensiunea unui obiect o se mai notează cu $|o|$. Resursele de calcul asociate execuției unui algoritm sunt legate de *spațiul* de memorie utilizat în decursul execuției și *timpul* necesar finalizării acesteia. Ne vom ocupa mai pe larg de resursa „timp”. Resursa „spațiu” se tratează cu totul similar (se mai pot consulta [JT], [K1], [K2], [L], [MY]). Este bine să subliniem însă faptul că aceste resurse sunt puternic corelate, astfel încât de obicei, dacă o problemă are un timp de execuție convenabil, spațiul necesar va fi mare, și reciproc.

Fie astfel o problemă P și un algoritm K ce rezolvă P (în onoarea lui Muhammad ibn Musa al-Khwarizmi, secolele VIII-IX, considerat părintele algoritmicii moderne). Vom nota cu $T_K(P)$ timpul necesar lui K pentru a calcula/rezolvă instanța $p \in P$. $T_K(P)$ va fi de fapt numărul operațiilor elementare efectuate de K în decursul execuției complete, pentru găsirea lui $P(p)$. Presupunem și că resursa timp este studiată independent de sistemul de calcul sau limbajul în care se face implementarea algoritmului. Aceasta înseamnă că execuția unei instrucțiuni nu depinde în nici un fel de operanzii implicați sau de timpul efectiv folosit pentru memorarea rezultatelor. Comportarea (în sens temporal, nu uităm) *în cazul cel mai nefavorabil* a lui K pe o intrare de dimensiune n este

$$T_K(n) = \sup\{T_K(p) \mid p \in P \text{ și } g_p(P) = n\}.$$

Analizând algoritmi într-o asemenea manieră, există avantajul de a ne asigura de faptul că timpul de lucru este mărginit superior de $T_K(n)$, pentru fiecare n . În practică este posibil însă ca $T_K(n)$ să fie determinat numai de anumite instanțe speciale, care apar foarte rar. De aceea, o alternativă ar fi să apelăm la teoria probabilităților ([CCȘ], [K]), și anume la studiul *comportării în medie* a unui algoritm. Aceasta impune următoarele :

a) Precizarea unei distribuții de probabilitate pe mulțimea instanțelor $p \in P$.

b) Determinarea mediei pentru $T_K(P)$, privită ca o variabilă aleatoare :

$$T_{K,med}(n) = \text{media}(\{T_K(p) \mid p \in P \text{ și } gp(P) = n\}).$$

Calculul mediei de mai sus se reduce de obicei la determinarea valorii unor sume (finite), câteodată existând totuși dificultăți mari de evaluare. Problema cea mai complicată nu este însă aceasta, ci efectuarea într-un mod realist a etapei precedente, notată cu a). Din acest motiv, vom insista doar asupra determinării lui $T_K(n)$. Și acest lucru este uneori foarte dificil, fără a lua în calcul considerarea unor detalii de implementare. Suntem nevoiți astfel să căutăm margini superioare (uneori, chiar inferioare) pentru $T_K(n)$, care sunt mai accesibile, și vom studia așa-numita *comportare asimptotică* a acestuia sau ordinul său de creștere. Vom adopta anumite notații uzuale pentru clasa funcțiilor (totale) de la N la N (pe scurt : $[N \rightarrow N]$). Mai exact, pentru fiecare $f \in [N \rightarrow N]$, numită în acest context și funcție de complexitate, punem :

$O(f) = \{g \mid (g: N \rightarrow N)(\exists c \in R, c > 0)(\exists n_0 \in N)(g(n) \leq c \times f(n), \text{ pentru fiecare } n \geq n_0)\}.$

$\Omega(f) = \{g \mid (g: N \rightarrow N)(\exists c \in R, c > 0)(\exists n_0 \in N)(g(n) \geq c \times f(n), \text{ pentru fiecare } n \geq n_0)\}.$

$\Theta(f) = \{g \mid (g: N \rightarrow N)(g \in O(f) \text{ I } \Omega(f))\}.$

În loc de $g \in O(f) \text{ } (\Omega(f), \Theta(f))$, se poate scrie și $g = O(f)$ (respectiv $\Omega(f)$, $\Theta(f)$). În sfârșit, comportarea asimptotică pentru $T_K(n)$ definită mai sus se va numi *complexitatea timp* a algoritmului K .

Revenind, dacă P este o problemă algoritmică, atunci o margine superioară pentru complexitatea ei („de tip” O) se poate stabili în practică prin proiectarea și analiza unui algoritm care să o rezolve. De exemplu, vom spune că P are complexitatea (*timp*) $O(f(n))$ dacă există un algoritm K care rezolvă P și K are complexitatea $T_K(n) = O(f(n))$.

Analog, P are complexitatea (*timp*) $\Omega(f(n))$ dacă orice algoritm K care rezolvă P are complexitatea $T_K(n) = \Omega(f(n))$. Mai mult, vom spune că un algoritm K pentru rezolvarea problemei P este optimal (relativ la timp) dacă P are complexitatea $\Omega(T_K(n))$. Dar a dovedi că un algoritm dat este optimal pentru o problemă este o sarcină foarte dificilă, existând puține rezultate generale și realizări practice în acest sens. De aceea ne limităm de obicei la considerarea marginilor superioare (sau inferioare, dar mai rar), adică ne vom raporta la clasa $O(f)$.

Vom face în continuare câteva precizări legate de nedeterminism, clasele formale de complexitate ale problemelor algoritmice, calculabilitate și decidabilitate pentru probleme/algoritmi și tratabilitatea algoritmilor.

Un algoritm K având proprietatea că $T_K(n) = O(f(n))$, unde f este un polinom (de grad oarecare), se va numi polinomial (va avea *complexitate polinomială*). Variante

(depinzând de aspectul funcției f): *complexitate logaritmică*, *liniară* (e vorba de polinoame de grad 1), *exponențială* etc. Exceptând problemele care nu admit rezolvări algoritmice (vezi în continuare), s-ar părea că pentru a rezolva o problemă este suficient să-i atașăm un algoritm corespunzător. Nu este chiar așa, deoarece complexitatea poate crește atât de rapid (cu dimensiunea intrării) încât timpul destinat rezolvării unei instanțe de dimensiune mare poate fi prohibitiv pentru om (indiferent de capacitatea de calcul a unui computer). Forma funcției f contează în mod esențial, deși am putea argumenta că „ 10^n este mai mic decât $n^{10.000}$ în destule cazuri”. Da, dar acest lucru se întâmplă pentru valori mici ale lui n și pentru un număr finit de numere naturale n . De aceea se justifică împărțirea clasei problemelor algoritmice nu numai în rezolvabile (există măcar un algoritm care o rezolvă, în sensul precizat) și nere-zolvabile, ci și a clasei problemelor rezolvabile în probleme tratabile (*tractable*) și netratabile (*intractable*). Vom adopta și :

Paradigmă. O problemă pentru care nu se cunosc algoritmi polinomiali (determiniști !) se consideră a fi netratabilă.

Considerăm util să prezentăm chiar acum câteva exemple didactice (preluate, aproape în totalitate, din [Cro]) pentru a justifica paradigma anterioară și faptul că o analiză competentă a complexității nu este ușor de realizat.

Exemplul 1. Să presupunem pentru simplitate că orice pas (operație elementară) al oricărui algoritm implementat necesită 10^{-6} secunde, adică $O(1) = 10^{-6}$. În aceste condiții, un algoritm cu funcția de complexitate dată de $f(n) = n$ va lucra 0,00002 secunde pentru $n = 20$ și 0,00004 secunde pentru $n = 40$. Un algoritm cu funcția de complexitate dată de $f(n) = n^5$ va lucra 3,2 secunde pentru $n = 20$ și 1,7 minute pentru $n = 40$. Un algoritm cu funcția de complexitate dată de $f(n) = 2^n$ va lucra 1,0 secunde pentru $n = 20$ și 12,7 zile pentru $n = 40$. Un algoritm cu funcția de complexitate dată de $f(n) = n^n$ va lucra 3×10^{10} secole pentru $n = 20$ și... cât oare, pentru $n = 40$? !

Exemplul 2. Fie P problema găsirii (calculării) lui a^m , unde $a \in \mathbb{N}$, $a \geq 2$, este dat. Deci P este funcția (notată la fel) având atât domeniul, cât și codomeniul egale cu \mathbb{N} și dată prin $P(m) = a^m$. Conform celor spuse anterior (ne reamintim de codificarea binară a informației), dimensiunea problemei (care depinde de fiecare instanță m , dar și de a în acest moment) va fi $g_m(P) = \lceil \log_2 a \rceil + \lceil \log_2 a \rceil \lceil \log_2 m \rceil$ este funcția „parte întreagă superioară”, de la \mathbb{R} la \mathbb{N}). Ca o observație, deoarece a este fixat, pentru m suficient de mare valoarea $\lceil \log_2 a \rceil$ practic nu mai contează și dimensiunea poate fi aproximată la $n = \lceil \log_2 m \rceil$. Deoarece folosim funcția amintită, am putea pune aproximativ și $2^n = 2^{\lceil \log_2 m \rceil} = m$ (vom proceda și în viitor în acest mod). Chiar fără vreo demonstrație formală, admitem că cel mai simplu (în toate sensurile !), trivial și determinist, algoritm (să-l notăm $A1$) care rezolvă problema este :

```

citire m;
alam:= 1;
i:= 1;
Cattimp (i ≤ m) executa
    alam:= alam * a;
    i:= i + 1
Sfcattimp

```

Numărul de operații executate de algoritm pentru instanța m este, conform observației anterioare, 2^n , adică $O(m) = O(2^n)$. Prin urmare, „cel mai simplu algoritm” al nostru este exponențial. Intuitiv, algoritmi determiniști satisfac proprietatea că, după execuția oricărui pas, pasul care urmează este unic determinat (de rezultatul execuției precedente). Nedeterminismul (definiția se obține, desigur, negând afirmația precedentă) pare o proprietate lipsită de temei, mai ales din punctul de vedere al practicii (cine își dorește un calculator despre care să nu poată fi sigur *ce* operație execută la un moment dat? !). Dar valoarea teoretică a acestui concept este inestimabilă (vezi mai jos influența asupra claselor de complexitate). În plus, situații nedeterminate chiar apar în practică (să ne gândim doar la execuția simultană, concurență, a mai multor programe/procese secvențiale, realizate pe un același calculator, dar pe procesoare diferite, având viteze diferite de efectuare a operațiilor elementare).

Să considerăm acum un algoritm recursiv (echivalent cu cel anterior, în sensul calculării aceleiași funcții), bazat pe următoarea proprietate (tot trivială) a funcției exponențiale cu baza a :

$$f(a) \begin{cases} 1, & \text{dacă } a = 0; \\ a^m = (a^2)^{m \text{ div } 2}, & \text{dacă } m \text{ este număr impar}; \\ a * a^{m-1}, & \text{dacă } m \text{ este număr par} \end{cases}$$

Algoritmul A2, rezultat prin derecursivarea proprietății anterioare, va fi:

```

citire m;
alam := 1;
Cattimp (m > 0) executa
    Daca (odd(m)) atunci
        alam:= alam × a;
        m:= m - 1
    altfel
        a:= a × a;
        m:= mdiv2
    Sfdaca
Sfcattimp

```


Să notăm și faptul că (nici) acum nu ne interesează limbajul concret de descriere a unui algoritm sau demonstrarea formală a faptului că acesta se termină și este corect din punctul de vedere al specificațiilor. Presupunem, de asemenea, că intrările și ieșirile sunt gestionate separat. După cum am precizat, ne ocupăm de cazul cel mai nefavorabil și găsim că $T_{A_2}(m)$ (numărul operațiilor elementare efectuate de A_2 pentru rezolvarea instanței m , problema pentru această instanță având dimensiunea structurală convenită deja de $n = \lceil \log_2 m \rceil$) este de ordinul $2 \times n$. Aceasta pentru că, dacă m chiar coincide cu 2^n (altfel spus, $m - 1 = 2^n - 1 = 1 + 2^1 + 2^2 + \dots + 2^{n-1}$, conform dezvoltării unei diferențe $a^n - b^n$), numărul de împărțiri executate în bucla *Cattimp* va fi de aproape n , iar numărul de operații va fi $O(2 \times n)$, deci cam $2 \times n$ (nu uităm nici de inițializarea lui *alam*, care reprezintă și ea o operație, deși nesemnificativă), ceea ce reprezintă $T_{A_2}(n)$. Prin urmare, problema noastră P va avea complexitatea $2 \times n = T_{A_2}(n) = g(n)$, iar $g \in O(f(n))$, unde $f(n) = n$. Aceasta înseamnă că pentru rezolvarea lui P am găsit un algoritm de complexitate liniară (!), ceea ce este evident o imposibilitate. Analiza este prin urmare greșită. Unde este greșeala? Ea provine din faptul că am considerat că operațiile aritmetice se fac între numere de lungime (binară) fixă. Dar ordinul de mărime al valorilor variabilelor implicate (*alam* și a) va crește odată cu creșterea valorii lui m (nu uităm că utilizarea calculatorului și a conceptelor de față sunt necesare doar în cazul valorilor mari). Astfel că, în realitate, numărul de operații elementare necesare înmulțirii unui număr întreg având o reprezentare binară de lungime minimă p (adică folosind p biți) cu altul de lungime q (folosind algoritmul uzual de înmulțire binară) este $O(p \times (p + q))$. În algoritmul anterior vor fi necesare operații (mai exact, înmulțiri) pentru aflarea succesivă a valorilor $a^2 = a \times a$, $a^4 = a^2 \times a^2$, \dots , $a^{4n} = a^{2n} \times a^{2n}$, precum și pentru a calcula $a^3 = a \times a^2$, $a^7 = a^3 \times a^4$, $a^{15} = a^7 \times a^8$ etc. Dacă vom considera drept operație elementară înmulțirea a două numere (binare) de lungime t ($= \lceil \log_2 a \rceil$), atunci în precedentul prim șir de înmulțiri se efectuează întâi o înmulțire (de tip $t \times t$), ceea ce ia timp $O(1)$; apoi o înmulțire (de tip $2t \times 2t$), care necesită $4 \times O(1)$ operații, apoi o înmulțire ($2^{n-1} \times t$) \times ($2^{n-1} \times t$) necesitând $2^{2 \times (n-1)} \times O(1)$ operații ș.a.m.d. Avem, prin urmare, un număr total de $O(2^{2 \times n})$ operații (elementare) pentru prima secvență. Procedăm similar și cu a doua secvență de mai sus, deducând la final că și algoritmul A_2 are (tot) complexitate timp exponențială.

Pentru a putea grupa formal problemele algoritmice (rezolvabile!) în clase de complexitate, este nevoie de o definiție formală a noțiunii de algoritm (semialgoritm). Aceasta poate fi introdusă cu ajutorul unor concepte precum (nu putem insista, presupunându-le a fi cunoscute cât de cât de către cititor): mulțimi și funcții recursive (calculabile prin algoritmi) și recursiv enumerabile (semicalculabile prin [semi]algoritmi); mașini Turing; Random Access Machines (vezi [MY]) etc. Tot fără a aprofunda, să spunem că într-un asemenea cadru se poate defini formal și (ne)determinismul. În general, orice obiect determinist este și nedeterminist, nu și reciproc. Vom putea astfel preciza formal și ce înseamnă calcul, accesibilitate, nedeterminism angelic (de tip „există”) sau demonic (de tip „orice”), terminare/oprire, acceptare, pre- și

post-condiții, invarianți, specificații, corectitudine etc. Se poate, de asemenea, fixa legătura dintre aceste concepte sau legătura dintre ele și calculatoarele reale. Folosind, în particular, noțiunea de mașină Turing, putem vorbi despre (similar cu ceea ce am definit anterior în mod informal): bandă de lucru, intrare, ieșire (acestea având lungime/dimensiune), configurație, pas de calcul, calcul cu succes, limbaj acceptat, algoritm atașat (funcție calculată), complexitate timp pentru o intrare x (cuvânt peste un alfabet Σ), complexitate timp pentru o mașină Turing etc. Din nou, se poate consulta $\langle [MY] \rangle$. Similar cu dezvoltările anterioare, vom nota această ultimă complexitate cu T_{MT} și ea va fi o funcție $T_{MT}: N \rightarrow N$, dată prin:

$$T_{MT}(n) = \sup_{x \in \Sigma^*, |x| = n} \{k \mid k \text{ este lungimea unui calcul de oprire al lui MT pe intrarea } x\}$$

și aceasta dacă mașina este deterministă, respectiv

$$T_{MT}(n) = \sup_{x \in L(MT), |x| = n} (\min\{k \mid k \text{ este lungimea unui calcul de acceptare al lui MT pe intrarea } x\}), \text{ dacă mașina este nedeterministă.}$$

Mai precis, dacă Σ este un *alfabet* (mulțime finită și nevidă) și MT este o mașină Turing oarecare, deterministă sau nu (având ca intrări cuvinte peste Σ), *limbajul acceptat* de MT este:

$$L(MT) = \{x \mid x \in \Sigma^* \text{ astfel încât există un calcul de acceptare al lui MT pentru intrarea } x\}.$$

Calcululele de acceptare sunt calcule de oprire care satisfac (eventual, în plus) o condiție specifică. Dacă h este o funcție pe Σ^* , spunem că h este calculabilă de mașina Turing deterministă MT dacă pentru fiecare intrare $x \in \Sigma^*$ calculul (mașina) se oprește având ieșirea $h(x)$. În cazul nedeterminist (care este de tip angelic) putem vorbi de calculabilitatea funcțiilor parțiale. Dacă $T_{MT} \in O(f)$ și f este un polinom notat p , peste N (ceea ce, reamintim, se mai scrie $T_{MT}(n) = O(p(n))$), se spune că funcția h calculată de MT este polinomial calculabilă (în cazul problemelor de decizie cuvintele calculabil/rezolvabil pot fi înlocuite de decidabil). Oricum, peste fiecare alfabet dat Σ , putem defini două clase importante de limbaje:

$$P = \{L \subseteq \Sigma^* \mid \text{există o MT deterministă și un polinom } p \text{ peste } N \text{ astfel încât } L = L(MT) \text{ și } T_{MT}(n) \leq p(n), \text{ pentru fiecare } n\}$$

și

$$NP = \{L \subseteq \Sigma^* \mid \text{există o MT nedeterministă și un polinom } p \text{ peste } N \text{ astfel încât } L = L(MT) \text{ și } T_{MT}(n) \leq p(n), \text{ pentru fiecare } n\}.$$

O mașină Turing deterministă este și nedeterministă, prin definiție. În plus, definiția timpului de lucru, $T_{MT}(n)$, al unei mașini deterministe, este mai restrictivă decât în cazul unei mașini nedeterministe, de unde rezultă $P \subseteq NP$. Încă nu se cunoaște dacă incluziunea precedentă este strictă, problema fiind una deschisă și cu implicații covârșitoare asupra teoriei generale a complexității ([1]). Ceea ce putem remarca este faptul că orice intrare x pentru o problemă algoritmică, pentru un algoritm, pentru o mașină Turing etc. poate fi presupusă (dacă nu, cazul este neinteresant din punctul de vedere al prelucrărilor electronice!) ca fiind codificată ca un cuvânt (aparținând unui Σ^* ; sau chiar lui N , cele două mulțimi având același număr de elemente dacă Σ este finit). Atunci, o problemă de decizie P poate fi privită ca o întrebare cu răspuns binar, de exemplu $P(x) = 0$ sau $P(x) = 1$. Cum atât P , cât și NP au fost definite drept clase de *limbaje*, fiecărei asemenea probleme (până la urmă, oricărei probleme algoritmice, deoarece din punctul de vedere al resurselor folosite nu ne interesează cu exactitate ieșirea $P(x)$, ci doar dacă ea există) i se poate atașa limbajul

$$L = \{x \mid x \in \Sigma^*, P(x) = 1\}.$$

Funcția caracteristică ([MY]) a acestui limbaj va fi dată chiar de P , iar rezolvarea lui P va fi același lucru cu a testa apartenența unui element x la limbajul L (problema *membership* pentru mulțimea L). Dacă $L \in P$, acest lucru va însemna că există un algoritm (privit ca o mașină Turing deterministă; neesențial, mașina Turing fiind acceptată drept un model universal pentru orice calculator) polinomial/scurt în ceea ce privește timpul necesar, care rezolvă P . Dacă $L \in NP$, algoritmul polinomial care există este nedeterminist, ceea ce este echivalent cu a spune că „putem rezolva repede/polinomial în $|x|$ problema P ” dacă $P(x) = 1$, dar dacă cumva $P(x) = 0$, atunci algoritmul poate să nu se termine (altfel spus, problema P descrie, în cazul general, o funcție parțială și avem de-a face cu un semialgoritm). Continuând, dacă sunt date două probleme de decizie $P_1: I_1 \rightarrow \{0, 1\}$ și $P_2: I_2 \rightarrow \{0, 1\}$, vom spune că P_1 se reduce polinomial la P_2 (notat $P_1 < P_2$), dacă există o funcție polinomial calculabilă $j: I_1 \rightarrow I_2$ (nu uităm că atât I_1 , cât și I_2 pot fi codificate în N , sau într-un același Σ^*), astfel încât să avem: $P_1(x) = P_2(j(x))$, pentru fiecare $x \in I_1$. O problemă de decizie P este *NP-completă* dacă $P \in NP$ și pentru fiecare $P' \in NP$ avem $P < P'$. Clasa problemelor *NP-complete* este nevidă.

Teoremă (S.A. Cook). Problema *SAT*, a satisfiabilității formulelor booleene ([Mas3]), este *NP-completă*.

Importanța clasei NP este evidentă: dacă o problemă $P \in NP$ și dacă pentru ea găsim (și) un algoritm polinomial (determinist) care să o rezolve (adică avem și $P \in P$), atunci orice altă problemă P' din NP se va putea rezolva (și) în timp polinomial (prin transformarea polinomială – conform definiției – a oricărei instanțe a lui P' într-o instanță a lui P , care poate fi rezolvată polinomial). Ceea ce ar însemna că $P = NP$.

Continuăm cu alte câteva exemple (în majoritate tot didactice și folosind pentru descrierea algoritmilor același limbaj pseudocod).

- *Problemă.* Să se calculeze suma primelor n numere naturale.

Soluție. Primul algoritm *Alg1* propus se bazează pe ideea de a construi o funcție care să calculeze succesiv sumele 0 , $0 + 1$, $0 + 1 + 2 \dots$, funcție care va returna în final valoarea sumei $1 + 2 + 3 + \dots + n$.

```

Functia suma(n)
s:= 0;
i:= 1;
Cattimp (i ≤ n) executa
    s:= s + i;
    i:= i + 1
Sfcattimp
suma:= s;
SfFunctia

```

Funcția numită *suma* va ocupa un spațiu de memorie fix pentru parametru, variabilele locale, pentru adresa de revenire și, evident, pentru cod. Nu există spațiu variabil suplimentar, deci $S_{Alg}(n) = O(1)$.

Un al doilea algoritm (*Alg2*) pe care-l propunem presupune construirea unei funcții recursive care calculează suma menționată, folosindu-se de relația de recurență $s(i) = s(i - 1) + i$, pornind inițial cu $s(0) = 0$:

```

Functia suma(n)
Daca (n = 0) atunci
    suma:= 0;
altfel
    suma:= suma(n - 1) + n;
Sfdaca
SfFunctia

```

Pentru fiecare apel al funcției precedente vor fi ocupați 4 octeți: unul pentru memorarea parametrului n , unul pentru valoarea funcției și 2 octeți pentru adresa de revenire. Se fac n apeluri recursive, deci spațiul de memorie variabil este de $5 \times n$ octeți. Algoritmul *Alg2* folosește mai mult spațiu efectiv (real) de memorie decât *Alg1*. Vom avea, de fapt, $S_{Alg}(n) = O(n)$.

Următorul exemplu prezintă un algoritm a cărui complexitate timp nu depinde decât de volumul datelor de intrare, și nu de alte caracteristici atipice (vezi și subcapitolul 4.4.3).

- *Problemă.* Să se ordoneze crescător elementele vectorului a având n componente.

Soluție (sortarea prin selecție, cu alegerea minimului; vezi și subcapitolul 4.2). Algoritmul, notat (local) cu *Alg*, se bazează pe o selecție (repetată) a celui mai mic element dintr-un subvector (subșir) al vectorului (șirului) dat. Codul este în subcapitolul 4.2.

La o iterație a buclei mari (folosind variabila i), se determină minimul subșirului a_{i+1}, \dots, a_n și elementul minim este plasat pe poziția i (elementele de la 1 la $i - 1$ fiind deja plasate pe pozițiile lor definitive). Pentru a calcula minimul dintr-un șir de k elemente sunt necesare $k - 1$ operații elementare (se presupune primul element din șir ca fiind cel minim, apoi se fac $k - 1$ comparații și eventual atribuirii până la epuizarea elementelor șirului). În total, în cazul de mai sus, se fac $(n - 1) + (n - 2) + \dots + 2 + 1 = n \times (n - 1)/2$ comparații, deci ordinul de complexitate timp este $O(n^2)$. Să subliniem faptul că timpul de execuție, în sensul situației celei mai defavorabile, nu depinde de ordinea inițială a elementelor vectorului.

În următorul exemplu vom analiza complexitatea timp în cazul cel mai defavorabil. Deși n-am dezvoltat complet partea teoretică necesară pentru a analiza complexitatea în medie, o vom prezenta (pe scurt) și pe aceasta.

- *Problemă*. Este practic aceeași cu problema precedentă: să se ordoneze crescător elementele unui vector.

Soluție (sortarea prin inserție directă, vezi și subcapitolul 4.3). Algoritmul (*Alg*) propus va ordona, în fiecare moment, un subșir obținut din cel anterior (deja ordonat) prin adăugarea unui nou element. El pornește de la subșirul cu un singur element (care este deja ordonat) și, odată cu adăugarea unui nou element pe următoarea poziție din șir, acesta este promovat până când noul subșir devine iarăși ordonat. Descrierea în pseudocod este:

```
i := 2;
Cattimp (i ≤ n) executa
    j := i;
    Cattimp (a[j - 1] > a[j] ∧ j > 1) executa
        k := a[j - 1];
        a[j - 1] := a[j];
        a[j] := k;
        j := j - 1;
    Sfcattimp
Sfcattimp
```

Analizăm complexitatea timp a algoritmului în funcție de n , dimensiunea presupusă a vectorului a ce urmează a fi sortat. La fiecare iterație a buclei principale (după i), elementele $a[1], a[2], \dots, a[i - 1]$ sunt deja ordonate și trebuie să interschimbăm elementele de forma $a[j]$ cu cele de forma $a[j - 1]$ (inițial avem $j = i$),

până când noul șir va deveni ordonat. În cazul cel mai defavorabil, când fiecare element adăugat la șir este mai mic decât cele adăugate anterior, elementul $a[i]$ adăugat va fi deplasat până pe prima poziție, iar ciclul interior se va executa de $i - 1$ ori în cadrul fiecărei execuții a ciclului exterior. Considerând astfel drept operație elementară compararea elementului $a[j - 1]$ cu elementul $a[j]$ și interschimbarea acestor elemente (atâta timp cât $a[j - 1] > a[j]$), vom avea în cazul cel mai defavorabil executate $1 + 2 + \dots + (n - 1) = n \times (n - 1)/2$ operații elementare, deci complexitatea algoritmului este $O(n^2)$.

Pentru analiza comportării în medie, mai întâi vom considera că orice permutare a elementelor șirului are aceeași probabilitate de apariție (adică orice ordine inițială este egal probabilă). Atunci, probabilitatea ca valoarea a_i , nou-adăugată la șirul a_1, a_2, \dots, a_{i-1} , să fie plasată în final pe o poziție oarecare k , din a_1, a_2, \dots, a_i ($1 \leq k \leq i$), este aceeași, adică $1/i$. Calculăm apoi numărul mediu de operații elementare (interschimbări de elemente) pentru ca elementul a_i să ajungă pe poziția k (ce va fi egal cu numărul de schimbări ce se efectuează înmulțit cu probabilitatea ca aceste schimbări să aibă loc) și numărul mediu total de operații elementare pentru un i fixat. Obținem imediat că și complexitatea algoritmului, în medie, este tot $O(n^2)$.

Încheind pe moment cu exemplificările legate de studiul complexității, să menționăm că foarte important în practică este și studiul coerent al terminării și corectitudinii programelor (nu doar verificarea *a posteriori* a acestora prin utilizarea diverselor date de test și teste experimentale). Ideea este că trebuie să ne asigurăm – dacă se poate, tot prin demonstrații formale – că programele concepute se termină pentru orice instanță admisibilă a datelor și că ele execută ceea ce vrem, înainte de a fi executate ([Me]). Presupunând astfel că informațiile de intrare admisibile sunt cele care satisfac o anumită condiție, exprimată printr-un predicat sau o condiție P (*precondiție*), rămâne să arătăm că programul se termină pentru orice asemenea instanță și că, în acest caz, informațiile de ieșire satisfac un alt predicat Q (*postcondiție*). Nu este foarte dificil de tratat astfel programele care nu conțin bucle, aceste construcții sintactice fiind singurele generatoare de informații necontrolabile sau de execuții infinite. Pentru a stăpâni ciclurile, se folosesc *predicatele invariante/invarianții*. Un predicat invariant R asociat unei bucle este adevărat înainte de prima execuție a acesteia și satisface în plus condiția că, dacă este adevărat înainte de o execuție a corpului buclei, atunci va fi adevărat și după terminarea acelei execuții. Un invariant pentru bucla exterioară din algoritmul de mai sus al sortării prin selecție poate fi exprimat prin afirmația/condiția/predicatul informal: Șirul a_1, \dots, a_i este ordonat crescător. Pentru alte detalii, vezi și [R] sau [W2].

- *Problemă*. Să se găsească elementul minimal (primul întâlnit, de la stânga la dreapta, dacă sunt mai multe) al unui șir dat de numere naturale și să se demonstreze terminarea și corectitudinea algoritmului folosit.

Soluție. Algoritmul de mai jos (des amintit în lucrarea de față), notat (local) tot prin *Alg*, este adaptat scopului de moment și este suficient de simplu pentru a nu-i

mai prezenta o descriere suplimentară. Intrarea este secvența ($k \in \mathbb{N}$, $k > 1$) a_1, a_2, \dots, a_k . Ieșirea va fi memorată în variabila *min*, care va conține acea valoare a_i care este cea mai mică din secvența anterioară ($i \in \{1, 2, \dots, k\}$) fiind cel mai mic cu proprietatea amintită). Descrierea lui *Alg* în pseudocod :

```

min := a1;
i := 2;
Cattimp (i ≤ k) executa
    Daca (min > ai) atunci
        min := ai
    Sfdaca
    i := i + 1
Sfcattimp
    
```

Precondiția (P): Secvența de intrare (să o notăm, generic, cu *a*) are cel puțin două elemente și acestea sunt numere naturale.

Postcondiția (Q): Variabila *min* conține (un) cel mai mic număr dintre elementele secvenței de intrare.

Invariantul (R), atașat unicei bucle prezente în program : pentru fiecare $i \in \{1, 2, \dots, k\}$, există $j \in \{1, 2, \dots, i\}$, astfel încât variabila *min* conține cel mai mic element al subsecvenței a_1, a_2, \dots, a_i . Această condiție poate fi scrisă pe scurt ca $(\forall i)(R(i))$, unde $R(i)$: există $j \in \{1, 2, \dots, i\}$, astfel încât variabila *min* conține cel mai mic element al subsecvenței a_1, a_2, \dots, a_i .

Să arătăm mai întâi că *Alg* se termină. În acest caz, lucrurile sunt simple deoarece programul conține o singură buclă și aceasta are un număr finit de pași/ execuții ale corpului său, cunoscut aprioric (anume $k - 1$). În privința corectitudinii, trebuie să arătăm că, dacă plecăm cu o secvență de intrare care satisface precondiția *P* și acesteia îi aplicăm algoritmul *Alg*, atunci la finalul execuției (am demonstrat deja că *Alg* se termină) este satisfăcută postcondiția *Q*. Vom arăta întâi că *R* este un invariant al unicei bucle. Mai exact, se vede direct, pentru început, că înainte de prima execuție a corpului buclei, *R* este adevărat pentru $i = 1$, adică *min* conține cel mai mic element (în sensul ordinii standard pe \mathbb{N}) al subsecvenței formate doar din a_1 . Altfel spus, înainte de prima execuție a corpului buclei, este adevărat $R(1)$. Să presupunem acum că *R* este satisfăcut înainte de cea de-a $i - a$ execuție a corpului buclei, $i \geq 2$ (pentru o anumită valoare $j \in \{1, 2, \dots, i\}$) și să arătăm că *R* este adevărat și după această execuție (pentru o anumită valoare $j' \in \{1, 2, \dots, i, i + 1\}$). Cu alte cuvinte, arătăm că, dacă înainte de cea de-a $i - a$ execuție a corpului buclei este adevărată $R(i)$, după această execuție va fi adevărată $R(i + 1)$ (deci... înainte de cea de-a $(i + 1)$ execuție !). Și aceasta, indiferent de valoarea curentă a lui i . Această ultimă afirmație este în mod evident adevărată, deoarece $R(i)$ adevărată implică faptul că *min* conține a_j , adică o cea mai mică valoare din subsecvența a_1, a_2, \dots, a_i . În cea de-a $i - a$ execuție a corpului buclei, se compară *min* cu noua valoare posibilă, adică a_{i+1} . Operația efectuată de algoritm (în urma acestei execuții)

va face, într-adevăr, ca \min să conțină o cea mai mică valoare a subsecvenței $a_1, a_2, \dots, a_i, a_{i+1}$, adică pe a_j . Astfel, după execuție va fi adevărată $R(i+1)$. În concluzie, după terminarea execuției buclei (adică după cea de-a $(k-1)$ execuție a corpului său), $R(k)$ va fi adevărată și deci \min va conține (o) cea mai mică valoare prezentă în secvența a_1, a_2, \dots, a_k .

Desigur că, pentru algoritmi mai complicați, care conțin cicluri cu un număr necunoscut (aprioric) de pași, s-ar putea să fie nevoie să se utilizeze anumite trucuri tehnice (cum ar fi salvarea variabilelor de intrare în anumite variabile de lucru temporare) și considerarea unor predicate invariante mai puternice (de exemplu, afirmații care să implice \mathcal{Q} , nu să coincidă cu acesta). De asemenea, demonstrația anterioară (privind păstrarea adevărului unui invariant pe parcursul execuției unei bucle), care mimează practic o demonstrație prin inducție matematică (în N), s-ar putea să trebuiască să fie înlocuită cu o demonstrație prin inducție structurală (într-o mulțime definită constructiv). Sunt astfel necesare cunoștințe mai profunde de logică și (măcar) de algebră și probabilități. Din lipsă de spațiu, indicăm doar câteva referințe bibliografice: [An], [CS], [L], [Mas3], [Po], neinsistând pe partea teoretică, și așa destul de complexă și, poate, inaccesibilă elevilor în mod direct (fără explicații suplimentare). Cunoașterea unei teorii generale a structurilor de date ar fi totuși benefică, la fel ca și anumite lucruri mai detaliate legate de compilatoare, programare neimperativă, sisteme de operare, medii distribuite, sisteme multimedia etc. Din nou vom apela doar la câteva referințe bibliografice: [Bac2], [Bare], [Hoa], [Hor], [HS], [JT], [Me], [Sa].

Capitolul 5

Structuri de date : liste, stive, cozi

În acest capitol vom prezenta câteva exemple (sperăm) semnificative de structuri de date clasice și de algoritmi standard care operează asupra acestor structuri.

5.1. Liste

Lista este o *multimulțime dinamică*, adică o colecție/mulțime cu un număr variabil de elemente, care se pot repeta. Elementele sunt de același *tip*. În general, elementele unei liste sunt tipuri abstracte specifice, altfel spus, tipuri de date abstracte ce configurează parțial sau total un model. Elementele unei liste se numesc *noduri*. Dacă între nodurile unei liste există o relație de succesiune, atunci lista se numește *simplu înlănțuită*. În acest caz există informația de a trece de la un nod la altul (succesor), dar nu putem determina nodul precedent unui nod dat. Dacă între nodurile unei liste există posibilitatea de a determina atât succesorul unui nod, cât și precedentul acestuia, atunci lista se numește *dublu înlănțuită*. În legătură cu listele, se au în vedere unele operații de interes general :

- Crearea unei liste.
- Accesul la un nod oarecare al listei.
- Adăugarea unui nod la o listă.
- Inserarea unui nod într-o listă.
- Ștergerea unui nod dintr-o listă.
- Ștergerea unei liste.

O listă simplu înlănțuită poate fi reprezentată grafic astfel :

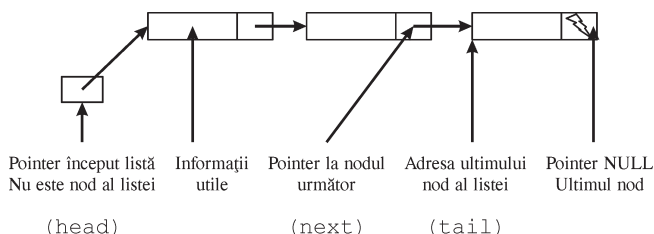


Figura 5.1. Reprezentarea grafică a unei liste simplu înlănțuite

Observații

1. *head* și *tail* sunt variabile declarate în cadrul codului în care se memorează adresa primului, respectiv al ultimului nod al listei. *next* este „pointer la tipul nodului”, face parte din definiția nodului și conține adresa următorului nod din listă sau valoarea NULL.
 2. Din figura 5.1 observăm că un nod al listei conține, alături de informația utilă, și adresa următorului nod din listă (dacă există).
 3. În cazul unei liste dublu înlănțuite, un nod al listei conține în plus și adresa nodului precedent din listă.
-

5.1.1. Liste liniare simplu înlănțuite

Între nodurile unei liste simplu înlănțuite este definită o relație numită *succesor*, mai exact fiecare nod conține un pointer a cărui valoare reprezintă adresa nodului următor din listă, cu condiția ca acesta să nu fie ultimul nod al listei. În acest caz, un nod din listă nu poate determina nodul precedent. În mod asemănător se poate defini relația de *precedent*, în care fiecare nod al listei conține un pointer a cărui valoare reprezintă adresa nodului precedent. În funcție de relația definită, putem itera nodurile unei liste plecând de la primul nod al listei către ultimul nod, relația *succesor*, sau plecând de la ultimul nod către primul nod, relația *precedent*. Listele pentru care nodurile sale satisfac și relația *succesor* și relația *precedent* se numesc *liste dublu înlănțuite*. În cele ce urmează ne vom limita numai la listele simplu înlănțuite pentru care nodurile satisfac relația *succesor*. O asemenea listă se caracterizează prin faptul că există întotdeauna un nod și numai unul care nu are nod următor (*succesor*, *fiu*), adică nodul final al listei, precum și un nod unic, care nu este următorul (*succesorul*) nici unui alt nod, adică nodul de început al listei. Aceste noduri formează *capetele* listei simplu înlănțuite. Pentru a gestiona nodurile unei liste simplu înlănțuite, 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 *head* și cu *tail*, pointerul spre nodul care nu are *succesor* în listă, adică ultimul nod al listei. Acești pointeri vor fi utilizați în toate exemplele pe care le vom avea în vedere în prelucrarea listelor simplu înlănțuite. Ei pot fi definiți fie ca variabile globale, fie ca parametri pentru funcțiile de prelucrare a listei, fie ca date-membru ale unui obiect. Tipul unui nod într-o listă simplu înlănțuită se poate defini folosind o declarație de forma (C/C++):

```
typedef struct _tagNod {  
    //Declarații. Informații utile continute de nod.  
    struct _tagNod *next;  
} MPI_Nod;
```

Pointerul *next* va conține adresa următorului nod al listei, adică definește relația *succesor* pentru nodurile listei.

Observație

Tipul unui nod pentru o *listă dublu înlănțuită* se poate defini astfel :

```
typedef struct _tagNod {  
    //Declarații. Informații utile continute de nod.  
    struct _tagNod *previous;  
    struct _tagNod *next;  
} MPI_Nod;
```

Pointerul *previous* va conține adresa nodului precedent din listă. Primul nod din listă va avea *previous = null*, iar ultimul nod din listă va avea *next = null*.

Pointerii *head* și *tail* se declară în afara oricărei funcții (de obicei înaintea definirii funcției *main()* a programului principal, deci variabile globale) prin :

```
MPI_Nod*head, *tail;
```

Pentru C# putem defini un nod și clasa ce implementează lista astfel :

```
public class Node  
{  
    public Object Data {get; set;} // Object clasa de bază în  
    Framework Class Library - FCL  
    public Node next;  
}  
  
public class LinkedList  
{  
    private Node head; // inițializat în constructor  
    private Node tail; // inițializat în constructor  
    // Metode  
}
```

Observații

1. În definiția din C#, *head* și *tail* sunt declarați ca *private*. Avem acces la listă numai prin instanța tipului *LinkedList*. În mod asemănător putem proceda și pentru C++.

2. Putem defini două proprietăți *Read Only* în *LinkedList* pentru a returna valoarea lui *head* și *tail* astfel:

```
public Node Head { get { return head;}}  
public Node Tail { get { return tail;}}
```

3. FCL din .NET conține implementări pentru aceste structuri de date.

În C++, folosind clase, putem defini un nod și clasa ce implementează lista simplu înlănțuită astfel:

```
class Node  
{  
public: int Key; // Data memorată în nod  
    Node* next;  
public:  
    Node(int key);  
    ~Node(void);  
};  
  
#include "Node.h"  
class Lsi  
{  
    Node *head;  
    Node *tail;  
public:  
    int Add(Node* node);  
    int Insert(int key, Node* newNode);  
    // result: 0 = nod negasit; 1 = nod gasit  
    Node* FindPreviousForKey(int key, int &result);  
    Node* FindCurrentForKey(int key);  
    int Delete(Node* node);  
    int Delete(int key);  
    int DeleteAll();  
    void PrintList();  
    int Count; // Număr noduri în listă  
public:  
    Lsi(void);  
    ~Lsi(void);  
};
```

Pentru clasa *Lsi* definită mai sus am definit câteva metode doar pentru exemplificare.

Pe tot parcursul acestui capitol ne vom mărgini să descriem ordinea instrucțiunilor din cadrul unor operații, ordine ce trebuie respectată atunci când lucrăm cu diverse structuri de date. Menționăm de la început că nu urmărim o optimizare a codului, ci o înțelegere corectă a operațiilor ce trebuie efectuate și claritatea codului scris. De asemenea, acest capitol nu constituie o tratare completă a structurilor prezentate, ci doar o sinteză a modelului lor de gestionare.

5.1.1.1. Crearea unei liste simplu înlănțuite

Pentru claritate, crearea unei liste ar trebui tratată după cum urmează :

LS1. Lista este vidă și trebuie creat primul nod al listei.

LS2. Lista nu este vidă și se adaugă un nou nod la sfârșitul listei.

Observații

1. Am evidențiat mai sus operațiile de adăugare și inserare a unui nod în listă. Trebuie să menționăm că prima operație va adăuga un nod la sfârșitul listei, iar a doua operație va insera un nod înaintea altui nod. În concluzie, operația de *inserare* a unui nod presupune inserarea unui nod la începutul listei.
 2. Dacă am defini că operația de inserare va insera un nod după un alt nod, atunci nu putem insera un nod la începutul listei și operația de adăugare nu-și mai are sensul.
 3. Adoptăm observația 1. în materialul ce urmează.
-

Gestionarea listei simplu înlănțuite o vom face cu ajutorul a două variabile, numite *head* și *tail*, definite mai sus.

Pentru LS1, ordinea operațiilor este următoarea : se inițializează pointerii *head* și *tail* cu valoarea *NULL*, deoarece la început lista este vidă.

```
head = NULL;  
tail = NULL;
```

LS1.1. Se rezervă o zonă de memorie în memoria *heap* pentru nodul curent.

LS1.2. Se încarcă nodul curent cu informațiile utile.

LS1.3. Se atribuie pointerilor *head* și *tail* adresa din memoria *heap* a nodului curent. Pointerii *head* și *tail* au aceeași valoare când lista este vidă (valoarea *NULL*) sau are un singur nod.

LS1.4. Se atribuie valoarea *NULL* pointerului *next* din nodul curent creat.

Codul ar putea arăta astfel :

```

...
// LS1.0.
head = NULL;
tail = NULL;
// LS1.1.
pTemp = (MPI_Nod*)malloc(sizeof(MPI_Nod));
if (pTemp == NULL)
{
    printf ("Memorie insuficienta la crearea unui nod al
listei\n");
    exit(1);
}
// LS1.2.
// Acțiuni specifice de inițializare a datelor-membru
// din nodul listei
// LS1.3.
head = pTemp;
tail = pTemp;
// LS1.4.
tail->next = NULL;

```

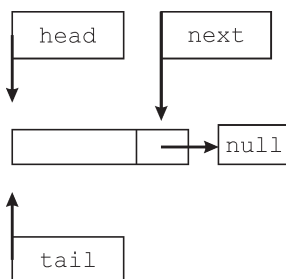


Figura 5.2. *Reprezentare grafică a listei*

Pentru LS2, ordinea operațiilor este următoarea :

LS2.1. Se rezervă o zonă de memorie în memoria *heap* pentru nodul curent.

LS2.2. Se „încarcă” nodul curent cu informațiile suplimentare.

LS2.3. Se atribuie pointerului *tail->next* adresa din memoria *heap* a nodului creat.

LS2.4. Se atribuie pointerului *tail* adresa din memoria *heap* a nodului creat.

LS2.5. Se atribuie valoarea *NULL* pointerului *tail->next*.

Codul ar putea fi următorul :

```
// LS2.1.
pTemp = (MPI_Nod*)malloc(sizeof(MPI_Nod));
if (pTemp == NULL) {
    printf ("Memorie insuficienta la crearea unui nod al
    listei\n");
    exit(1);
}
// LS2.2.
// Operații specifice de inițializare a nodului
// LS2.3. Se face legătura dintre ultimul nod
// al listei cu noul nod creat
tail->next = pTemp;
// LS2.4. Noul nod creat va deveni ultimul nod al listei
tail = pTemp;
//LS2.5. Acum tail punctează spre noul nod creat
// care nu are succesori
tail->next = NULL;
```

Observație

LS2.3 și LS2.4 trebuie să se execute în această ordine.

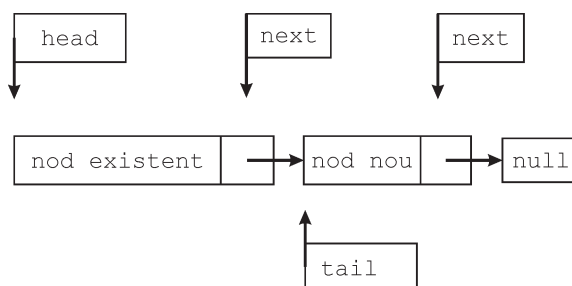


Figura 5.3. Operația de adăugare a unui nod la o listă existentă

5.1.1.2. Accesul la un nod al listei simplu înlănțuite

Pentru a găsi un anumit nod al listei, va trebui să definim anumite *criterii de identificare* pentru acesta (de exemplu, numărul de ordine al nodului, nodul care conține o anumită informație etc.). Considerăm util să arătăm modul de iterare, de parcurgere a unei liste simplu înlănțuite. Algoritmul în sine este foarte simplu : se pleacă de la primul nod și cu ajutorul datei-membru *next* (aceasta trebuie să fie diferită de valoarea *null*) din cadrul nodului obținem acces la următorul nod din listă.

Parcurgerea totală a listei pentru a afișa și/sau a efectua anumite operații poate fi realizată cu ajutorul următorului cod :

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

Observație

Variabilele *head* și *tail* nu se vor modifica în afara operațiilor de adăugare, inserare și ștergere a unui nod sau ștergere a unei liste. În exemplul de mai sus, valoarea variabilei *head* este preluată în variabila locală *pTemp*.

Presupunem că informațiile utile din nod conțin o dată-membru numită *key* care este de tip întreg.

Dacă definim drept criteriu de căutare, de identificare a unui nod după o anumită valoare a datei-membru *key*, atunci determinarea nodului respectiv se va face parcurgând lista de la început și comparând valoarea datei-membru *key* din fiecare nod parcurs cu valoarea memorată într-o variabilă locală (în general preluată de la tastatură sau rezultată în urma unor calcule anterioare sau este un parametru formal al unei funcții). Presupunem că valoarea este păstrată în variabila temporară *tempKey*. Codul poate arăta astfel (în cadrul unei funcții) :

```
...
MPI_Nod *pTemp;
// int tempKey;
// este dat ca parametru formal al metodei
...
pTemp = head;
while (pTemp != NULL)
{
    if (pTemp->key == tempKey)
        return pTemp; // în pTemp avem adresa nodului căutat
    pTemp = pTemp->next;
}
return NULL; //nu există un asemenea nod
...
```


5.1.1.3. Inserarea unui nod într-o listă simplu înlănțuită

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

1. Înaintea primului nod.
2. Înaintea unui nod precizat printr-o cheie.
3. După un nod precizat printr-o cheie.

Observație

Inserarea unui nod după un nod precizat printr-o cheie (punctul 3 de mai sus) poate coincide cu operația de adăugare a unui nod în cazul în care nodul determinat prin acea cheie este ultimul al listei.

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

Presupunem că lista nu este vidă. Dacă lista este vidă, vezi LS1 descris la 5.1.1.1.

Adresa primului nod al listei se păstrează în variabila *head*. Operațiile care trebuie efectuate, precum și ordinea acestora sunt descrise în continuare :

1. Alocare de memorie pentru noul nod, adresa se obține, de exemplu, în variabila *pTemp*. Dacă operația s-a desfășurat cu succes, se continuă cu 2, altfel se renunță la inserare.
2. Pointerul *pTemp->next* va păstra adresa următorului nod, care este de fapt fostul prim nod al listei, deci valoarea lui *head*.
pTemp->next = head;
3. Pointerul *head* va primi ca valoare adresa noului nod creat :
head = pTemp;

Observație

Dacă se inversează etapele (2) cu (3), atunci *am pierdut* lista. Pointerul *tail* va puncta spre ultimul element al listei, pointerul *head* va puncta spre noul nod creat, iar *head->next* va puncta tot spre noul nod creat. O încercare de a parcurge lista în acest moment va duce la intrarea programului într-o buclă infinită.

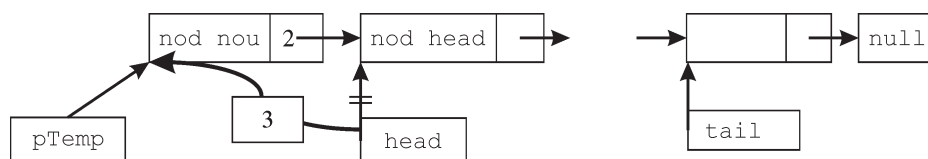


Figura 5.4. Reprezentare grafică (prin 2 și 3 am marcat operațiile descrise mai sus)

Notă: a) *nod nou* – devine nodul de început al listei, adresa păstrată în variabila *head*.

b) *nod head* – devine primul nod de după *head*, un nod normal al listei.

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

Presupunem că valoarea cheii memorate în nod este în data-membru *key*. Să reprezentăm grafic ce ar trebui să facem în această situație. Prin nod curent înțelegem nodul din listă care satisface condiția *key == tempKey* (*key* este data-membru din nod, iar *tempKey* o variabilă locală). Deoarece parcurgerea listei simplu înlănțuite se face numai înainte, va trebui să avem tot timpul adresa nodului anterior. După ce am determinat nodul înaintea căruia trebuie inserat noul nod, operația în sine se transformă în inserare nod după nod. Adresa nodului anterior o vom păstra în variabila numită *pNodAnterior*, iar adresa nodului ce satisface condiția o vom păstra în variabila *pNodCurent*. O situație specială apare atunci când primul nod al listei satisface condiția de căutare. În acest caz suntem în situația descrisă în subcapitolul anterior – „inserarea unui nod într-o listă simplu înlănțuită înaintea primului nod”, și deci vom apela la metoda deja descrisă. În celelalte cazuri (presupunând că există nodul înaintea căruia să facem inserarea) avem nevoie de adresele nodurilor păstrate în *pNodAnterior* și *pNodCurent*. Dacă păstrăm adresa noului nod în variabila *pTemp*, în acest caz nu există operații critice (ne referim la ordinea de executare a acestora). Actualizăm *pNodAnterior->next* cu *pTemp* și *pTemp->next* cu *pNodCurent*.

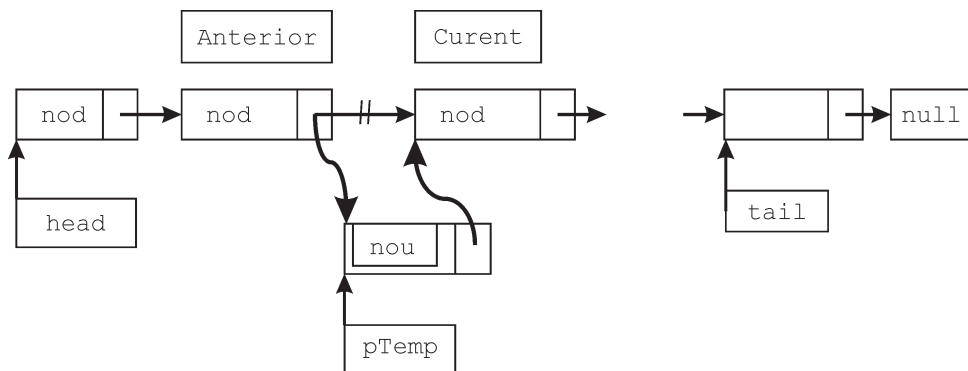


Figura 5.5. Reprezentare grafică

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_key;
...
// Determinare nod. Acest cod poate fi plasat într-o funcție
// ce va returna pNodAnterior și pNodCurent
// Prototipul funcției poate fi
// int Find(MPI_Nod* pNodAnterior, MPI_Nod* pNodCurent)
pTemp = head;
pNodAnterior = head;
pNodCurent = NULL;
while (pTemp != NULL)
{
    // Caut nodul ce satisface condiția...
    if (pTemp->key == m_key)
    {
        pNodCurent = pTemp;
        break;
    }
    pNodAnterior = pTemp;
    pTemp = pTemp->next;
}
// Sfârșit determinare nod

if (pNodCurent == NULL)
{
    printf("Nu exista cheia %d Lista nemodificata...",
m_key);
    return 1;
}
if (pNodCurent == head)
{
    // Nodul se inserează la începutul listei.
    // Se va apela funcția care tratează acest caz.
    return 0;
}
else
{
    // Alocăm 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 returnează un cod de eroare
    }

    // Inițializări pentru nodul creat
    pNodAnterior->next = pTemp;
    pTemp->next = pNodCurent;
    return 0; // Operație încheiată cu succes
}
```

5.1.1.3.3. Inserarea unui nod într-o listă simplu înlanțuită, după un nod precizat printr-o cheie

La fel ca mai sus, prin *nod curent* înțelegem nodul din listă care satisface condiția $key == m_key$.

Cazul special ce se întâlnește aici este acela când nodul ce satisface condiția este ultimul nod al listei. În acest caz, problema se transformă în „adăugarea unui nod la o listă simplu înlanțuită”, care este tratată în subcapitolul 5.1.1.1, cazul LS2.

În celelalte cazuri, această problemă se transformă conform situației de la subcapitolul 5.1.1.3.2, având în vedere următoarele observații: memorăm în variabila *pNodAnterior* adresa nodului ce satisface condiția de căutare și în *pNodCurent* adresa următorului nod din listă, adică $pNodCurent = pNodAnterior \rightarrow next$. În continuare apelăm la metoda descrisă în subcapitolul 5.1.1.3.2.

5.1.1.4. Ștergerea unui nod dintr-o listă simplu înlanțuită

Ștergerea se poate realiza în mai multe moduri, având în vedere următoarele situații :

- S1. Ștergerea primului nod al unei liste simplu înlanțuite.
- S2. Ștergerea unui nod precizat printr-o cheie.
- S3. Ștergerea ultimului nod al unei liste simplu înlanțuite.

Observații

Am evidențiat separat ștergerea primului, respectiv a ultimului nod al unei liste pentru simplul motiv că aceste operații conduc la modificarea valorii variabilei *head*, respectiv *tail*. Ștergerea unei liste simplu înlanțuite conduce de asemenea la modificarea valorii variabilelor *head* și *tail*.

Vom analiza fiecare situație în parte, punând în evidență operațiile care trebuie efectuate, precum și ordinea lor. Operația comună tuturor cazurilor luate în considerare este cea a eliberării memoriei alocate. Pentru fiecare funcție (operator) din C/C++ de alocare de memorie din memoria *heap* există definită și funcția (operatorul) corespunzător de eliberare a memoriei ocupate.

Observație

În .NET, eliberarea memoriei alocate în *heap* este realizată de către *Garbage Collector*.

5.1.1.4.1. Ștergerea primului nod al unei liste simplu înlanțuite

Ștergerea primului nod presupune reactualizarea valorii pointerului *head* cu valoarea pointerului $head \rightarrow next$. Ordinea operațiilor este următoarea :

1. Dacă valoarea pointerului *head* este *NULL*, atunci lista este vidă și nu avem ce șterge (operație terminată).

2. Dacă $head=tail$ și sunt diferite de $NULL$, atunci lista are un singur nod și vom elibera memoria ocupată de acel nod, după care vom atribui valoarea $NULL$ pentru pointerii $head$ și $tail$ (operație terminată) – în caz contrar, se trece la (3).
3. Memorăm adresa de început a listei într-o variabilă temporară, $pTemp$.
4. Atribuim pointerului $head$ valoarea pointerului $head->next$.
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.

În C/C++, operatorul folosit pentru eliberarea zonei de memorie depinde de operatorul folosit pentru alocarea acesteia. Pentru operatorul *new* (alocare memorie în *heap*), eliberarea memoriei se face folosind operatorul *delete*, iar pentru operatorul *malloc* de alocare de memorie în *heap* se folosește operatorul *free*.

5.1.1.4.2. Ștergerea unui nod precizat printr-o cheie

Ștergerea unui nod precizat printr-o cheie (nodul nu este primul și nici ultimul din listă) presupune refacerea legăturilor dintre nodul precedent și succesorul nodului șters, precum și eliberarea zonei de memorie alocate. Presupunem că lucrăm cu următoarele variabile de memorie :

- $pNodAnterior$ – conține adresa nodului precedent celui ce trebuie șters.
- $pNodCurent$ – conține adresa nodului ce trebuie șters.
- $pNodUrmator$ – conține adresa nodului succesor celui ce trebuie șters, care se obține astfel :

```
pNodUrmator = pNodCurent->next;
```

Cu aceste notații, ordinea operațiilor este următoarea :

1. Se actualizează valoarea pointerului $pNodAnterior->next$ cu valoarea pointerului $pNodUrmator$.
2. Se eliberează zona de memorie dată de $pNodCurent$.

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

Observație

Dacă nodul ce trebuie șters este primul nod al listei ($pNodCurent = head$), atunci se aplică soluția indicată în paragraful anterior.

5.1.1.4.3. Ștergerea ultimului nod al unei liste simplu înlănțuite

Această operație presupune următoarele acțiuni :

1. Dacă lista este vidă, atunci nu avem ce șterge, operația 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 *tail*.
4. Actualizarea valorii pointerului *tail* cu valoarea variabilei *pTemp*.
5. Setarea pe *NULL* a pointerului *tail->next*.

Observație

Aplicarea efectivă necesită de fiecare dată parcurgerea listei în totalitate, ceea ce pentru liste mari presupune un timp mai îndelungat.

Etapele de mai sus nu tratează cazul când lista are exact un singur nod. În această situație, înainte de etapa (2) ar trebui verificat dacă *head=tail* și sunt diferite de *NULL*. În caz afirmativ, se execută etapele descrise la ștergerea primului nod al listei. De asemenea, trebuie verificat mereu dacă lista nu este vidă.

Observații

În cazul în care folosim *POO* pentru crearea unei liste simplu înlanțuite, definim o proprietate *Count* în cadrul clasei, proprietate ce va conține numărul nodurilor din cadrul listei. Proprietatea *Count* va fi actualizată în cadrul operațiilor de adăugare, inserare și ștergere de noduri. Cu ajutorul acestei proprietăți determinăm dacă lista este vidă, *Count=0*, are un singur nod, *Count=1*, sau mai multe noduri, *Count>1*.

Determinarea penultimului nod al listei se face prin testarea egalității dintre data-membru *next* a nodului cu valoarea variabilei *tail*. Secvența de cod poate fi :

```
...
pTemp = head;
MPI_Nod *pNod; // Va păstra adresa penultimului nod din lis-
tă.
pNod = null; // Inițializare
while(pTemp != null)
{
    if (pTemp->next == tail)
    {
        pNod = pTemp;
        break;
    }
    pTemp = pTemp->next;
}
...
```

Observație

Am putea renunța la declararea variabilei locale *pNod*. La ieșirea din bucla *while*, variabila *pTemp* va conține adresa penultimului nod al listei. În acest caz, funcția ce determină penultimul nod al listei ar putea avea următorul cod C/C++ :

```

MPI_Nod* PenultimNod()
{
    // Presupunem că head și tail sunt variabile globale
    // pentru a putea fi accesate în cadrul funcției
    MPI_Nod *pTemp;
    pTemp = head;
    if (head == tail) // Lista este vidă sau conține
                        // un singur nod.
        return null; // Nu există un penultim nod.
    while (pTemp != null)
    {
        if (pTemp->next == tail)
            break;
        pTemp = pTemp->next;
    }
    return pTemp;
}

```

5.1.1.4.4. Ștergerea unei liste simplu înlănțuite

Ștergerea unei liste simplu înlănțuite se poate face prin aplicarea repetată a acțiunii de ștergere a primului nod din listă. Se repetă acest procedeu până când valoarea pointerului *head* devine *NULL*. În final, *head* și *tail* trebuie să aibă valoarea *null*.

5.1.2. Liste circulare simplu înlănțuite

Lista liniară simplu înlănțuită pentru care valoarea pointerului *tail* -> *next* este egală cu valoarea pointerului *head* (ultimul nod al listei punctează spre primul nod al listei) se numește *listă circulară simplu înlănțuită*.

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

```

MPI_Nod *head;

```

Operațiile posibile asupra acestui tip de listă sunt :

1. Adaugă nod.
2. Șterge nod.
3. Șterge listă.
4. Determinare nod.

5.1.2.1. Crearea unei liste circulare simplu înlănțuite

Crearea listei circulare simplu înlănțuite se face asemănător cazului listei simplu înlănțuite. Pentru început, variabila *head* va avea valoarea *NULL*. Nodurile care se vor adăuga vor fi plasate după nodul a cărui adresă se află memorată în *head*. Variabila *head* va menține mereu adresa ultimului nod adăugat în listă. Etapele creării listei circulare înlănțuite sunt :

1. Alocarea zonei de memorie pentru nodul care se va crea, adresa este în *pTemp* ;
2. Noul nod va fi plasat după nodul gestionat de *head*. În aceste condiții vom avea atribuirile :

```
pTemp->next= head->next;  
head->next = pTemp.
```

Un cod (parțial) 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 (head == NULL) // Lista vidă  
    {  
        head = pTemp;  
        head->next = pTemp;  
        // inițializări informații nod  
        ...  
    }  
    else  
    {  
        // Inserare după nodul identificat de head  
        pTemp->next = head->next;  
        head->next= pTemp;  
    }  
}
```


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

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

Inserarea unui nod într-o listă circulară poate fi făcută după nodul identificat de variabila *head* sau după un nod identificat printr-o *cheie*. În cadrul acestor operații de inserare trebuie avut în vedere faptul că ordinea efectuării operațiilor este critică. În caz contrar, se poate produce o distrugere a listei, memoria *heap* alocată este pierdută de către program, au loc 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 *head* este jucat de *pNodCurent*). În cazul inserării unui nod înaintea altui nod precizat printr-o cheie, în procesul de identificare a nodului înaintea căruia se face inserarea suntem obligați să memorăm și adresa nodului anterior. Dacă am determinat această adresă (a nodului anterior), problema se transformă într-o *inserare de nod după un nod cunoscut*. Presupunem că am obținut adresa nodului după care vom insera, adresă păstrată în variabila *pNod*. Codul de inserare este :

```
// Alocare în memoria heap a noului nod
pTemp = (MPI_Nod*)malloc(sizeof(MPI_Nod));
if (pTemp == NULL)
{ printf("Memorie insuficienta. Alocare esuata...");
}
else
{
    // Inserare după nodul identificat de head
    // Actualizare informații nod
    pTemp->next = pNod->next;
    pNod->next= pTemp;
}
}
```

5.1.2.3. Ștergerea unui nod dintr-o listă circulară simplu înlănțuită

Această problemă coincide cu problema ștergerii unui nod care are succesor și este succesorul altui nod dintr-o listă simplu înlănțuită.

5.1.2.4. Parcurgerea unei liste circulare simplu înlănțuite

Algoritmul de parcurgere a listei este asemănător celui de la liste liniare simplu înlănțuite, trecerea de la un nod la altul realizându-se folosind relația de succesor ce

există între noduri. Ultimul nod ce trebuie afișat îndeplinește condiția că adresa dată de *next* este egală cu adresa nodului de unde am început parcurgerea listei. Presupunând că în variabila *head* avem adresa unui nod al listei, atunci codul este următorul :

```
...
MPI_Nod *pTemp = head;
// Ar trebui verificat că lista are noduri.
// if (head == NULL) return;
do
{
    // Afișare informații nod. Calcule.
    pTemp = pTemp->next;
} while (pTemp != head);
...
```

5.1.3. Liste liniare dublu înlănțuite

Listele liniare simplu înlănțuite, precum și cele circulare discutate până acum au marele dezavantaj că relația dintre noduri este ori de precedență, ori de succesiune. Cu alte cuvinte, parcurgerea acestor liste se face într-o singură direcție, întotdeauna putându-se identifica cel mult un vecin al unui nod. *Lista dublu înlănțuită* se definește în același mod ca o listă simplu înlănțuită, cu observația că între noduri există relația de succesor și de precedență și astfel putem parcurge (itera) lista de noduri în ambele direcții.

Tipul unui nod pentru o listă dublu înlănțuită se poate defini astfel :

```
typedef struct _tagNod {
    //Declarații. Informații utile conținute de nod.
    struct _tagNod *previous;
    struct _tagNod *next;
} MPI_Nod;
```

Pointerul *previous* va conține adresa nodului precedent din listă. Primul nod din listă va avea *previous = null*, iar ultimul nod din listă va avea *next = null*.

Pentru a gestiona o listă liniară dublu înlănțuită, vom utiliza variabilele *head* și *tail* ca la listele simplu înlănțuite. Aceste variabile *punctează* spre nodurile de început și de sfârșit ale listei, noduri ce se caracterizează prin următoarele aspecte :

- Primul nod al listei nu are precedent (*head->previous = null*);
- Ultimul nod al listei nu are succesor (*tail->next = NULL*).

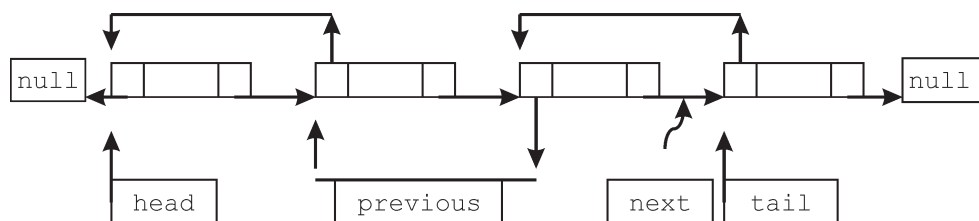


Figura 5.6. Listă liniară dublu înălănțuită

Notă : Săgețile spre dreapta indică relația de succesor, iar cele spre stânga relația de precedent.

În legătură cu listele dublu înălănțuite se pot defini aceleași operații ca în cazul listelor simplu înălănțuite :

1. Creare listă dublu înălănțuită.
2. Acces la un nod al listei.
3. Adăugare nod (la sfârșitul listei).
4. Inserare nod într-o listă dublu înălănțuită – înainte sau după un nod precizat.
5. Ștergere nod din listă.
6. Ștergere listă.

5.1.3.1. Crearea unei liste dublu înălănțuite

În momentul creării unei liste dublu înălănțuite distingem două situații :

- a) Lista este vidă și se adaugă primul nod la listă.
- b) Lista conține noduri, *adăugarea* făcându-se *după ultimul nod* (la sfârșitul listei).

Pentru cazul a), ordinea operațiilor este următoarea (se reiau operațiile de la listele simplu înălănțuite și se modifică pentru a fi funcționale pentru listele dublu înălănțuite) :

LD1. Se inițializează pointerii *head* și *tail* cu valoarea *NULL*, deoarece la început lista este vidă.

```
head = NULL;
```

```
tail = NULL;
```

LD2. Se rezervă o zonă de memorie în memoria *heap* pentru nodul curent.

LD3. Se încarcă nodul curent cu informațiile suplimentare.

LD4. Se atribuie pointerilor *head* și *tail* adresa din memoria *heap* a nodului curent. Pointerii *head* și *tail* au aceeași valoare și dacă lista este vidă (valoarea *NULL*) sau are un singur nod.

LD5. Se atribuie valoarea *NULL* pointerului *head->next*.

LD6. Se atribuie valoarea *NULL* pointerului *head->previous*.

Codul ar putea arăta astfel :

```

...
// LD1.
    head = NULL;
    tail = NULL;
// LD2.
pTemp = (MPI_Nod*)malloc(sizeof(MPI_Nod));
if (pTemp == NULL)
{
    printf ("Memorie insuficienta la crearea nodului...\n");
    exit(1);
}
// LD3.
    // Acțiuni specifice de inițializare a datelor membru
    // din nodul listei.
    // LD4.
head = pTemp;
tail = pTemp;
// LD5.
head->next = NULL;
head->previous = NULL;

```

Cazul b) presupune adăugarea nodului la sfârșitul listei. Operațiile sunt asemănătoare cu cele de la liste simplu înlanțuite, singura diferență fiind aceea că trebuie să actualizăm pointerul *previous* al nodului creat cu valoarea dată de *tail*.

Ordinea operațiilor este următoarea :

- LDA1. Se rezervă o zonă de memorie în memoria *heap* pentru nodul curent, *pTemp*.
- LDA2. Se încarcă nodul curent cu informațiile suplimentare.
- LDA3. Se atribuie pointerului *tail->next* adresa din memoria *heap* a nodului creat.
- LDA4. Se atribuie pointerului *pTemp->previous* valoarea lui *tail*.
- LDA5. Se atribuie pointerului *tail* adresa din memoria *heap* a nodului creat, *tail=pTemp*.
- LDA6. Se atribuie valoarea *NULL* pointerului *tail->next*.

Ordinea operațiilor care urmează este strictă :

1. *tail->next* ia valoarea lui *pTemp*;
2. *pTemp->previous* din nodul alocat ia valoarea variabilei *tail*.
3. Noul nod devine ultimul nod al listei, *tail* se schimbă corespunzător, *tail=pTemp*.
4. *pTemp->next* din ultimul nod adăugat ia valoarea *NULL*.

5.1.3.2. 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 sfârșit (se va folosi pointerul *next*) sau de la sfârșit spre început (se va folosi pointerul *previous*). Metoda a fost descrisă la liste simplu înlănțuite. Nu o mai reluăm aici.

5.1.3.3. 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 sfârșitul listei.

Observație

Inserarea la sfârșitul listei, adică operația (c) de mai sus, coincide cu operația de adăugare a unui nod la sfârșitul listei, operație deja descrisă.

Inserarea 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, *previous*.

Reluăm ceea ce am descris pentru listele simplu înlănțuite.

Adresa primului nod al listei (dacă nu este vidă) se păstrează în pointerul *pInceputLista*. Operațiile care trebuie efectuate, precum și ordinea lor sunt descrise în continuare :

1. Alocăm memorie pentru noul nod ; adresa se obține – de exemplu – în *pTemp*.
2. Pointerul *pTemp->next* va păstra adresa următorului nod, care este de fapt fostul prim nod al listei, deci valoarea lui *head* :

```
pTemp->next= head;
```

3. Pointerul *head->previous* va păstra adresa noului nod creat, *pTemp*.
4. Pointerul *head* va primi ca valoare adresa noului nod creat, *pTemp* :

```
head = pTemp;
```

5. Pointerul *head->previous* va primi valoarea *NULL* (este noul nod de început al listei).

Invităm cititorul să scrie singur codul complet pentru această funcție.

Inserarea 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*);
- adresa nodului succesori (*pNodUrmator*);
- adresa noului nod ce va fi inserat între cele două noduri.

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 *previous*). Adresele nodului anterior și nodului următor se obțin printr-o procedură asemănătoare cu cea descrisă la liste simplu înlanțuite. Înainte de a face inserarea, situația legăturilor (valorile pointerilor *next* și *previous*) din cele două noduri sunt :

```
pNodAnterior->next = pNodUrmator;    // succesori  
pNodUrmator->previous = pNodAnterior; // precedent
```

Ordinea operațiilor pentru inserare este :

1. Alocarea memoriei pentru noul nod ; adresa se păstrează în *pTemp* (presupunem că acțiunea de alocare s-a desfășurat cu succes).
2. Stabilirea precedenței pentru noul nod :
pTemp->previous = pNodAnterior;
3. Stabilirea nodului succesori pentru noul nod :
pTemp->next = pNodSuccesor;
4. Stabilirea nodului succesori pentru nodul anterior :
pNodAnterior->next = pTemp;
5. Stabilirea nodului precedent pentru nodul succesori :
pNodUrmator->previous = pTemp;

Observație

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

5.1.3.4. Ștergerea unui nod dintr-o listă dublu înlănțuită

După modul de amplasare a nodului care trebuie șters, distingem următoarele cazuri :

1. Ștergerea primului nod al listei.
2. Ștergerea ultimului nod al listei.
3. Ștergerea unui nod care nu este capăt al listei.

Ștergerea primului nod al listei

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

În acest caz, operația de ștergere poate fi descrisă astfel :

1. Dacă lista este vidă, operația este terminată.
 2. Memorăm adresa nodului următor :
- ```
pTemp = head->next;
```
3. Eliberăm memoria punctată de *head*.
  4. Actualizăm valoarea lui *head* cu *pTemp* (nodul următor devine primul nod) ;
  5. Noul nod de început al listei nu are precedenți :

```
head->previous = NULL;
```

#### Observații

1. 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*.
2. 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, deoarece încercăm să accesăm o zonă de memorie interzisă (adresa *0x00000000*).
3. Ce trebuie făcut? Modificăm (2) astfel :  
Dacă *pInceputLista->pElementUrmator = NULL*, atunci eliberăm zona de memorie a cărei adresă este memorată în *head*, după care setăm pe *NULL* pointerii ce mențin informația despre capetele listei (*head* și *tail*). Operația se consideră terminată și nu se mai execută celelalte etape.

#### Ștergerea ultimului nod al unei liste dublu înlănțuite

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

1. Memorăm adresa nodului precedent în *pNodPrecedent*.
- ```
pNodPrecedent = tail->previous;
```

2. Eliberăm zona de memorie „punctată” de *tail*.
3. Reactualizăm valoarea pointerului *tail* cu valoarea pointerului *pNodPrecedent*.
4. Ultimul nod al listei nu are succesori:

```
tail->next = NULL.
```

Ștergerea unui nod neterminal al listei

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

1. Memorarea adresei nodului precedent în *pNodPrecedent*:

```
pNodPrecedent = pTemp->previous;
```

2. Memorarea adresei nodului următor în *pNodUrmator*:

```
pNodUrmator = pTemp->next;
```

3. Eliberarea zonei de memorie a cărei adresă este în *pTemp*.
4. Refacere legături:

- Legătura cu nodul precedent:

```
pNodUrmator->previous = pNodPrecedent;
```

- Legătura cu nodul următor:

```
pNodPrecedent->next = pNodUrmator;
```

5.1.3.5. Ștergerea unei liste dublu înălțuite

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

```
...
MPI_NodLD *pTemp1, *pTemp;
pTemp = head;
while (pTemp != NULL)
{
    pTemp1 = pTemp->next;
    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 în așa fel încât acesta să facă o distincție clară între *definiția formală a unei structuri de date*, *reprezentarea sa grafică (vizuală)* și *diversele tipuri de implementare*.

5.2. Stive

O *stivă* este o listă simplu înlănțuită gestionată conform principiului LIFO (*Last In First Out*), adică ultimul nod pus în stivă este primul nod scos din stivă.

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

1. Adaugă element în stivă (*push*) ;
2. Scoate un element din stivă (*pop*) ;
3. Șterge stiva (*clear*).

Primele două operații afectează *vârful* stivei.

Pentru a implementa o stivă printr-o listă simplu înlănțuită, va trebui să identificăm baza și vârful stivei cu capetele listei simplu înlănțuite. Distingem două posibilități :

- ST1. Nodul spre care punctează variabila *head* este baza stivei, iar nodul spre care punctează variabila *tail* este vârful stivei ;
- ST2. Nodul spre care punctează variabila *head* este vârful stivei, iar nodul spre care punctează variabila *tail* este baza stivei.

Cazul ST1. Funcțiile *push* și *pop* se identifică cu operațiile de adăugare a unui nod la sfârșitul listei simplu înlănțuite, respectiv cu ștergerea ultimului nod al unei liste simplu înlănțuite. Dacă revenim la operația de ștergere a ultimului nod al unei liste simplu înlănțuite, atunci constatăm că funcția *pop* este inefficientă în acest caz, deoarece nu avem acces direct la penultimul nod al listei simplu înlănțuite.

Cazul ST2. Funcțiile *push* și *pop* se identifică cu operațiile de adăugare a unui nod la începutul listei simplu înlănțuite, respectiv de ștergere 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 ST2. În ambele situații, funcția *clear* – șterge stiva – se implementează la fel ca în cazul ștergerii unei liste simplu înlănțuite.

Observații

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

```
int stiva[100],
```

caz în care funcțiile *push*, *pop* și *clear* au o cu totul altă implementare. În acest caz, numărul maxim de elemente ale 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 să se verifice dacă nu se depășește valoarea maximă a indicelui (99 în acest caz), care este incrementarea valorii indicelui, urmată de actualizarea elementului stivei. Eliminarea unui element din stivă va însemna preluarea valorii curente date 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 ștergere a stivei este echivalentă cu setarea pe 0 a indicelui stivei.

5.3. Cozi

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

CO1. Adaugă element în coadă.

CO2. Scoate element din coadă.

CO3. Șterge coada.

Implementarea acestor funcții este aceeași cu implementarea funcțiilor de adăugare a unui nod la sfârșitul unei liste simplu înălțuite, respectiv de ștergere a unui nod de la începutul aceleiași liste.

Observație

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

5.4. Baze de date

Bazele de date sunt tratate în manualele pentru liceu urmând abordări de genul utilizării unui mediu de dezvoltare specific unui anumit tip de baze de date, de exemplu Visual Fox. Acest mediu de dezvoltare permite atât crearea bazei de date, cât și realizarea de aplicații ce folosesc acele baze utilizând un limbaj specific. Intenția noastră este să prezentăm posibilitățile de dezvoltare a unor aplicații ce utilizează baze de date, folosind platforma .NET Framework și limbajul C#. Exemplele pe care le furnizăm au în vedere serverul de baze de date SQL Server Express Edition, dezvoltat de Microsoft. În liceu există cursuri opționale ce tratează limbajul C# și platforma .NET Framework. Nu vom insista asupra utilizării limbajului de definire a bazei de date, DDL (*Data Definition Language*), și nici asupra DML (*Data Manipulation Language*). Vom supune atenției cititorului acea parte din .NET Framework, ADO.NET, ce permite să lucrăm cu baze de date. ADO.NET include furnizori de date pentru conectarea la o bază de date, executarea de comenzi și regăsirea rezultatelor.

Observație

Componente principale pentru .NET :

- CLR – *Common Language Runtime* ;
- BCL/FCL – *Base Class Library/Framework Class Library*.

.NET acceptă mai multe limbaje de programare (F# – Fortran, VB.NET – Visual Basic pentru .NET etc.), dintre care cel mai cunoscut și folosit este C#.

„The Common Language Runtime (CLR), the virtual machine component of Microsoft's .NET framework, manages the execution of .NET programs. A process known as just-in-time compilation converts compiled code into machine instructions which the computer's CPU then executes. The CLR provides additional services including memory management, type safety, exception handling, garbage collection, security and thread management. All programs written for the .NET framework, regardless of programming language, are executed by the CLR. All versions of the .NET framework include CLR” (https://en.wikipedia.org/wiki/Common_Language_Runtime).

5.4.1. Furnizori ADO.NET

Fiecare dezvoltator de servere de baze de date trebuie să implementeze o serie de clase pentru a permite accesul la baza de date folosind .NET Framework. Mulțimea *assemblies*, ce conține asemenea implementări, formează ceea ce se numește furnizor ADO.NET. Pentru SQL Server, clasele sunt prefixate cu SQL, pentru MySQL server,

clasele sunt prefixate cu MySQL, iar pentru Firebird Database, clasele sunt prefixate cu Fb etc.

Exemplele ce urmează au la bază următoarele două tabele ce se creează în baza de date numită Blogs.

```
CREATE TABLE [dbo].[Posts] (  
    [PostId] int IDENTITY(1,1) NOT NULL,  
    [Title] nvarchar(max) NOT NULL  
);  
GO  
CREATE TABLE [dbo].[Comments] (  
    [CommentId] int IDENTITY(1,1) NOT NULL,  
    [CommentText] nvarchar(max) NOT NULL,  
    [PostPostId] int NOT NULL  
);  
GO  
-- Creare cheie primară [PostId] în tabela 'Posts'  
ALTER TABLE [dbo].[Posts]  
ADD CONSTRAINT [PK_Posts]  
    PRIMARY KEY CLUSTERED ([PostId] ASC);  
GO  
-- Creare cheie primară [CommentId] în tabela 'Comments'  
ALTER TABLE [dbo].[Comments]  
ADD CONSTRAINT [PK_Comments]  
    PRIMARY KEY CLUSTERED ([CommentId] ASC);  
GO  
  
-- Creare cheie străină [PostPostId] în tabela 'Comments'  
ALTER TABLE [dbo].[Comments]  
ADD CONSTRAINT [FK_PostComment]  
    FOREIGN KEY ([PostPostId])  
    REFERENCES [dbo].[Posts]  
        ([PostId])  
    ON DELETE NO ACTION ON UPDATE NO ACTION;  
GO
```

Observații

Cheile primare, de tip `int IDENTITY(1,1)`, din cele două tabele sunt create de serverul bazei de date și, ca atare, nu vom furniza valori pentru acestea în cadrul operațiilor de adăugare de înregistrări.

Între cele două tabele există relația de „unu la mai multe” (*one-to-many*), iar unei înregistrări din tabela `Posts` îi corespund zero sau mai multe înregistrări din tabela `Comments`.

5.4.2. Conectarea la baza de date

Conectarea la baza de date se realizează prin instanțierea clasei `SqlConnection`, furnizând în constructorul clasei string-ul pentru conexiune urmat de apelul metodei `Open()`.

```
string connectionString = @"Data Source=.\SQLEXPRESS; Initial
Catalogue=Blogs; Integrated Security=SSPI";
SqlConnection connection = new SqlConnection(connectionString);
connection.Open();
```

În codul de mai sus se specifică drept server bază de date SQL Express, baza de date are numele Blogs, iar autentificarea folosită este cea dată de Windows. Metoda `Open()` realizează conexiunea cu baza de date, ceea ce înseamnă că putem folosi tabelele, procedurile catalogate, *view*-urile etc. ce sunt definite în baza de date. Închiderea conexiunii la baza de date se realizează prin metoda `Close()` pe obiectul `connection`:

```
connection.Close();
```

5.4.3. Execuția comenzilor

Comenzile realizate asupra bazei de date pot fi pentru inserare (adăugare) (`INSERT INTO`), actualizare (`UPDATE`), ștergere (`DELETE`), regăsire (`SELECT`) înregistrări în/din tabele. Pentru execuția acestor operații va trebui să folosim un obiect de tip `SqlCommand`, pe care vom apela metodele `ExecuteNonQuery()` sau `ExecuteReader()` în cazul regăsirii de informații din baza de date. Vom exemplifica fiecare operație în parte.

5.4.3.1. Adăugarea înregistrărilor

Pentru a adăuga înregistrări în tabele se folosește comanda `SQL`, `INSERT INTO`. Metoda pentru adăugarea de înregistrări în tabela Posts poate avea următorul cod:

```
public void InsertIntoPosts(string connectionString, string
title)
{
    string query="Insert Into dbo.Posts (Title) Values(@Title)";
    using (SqlConnection connection = new SqlConnection(connection
String))
    {
        using(SqlCommand cmd = new SqlCommand(query, connection))
        {
            cmd.Parameters.Add("@Title", SqlDbType.NVarChar, 4000).
Value = title;
            connection.Open();
            cmd.ExecuteNonQuery();
            connection.Close();
        }
    }
}
```

Adăugarea de înregistrări în tabela Comments presupune existența unei înregistrări în tabela Posts (relația de unu-la-mai-multe). Metoda pentru adăugarea unei asemenea înregistrări este furnizată în continuare.

```
public void InsertIntoComments(string connectionString, int
postId, string commentText)
{
    string query = "Insert Into Comments(CommentText, PostPostID) " +
        "Values(@CommentText, @PostId)";
    using (SqlConnection connection = new
SqlConnection(connectionString))
        using (SqlCommand cmd = new SqlCommand(query, connection))
        {
            cmd.Parameters.Add("@CommentText", SqlDbType.NVarChar,
4000).Value = commentText;
            cmd.Parameters.Add("@PostId", SqlDbType.int).Value = postId;
            connection.Open();
            cmd.ExecuteNonQuery();
            connection.Close();
        }
}
```

Observație

Codul de mai sus face presupunerea că există în tabela Posts înregistrarea ce are *PostId* egal cu cel furnizat în parametrul metodei.

5.4.3.2. Actualizarea înregistrărilor

Actualizarea/ștergerea înregistrărilor dintr-o tabelă presupun definirea unui filtru pe mulțimea înregistrărilor din tabela respectivă, filtru specificat în clauza *WHERE* a comenzilor *SQL UPDATE* sau *DELETE*. Vom crea o metodă ce are ca scop modificarea valorii coloanei *CommentText* din tabela Comments pentru o cheie specificată, *CommentId*.

```
public void UpdateComments(string connectionString, int
commentId, string newText)
{
    string query = "Update Comments set CommentText = @p1
Where commentId = @p2)";
    using (SqlConnection connection = new SqlConnection
(connectionString))
        using (SqlCommand cmd = new SqlCommand(query, connection))
```

```

{
    cmd.Parameters.Add("@p1", SqlDbType.NVarChar, 4000).Value
= newText;
    cmd.Parameters.Add("@p2", SqlDbType.int).Value = commentId;
    connection.Open();
    cmd.ExecuteNonQuery();
    connection.Close();
}
}

```

Observații

Metoda `UpdateComments` de mai sus diferă de metodele `InsertIntoPosts` și `InsertIntoComments` numai prin conținutul *string*-ului *query*. Cele trei metode prezintă o parte comună de cod ce se execută după ce un obiect de tip *SqlCommand* a fost creat (`Open()`, `ExecuteNonQuery()` și `Close()`).

5.4.4. Regăsirea informațiilor

Regăsirea informațiilor din baza de date presupune execuția unei comenzi SQL de tip *SELECT*. Scenariul pe care îl avem în vedere este acela de a regăsi toate înregistrările din `Comments` ce țin de o anumită înregistrare din `Posts`, înregistrare determinată pe baza cheii primare. Metoda este descrisă în continuare.

```

public void PostComments(string connectionString, int postId)
{
    string query = "SELECT p.PostId, p.Title, c.CommentText" +
        "From Posts as p, Comments as c " +
        "WHERE p.PostId = @p1";
    using (SqlConnection connection = new
SqlConnection(connectionString))
    using (SqlCommand cmd = new SqlCommand(query, connection))
    {
        cmd.Parameters.Add("@p1", SqlDbType.Int).Value = postId;
        connection.Open();
        SqlDataReader reader = cmd.ExecuteReader();
        bool firstPost = false;
        while(reader.Read())
        {
            if (!firstPost)
            {
                Console.WriteLine("PostId = {0}, Title = {1}",
                    reader.GetInt(0), reader.GetString(1));
                firstPost = true;
            }
        }
    }
}

```

```
        Console.WriteLine("\tCommentText = {0}", reader.  
GetString(2));  
    }  
}  
connection.Close();  
}
```

Metoda de mai sus afișează rezultatul pe ecran. În cadrul metodelor de regăsire a informațiilor folosim un obiect de tip *SqlDataReader*, ce reprezintă un cursor ce poate fi parcurs numai înainte. *SqlDataReader* expune o mulțime de metode de tip *Get...* pentru a extrage valorile din obiectul *SqlDataReader*. Pentru aceste metode *Get...* am folosit prototipul în care specificăm drept parametru indexul coloanei din înregistrarea conținută în cursor. Prima coloană are indexul zero.

Capitolul 6

Teoria grafurilor și a arborilor

6.1. Grafuri și arbori

În teoria grafurilor se definesc noțiuni, se demonstrează proprietăți, se descriu modalități de memorare și algoritmi specifici pentru prelucrarea informațiilor ce pot fi reprezentate cu ajutorul unei mulțimi de noduri conectate printr-o mulțime de muchii sau arce. În liceu se studiază *grafurile neorientate și grafurile orientate*.

Vom aminti acum câteva noțiuni fundamentale din teoria grafurilor. Pentru detalii se mai pot consulta [CLR], [CȘ], [IP], [K1], [L], [MM], [W2].

Un *graf neorientat* este o pereche ordonată de mulțimi, notată $G = (X, U)$, unde $X = \{x \mid x \in X\}$, este mulțimea nodurilor sau vârfurilor, iar $U = \{[x, y] \mid x, y \in X\}$, mulțimea muchiilor (perechi neordonate de noduri, $[x, y] = [y, x]$).

Un *graf orientat* este o pereche ordonată de mulțimi, notată $G = (X, U)$, unde $X = \{x \mid x \in X\}$ este mulțimea nodurilor sau vârfurilor, iar $U = \{(x, y) \mid x, y \in X\}$, mulțimea arcelor (perechi ordonate de noduri, $(x, y) \neq (y, x)$).

Un *nod/vârf* poate fi reprezentat în plan printr-un punct (cerc etc.), eventual numerotat, o muchie, printr-un segment de dreaptă, iar un arc poate fi reprezentat în plan printr-o săgeată direcționată.

Definiții :

- *adiacență* = proprietatea a două noduri de a fi unite prin muchie/arc ; dacă $[x, y] \in U$ sau $(x, y) \in U$, spunem că nodurile x și y sunt adiacente ;
- *incidență* = proprietatea unei muchii sau a unui arc de a uni două noduri ; dacă $[x, y] \in U$, spunem că muchia este incidentă cu nodurile x și y (analog pentru arcul (x, y)).

Într-un graf neorientat :

- *gradul nodului* x , notat cu $d(x)$ = numărul de muchii incidente cu nodul x ;
- *nod izolat* = nod cu gradul 0 ; $d(x) = 0$;
- *nod terminal* = nod cu gradul 1 ; $d(x) = 1$;
- *lanț* = succesiune de noduri cu proprietatea că oricare două noduri consecutive din lanț sunt adiacente ;

- *lanț compus* = lanț în care muchiile se pot repeta ;
- *lanț simplu* = lanț în care fiecare muchie apare o singură dată, dar nodurile se pot repeta ;
- *lanț elementar* = lanț în care nodurile sunt distincte ;
- *ciclu* = lanț în care primul nod coincide cu ultimul nod ;
- *ciclu elementar* = ciclu în care nodurile sunt distincte, cu excepția primului și ultimului nod.

Într-un graf orientat :

- *gradul intern al nodului x* , notat cu $d^-(x)$ = numărul de arce care au extremitatea finală în nodul x ;
- *gradul extern al nodului x* , notat cu $d^+(x)$ = numărul de arce care au extremitatea inițială în nodul x ;
- *nod izolat* = nod cu gradele intern și extern egale cu 0 ; $d^-(x) = d^+(x) = 0$;
- *drum* = succesiune de noduri cu proprietatea că oricare două noduri consecutive sunt adiacente (arcele păstrează aceeași orientare) ;
- *drum simplu* = drum în care nodurile se pot repeta, dar arcele care îl compun sunt distincte ;
- *drum elementar* = drum în care nodurile sunt distincte ;
- *circuit* = drum în care primul nod coincide cu ultimul nod ;
- *drum elementar* = drum în care nodurile sunt distincte, cu excepția primului și ultimului nod ;
- *lungimea unui lanț/ciclu/drum/circuit* = numărul de muchii/arce din care este format.

Definiții :

- *graf parțial* = graf care se obține din graful inițial prin eliminarea unor muchii/arce, nu și a nodurilor ;
- *subgraf* = graf care se obține din graful inițial prin eliminarea unor noduri și a tuturor muchiiilor/arcilor incidente cu acestea (nu pot fi eliminate alte muchii/arce decât cele incidente cu nodurile eliminate).

Proprietăți :

- Numărul grafurilor neorientate cu n noduri este $2^{C_n^2} = 2^{\frac{n(n-1)}{2}}$;
- Numărul grafurilor orientate cu n noduri este $4^{C_n^2} = 4^{\frac{n(n-1)}{2}} = 2^{n(n-1)}$;
- Numărul grafurilor parțiale ale unui graf cu n noduri și m muchii/arce este 2^m .

Tipuri particulare de grafuri orientate :

- *graf plin* = graf orientat în care $\forall x, y, x \neq y$, există ambele arce (x, y) și (y, x) ; numărul de arce într-un graf orientat plin este $n(n - 1)$;
- *graf complet* = graf în care $\forall x, y, x \neq y$, există cel puțin un arc (x, y) sau (y, x) ; graful complet orientat nu este unic ; numărul de arce într-un graf orientat complet este cel puțin $n(n - 1)/2$ și $n(n - 1)$;

- *graf tare conex* = graf orientat în care, $\forall x, y, x \neq y$, există drum de la x la y și există drum de la y la x (între x și y există drum dus-întors);
- *componentă tare conexă* = un subgraf tare conex și maximal în raport cu această proprietate.

Observații

1. Un nod izolat constituie o componentă tare conexă.
2. Numărul minim necesar de arce într-un graf orientat G cu n vârfuri, astfel încât G să fie tare conex este n , dacă G nu conține inițial nici un arc. În acest caz, se construiește ușor un circuit elementar care trece prin toate vârfurile, fiecare arc nou ales având extremitatea inițială egală cu extremitatea finală a arcului precedent ales (nu contează cu care vârf începem).
3. Să ne punem aceeași problemă ca la punctul anterior, renunțând la ideea circuitului elementar. Mai exact, fiind dat un graf orientat G cu n vârfuri, numărul minim necesar de arce astfel încât G să fie tare conex, oricum ar fi dispuse arcele, este $(n - 1)(n - 2) + n$. Astfel, în cel mai rău caz, construim mai întâi un subgraf plin în graful dat, folosind $n - 1$ vârfuri, ceea ce înseamnă că avem nevoie de cel mult $(n - 1)(n - 2)$ arce. Apoi rămânem cu acest subgraf plin și cu un vârf (presupus) izolat. Mai avem nevoie de cel mult $n - 1$ arce (care să aibă ca extremitate inițială vârful izolat și ca extremități finale, pe rând, cele $n - 1$ vârfuri deja selectate pentru subgraful plin), plus încă un arc care să aibă ca extremitate finală nodul izolat.

Tipuri particulare de grafuri neorientate :

- *graf regulat* = graf în care toate nodurile au grade egale ;
- *graf complet* = graf în care oricare două noduri distincte sunt adiacente ; numărul de muchii într-un graf complet este $C_2^n = n(n - 1)/2$;
- *graf aciclic* = graf în care nu există nici un ciclu ;
- *graf conex* = graf în care, oricare ar fi două noduri distincte, există cel puțin un lanț care le unește ;
- *componentă conexă* = un subgraf conex și maximal în raport cu această proprietate (nu există lanț între un nod din subgraf și un nod care nu aparține subgrafului).

Observații

Un nod izolat constituie o componentă conexă.

Numărul minim de muchii necesar pentru ca un graf neorientat cu n vârfuri să fie conex este $n - 1$ (graful este arbore).

Numărul minim de muchii necesar pentru ca un graf neorientat cu n vârfuri să fie conex, oricum ar fi dispuse muchiile, este $(n - 1)(n - 2) + 1$.

Un *arbore* este un graf conex fără circuite. În 1857, matematicianul britanic Arthur Cayley a folosit pentru prima dată termenul „arbore” în teoria grafurilor, plecând de la o analogie cu înțelesul său din domeniul botanicii. Arborii au fost studiați intensiv de numeroși matematicieni și fizicieni, precum Cayley, pe care l-au interesat aplicațiile lor în chimia organică, sau fizicianul german G.R. Kirchhoff, care a studiat această categorie pornind de la studiul rețelilor electrice.

Teorema de caracterizare a arborilor. Următoarele afirmații sunt echivalente :

1. A este arbore cu n vârfuri ;
2. A este conex cu $n - 1$ muchii ;
3. A este aciclic cu $n - 1$ muchii ;
4. A este conex minimal (dacă se elimină o muchie, se distruge conexitatea) ;
5. A este aciclic maximal (dacă se adaugă o muchie, se formează un ciclu).

Proprietate : Oricare ar fi două noduri distincte în arbore, există un lanț elementar unic care le unește.

În practică sunt multe situații când se impune ierarhizarea informațiilor ce trebuie prelucrate, cum ar fi descrierea arborelui genealogic, a organigramei unei întreprinderi sau organizarea meciurilor dintr-un campionat sportiv. Structurile de date arborescente sunt modele abstracte utilizate pentru memorarea și prelucrarea unor astfel de informații. Aceste structuri se vor numi tot arbori.

Altfel spus, prin arbore cu rădăcină, înțelegem, în primul rând o mulțime finită de noduri/vârfuri care, dacă nu este vidă, este formată din n elemente, $A = \{x_1, x_2, \dots, x_n\}$, $n > 0$, satisfăcând proprietățile :

- există un nod și numai unul care se numește *rădăcina* arborelui, notat (să spunem) x ;
- celelalte noduri se grupează în A_1, A_2, \dots, A_k submulțimi disjuncte ale lui A , care formează la rândul lor câte un arbore cu rădăcina, respectiv J_1, J_2, \dots, J_k ; acești arbori se numesc *subarbori ai rădăcinii*.

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

De asemenea, într-un arbore cu rădăcină există (desigur) și arce/muchii, ele fiind însă doar „între rădăcini” (adică, de la x la y_1, y_2, \dots, y_k , iar y_j -urile pot juca, la rândul lor, rolul unui x).

În reprezentarea ierarhizată a informațiilor descrisă cu ajutorul arborilor cu rădăcină, nodurile sunt dispuse pe niveluri, rădăcina având nivelul 0.

Fie x un nod situat pe nivelul n . Definim :

- *descendent al nodului x* = orice nod care se află pe un drum elementar ce pleacă din x , altul decât cel care unește rădăcina de x ;
- *fiu/descendent direct al nodului x* = descendent al nodului x , adiacent cu x , aflat pe nivelul $n + 1$;

- *ascendent al nodului* x = nod care se află pe drumul elementar care unește rădăcina de nodul x ;
- *părinte/tată/ascendent direct al nodului* x = ascendent al nodului x , aflat pe nivelul $n - 1$; nodul rădăcină este singurul nod din arbore care nu are părinte;
- *adâncimea/înălțimea arborelui* = lungimea maximă a unui drum elementar care unește rădăcina cu o frunză (numărul maxim de niveluri);
- *arbore degenerat* = arbore în care orice nod care nu este terminal are exact un fiu;
- *arbore ordonat* = arbore în care, pentru orice nod x , subarborii lui x sunt ordonați, în sensul valorilor asociate rădăcinilor.

În această viziune, arborii sunt structuri de date recursive ce pot fi reprezentate în memorie în mod *ascendent* (memorând părintele fiecărui nod) sau *descendent* (memorând lista de fii ai fiecărui nod). Implementarea acestora poate fi *statică* sau *dinamică*.

Pentru arborii cu rădăcină, cea mai eficientă modalitate de memorare este bazată pe reprezentarea ascendentă, în care se reține pentru fiecare nod ascendentul său direct (părintele/tatăl său).

6.2. Arbori binari

O categorie foarte importantă de arbori cu rădăcină o constituie *arborii binari*. Un arbore binar este un arbore cu rădăcină, în care toți descendenții rădăcinii se împart în două submulțimi disjuncte, ce formează câte un arbore binar între care se face distincție clară: *subarboarele stâng* și *subarboarele drept*. Arborele binar este ordonat, deoarece, în fiecare nod, subarboarele stâng se consideră că precedă subarboarele drept. Așadar, orice nod al unui arbore binar are cel mult doi fii (descendenți direcți) numiți *fiul stâng* și *fiul drept*.

Clase speciale de arbori binari :

- *Arbore binar plin* – are k niveluri numerotate de la 0 la $k - 1$ și pe fiecare nivel x are 2^x noduri. Numărul total de noduri dintr-un arbore binar plin va fi $n = 2^k = 2^0 + 2^1 + \dots + 2^{k-1}$.

Observație

Orice lanț elementar de la rădăcină la un nod din arbore va avea lungimea de cel mult $k = \lceil \log_2 n \rceil$.

- *Arbore binar strict* – orice nod care nu este frunză are exact doi fii.
- *Arbore binar complet* – se obține dintr-un arbore binar plin prin eliminarea, în ordine, a unor frunze de pe ultimul nivel, de la dreapta spre stânga.

- *Arbore binar echilibrat* – arbore binar în care, pentru orice nod, numărul de noduri din subarboarele său stâng diferă de numărul de noduri din subarboarele său drept prin cel mult o unitate.
- *Arbore binar degenerat* – arbore binar în care fiecare nod, cu excepția frunzei, are exact un fiu (stâng sau drept). Un arbore degenerat cu n vârfuri va avea n niveluri.
- *Un arbore binar de căutare* (ABC) este un arbore binar care are asociat fiecărui nod o informație-cheie, iar nodurile sunt dispuse astfel încât cheia rădăcinii oricărui subarbore să fie mai mare decât cheia fiului său stâng și mai mică decât cea a fiului său drept. Cheile sunt valori distincte dintr-o mulțime finită și ordonată.

Tabelul 6.1. Structura de date ce implementează reprezentarea ascendentă a unui arbore binar

Implementare statică	Implementare dinamică
<pre>typedef struct{ //declaratii int tata; int fiu; //fiu = -1, dacă e fiu stâng } Nod; //fiu = 1, dacă e fiu drept Nod ArbBin[NMax]; //NMax = numărul maxim de noduri int Radacina;</pre>	<pre>typedef struct Arbore{ //declarații struct Arbore* pTata; int fiu; } ArboreBinar; ArboreBinar* pRadacina;</pre>

Reprezentarea ascendentă poate fi folosită în reprezentarea eficientă a unor clase de mulțimi disjuncte sau în optimizarea operațiilor cu mulțimi/intervale (reuniune, apartenență etc.).

Reprezentarea statică a unui arbore binar complet utilizează un vector cu n componente (n = numărul de noduri din arbore). Considerăm că nodurile de pe nivelul k sunt numerotate de la stânga la dreapta cu valorile $2^k, 2^k + 1, 2^k + 2, \dots, 2^{k+1} - 1$. Astfel, pentru orice nod x , fiul stâng va fi $2 \cdot x$, dacă $2 \cdot x \leq n$, fiul drept va fi $2 \cdot x + 1$, dacă $x \cdot 2 + 1 \leq n$, iar părintele său va fi $\lfloor x/2 \rfloor$, dacă $x \geq 2$.

Această reprezentare este eficientă din punctul de vedere al spațiului de memorare ocupat și se utilizează în implemetarea *heap*-urilor.

Tabelul 6.2. Structura de date ce implementează reprezentarea descendentă a unui arbore binar

Implementare statică	Implementare dinamică
<pre>typedef struct{ // informații asociate nodului int FiuStang, FiuDrept; } Nod; Nod ArbBin[NMax]; //NMax = numărul maxim de noduri int Radacina;</pre>	<pre>typedef struct Arbore{ //informații asociate nodului struct Arbore* pStang; struct Arbore* pDrept; } ArboreBinar; ArboreBinar* pRadacina;</pre>

Practic, în cazul alocării dinamice, tipul de date este același cu cel al unei liste liniare dublu înălțuite (o zonă de informații utile și o zonă de legătură formată din doi pointeri). Pentru a fi mai intuitivă declararea listei, am schimbat numele câmpurilor și al variabilelor folosite.

Situația menționată este un exemplu edificator pentru faptul că o structură de date trebuie văzută nu numai ca o colecție de informații organizate într-un anumit mod (simplă, compusă, *array*, *struct* etc.), ci și împreună cu mulțimea de operații admise a se efectua asupra ei: asupra arborelui, operațiile admise sunt diferite față de cele asupra listelor înălțuite, stivei, cozii etc.

În continuare vom insista asupra operațiilor cel mai des utilizate în prelucrarea arborilor binari, iar în exemplele și secvențele de cod care urmează vom folosi implementarea dinamică a acestor structuri în C++:

1. Inserarea unui nod frunză.
2. Accesul la un nod al unui arbore binar.
3. Parcurgerea unui arbore binar.
4. Ștergerea unui arbore binar.

Operațiile de inserare și acces la un nod presupun, la fel ca la liste, definirea unui *criteriu de poziționare* sau *de identificare* a unui anumit nod în cadrul arborelui. Gestiunea nodurilor unui arbore binar se realizează cu *ajutorul unei variabile ce punctează spre rădăcina (sub)arborelui*. Notăm această variabilă cu *pRadacina*, definită astfel:

```
ArboreBinar* pRadacina.
```

Această variabilă are ca valoare adresa de început a zonei de memorie în care este alocată rădăcina arborelui. Dacă arborele este vid, *pRadacina* are valoarea *NULL*.

6.2.1. *Inserarea unui nod frunză într-un arbore binar alocat dinamic*

Etapele ce trebuie parcurse pentru a realiza această operație sunt:

1. Se alocă o zonă de memorie pentru nodul care urmează să se insereze în arbore. Notăm cu *pTemp* pointerul care are ca valoare adresa obținută în urma cererii de alocare de memorie. 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 afișarea unui mesaj de eroare.
2. Se atribuie valori câmpurilor ce descriu acest nod. Dacă asignările se termină cu succes, se trece la etapa următoare.
3. Deoarece noul nod va fi o frunză, se atribuie valoarea *NULL* pointerilor *pStang* și *pDrept* pentru noul nod *pTemp*:

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

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

```
pRadacina = pTemp;
```

și procesul se termină. În caz contrar, se determină poziția în care trebuie inserată noua frunză (dacă va fi fiu stâng sau drept al părintelui său) conform criteriului de identificare menționat mai sus. Presupunem că adresa tatălui noului nod va fi determinată și memorată în variabila pointer *pNodTata*. În cazul în care operația de inserare nu poate fi efectuată, eliberăm zona de memorie a cărei adresă se află în *pTemp* și procesul se termină.

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

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

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

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

Etapă a patra 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 a cărui inserare se dorește și cele existente deja în nodurile alocate. Trebuie reținut că în acest caz se va începe cu cercetarea nodului-rădăcină și că operațiile care se efectuează sunt aceleași pentru fiecare nod. Acest criteriu de identificare poate fi implementat ca o funcție recursivă 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 ieșire*. Prototipul funcției ar putea fi :

```
int identificare(ArboreBinar* pNodCurent, ArboreBinar* &pNodTata);
```

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

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

Implementarea funcției de inserare în C/C++ poate fi :


```

int identificare(ArboreBinar* pNodCurent, ArboreBinar* &pNodTata);
/*functia returneaza 0 daca nodul cautat nu a fost gasit in
arbore, o valoare nenula daca a fost identificat: -1 daca
este fiu stang al parintelui sau si 1 daca este fiu drept*/

void inserare(ArboreBinar* &pRadacina, int x){
    //x este valoarea de inserat
    ArboreBinar* pTemp;
    ArboreBinar* pTata;
    pTemp = new ArboreBinar;
    if (pTemp == NULL) {
        cout << "alocare imposibila";
        delete pTemp;
        return;
    }
    pTemp->val = x;
    pTemp->pStanga = pTemp->pDreapta = NULL;
    if (pRadacina == NULL) {
        pRadacina = pTemp; return;
    }
    int ok = identificare(pRadacina, pTata);
    if (!ok){
        cout << "Eroare la inserare.";
        delete pTemp;
        return;
    }
    if (ok < 0) pTata->pStanga = pTemp;
    else pTata->pDreapta = pTemp;
}

```

6.2.2. Parcurgerea unui arbore binar

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

- în preordine ;
- în inordine ;
- în postordine.

Parcurgerea unui arbore binar *în preordine* înseamnă accesul la rădăcina arborelui și apoi parcurgerea celor doi subarbori, întâi a celui stâng și apoi a celui drept. Subarborii, fiind la rândul lor arbori binari, se parcurg în același mod. Parcurgerea unui arbore binar *în inordine* înseamnă parcurgerea mai întâi a subarborelui stâng, apoi accesul la rădăcină și, în continuare, parcurgerea subarborelui drept. Cei doi subarbori se parcurg în același mod. Parcurgerea unui arbore binar *în postordine* înseamnă parcurgerea mai întâi a subarborelui stâng, apoi a subarborelui drept și, în

final, accesul la rădăcina arborelui. Cei doi subarbori se parcurg în același mod. Pentru fiecare dintre cele trei metode construim funcțiile *Preordine*, *Inordine* și *Postordine*, care au următorul prototip :

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

Pentru descrierea recursivă a acestora vom folosi și funcția :

```
void prelucrare_Radacina(ArboreBinar* pNod);
```

prin care vom descrie anumite operații specifice de prelucrare a informațiilor asociate rădăcinii subarborilor stâng și drept. Algoritmul pentru parcurgerea în preordine este descris în continuare. Dacă pointerul spre rădăcină nu este *NULL*, se execută etapele :

- a) se apelează funcția *prelucrareRadacina* cu valoarea parametrului *pointer spre rădăcină* ;
- b) fiul stâng devine noua rădăcină și se apelează funcția *Preordine*, cu valoarea parametrului *pointer spre noua rădăcină* (astfel se parcurge în preordine subarboarele stâng) ;
- c) fiul drept devine noua rădăcină și se apelează funcția *Preordine*, cu valoarea parametrului *pointer spre noua rădăcină*.

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

Codurile C/C++ pentru funcțiile *Inordine* și *Postordine* se construiesc în mod similar, repoziționând apelul funcției *prelucrareRadacina* conform tipului de parcurgere.

Propunem utilizarea metodelor de parcurgere pentru :

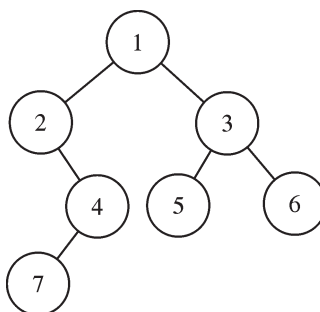
- verificarea egalității a doi arbori binari ;
- copierea unui arbore binar ;
- determinarea înălțimii unui arbore binar/numărul maxim de niveluri ;
- determinarea nodurilor frunză ;

- determinarea nodurilor de pe un nivel dat ;
- determinarea tuturor descendenților unui nod dat ;
- verificarea dacă arborele este echilibrat/strict/complet.

6.2.3. Crearea arborilor binari plecând de la parcurgerile în preordine și inordine

Considerăm cunoscute parcurgerile în preordine și inordine ale arborelui binar din imaginea alăturată :

1 2 4 7 3 5 6 (preordine)
2 7 4 1 5 3 6 (inordine)



Deducem că 1 este rădăcina din preordine, iar din inordine, deducem că toate nodurile din stânga lui 1 vor fi în subarborile stâng, 2 7 4, iar cele din dreapta, 5 3 6, vor fi în subarborile drept al rădăcinii. Parcurgem în același mod secvențele din cele două parcurgeri ale subarborilor și deducem rădăcinile și subarborii acestora. Astfel, 2 este rădăcina subarborului stâng al lui 1, iar 7 4 sunt în subarborile lui 2. Cum 7 4 sunt la dreapta lui 2 în inordine, deducem că sunt în subarborile drept al lui 2 ș.a.m.d.

Secvențele obținute prin parcurgerea în preordine și inordine sunt memorate în secvența de cod C++ de mai jos sub forma a doi vectori: *inord[NMax]*, *preord[NMax]*.

```

int preord[NMax], inord[NMax], n;
void Create(ArboreBinar* &pRadacina,int xRadacina,int xStanga,
int xDreapta){
    pRadacina = new ArboreBinar;
    if (pRadacina == NULL) {
        fout << „alocare imposibila”;
        delete pRadacina;
        return;
    }
    pRadacina->val = preord[xRadacina];
    int i = xStanga;
    while (inord[i] != preord[xRadacina])
        i++;

```

```

    if (i == xStanga) pRadacina->pStang = NULL;
    else creare(pRadacina->pStang, xRadacina + 1, xStanga, i - 1);
    if (i == xDreapta) pRadacina->pDrept = NULL;
    else
        creare(pRadacina->pDrept, xRadacina + i - xStanga + 1,
            i + 1, xDreapta);
}

void Preordine(ArboreBinar* pNod){
    if (pNod != NULL) {
        fout << pNod->val<<" ";
        Preordine(pNod->pStang); // parcurge subarborele stâng
        Preordine(pNod->pDrept); // parcurge subarborele drept
    }
}

```

Vom demonstra acum că arborele creat din parcurgerile în preordine și inordine este unic folosind metoda inducției matematice după numărul de noduri din arbore.

Fie n numărul de noduri și $inord[NMax]$, $preord[NMax]$ vectorii care memorează secvențele obținute prin parcurgerea în inordine, respectiv preordine.

$P(1)$: Arborele are un singur nod și acela este rădăcina.

$P(n)$: Orice arbore cu k noduri, $k \leq n$, construit pe baza parcurgerilor în preordine-inordine este unic.

$P(n + 1)$: Demonstrăm că un arbore cu $n + 1$ noduri construit pe baza parcurgerilor în preordine-inordine este unic, plecând de la presupunerea că $P(n)$ este adevărată.

În arborele cu $n + 1$ noduri, din parcurgerea în preordine deducem că primul nod este rădăcina. Identificându-l în secvența în inordine, deducem că nodurile aflate la stânga lui formează secvența în inordine a subarborelui stâng, iar cele din dreapta, secvența în inordine a subarborelui drept corespunzător. Deci rădăcina este determinată în mod unic, iar cei doi subarbori, fiecare cu cel mult n noduri, sunt unici, conform $P(n)$.

6.2.4. Accesul la un nod al unui arbore binar

Accesul la un nod al unui arbore binar presupune existența unui criteriu care să permită căutarea nodului respectiv, lucru pe care l-am discutat la *etapa 4*. Identificarea nodului se realizează prin intermediul unui mecanism bazat pe una dintre cele trei tipuri de parcurgeri (preordine, inordine, postordine). Dacă nodul nu există în arbore, se va returna un cod de eroare.

6.2.5. Ștergerea unui arbore binar

Pentru a șterge un arbore binar sunt necesare parcurgerea lui și ștergerea fiecărui nod. Arborele va fi parcurs în postordine (rădăcina arborelui trebuie ștearsă ultima). Codul pentru această funcție este :

```
void StergArbore(ArboreBinar* pNod) {  
    if (pNod != NULL) {  
        StergArbore(pNod->pStang);  
        StergArbore(pNod->pDrept);  
        delete pNod;  
    }  
}
```

Observație

Nu s-au pus condiții asupra valorii pointerilor înainte de a elibera memoria punctată de aceștia.

Considerăm util să continuăm cu un proiect didactic, pe care nu-l vom mai include în anexe.

Proiect de tehnologie didactică

Școala :

Disciplina : Informatică

Clasa : a XI-a

Profilul : Matematică-informatică, intensiv informatică

Data :

Profesor :

Unitatea de învățare : Structuri de date arborescente

Tema lecției : Operații cu arbori binari de căutare

Tipul lecției : Aplicații practice de laborator

Durata : 2 h 50 min.

Competențe generale :

(CG1) Identificarea datelor care intervin într-o problemă și aplicarea algoritmilor fundamentali de prelucrare a acestora.

(CG2) Elaborarea algoritmilor de rezolvare a problemelor.

(CG3) Implementarea algoritmilor într-un limbaj de programare.

Competențe specifice :

(CS1) Descrierea operațiilor specifice structurilor arborescente și elaborarea unor subprograme care să implementeze aceste operații.

(CS2) Analizarea în mod comparativ a avantajelor utilizării diferitelor metode de structurare a datelor necesare pentru rezolvarea unei probleme.

(CS3) Aplicarea în mod creativ a algoritmilor fundamentali în rezolvarea unor probleme concrete.

Competențe derivate

La sfârșitul activității didactice, elevii vor fi capabili să realizeze următoarele activități :

(CD1) Analizarea problemei propuse și identificarea structurilor de date adecvate pentru o reprezentare eficientă.

(CD2) Implementarea corectă a operațiilor cu arbori binari de căutare : inserarea, crearea, căutarea, parcurgerea, ștergerea unui nod.

(CD3) Utilizarea corectă a funcțiilor pentru rezolvarea problemei propuse.

(CD4) Testarea, depanarea și corectarea propriei aplicații, dacă este necesar.

*Competențe psihopedagogice***C1. Cognitive**

La sfârșitul lecției, elevii vor fi capabili :

(C1.1) să analizeze o problemă, să descrie etapele de rezolvare.

(C1.2) să identifice etapele de proiectare a aplicațiilor propuse.

(C1.3) să scrie funcții C++ pentru rezolvarea cerințelor propriu-zise.

(C1.4) să aprecieze corectitudinea implementării, rulând aplicația pe mai multe date de test.

C2. Competențe afective :

(C2.1) să se autoevalueze corect.

(C2.2) să dovedească curiozitate și interes pentru noțiunile utilizate.

C3. Competențe atitudinale/comportamentale :

(C3.1) să conștientizeze importanța alegerii unor structuri de date adecvate pentru rezolvarea problemei propuse.

(C3.2) să argumenteze eficiența structurilor alese din punctul de vedere al economiei de memorie.

C4. Competențe acționale

(C4.1) să identifice structura de date de tip arbore binar de căutare.

(C4.2) să descrie operațiile specifice necesare în algoritmul de rezolvare.

(C4.3) să implementeze corect funcții C++ pentru operațiile propuse.

(C4.4) să testeze, să depaneze și să corecteze eventualele erori din propria aplicație.

Strategii didactice

a) Principii didactice :

- principiul legării teoriei de practică ;
- principiul sistematizării și continuității cunoștințelor ;
- principiul accesibilității ;
- principiul individualizării și diferențierii învățării.

b) Metode și procedee didactice :

- problematizarea (P), algoritmizarea (A), conversația frontală și individuală (Cv), explicația (E), munca independentă (M).

c) Forme de organizare : lucrul individual.

d) Forme de dirijare a învățării :

- independentă ;
- dirijată de către profesor prin mijloacele de învățare.

e) Metode de evaluare :

- evaluare continuă pe parcursul lecției ;
- apreciere verbală ;
- evaluare practică.

f) Resurse materiale :

- calculatoare cu mediul de programare instalat ;
- fișe de aplicații ;
- material bibliografic : [CLR], [L], [CȘ], [IP], [MM].

C3.1, C3.2	85'	<ul style="list-style-type: none"> - Parcurgerea în ordine ar produce lista ordonată a cheilor în ABC. - Inserarea unui nod nou în ABC se va face ca frunză după ce a fost căutată poziția acestuia conform definiției ABC. - Timpul maxim necesar regăsirii unei chei în ABC este egal cu înălțimea arborelui. - Ștergerea unui nod din ABC presupune identificarea nodului și tratarea diferită a cazului în care se află : este frunză, are doar un fiu sau are doi fii. <p>Pot veni elevi la tablă care să exemplifice funcționalitatea algoritmului pe date concrete, realizând eventual o descriere grafică a operațiilor, pentru o mai bună înțelegere.</p> <p>Profesorul monitorizează activitatea individuală a elevilor, le acordă permanent suport. Dacă au fost identificate greșeli repetate la mai mulți elevi, se întrerupe lecția pentru o discuție cu toată clasa și lămurirea neclarităților.</p> <p>Profesorul încurajează elevii cu performanțe superioare să rezolve cerințele suplimentare, de exemplu transferarea arborelui binar de căutare într-un arbore binar de căutare echilibrat.</p> <p>Elevii implementează individual, testează și depanează propriile aplicații.</p> <p>Profesorul monitorizează activitatea elevilor, ghidând aplicarea noțiunilor predate și acordându-le suport când este solicitat.</p>	E
CD2, CD3, CD4		<p><i>Evaluarea</i></p> <ul style="list-style-type: none"> - pe parcurs, prin aprobare și dezaprobare verbală în urma răspunsurilor date de elevi la întrebări sau în urma observării sistematice a muncii independente ; - la finalul orei, cu note argumentate de rezultatele aplicației realizate. 	M, A
C2.1	5'	<p><i>Aprecieri activități</i></p> <p>Profesorul face aprecieri privind performanțele elevilor, face recomandări de recuperare celor care nu au reușit să rezolve sarcinile obligatorii și îi notează pe cei care au fost activi.</p> <p>Profesorul poate cere elevilor să se autoevalueze.</p>	
	4'	<p>Tema pentru acasă</p> <p>Profesorul propune tema pentru acasă, iar elevii notează în caiete.</p> <p>Tema pentru acasă ar putea fi o aplicație asemănătoare în care se creează o agendă de contacte, cu nume, număr de telefon, adresă de e-mail. Operații posibile asupra agendei :</p> <ul style="list-style-type: none"> - adăugarea unui contact ; - crearea agendei ; - modificarea datelor de contact ; - căutarea unui contact ; - afișarea listei de contacte. 	

Clasa :

Tema lecției :

Arbori binari de căutare – aplicații practice de laborator

Fișă de lucru

Concepeți o aplicație în C++ care să realizeze operații pe o listă de produse, a cărei configurație se modifică prin operații de tipul : adăugare produs, căutare produs, modificare stoc produs, ștergere produs, afișare produse, determinare valoare stoc total.

Din fișierul text *date.in* se citește o listă de produse cu informații date sub forma :
Cod Pret Stoc, unde :

- *Cod* este un număr natural nenul de 8 cifre, reprezentând codul unic al unui produs din listă ;
- *Pret* este un număr real pozitiv care reprezintă valoarea unei unități de produs ;
- *Stoc* este un număr natural care reprezintă cantitatea din produsul respectiv disponibilă.

Alegeți un tip de date adecvat, știind că se modifică dinamic configurația listei.
Scrieți funcții utilizator pentru :

1. Adăugarea unui element nou în listă, dacă nu există codul deja ; dacă produsul există, stocul se va suplimenta cu o nouă cantitate.
2. Căutarea unui produs după cod și afișarea informațiilor acestuia.
3. Determinarea numărului de produse din listă la un moment dat.
4. Calcularea valorii totale a stocului la un moment dat.
5. Afișarea listei în ordinea crescătoare a codurilor.
6. Afișarea unui meniu care să permită selectarea și efectuarea repetată a operațiilor din lista de mai sus.
7. Ștergerea din listă a produselor cu stoc 0.
8. Afișarea listei în ordine crescătoare a valorii totale a stocului pentru fiecare produs.

Notă : Subpunctele 7 și 8 sunt suplimentare.

Variantă de implementare

```
#include <fstream>
#include <iostream>
#define NMax 1000
using namespace std;
ifstream fin („abc.in”);
ofstream fout („abc.out”);
```

```
///Produs - descrierea unui produs prin cele 3 campuri: Cod,
Pret, Stoc
typedef struct {
    int Cod, Pret, Stoc;
} Produs;
///ABC = Arbore Binar de Cautare
typedef struct Nod{
    Produs prod;
    struct Nod* pStang;
    struct Nod* pDrept;
} ABC;
ABC* pRadacina;
int n;

void Inserare(ABC* &pRadacina, Produs P)
{
    ABC* pTemp = new ABC;
    if (pTemp == NULL) {
        fout << „alocare imposibila”;
        delete pTemp;
        return;
    }
    pTemp->prod = P;
    pTemp->pStang = pTemp->pDrept = NULL;
    if (pRadacina == NULL) {
        pRadacina = pTemp;
        return;
    }
    ABC *r=pRadacina;
    while (1)
    {
        if (r->prod.Cod == pTemp->prod.Cod)
            { r->prod.Stoc += P.Stoc; return; }
        if (r->prod.Cod > pTemp->prod.Cod)
            if (r->pStang) r = r->pStang;
            else { r->pStang = pTemp; return;}
        else if (r->pDrept) r = r->pDrept;
            else { r->pDrept = pTemp; return;}
    }
}

void Creare(ABC* &pRadacina)
{
    Produs P;
    pRadacina = 0;
    while (fin >> P.Cod >> P.Pret >> P.Stoc)
        Inserare(pRadacina, P);
}
```

```

void Inordine(ABC* pNod)
{
    if (pNod != NULL)
    {
        Inordine(pNod->pStang); /* parcurge subarborele stâng în
inordine*/
        cout << pNod->prod.Cod << " , " << pNod->prod.Pret << " , "
<< pNod->prod.Stoc << " , \n";
        Inordine(pNod->pDrept); /* parcurge subarborele drept în
inordine*/
    }
}

ABC* CautaNod(ABC* pRadacina, int x)
{
    ABC *r=pRadacina;
    while (r)
    {
        if (r->prod.Cod == x) return r;
        if (r->prod.Cod > x) r = r->pStang;
        else r = r->pDrept;
    }
    return NULL;
}

int NumarProduce(ABC* pNod)
{
    if (pNod == NULL) return 0;
    else return 1 + NumarProduce(pNod->pStang) +
NumarProduce(pNod->pDrept);
}

int ValoareStoc(ABC* pNod)
{
    if (pNod == NULL) return 0;
    else return pNod->prod.Pret * pNod->prod.Stoc +
NumarProduce(pNod->pStang) + NumarProduce(pNod->pDrept);
}

void StergeNodStoc0(ABC* &pRadacina, ABC *pNod, ABC *pTata)
{
    if(pNod != NULL)
    {
        StergeNodStoc0(pRadacina, pNod->pStang, pNod);
        StergeNodStoc0(pRadacina, pNod->pDrept, pNod);
        if (pNod->prod.Stoc == 0)

```

```

{
    if(pNod->pDrept != NULL && pNod->pStang != NULL)
        // nodul de sters pNod are ambii fii
    {
        /* inlocuiesc informatia din radacina cu cea din
        /* cel mai din dreapta nod al subarborelui stang
        /// care are cheia cea mai mare mai mica decat rada-
        cina*/
        ABC *p = pNod->pStang;
        pTata = pNod;
        while (p->pDrept != NULL)
        {
            pTata = p;
            p = p->pDrept;
        }
        pNod->prod = p->prod;
        pNod = p; /*nodul de sters devine p; p are cel mult
2 fii*/
    }
    if(pNod->pDrept == NULL) // are cel mult fiu stang
    {
        if (pTata == NULL) pRadacina = pNod->pStang;
        else if (pTata->pDrept == pNod)
            pTata->pDrept = pNod->pStang;
        else pTata->pStang = pNod->pStang;
    }
    else
    if(pNod->pStang == NULL) // are cel mult fiu drept
    {
        if (pTata == NULL) pRadacina = pNod->pDrept;
        else if (pTata->pDrept == pNod)
            pTata->pDrept = pNod->pDrept;
        else pTata->pStang = pNod->pDrept;
    }
    delete pNod;
}
}
}

void StergArbore(ABC* pNod)
{
    if (pNod != NULL) {
        StergArbore(pNod->pStang);
        StergArbore(pNod->pDrept);
        delete pNod;
    }
}

```

```

void afisareMeniu()
{
    cout<<"\t1. Adauga produs\n";
    cout<<"\t2. Cauta produs\n";
    cout<<"\t3. Afiseaza lista de produse\n";
    cout<<"\t4. Valoare stoc\n";
    cout<<"\t5. Numar de produse din lista\n";
    cout<<"\t6. Stergere produse cu stoc 0\n";
    cout<<"\t7. Iesire\n";
}
int main()
{
    int optiune, cod;
    Produs P;
    ABC* pNod;
    Creare(pRadacina);
    while(1)
    {
        afisareMeniu();
        cin>>optiune;
        switch (optiune)
        {
            case 1: cout << „dati produsul de adaugat (cod, pret,
cantitate): „; cin >> P.Cod >> P.Pret >> P.Stoc;
                Inserare(pRadacina, P);
                break;
            case 2: cout << "dati codul produsului cautat: ";
                cin >> cod;
                pNod = CautaNod(pRadacina, cod);
                if (pNod == NULL) cout << "Produsul nu este in lista";
                else cout << pNod->prod.Cod << ' ' << pNod->prod.
Pret << ' ' << pNod->prod.Stoc<<' \n';
                break;
            case 3: Inordine(pRadacina); cout<<"\n\n";
                break;
            case 4: cout << "Valoarea totala a stocului = " <<
                ValoareStoc(pRadacina)<<"\n\n";
                break;
            case 5: cout << "Numarul de produse din lista = " <<
                NumarProduse(pRadacina)<<"\n\n"; break;
            case 6: StergeNodStoc0(pRadacina, pRadacina, NULL);
                break;
            case 7: StergArbore(pRadacina); return 0;
            default: cout << "Optiune incorecta!";
        }
    }
    StergArbore(pRadacina);
    return 0;}

```

Vom avea alte câteva exemple de proiecte de lecție și în Anexa 1, nu neapărat legate de structurile arborescente. Datorită importanței acestora însă, ar fi meritat (poate) o tratare mai extinsă și mai profundă. Putem să considerăm un lucru pozitiv totuși faptul că au fost discutate într-un capitol separat de celelalte structuri clasice.

Concluzii

Datorită trecerii a doisprezece ani de la momentul publicării, ceea ce pentru domeniul comunicațiilor și tehnologiei informației înseamnă enorm, ediția precedentă a volumului a trebuit să fie rescrisă aproape în întregime, chiar structura lui generală trebuind să fie schimbată.

Gestionarea unor evidențe simple, procesarea imaginilor, realizarea de conferințe (comunicare) online, urmărirea și reglarea unor procese industriale, simularea unor activități simple de tipul conducerii unui automobil, dirijării decolărilor/aterizărilor de (pe) un aeroport sau chiar completa robotizare a unor procese complexe reprezintă astăzi lucruri uzuale. Trebuie să avem însă în vedere și partea economică. De exemplu, crearea și utilizarea unui laborator pentru simularea conducerii unui avion sunt mult mai puțin costisitoare decât pregătirea piloților pe un avion modern real. Asta pentru a nu mai vorbi despre utilitatea existenței unui robot corespunzător pentru intervențiile microchirurgicale.

În acest sens, abordarea învățământului de informatică într-un mod exhaustiv, având în vedere toate disciplinele pe care acesta s-ar presupune că ar trebui să le conțină (și ar putea fi introduse într-o programă analitică de liceu sau facultate ori chiar în procesul de educație permanentă), nu ni se pare o idee prea fericită. Limitările și restricțiile au devenit absolut necesare din moment ce ne-am simțit obligați să spunem câteva cuvinte despre societatea informațională și e-educație în România de astăzi, chiar din introducere. Lumea în care trăim este practic dependentă (în totalitate, am putea spune) de computere și dispozitive mobile sau *wireless* de comunicare. Acest lucru a influențat într-un mod esențial conținutul învățământului, modalitățile de transmitere și receptare a cunoștințelor, organizarea și administrarea unităților de învățământ în ansamblu.

Conceptele clasice ale didacticii și metodicii predării oricărei discipline (și în special ale celor tehnice, ale informaticii în particular) s-au îmbogățit, stratificat, cristalizat în noi forme. Discutăm astfel, în primele trei capitole, despre *curriculum*, *instruire*, *evaluare*, despre *principii didactice* și *didactica formării de competențe*, precum și despre *metode*, *tehnici* și *procedee didactice* pentru învățământul preuniversitar, la nivel conceptual (în primul rând) și punând evident accent pe disciplina informatică. Capitolele 4, 5 și 6 sunt destinate tratării aspectelor aplicative ale predării-învățării în cazul noțiunilor fundamentale ale informaticii, în contextul actual : *paradigme de programare*, *tehnici de programare* și *modalități de proiectare a algoritmilor*,

algoritmi clasici, analiza complexității, corectitudinii și terminării algoritmilor și programelor, structuri (liste, stive, cozi) și baze de date, teoria grafurilor și (în particular) a arborilor.

Anexa 1 reunește câteva *proiecte de tehnologie didactică* suplimentare, necesare pentru o înțelegere mai profundă a semnificației practice a folosirii conceptelor meto-dico-didactice specifice în pedagogia învățării. Separat, în Anexa 2, am grupat câteva subiecte date în ultimii ani la bacalaureat, la olimpiadele școlare de profil, la admiterea la Facultatea de Informatică (Universitatea „Alexandru Ioan Cuza”, Iași), precum și la unele concursuri de promovare pentru profesorii din învățământul pre-universitar. În Anexa 3, deoarece accesul rapid la anumite informații de bază este determinant pentru îndeplinirea rapidă a unor sarcini și solicitări, am indicat numeroase site-uri și portaluri de Internet la care nu se face referire în mod explicit în lucrare, dar au o mare importanță și adresabilitate.

Bibliografia doar pare vastă, lucrările nefiind, într-adevăr, citate în totalitate pe parcurs. Suntem însă convinși că în text au fost transpuse măcar unele idei importante din fiecare. Mai mult, pentru că domeniul tratat este atât de larg, important și dinamic, trebuie să ne cerem scuze tuturor celor care n-au fost deloc citați, deși contribuția lor ar fi putut fi etichetată drept fundamentală.

Autorii și-au propus astfel să sublinieze existența unei legături coerente și profunde între informatica trecutului, cea a prezentului și cea a viitorului, între învățământ și societate, între transmițători și receptori, între computere și umanitate.

Avem speranța că demersul nostru va fi considerat benefic și că am acoperit o zonă de interes maxim în ceea ce privește pregătirea profesorilor de informatică din învățământul preuniversitar. Dar e posibil ca o nouă ediție revăzută și adăugită să fie necesară mult mai repede decât s-ar putea crede.

ANEXE

Anexa 1

Proiecte de tehnologie didactică

Această anexă este rezervată prezentării câtorva proiecte de lecție, prin care se dorește ilustrarea unor domenii și a problematicii de specialitate importante la nivel teoretic : paradigme de programare, recursivitate, metode de proiectare a algoritmilor, analiza algoritmilor, parcurgerea grafurilor, metode de sortare, lucrul cu diverse structuri și baze de date. Din punct de vedere practic (în afara implementării unor rezolvări/algoritmi pentru probleme de tipul menționat), am intenționat să prezentăm și câteva aplicații de laborator ce țin de tehnologia informației și a comunicațiilor sau de învățarea preliminară a unor limbaje noi, posibil derivate din cele deja presupuse a fi cunoscute (cum ar fi C#). Revenind la didactică, am avut de ales între lecții de predare-învățare, de recapitulare și aprofundare de cunoștințe, de evaluare etc., încercând să exemplificăm folosirea mai multor metode, tehnici și procedee didactice (punctând competențele urmărite).

Aceste proiecte de tehnologie didactică pot fi folosite ca model în vederea realizării altor proiecte. De asemenea, pot fi îmbunătățite și/sau modificate conform specificului clasei de elevi.

Menționăm că în capitolele 4, 6 (mai ales) și 5 sunt redate rezolvările unor probleme tot sub forma unor asemenea proiecte didactice. Unii algoritmi sunt prezentați în Pascal (deși nu am dat explicit detalii asupra acestui limbaj), considerând că cititorii sunt avizați.

Proiect de tehnologie didactică

Limbaajul C/C++ : Funcții în C/C++

Disciplina : –

Clasa : –

Data : –

Profesor : –

Unitatea de învățare : Funcții în C/C++

Detalii de conținut : Prototipul unei funcții care nu returnează valori și nu are parametri, respectiv care are parametri și returnează valori

Tipul lecției : Predare de noi cunoștințe

Se are în vedere realizarea deprinderilor de a lucra eficient, modularizat, productiv, utilizând descompunerea unei probleme date în subprobleme a căror rezolvare este mai simplă.

La sfârșitul lecției, elevii vor fi capabili :

- să declare o funcție ;
 - să recunoască componentele principale ale unei funcții (numele ei, valoarea returnată, argumente) sau, altfel spus, prototipul unei funcții ;
 - să descrie modul de folosire a argumentelor unei funcții (prin valoare, prin referință) ;
 - să deprindă folosirea valorilor returnate de către o funcție.

Metode și procedee didactice : Conversația frontală și individuală, explicația, expunerea etc.

Mijloace didactice : Manuale, culegeri de probleme. Material bibliografic suplimentar : D. Hrinciuc Logofătu, C++. *Probleme rezolvate și algoritmi*, Editura Polirom, Iași, 2001 ; L. Negrescu, *Limbajul C*, Editura Computer Libris Agora, Cluj-Napoca, 1997 ; B. Stroustrup, *The C++ Programming Language*, ed. a II-a, AT&T Bell Telephone Laboratories, 1991.

Momentele lecției : Presupunem că organizarea clasei durează 60s și verificăm și alte lucruri legate de mediu. Importantă este însă enunțarea scopului lecției. Astfel, aici putem spune că în cadrul lecției care urmează vor fi expuse următoarele probleme legate de funcții în C/C++ : declararea funcțiilor ; definirea lor ; utilizarea argumentelor funcției și a valorii sale de retur :

- Reactualizarea cunoștințelor dobândite anterior.
- Se pot pune întrebări, cum ar fi : Ce înseamnă `#include <string.h>` ? Care este rolul directivei `#include` ? Unde și de ce am folosit-o ? Ce informații poate conține un fișier antet (în mod uzual are *extensia* `.h`) ?

Funcțiile care vor fi prezentate pot fi identificate pornind de la necesitatea evidentă că într-un program C/C++ apelăm la funcții pentru a realiza anumite prelucrări într-un mod repetat și mai simplificat. De exemplu, afișarea unui text pe ecran (`stdout`) se poate face apelând funcția `printf (...)`, citirea unui caracter de la tastatură (`stdin`) se face apelând funcția `scanf ()` etc. Definirea unei funcții constituie modalitatea de a specifica felul în care trebuie executată o anumită operație. O funcție nu poate fi apelată atât timp cât nu este declarată.

Declararea funcției realizează următoarele lucruri : atribuie un nume funcției ; fixează tipul valorii returnate de către funcție (dacă există) ; fixează numărul și tipul (argumentelor) parametrilor funcției, parametri care vor fi furnizați în momentul apelului acestei funcții. La declararea și definirea funcției, aceștia se numesc *parametri formali* (*argumente formale*). De exemplu,

```
void StergeEcran(int, int, int, int);
int AfiseazaFereastră (int , dword);
char* ObțineNumereFereastră (int IdentificatorFereastră);
```

În cadrul declarării funcției putem folosi nume pentru argumente. Acest lucru îi dă utilizatorului posibilitatea să înțeleagă semnificația parametrului respectiv și valorile posibile care trebuie folosite. Compilatorul nu ia în considerare aceste nume, ci numai tipul parametrilor și ordinea lor.

Definirea funcției. O definire de funcție înseamnă o declarare de funcție în care este prezentat și corpul funcției (codul). Orice funcție folosită într-un program trebuie să fie definită undeva în cadrul programului, dar numai o singură dată. De exemplu :

```
extern double precizie ();
double precizie ()
{
    double epsilon = 1.0;
    while (1.0 + epsilon > 1.0 )
        epsilon = epsilon / 2.0;    // epsilon /= 2.0;
    return 2.0 * epsilon;
}
extern int maxim (int, int);
int maxim (int n1, int n2)
    return n1 < n2 ? n2: n1;
}
```

Putem observa că definirea funcțiilor cu parametri care nu sunt folosiți nu este recomandabilă. Trebuie totuși știut că există funcții care au parametri ce nu sunt utilizați, aceștia rezultând din planificările efectuate înainte pentru dezvoltarea viitoare a funcției (codului).

Transmiterea valorii argumentelor (parameter passing). În momentul apelului unei funcții, fiecare argument formal este inițializat cu argumentul actual corespunzător. În această fază se fac verificări asupra tipurilor de parametri și toate tipurile de conversii standard și cele definite de utilizator sunt realizate. Există o regulă specială pentru transmiterea tablourilor, o facilitare pentru transmiterea argumentelor fără verificare și o facilitare pentru specificarea argumentelor implicite. De exemplu :

```
void f (int valoare, int& referinta)
{   valoare++;
    referinta++;
}
```

Când funcția *f* este apelată, *valoare++* *incrementează o copie locală* a primului argument actual, în timp ce *referinta++* *incrementează al doilea argument formal*. Trebuie sesizată diferența. Un apel al acestei funcții este dat în cele ce urmează.

```
void main()
{
    int _valoare = 10;
    int _referinta = 100;

    printf ("\nInainte de apel functie f\n _valoare = %d \n _referinta = %d", _valoare, _referinta);
    f (_valoare, &_referinta);
    printf ("\nDupa apel functie f \n _valoare = %d \n _referinta = %d", _valoare, _referinta);
}
```

Rezultatul execuției este : înainte de apelul funcției *f* vom avea *_valoare = 10* și *_referinta = 100*, iar după apelul funcției *f* vom găsi *_valoare = 10* și *_referinta = 101*. Primul argument este dat *prin valoare*, iar al doilea *prin referință*. Deci primul parametru pentru *f* se comportă ca o *variabilă locală*. Deși nu este indicată transmiterea parametrilor prin referință, acest lucru este absolut necesar atunci când se transferă obiecte *mari* prin argumentele unei funcții. Pentru a indica faptul că obiectele transmise prin referință nu trebuie modificate de către funcție, argumentul trebuie declarat *const*. De asemenea, declarând un argument de tip pointer ca fiind *const*, înseamnă că obiectul punctat de acest pointer nu poate fi schimbat de funcție.

Obținerea valorii de retur. O funcție care nu este declarată ca având tipul *void* trebuie să returneze o valoare. Valoarea returnată este specificată de instrucțiunea *return*. Într-o funcție pot exista mai multe instrucțiuni *return*. De exemplu :

```
int f () { }
```

va genera o eroare la compilare, căci nu există o instrucțiune *return* care să returneze o valoare (de tip *int*), în timp ce

```
void g () { }
```

va fi considerată corectă, netrebuind să returneze nimic.

Observație

O instrucțiune *return* inițializează o variabilă de tipul returnat. Tipul valorii returnate de o funcție trebuie să corespundă cu tipul funcției – în caz contrar, se vor efectua conversiile standard și cele definite de utilizator asupra acestei valori.

Important

La fiecare apel al funcției se realizează o copie a argumentelor sale și a variabilelor de tip *automatic*. Memoria utilizată pentru aceasta este refolosită după ce funcția își termină execuția. Din această cauză este interzisă returnarea unui pointer la o variabilă locală.

Fixarea cunoștințelor : Prin întrebări simple, clasa poate fi condusă spre a sesiza momentele principale ale lecției referitoare la funcții în C/C++ : declarare, definire, transmiterea argumentelor, folosirea practică a funcțiilor, obținerea corespunzătoare a valorilor de retur.

Activitatea (suplimentară a) profesorului : se va insista asupra faptului că ceea ce poate conține un fișier antet nu este o cerință a limbajului de programare, ci mai degrabă o convenție pentru a separa anumite declarații (de tip, de funcții etc.) de codul-sursă propriu-zis.

Activitatea elevilor : ei trebuie să furnizeze la momentul oportun răspunsuri corecte, cum ar fi cele menționate mai sus și cele descrise în continuare. Mecanismul definit de directiva de compilare *#include* se referă la facilitatea de gestionare a fișierelor ce conțin un cod-sursă de a uni fragmente de program într-o singură unitate (fișier) pentru compilare. Directiva *#include*, a cărei sintaxă este

```
#include "de_inclus" sau #include <de_inclus>
```

înlocuiește linia de program unde apare cu conținutul fișierului *de_inclus*.

Tema pentru acasă : Realizați funcții care să satisfacă următoarele cerințe (nu simultan) :

- Returnează elementul minim dintre cele trei argumente de același tip ale funcției.
 - Inițializează elementele unei structuri (argumentul va fi transmis prin referință, dar nu *const*), nu are valoare de retur.
-

Observație

Această lecție va trebui să fie completată cu o lecție de laborator.

Proiect de tehnologie didactică

Liste simplu înlănțuite, alocate dinamic

Disciplina : –

Clasa : –

Profesor : –

Data : –

Unitatea de învățare : Structuri de date de tip listă simplu înlănțuită, alocată dinamic

Detalii de conținut : Abordarea lecției presupune cunoașterea de către elevi a alocării dinamice a variabilelor, alocarea și eliberarea efectivă a zonelor de memorie aferente, lucrul cu tipul pointer și tipul referință, deosebirea dintre variabilele de tip pointer și de tip referință

Tipul lecției : Mixtă (verificare și comunicare de cunoștințe)

Competențe acționale

La sfârșitul acestei lecții, elevii vor fi capabili :

- să poată defini o structură de date de tip listă simplu înlănțuită ;
- să cunoască și să folosească în aplicații principalele operații care se pot efectua cu nodurile (elementele) unei liste simplu înlănțuite ;
- să identifice situațiile în care este recomandabilă utilizarea acestei structuri de date ;
- să folosească în mod optim în aplicații liste simplu înlănțuite.

Metode și procedee didactice : Conversația frontală și individuală, explicația, metoda analitică, munca independentă, învățarea prin descoperire, problematizarea etc.

Mijloace didactice : Manualul, culegeri de probleme, tabla, calculatorul, retro(video)proiectorul

Momentele lecției :

- Momentul organizatoric : maximum două minute, obiectivul urmărit fiind, desigur, crearea unei atmosfere specifice bunei desfășurări a activității didactice care urmează.
 - Verificarea temei pentru acasă și a cunoștințelor dobândite anterior : cel mult 8 minute (aici se poate cumva detalia activitatea concretă a profesorului și ceea ce se așteaptă de la elevi).
 - Captarea atenției, prezentarea titlului și a obiectivelor lecției noi : maximum două minute.
- Activitatea profesorului* poate fi detaliată sub forma :
- a) Se prezintă o *situație-problemă* prin care se urmărește scoaterea în evidență a importanței utilizării unei structuri de date tip listă simplu înlănțuită, insistându-se asupra avantajelor pe care le prezintă această structură în condițiile impuse de problemă.
 - b) Se subliniază gama largă de probleme (gestionarea documentelor și reprezentărilor/vizualizărilor acestora în cadrul unei aplicații concrete, precum și a spațiului de memorie liber și/sau alocat explicit etc.) care vor putea fi soluționate elegant folosind structura de date tip listă.

Prezentarea noilor conținuturi : Durata estimată poate fi de 12 minute. Profesorul poate prezenta succesiv :

- *Definirea nodului* – a se revedea capitolul 5.
- *Operațiile* care se pot face cu nodurile unei liste simplu înlănțuite : *adăugarea* de noduri la sfârșitul unei liste (se scoate în evidență cazul când lista este vidă și cazul când mai există alte noduri în listă) ; *modificarea* unui nod identificat prin informația de nod (se scoate în evidență cazul când nodul de modificat nu a fost găsit sau lista este vidă) ; *ștergerea* unui nod identificat prin informația de nod (se scoate în evidență cazul când nodul șters este primul sau ultimul din listă și cazul când acesta nu a fost găsit sau lista este vidă) ; *inserarea* unui nod în listă, înainte sau după un nod reper (se va scoate în evidență cazul când inserarea se va realiza înaintea primului nod sau după ultimul nod și cazul când nodul reper nu a fost găsit sau lista este vidă) ; *parcursarea* (afișarea conținutului) unei liste simplu înlănțuite (se va scoate în evidență cazul când lista este vidă).
- *Modul de recunoaștere a anumitor instanțe* care ușurează implementarea (păstrarea sfârșitului listei pentru a facilita adăugirile, reținerea nodului „urmă” pentru a înlesni ștergerile și inserările înaintea unui nod specificat etc.).

Observație

Prezentarea operațiilor cu nodurile unei liste simplu înlănțuite va fi însoțită de scheme și desene sugestive. Se pot introduce și aici lucruri concrete privind activitatea elevului sau (alte) metode și procedee didactice.

Asigurarea conexiunii inverse (feedback): Durata estimată este de 4 minute. Profesorul poate solicita elevilor prezentarea unor situații-problemă similare cu cele menționate anterior și pentru rezolvarea cărora se poate folosi o listă simplu înlănțuită. Se va cere și să se argumenteze exemplele date. Elevii vor identifica asemenea probleme și vor propune soluții în care se va folosi structura de date învățată, prezentându-se avantajele unei asemenea implementări.

Prezentarea temei pentru acasă: Durata estimată este de două minute. Din nou, putem prezenta totul sub forma activitate profesor/activitate elev. Astfel, activitatea profesorului poate presupune realizarea unei implementări nerecursive a funcției lui Ackermann folosind o listă simplu înlănțuită (problema a fost rezolvată folosind alocarea statică a variabilelor necesare implementării). Activitatea elevului ar putea fi descrisă în felul următor: notează problema propusă spre rezolvare și *indicațiile* de implementare ale profesorului și cere eventuale lămuriri suplimentare.

Proiect de tehnologie didactică

Tipul record

Disciplina: –

Clasa: –

Data: –

Profesor: –

Unitatea de învățare: Tipuri structurate. Tipul înregistrare (Record)

Detalii de conținut: Identificarea unor noi tipuri de date; proiectarea aplicațiilor pentru rezolvarea unor probleme utilizând instrumente specifice de prelucrare a datelor

Tipul lecției: Comunicare de noi cunoștințe

Timpul acordat: 50 de minute

Obiectiv fundamental: Însușirea cunoștințelor cu privire la crearea propriului tip de dată și a funcțiilor necesare să lucreze cu acesta

Competențe acționale:

- Dezvoltarea capacității de gândire independentă și a capacității de comunicare, folosind limbajul de programare studiat.
- Gruparea datelor de tipuri diferite în structuri cu organizare omogenă.
- Stăpânirea modalităților de acces la componentele unei înregistrări și efectuarea diferitelor operații cu acestea.

Competențe specifice cu efecte asupra personalității:

- Afective: cultivarea satisfacției de a răspunde corect la problemele utilizate la lecție.
- Psihomotorii: viteză și precizie în exprimarea orală prin utilizarea limbajului de specialitate.

Metode și procedee didactice: Conversația, explicația, argumentarea unor enunțuri, problematizarea, demonstrația, rezolvarea prin algoritmizare.

Mijloace didactice : Manualul, tabla, creta, video(retro)proiectorul.

Tipuri de activitate : Frontală, pentru realizarea feedbackului; individuală, pentru ca elevii să poată fi permanent supravegheați și îndrumați de către profesor în vederea corectării erorilor de exprimare sau de gândire și a bunei înțelegeri a noilor cunoștințe transmise.

Material bibliografic suplimentar : T. Sorin, *Informatică pentru clasa a IX-a*, Editura L&S Infomat, București, 2001 ; Violeta Hamciuc, *Caiet de laborator pentru clasa a IX-a. Teorie și aplicații*, Editura Dacia Educațional, Cluj-Napoca, 2001 ; Mihaela Morar, Corina Mocanu, Garofița Dițu, *Turbo Pascal. Exerciții și probleme*, Liceul de Informatică, Suceava, 1999.

Momentele lecției :

- Organizare (un minut) : se trec în catalog elevii absenți, se stabilește liniștea și se asigură atmosfera adecvată activității didactice.

Activitatea profesorului (45 de minute) : se actualizează noțiunile studiate în lecțiile anterioare, comunicându-se elevilor enunțul unei probleme. De exemplu :

La examenul de capacitate s-au înscris n elevi. Pentru fiecare elev înscris trebuie memorate următoarele informații :

- numele elevului;
- nota la examenul de matematică;
- nota la examenul de română;
- media la examen;
- mesajul „Admis” sau „Respins”.

Se cere să se afișeze lista elevilor admiși la examen. Un elev este considerat „Admis” dacă media examenului de capacitate este mai mare sau egală cu cinci.

Profesorul poartă un dialog cu clasa despre modul de rezolvare a problemei propuse și despre modalitatea în care vor fi memorate datele. Elevii vor propune (probabil) memorarea informațiilor în cinci vectori, câte unul pentru fiecare tip de informație. Un elev va ieși la tablă pentru a construi un exemplu pentru $n = 5$ elevi. Profesorul va accentua faptul că, pentru a afla datele elevului 3, trebuie selectate elemente din cinci vectori : $a[3]$, $b[3]$, $c[3]$, $d[3]$, $e[3]$, deci operațiile de prelucrare a datelor memorate în acest mod sunt *difficil* de realizat. Le comunică elevilor că în limbajul Pascal există tipul înregistrare (Record) care permite lucrul cu date de tipuri diferite. Profesorul scrie pe tablă titlul noii lecții : „Tipul înregistrare (Record)” și prezintă elevilor definiția tipului : tipul înregistrare (Record) este un tip de date structurat, care poate conține un număr fix sau variabil de componente, de același tip sau de tipuri diferite. Componentele unei înregistrări sunt numite *câmpuri* sau *articole*. În funcție de numărul de articole, tipul înregistrare se clasifică astfel : *înregistrare fixă* – conține un număr fix de componente de tipuri diferite ; *înregistrare cu variante* – conține un număr variabil de componente de același tip sau de tipuri diferite.

În continuare, profesorul va scrie pe tablă modul general de definire a tipului Record și va explica elevilor particularitățile care apar la definire :

```
Type nume_tip = RECORD
    camp11,camp12,..., camp1i: tip1;
    camp21,camp22,..., camp2j: tip2;
    .....
    campn1,campn2,..., campnz: tipn;
END;
```

Împreună cu elevii, profesorul va defini tipul *elev*, care va conține datele din problema propusă spre rezolvare la începutul orei și va grupa cei *n* elevi într-un tablou unidimensional cu elemente de tip înregistrare.

```
Type elev = RECORD
    nume: string[20];
    notam: 1..10;
    notar: 1..10;
    media: real;
    obs: string[10];
END;
Vector = array[1..30] of elev;
Var a: vector;
    n: byte;
```

Activitatea elevilor (în aceeași perioadă de timp): Elevii vor fi atenți la întrebările profesorului și vor formula răspunsuri la acestea; vor fi antrenați în scrierea la tablă a formei tabloului unidimensional.

Putem *reveni* și spune că activitatea profesorului poate continua cu prezentarea operațiilor permise datelor de tip înregistrare.

Modalitatea de acces la componentele unui Record :

- folosind caracterul punct: *nume_variabilă.nume_câmp*;
- folosind instrucțiunea *WITH*:

```
with nume_variabila do
    begin
        prelucrare campuri inregistrare
    end;
```

- atribuirea : operație permisă între două variabile definite de tip înregistrare dacă și numai dacă au același tip.

În continuare, profesorul va prezenta rezolvarea problemei propuse exemplificând și punctând de fiecare dată noile cunoștințe, cum ar fi citirea datelor de intrare (nume, notam, notar), completarea prin program a câmpurilor *media* și *obs* sau afișarea rezultatului. Revenind, putem spune că este de dorit ca elevii să participe activ la oră, răspunzând la întrebările formulate de profesor, să noteze în caiete ceea ce nu știu și să fie atenți la explicațiile profesorului. Elevii vor fi antrenați și în scrierea codului programului, profesorul purtând un permanent dialog cu clasa.

Realizare feedback (două minute) : Profesorul confirmă, apreciază răspunsurile corecte și, dacă este cazul, intervine cu explicații suplimentare, în timp ce elevii rețin aprecierile și explicațiile suplimentare și fac anumite comentarii.

Ultimele trei proiecte de tehnologie didactică atașate sunt standardizate în întregime conform ultimelor cerințe educaționale. Sugerăm cititorului să aducă și proiectele pe care le-am prezentat anterior la această formă.

Proiect de tehnologie didactică

Școala :

Disciplina : Informatică

Clasa : a X-a

Profilul : Matematică-informatică, intensiv informatică

Data : –

Profesor : –

Unitatea de învățare : Tehnici de implementare a algoritmilor : subprograme.

Tema lecției : Subprograme utilizator : definiție, apel, mecanism de transfer a parametrilor.

Tipul lecției : Predare-învățare (transmitere de noi cunoștințe)

Durata : 50 de minute

Competențe generale :

(CG2) Identificarea datelor care intervin într-o problemă și a relațiilor dintre acestea.

(CG3) Elaborarea algoritmilor de rezolvare a problemelor.

(CG4) Aplicarea algoritmilor fundamentali în prelucrarea datelor.

(CG5) Implementarea algoritmilor într-un limbaj de programare.

Competențe specifice :

(CS2.1) Utilizarea corectă a subprogramelor predefinite și a celor definite de utilizator.

(CS2.2) Construirea unor subprograme pentru rezolvarea subproblemelor unei probleme.

(CS3.2) Recunoașterea situațiilor în care este necesară utilizarea unor subprograme.

(CS3.3) Analiza problemei în scopul identificării subproblemelor acesteia.

(CS5.1) Elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specializării.

Competențe derivate :

La sfârșitul activității didactice elevii vor fi capabili :

(CD1) Să analizeze problema propusă și să identifice subproblemele în care poate fi descompusă.

(CD2) Să definească subprograme utilizator corespunzătoare algoritmilor de rezolvare a fiecărei subprobleme.

(CD3) Să justifice alegerea modului de transfer al parametrilor.

(CD4) Să apeleze adecvat subprogramele realizate în vederea rezolvării problemei.

(CD5) Să implementeze secvențe de cod C++ pentru rezolvarea cerințelor problemelor propuse.

C1. Competențe cognitive

La sfârșitul lecției, elevii vor fi capabili :

(C1.1) : să analizeze fiecare problemă și să identifice subproblemele în care poate fi descompusă ;

(C1.2) : să conceapă algoritmul de rezolvare pentru fiecare subproblemă a aplicației propuse ;

(C1.3) : să identifice tipurile de parametri adecvați și modalitatea lor de transfer.

C2. Competențe afective

La sfârșitul lecției, elevii vor fi capabili :

(C2.1) : să conștientizeze necesitatea modularizării soluției prin folosirea subprogramelor ;

(C2.2) : să dovedească curiozitate și interes pentru noțiunile prezentate.

C3. Competențe atitudinale/comportamentale

La sfârșitul lecției, elevii vor fi capabili :

(C3.1): să explice importanța dezvoltării modulare a aplicațiilor complexe, cu ajutorul subprogramelor ;

(C3.2): să argumenteze alegerea modului de transfer al parametrilor.

C4. Competențe acționale

La sfârșitul lecției, elevii vor fi capabili :

(C4.1): să utilizeze corect în aplicații noțiunile teoretice însușite ;

(C4.2): să implementeze corect în C++ subprogramele discutate.

Strategii didactice

Principii didactice :

- principiul însușirii conștiente și active a cunoștințelor ;
- principiul accesibilității ;
- principiul legării teoriei de practică.

Metode și procedee didactice : problematizarea (P), algoritmizarea (A), conversația euristică (Cv), explicația (E), demonstrație (D).

Forme de organizare : lucrul frontal.

Forme de dirijare a învățării :

- independentă ;
- dirijată de profesor prin mijloacele de învățare.

Metode de evaluare :

- evaluare continuă pe parcursul lecției ;
- apreciere verbală.

Resurse materiale :

- culegeri auxiliare ;
- tablă ;
- proiector.

Referințe bibliografice : [CȘ], [K1], [CLR].

Structura lecției pe secvențe de instruire

CS/CD	Min.	Etapele lecției – activitatea elev-profesor	MD (metode didactice)
	2'	Moment organizatoric <i>Profesorul</i> : verifică frecvența elevilor, verifică existența resurselor materiale <i>Elevii</i> : se pregătesc pentru oră – deschid caietele	
	3'	Captarea atenției <i>Profesorul</i> : anunță tema lecției, competențele pe care urmărește să le formeze și explică modul de desfășurare a orei. <i>Elevii</i> : răspund solicitărilor profesorului, cer lămuriri.	
	40'	Desfășurarea lecției Dirijarea învățării <i>Profesorul</i> le propune elevilor spre analiză și rezolvare o problemă care să îi ajute să conștientizeze necesitatea descompunerii acestora în subprobleme și importanța rezolvării modulare. <i>Problema</i> : Date două numere naturale, nenule, a și b ($0 < a, b < 105$), să se decidă dacă sunt <i>numere prime gemene</i> sau <i>numere prietene</i> . Definiție : 1. a și b sunt <i>prime gemene</i> dacă și numai dacă sunt consecutive impare și prime (exemplu : 5, 7, 17, 19 etc.); 2. a și b sunt <i>prietene</i> dacă și numai dacă $b = 1 +$ (suma divizorilor proprii ai lui a) și $a = 1 +$ (suma divizorilor proprii ai lui b) (exemplu : 220, 284). <i>Elevii</i> : analizează problema, identifică secvențele de operații care se repetă pentru date diferite : testul de primalitate și suma divizorilor unui număr și conștientizează avantajul de a le rezolva modular. Profesorul va pune întrebări, stimulând prin conversație participarea activă a elevilor la oră. Întrebări posibile : <ul style="list-style-type: none"> • Ce prelucrări se repetă în algoritmul de rezolvare ? • Prin ce date diferă fiecare prelucrare care se repetă ? • Ar fi util să definim o singură dată o prelucrare, de exemplu testul de primalitate, apoi să o solicităm să se execute de oricâte ori am avea nevoie ? • Ce avantaje ar aduce această modularizare ? 	P
CD1			
CI.1.1.			
CD2			

<p>CD3</p>	<p>D</p> <p>Dacă rezultatul funcției NU este void, ea trebuie să conțină instrucțiuni de tip <code>return expresie;</code> prin care se va returna o valoare de tipul precizat.</p> <p>Exemplu.</p> <pre>int test(int i, int k, float j) /*antet corect de funcție*/ int test(int i, k; float j) /*antet greșit – lista incorectă de parametri formali*/ ATENȚIE!</pre> <p>NU se poate defini o funcție în interiorul altei funcții! NU se poate defini o funcție într-o altă funcție, deci toate funcțiile au aceeași sferă de influență.</p> <p><i>Apelul unei funcții</i></p> <pre>nume_funcție(lista_parametrilor_actuali)</pre>	<p>E, Cv</p>
------------	--	--------------

CD4 C1.2. C3.2.	<p>Exemplu :</p> <p>Funcție care interschimbă valorile a două variabile transmise ca parametri, valorile modificate ale acestora fiind furnizate tot prin a și b.</p> <pre>void schimba(int &a,int &b) { int aux=a; a=b; b=aux; } ... int a=5, b=2; schimba(a,b); cout<<a<<' '<<b; /* 2 5 */</pre>	<pre>void schimba(int a,int b) { int aux=a; a=b; b=aux; } ... int a=5, b=2; schimba(a,b); cout<<a<<' '<<b; /* 5 2 */</pre>	D
CD5 C3.1. C4.1.	<p>Elevii notează în caiete, cer lămuriri, răspund întrebărilor profesorului.</p> <p><i>Elevii</i> diferiți vor scrie la tablă secvențe de cod C++ care implementează algoritmi discutați pentru rezolvarea subproblemelor problemei propuse.</p> <p>Asigurarea transferului de cunoștințe</p> <p>Clasei i se cere să urmărească, să corecteze sau să îmbunătățească variantele scrise.</p> <pre>bool prim(int n) { if (n<2) return false; for (int d=2; d*d<=n; ++d) if (n%d==0) return false; return true; }</pre>		A, M
C4.2.	<pre>int sum_div(int n) { int s=0, d=1; for (d=1; d*d<n; d++) if (n%d==0) s=s+d+n/d; if (n%d==0) s=s+d; return s; }</pre>		

	<pre> ... int a, b; cin>>a>>b; if (a%2 && b%2 && prim(a) && prim(b)) cout<<"prime gemene"; else if (sum_div(a)==b && sum_div(b)==a) cout<<"prietene"; else cout<<"nu"; </pre> <p><i>Profesorul încurajează conversația, dezvoltarea limbajului de specialitate prin integrarea noilor noțiuni, acordă feedback răspunsurilor elevilor, intervenind cu explicații suplimentare acolo unde observă nelămuriri sau erori de logică ori de exprimare.</i></p>	
	<p>Evaluarea</p> <p>Are loc pe parcurs, prin observarea sistematică a reacțiilor elevilor și a muncii lor independente, prin aprobare, dezaprobare verbală în urma răspunsurilor date de elevi la întrebări.</p>	
3'	<p>Asigurarea retenției</p> <p>Profesorul pune câteva întrebări din lecția nouă :</p> <ul style="list-style-type: none"> • Ce este o funcție ? • Enumerați trei avantaje ale utilizării funcțiilor. • Care este diferența dintre parametrii formali și actuali ? • Care este diferența dintre parametrii transmiși prin valoare și cei transmiși prin referință ? 	Cv
2'	<p>Tema pentru acasă</p> <p>Profesorul propune tema pentru acasă, 1-3 probleme și/sau 5-10 exerciții de tip grilă din manual, asemănătoare cu ce s-a discutat în clasă, iar elevii notează în caiete.</p>	

Proiect de tehnologie didactică

Școala : –

Disciplina : Informatică

Clasa : a IX-a

Profilul : Matematică-informatică, intensiv informatică

Data : –

Profesor : –

Unitatea de învățare : Elaborarea algoritmilor de rezolvare a problemelor și implementarea lor într-un limbaj de programare

Tema lecției : Structura repetitivă condiționată anterior

Tipul lecției : Lecție de fixare și formare de deprinderi și priceperi

Durata : 100 de minute

Competențe generale :

(CG1) Identificarea datelor care intervin într-o problemă.

(CG3) Elaborarea algoritmilor de rezolvare a problemelor.

(CG4) Aplicarea algoritmilor fundamentali de prelucrare a datelor.

(CG5) Implementarea algoritmilor într-un limbaj de programare.

Competențe specifice :

(CS3.1) Analizarea enunțului unei probleme și stabilirea pașilor de rezolvare a problemei.

(CS3.2) Reprezentarea algoritmilor în pseudocod.

(CS3.3) Respectarea principiilor programării structurate în procesul de elaborare a algoritmilor.

(CS5.1) Elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specializării.

(CS5.2) Alegerea unui algoritm eficient de rezolvare a unei probleme.

Competențe derivate :

La sfârșitul activității didactice, elevii vor fi capabili să :

(CD1) Identifice necesitatea utilizării structurii repetitive condiționate anterior în algoritmul fundamental pentru descompunere în factori primi.

(CD2) Reprezinte în pseudocod algoritmul propus folosind structura *cat timp... execută*.

(CD3) Să argumenteze eficiența algoritmului propus.

(CD4) Să recunoască forme de aplicare a algoritmilor fundamentali discutați în diverse probleme.

C1. Competențe cognitive

La sfârșitul lecției, elevii vor fi capabili :

(C1.1) : să analizeze o problemă, să descrie etapele algoritmului de rezolvare ;

(C1.2) : să identifice tipul structurii repetitive adecvate ;

(C1.3) : să scrie în pseudocod algoritmi eficienți pentru rezolvarea cerințelor propriu zise.

C2. Competențe afective

(C2.1) : să argumenteze corect alegerea structurilor de control și eficiența algoritmului propus ;

(C2.2) : să se autoevalueze corect ;

(C2.3) : să dovedească curiozitate și interes pentru noțiunile utilizate.

C3. Competențe psihomotorii

(C3.1): să utilizeze corect în problemele propuse algoritmul fundamental de descompunere în factori primi.

C4. Competențe acționale

(C4.1): să identifice structura de control *cat timp... executa*;

(C4.2): să descrie modul de utilizare a algoritmului fundamental în rezolvarea problemei propuse;

(C4.3): să scrie în pseudocod algoritmul corect.

Strategii didactice

Principii didactice :

- principiul legării teoriei de practică ;
- principiul sistematizării și continuității cunoștințelor ;
- principiul accesibilității ;
- principiul individualizării și diferențierii învățării.

Metode și procedee didactice : problematizarea (P), algoritmizarea (A), conversația frontală și individuală (Cv), explicația (E), munca independentă (M), demonstrația (D).

Forme de organizare : lucrul individual și pe grupe de 2-3 elevi

Forme de dirijare a învățării :

- independentă ;
- dirijată de profesor prin mijloacele de învățare.

Metode de evaluare :

- evaluare continuă pe parcursul lecției ;
- apreciere verbală ;
- evaluare practică.

Resurse materiale :

- fișe de aplicații ;
- material bibliografic : culegere de probleme, manual pentru clasa a IX-a, [CȘ], [K1], [CLR], [L].

Structura lecției pe secvențe de instruire

CD	Min.	Etapele lecției – activitatea elev-profesor	MD
	2'	<p>Moment organizatoric <i>Profesorul</i> : verifică frecvența elevilor, verifică existența resurselor materiale. <i>Elevii</i> : se pregătesc pentru oră.</p>	
	4'	<p>Captarea atenției <i>Profesorul</i> : anunță tema lecției, competențele de format și distribuie fișele de lucru, apoi explică modul de desfășurare a orei. Fișele conțin un număr de sarcini obligatorii, iar pentru elevii cu performanțe mai bune se prevăd câteva sarcini suplimentare. <i>Vezi fișa.</i> <i>Elevii</i> : răspund solicitărilor profesorului, cer lămuriri.</p>	
CD2, C4.2.	5'	<p>Desfășurarea lecției Actualizarea cunoștințelor dobândite în lecțiile anterioare <i>Profesorul</i> solicită elevilor să descrie modul de funcționare a structurii repetitive condiționate anterior <i>cat timp... execută</i>, apoi să explice cum se aplică aceasta în cazul algoritmului fundamental de descompunere în factori primi, punând accent pe modalitățile de eficientizare a timpului de execuție. <i>Elevii</i> : răspund solicitărilor profesorului. Dacă răspunsurile așteptate nu apar în scurt timp, profesorul va numi elevi care să răspundă.</p>	Cv
	60'	<p>Dirijarea învățării Profesorul solicită elevilor să citească cerințele de pe fișa de lucru. Sunt discutate la tablă câteva exemple numerice P, Cv pentru o mai bună înțelegere a enunțului, evidențiindu-se posibilitățile de optimizare. Profesorul pune întrebări, stimulând prin conversație participarea activă a elevilor la oră. Întrebări posibile :</p> <ul style="list-style-type: none"> • Care este cel mai mic număr care are aceiași factori primi în descompunere ca și n ? • Cum putem adapta algoritmul elementar de descompunere în factori primi pentru a determina acest număr ? • Cum putem adapta algoritmul elementar de descompunere în factori primi pentru a verifica dacă numărul n este liber de pătrate (nu conține în descompunerea în factori primi niciun factor prim la putere pară) ? • Cum putem calcula numărul de divizori ai unui număr natural nenul n, folosind descompunerea în factori primi ? • Este un algoritm mai eficient decât parcurgerea divizorilor proprii posibili și numărarea celor care sunt divizori ai lui n ? • Cum calculăm numărul de cifre 0 în care s-ar termina produsul unor numere naturale citite pe rând până la citirea lui 0 ? • Putem aplica același algoritm pentru determinarea numărului de cifre 0 de la sfârșitul lui $n!$, $0 < n < 109$? <p>Dacă da, este eficient ?</p>	P, Cv

CD2.	<p>Asigurarea transferului</p> <p>Răspunsurile aşteptate ale elevilor :</p> <ul style="list-style-type: none"> - algoritmul elementar de descompunere în factori primi (cu parcurgerea optimizată a divizorilor primi posibili până la $\lfloor \sqrt{n} \rfloor$); 	E
C3.1.	<pre> d←2 cat timp d*d≤n executa p←0 cat timp n%d=0 executa n←[n/d] p←p+1 sfarsit cat timp daca p>0 atunci prelucreaza(d,p) sfarsit daca d←d+1 sfarsit cat timp daca n>1 atunci prelucreaza(n,1) sfarsit daca </pre>	A
C4.1.	<pre> sfarsit cat timp daca p>0 atunci prelucreaza(d,p) sfarsit daca d←d+1 sfarsit cat timp daca n>1 atunci prelucreaza(n,1) sfarsit daca </pre>	M
C1.1.	- cel mai mic număr natural care are aceeași factori primi în descompunere ca și n se obține făcând produsul factorilor primi distincți ai lui n , obținuți în urma descompunerii în factori primi a acestuia ;	E, Cv
C4.2.	- pentru verificarea unui număr liber de pătrate, $prelucreaza(d, p)$ se va înlocui cu verificarea dacă $p \% 2 = 0$, caz în care răspunsul este NU ;	
CD1.	- pentru numărul divizorilor lui n se poate aplica formula lui Euler bazată pe descompunerea în factori primi : dacă $n = d_1^{p_1} * d_2^{p_2} * \dots * d_k^{p_k}$, unde d_1, d_2, \dots, d_k sunt factorii primi din descompunerea lui n , iar p_1, p_2, \dots, p_k sunt puterile acestora, numărul divizorilor lui n este $(p_1+1) * (p_2+1) * \dots * (p_k+1)$. Acest algoritm are complexitatea timp $T(n) = O(\sqrt{n})$, în timp ce algoritmul care parcurge divizorii proprii posibili până la $\lfloor n/2 \rfloor$ are complexitatea $T(n) = O(n)$.	
CD3.	- pentru determinarea numărului de cifre 0 de la sfârșitul produsului numerelor naturale citite până la 0 este necesară calcularea puterii lui 2 și a puterii lui 5 în numerele date, apoi afișarea minimului acestor două puteri, deoarece un 0 de la sfârșitul produsului înseamnă o înmulțire cu $2 * 5$.	
C2.1.		
20'		

CD4.		<p>– pentru determinarea numărului de cifre 0 de la sfârșitul lui $n!$ nu se va aplica același algoritm deoarece se observă că în $n!$ este relevantă doar puterea lui 5, care se calculează cu formula $\frac{n}{5} + \frac{n}{5^2} + \frac{n}{5^3} + \dots + \frac{n}{5^k}$, unde 5^k este cea mai mare putere a lui 5, mai mică sau egală cu n.</p>	D E
C2.2. C1.3.	5'	<p>Asigurarea intensificării retenției</p> <p>Vor veni la tablă elevii din fiecare echipă, care să exemplifice funcționalitatea fiecărui algoritm propus pe date concrete, pentru o mai bună înțelegere a enunțurilor. Apoi, un alt elev din aceeași echipă va rezolva la tablă și va explica algoritmul propus, după care va argumenta eficiența acestuia.</p> <p>Elevii notează în caiete algoritmi prezenți, exemplele numerice și argumentele pentru eficiența acestora.</p> <p><i>Profesorul</i> monitorizează activitatea individuală a elevilor, le acordă permanent suport. Dacă au fost identificate greșeli repetate la mai mulți elevi, se întrerupe lecția pentru o discuție cu toată clasa și lămurirea neclarităților.</p> <p><i>Profesorul</i> încurajează elevii cu performanțe superioare să rezolve cerințele suplimentare, insistând asupra eficienței timp a algoritmilor și asupra cazurilor particulare.</p> <p>Evaluarea</p> <ul style="list-style-type: none"> – pe parcurs, prin aprobare, dezaprobare verbală în urma răspunsurilor date de elevi la întrebări sau în urma observării sistematice a munii independente; – la începutul orei, profesorul negociază cu clasa câteva criterii de evaluare a algoritmilor propuși pentru rezolvarea problemelor din fișa de lucru (corectitudinea algoritmului propus, eficiența, tratarea cazurilor particulare, modul de redactare a algoritmului în pseudocod); – fiecare echipă va prezenta la tablă una dintre problemele rezolvate, astfel încât în final să fie prezentate toate problemele propuse pe fișa de lucru; – evaluare reciprocă (sunt încurajate echipele care au rezolvat aceeași problemă cu cea prezentată la tablă să evalueze soluția propusă de colegii lor); – la finalul orei, cu note argumentate de rezultatele aplicației realizate. 	
C2.2.			
	5'	<p>Aprecieria activității</p> <p>Profesorul face aprecieri privind performanțele elevilor, recomandări de recuperare celor care nu au reușit să rezolve sarcinile obligatorii și îi notează pe cei care au fost activi.</p> <p>Profesorul poate cere elevilor să se autoevalueze.</p>	
	4'	<p>Tema pentru acasă</p> <p>Profesorul propune tema pentru acasă, iar elevii notează în caiete.</p> <p>Tema pentru acasă ar putea fi una sau două probleme asemănătoare cu cele de pe fișa de lucru și, pentru cei capabili de performanțe superioare, probleme date la olimpiade și concursuri de informatică care utilizează acest algoritm fundamental sau alți algoritmi studiați.</p>	

Clasa :

Tema lecției : Structura repetitivă condiționată anterior.

Fișă de lucru

Fie n un număr natural nenul, $0 < n < 108$. Se cere :

1. Să se determine cel mai mic număr natural care are aceiași factori primi în descompunere ca și n .
2. Să se verifice dacă n este liber de pătrate (în descompunerea sa în factori primi, niciun factor nu apare la putere pară). Se va afișa răspunsul corespunzător DA/NU.
3. Să se determine numărul de divizori ai lui n , folosind parcurgerea divizorilor proprii.
4. Să se determine numărul de divizori ai lui n , folosind formula lui Euler.
5. Să se determine numărul cifrelor de 0 de la sfârșitul produsului numerelor naturale citite până la citirea numărului 0.
6. Să se determine numărul de cifre 0 de la sfârșitul lui $n!$.

Temă

Fie n un număr natural nenul, $0 < n < 108$. Se cere :

1. Să se determine cel mai mare factor prim care apare în descompunerea lui n la o putere impară, dacă există sau să se afișeze mesajul *nu există*, în caz contrar.
2. Să se determine cel mai mic număr natural cu care trebuie înmulțit n astfel încât produsul obținut să fie pătrat perfect. Să se afișeze descompunerea în factori primi a acestui număr, câte o pereche de forma $d \ p$ (divizor exponent) pe o linie.
3. (**) Se citește x , număr natural, $1 < x < 105$. Să se verifice dacă $x!$ se divide cu n .

Proiect de tehnologie didactică

Școala : –

Disciplina : Informatică

Clasa : a XI-a

Profilul : Matematică-informatică, intensiv informatică

Data : –

Profesor : –

Unitatea de învățare : Grafuri neorientate și arbori cu rădăcină

Tema lecției : Grafuri hamiltoniene, euleriene și arbori cu rădăcină

Tipul lecției : Lecție de recapitulare-evaluare

Durata : 50 de minute

Competențe generale :

- (CG1) Identificarea datelor care intervin într-o problemă și aplicarea algoritmilor fundamentali de prelucrare a acestora.
- (CG2) Elaborarea algoritmilor de rezolvare a problemelor.
- (CG3) Implementarea algoritmilor într-un limbaj de programare.

Competențe specifice :

- (CS1.1) Transpunerea unei probleme din limbaj natural în limbaj de grafuri, folosind corect terminologia specifică.
- (CS1.2) Analizarea unei probleme în scopul identificării datelor necesare și alegerea modalităților adecvate de structurare a datelor care intervin într-o problemă.

(CS1.3) Descrierea unor algoritmi simpli de verificare a unor proprietăți specifice grafurilor.

(CS1.4) Descrierea algoritmilor fundamentali de prelucrare a grafurilor și implementare acestora într-un limbaj de programare.

(CS1.8) Aplicarea în mod creativ a algoritmilor fundamentali în rezolvarea unor probleme concrete.

Competențe derivate :

La sfârșitul activității didactice, elevii vor fi capabili :

(CD1) Să recunoască grafurile hamiltoniene și euleriene pe baza definițiilor și a condiției necesare și suficiente (la grafuri euleriene).

(CD2) Să construiască arborele parțial corespunzător parcurgerilor BF și DF.

(CD3) Să reprezinte un arbore cu rădăcină în formă ascendentă și descendentă.

(CD4) Să recunoască forme de aplicare ale algoritmilor fundamentali discutați în probleme diverse.

C1. Competențe cognitive

La sfârșitul lecției, elevii vor fi capabili :

(C1.1): să analizeze o problemă, să descrie etapele algoritmului de rezolvare ;

(C1.2): să identifice tipul grafurilor din enunț ;

(C1.3): să scrie în secvențe de cod pentru parcurgerea DF, BF a unui graf dat.

C2. Competențe afective

(C2.1): să argumenteze alegerile făcute în subiectele grilă date la test ;

(C2.2): să se autoevalueze corect ;

(C2.3): să dovedească curiozitate și interes pentru noțiunile utilizate.

C3. Competențe psihomotorii

(C3.1): să utilizeze corect noțiunile fundamentale de la grafuri neorientate și arbori în problemele propuse

C4. Competențe acționale

(C4.1): să afișeze muchiile selectate în arbore conform parcurgerii DF sau BF ;

(C4.2): să se încadreze în timpul de lucru alocat testului ;

(C4.3): să scrie în C++ funcții corecte care implementează cerințele.

Strategii didactice

Principii didactice

- principiul legării teoriei de practică ;
- principiul sistematizării și continuității cunoștințelor ;
- principiul accesibilității ;
- principiul individualizării și diferențierii învățării.

Metode și procedee didactice : problematizarea (P), conversația frontală și individuală (Cv), explicația (E), munca independentă(M).

Forme de organizare : subiectul scris al testului individual

Forme de dirijare a învățării :

- independentă ;
- dirijată de profesor prin mijloacele de învățare.

Metode de evaluare : evaluare continuă prin test docimologic.

Resurse materiale :

- subiectele scrise pentru test ;
- culegeri, manual.

Referințe bibliografice : [CȘ], [MM], [IP], [K2].

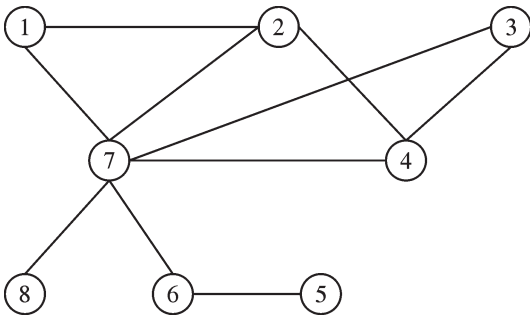
Nume și prenume

Data

Clasa

Grafuri – lucrare de verificare a cunoștințelor

Nr. 1

1. Scrieți declarațiile de date corespunzătoare și definiția completă a unei funcții *arb_DF* care primește ca parametru un nod x al unui graf conex cu n vârfuri, dat prin matricea de adiacență și afișează muchiile arborelui parțial obținut prin traversarea *DFS* a grafului dat.
2. Fie graful neorientat :
 
 - a) Adăugați un număr minim de muchii astfel încât graful să fie eulerian. Scrieți lista muchiilor adăugate.
 - b) Care este numărul maxim de muchii de eliminat din graful inițial astfel încât graful să rămână conex ?
 - c) Pe graful inițial, puneți în evidență un subgraf hamiltonian cu 5 vârfuri. Dacă există, scrieți nodurile care formează subgraful, altfel justificați de ce nu este posibil.
3.
 - a) Desenați arborele și scrieți vectorul de „tați” corespunzător arborelui cu rădăcină, cu 8 noduri, numerotate de la 1 la 8, dat prin lista alăturată a descendenților direcți (fiilor) ?
 - b) Care este înălțimea arborelui ?
 - c) Care sunt frunzele arborelui ?
4. Determinați numărul maxim de muchii din care poate fi format un graf eulerian cu 20 de vârfuri.
5. Scrieți declarațiile de date corespunzătoare și definiția completă a unei funcții *hamiltonian* care primește ca parametrii x , un șir de noduri, și prin și lg lungimea acestuia și returnează 1 dacă șirul reprezintă un ciclu hamiltonian al unui graf conex cu n vârfuri, dat prin matricea de adiacență și 0 altfel.

Barem

1. 1,5 puncte.
2. $0,75 * 3 = 2,25$ puncte.
3. $0,75 * 3 = 2,25$ puncte.
4. 1 punct.
5. 2 puncte.

Se acordă un punct din oficiu.

Timp de lucru : 30 min.

La acest test cred că iau nota :

Rezolvare și barem detaliat

1. 0,25 p. – declarare date; 0,25 p. – verificare adiacență; 0,25 p. – verificare nod nevizitat; 0,25 p. – afișare muchie; 0,25 p. – marcare nod vizitat; 0,25 p. – utilizare parcurgere DF.

```
int a[NMax+1][Max+1], n, uz[NMax];
/* variabile globale: n = număr noduri, a = matricea de adiacență;
uz[i]=1, dacă nodul i a fost parcurs în DF și 0, altfel */

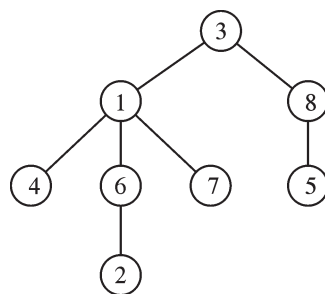
void arb_DF(int x)
{
    uz[x]=1;
    for (int i=1; i<=n; i++)
        if (a[x][i]==1 && uz[i]==0)
        {
            cout<<x<<' '<<i<<'\n';
            DF(i);
        }
}
```

2.

- $[2, 5], [8, 4]$.
- Din cele 10 muchii elimin 3 muchii. Vom obține un graf parțial *conex minimal* care va fi arbore și va avea $8 - 1 = 7$ muchii.
- Subgraful hamiltonian cu 5 vârfuri poate fi indus de mulțimea $\{1, 2, 3, 4, 7\}$.
- Desenați arborele și scrieți vectorul de „tați” corespunzător arborelui cu rădăcină, cu 8 noduri, numerotate de la 1 la 8, dat prin lista alăturată a descendenților direcți (fiilor)?

3.

- $Tata = (3, 6, 0, 1, 8, 1, 1, 3)$.
- Înălțimea arborelui = 3.
- Frunzele arborelui: 2, 4, 5, 7.



4. Numărul maxim de muchii din care poate fi format un graf eulerian cu 20 de vârfuri se obține pentru graful regulat cu 20 de vârfuri, fiecare având gradul 18 și este egal cu $(18 * 20) / 2 = 180$.

Graful complet cu 20 de vârfuri nu este eulerian deoarece toate vârfurile au gradul 19, care este impar.

5. 0,25 p. – declarare date; 0,25 p. – verificare lungime; 0,25 p. – verificare egalitate nod inițial și final în ciclul hamiltonian; 0,25 p. – verificare adiacență; 0,25 p. – verificare noduri distincte; 0,25 p. – marcare nod; 0,25 p. – antet corect; 0,25 p. – returnare rezultat corect.

```
int a[NMax+1][Max+1], n, uz[NMax];
/* variabile globale: n = număr noduri, a = matricea de adi-
acență;
uz[i]=1, dacă nodul i face parte din ciclul hamiltonian și
0, altfel */

int hamiltonian(int x[], int lg)
{
    if (lg!=n || x[1]!=x[lg]) return 0;
    for (int i=1; i<lg; i++)
    {
        if (a[x[i]][x[i+1]]==0) return 0; //nu sunt adiacente
        if (uz[x[i]]==1) return 0; //nodurile nu sunt distincte
        uz[x[i]]=1;
    }
    return 1; }
```

Anexa 2

Subiecte date la concursuri de specialitate

În această anexă furnizăm (ca modele) câteva seturi de subiecte date în decursul ultimilor ani la diverse concursuri ale elevilor și profesorilor. Acolo unde a fost posibil, subiectele sunt însoțite de baremele corespunzătoare de rezolvare, nu însă și de rezolvările în sine. Uneori, doar indicăm modalitatea în care s-a construit un barem, și nu baremul în sine. Gruparea pe seturi a subiectelor a fost făcută în funcție de scopul și finalitatea concursurilor, după cum urmează :

- A. Probleme date la bacalaureat și olimpiade.
- B. Probleme date la admiterea la Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași.
- C. Subiecte date la concursurile de obținere a definitivatului și/sau gradului II în informatică, pentru profesorii din învățământul preuniversitar (județele arondate centrului universitar și Inspectoratului Școlar Județean Iași).

A. Probleme date la bacalaureat și olimpiade

Examenul de bacalaureat național 2015

Proba E. d)

Informatică

Limbajul C/C++

Varianta 9

Filiera teoretică, profilul real, specializările : matematică-informatică și matematică-informatică intensiv informatică

Filiera vocațională, profilul militar, specializarea matematică-informatică

- Toate subiectele sunt obligatorii. Se acordă 10 puncte din oficiu.
- Timpul de lucru efectiv este de 3 ore.
- În rezolvările cerute, identificatorii utilizați trebuie să respecte precizările din enunț, iar în lipsa unor precizări explicite, notațiile trebuie să corespundă cu semnificațiile asociate acestora (eventual în formă prescurtată).
- În programele cerute, datele de intrare se consideră corecte, validarea acestora nefiind necesară.

SUBIECTUL I**(30 de puncte)**

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Variabila întreagă x memorează un număr natural cu cel puțin patru cifre nenule distincte. Expresia C/C++ a cărei valoare este egală cu cifra sutelor acestui număr este : **(4 p.)**
 - a) $x/100$;
 - b) $x\%100$;
 - c) $(x/10)\%10$;
 - d) $(x/100)\%10$
2. Se consideră algoritmul de mai jos, reprezentat în pseudocod.

```

citește n,k
(numere naturale, k>1)
pm ← 0
i ← 1
cât timp i ≤ n execută
    x ← i
    p ← 0
    cât timp x % k = 0 execută
        x ← [x/k]
        p ← p+1
    ■
    dacă p > pm atunci
        pm ← p
    ■
    i ← i+1
    ■
scrie pm

```

S-a notat cu $a \% b$ restul împărțirii numărului natural a la numărul natural nenul b și cu $[c]$ partea întreagă a numărului real c .

- a) Scrieți valoarea afișată dacă se citesc, în această ordine, numerele 7 și 2. **(6 p.)**
- b) Dacă pentru variabila k se citește numărul 5, scrieți cea mai mică și cea mai mare valoare care pot fi citite pentru variabila n astfel încât, în urma executării algoritmului, pentru fiecare dintre acestea, valoarea afișată să fie 3. **(4 p.)**
- c) Scrieți în pseudocod un algoritm echivalent cu cel dat, înlocuind prima structură `cât timp...execută` cu o structură repetitivă de tip `pentru...execută`. **(6 p.)**
- d) Scrieți programul C/C++ corespunzător algoritmului dat. **(10 p.)**

SUBIECTUL II**(30 de puncte)**

Pentru fiecare dintre itemii 1 și 2 scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Variabila c , declarată astfel,

```

struct carte
{ char titlu[21];
  float pret;
} c;

```


memorează titlul și prețul unei cărți. Expresia $C/C++$ a cărei valoare reprezintă prețul cărții respective majorat cu 50% este :

- a) $c.pret*3/2$; b) $pret.c*3/2$; c) $c(pret)*3/2$; d) $pret[c]*3/2$.

2. Un arbore cu 37 de noduri, numerotate de la 1 la 37, are ca rădăcină nodul numerotat cu 1, iar tatăl fiecărui nod i ($i \in [2, 37]$) este numerotat cu partea întreagă a rădăcinii pătrate a lui i ($\lfloor \sqrt{i} \rfloor$). Numărul de frunze ale arborelui este : **(4 p.)**

- a) 36 ; b) 31 ; c) 21 ; d) 6.

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

3. Un graf neorientat cu 8 noduri, numerotate de la 1 la 8, are muchiile $[1, 2]$, $[1, 6]$, $[4, 6]$, $[3, 6]$, $[6, 5]$, $[5, 3]$, $[3, 4]$, $[7, 8]$, $[8, 2]$. Enumerați trei noduri care nu aparțin nici unui ciclu în acest graf. **(6 p.)**

4. Fiind date două șiruri de caractere a și b , îl numim pe a prefix al lui b dacă a este egal cu b sau dacă b se poate obține din a prin alipirea la dreapta a unor noi caractere. Variabilele a și b pot memora câte un șir cu cel mult 20 de caractere. Știind că variabila b a fost inițializată cu un șir format dintr-un număr par de caractere, scrieți o secvență de instrucțiuni în urma executării căreia variabila a să memoreze un prefix al lui b a cărui lungime să fie jumătate din lungimea lui b .

Exemplu : dacă b memorează șirul *aurari*, atunci a memorează șirul *aur*. **(6 p.)**

5. Scrieți un program $C/C++$ care citește de la tastatură un număr natural n ($n \in [2, 20]$), apoi n numere naturale din intervalul $[0, 104]$, reprezentând, de la stânga la dreapta, în această ordine, valorile elementelor aflate pe prima linie a unui tablou bidimensional cu n linii și n coloane. Programul construiește în memorie tabloul, inițializând celelalte elemente, astfel încât fiecare linie să se obțină prin permutarea circulară a elementelor liniei anterioare, de la stânga spre dreapta, cu o poziție, ca în exemplul de mai jos.

Programul afișează pe ecran tabloul obținut, fiecare linie a tabloului pe câte o linie a ecranului, elementele de pe aceeași linie fiind separate prin câte un spațiu.

Exemplu : dacă se citesc numerele $n = 4$, apoi 1, 1, 3, 2, se obține tabloul : **(10 p.)**

```
1  1  3  2
2  1  1  3
3  2  1  1
1  3  2  1
```

SUBIECTUL III**(30 de puncte)**

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Utilizând metoda *backtracking*, se generează toate numerele naturale din intervalul [100, 999] care au suma cifrelor egală cu 5. Primele cinci soluții obținute sunt, în această ordine, 104, 113, 122, 131, 140. Utilizând același algoritm, se generează toate numerele naturale din intervalul [1000, 9999] care au suma cifrelor egală cu 6. Al treilea număr generat este : **(4 p.)**

- a) 1005 ; b) 1023 ; c) 1031 ; d) 1041.

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

2. Subprogramul F este definit alăturat. Scrieți ce se afișează în urma apelului de mai jos.
F('d') ; **(6 p.)**

```
void F(char c)
{ if(c>='a')
  { cout<<c;
    printf("%c", c);
    F(c-1);
  }
}
```

3. Șirul lui Fibonacci (1, 1, 2, 3, 5, 8, 13, 21...) se definește astfel :
 $f_1 = 1$, $f_2 = 1$ și $f_i = f_{i-1} + f_{i-2}$ pentru orice număr natural i , $i \geq 3$.

Subprogramul Fibo are un singur parametru, n , prin care primește un număr natural ($n \in [1, 30]$). Subprogramul returnează al n -lea termen impar al șirului lui Fibonacci. Scrieți definiția completă a subprogramului.

Exemplu : dacă $n = 6$, subprogramul returnează numărul 21. **(10 p.)**

4. Fișierul `bac.txt` conține un șir de cel mult un milion de numere naturale din intervalul [0, 102], separate prin câte un spațiu. Se cere să se determine toate perechile distincte formate din termeni ai șirului aflat în fișier, x și y ($y - x \geq 2$), astfel încât să nu existe nici un termen al șirului care să aparțină intervalului (x, y) . Numerele din fiecare pereche sunt afișate pe câte o linie a ecranului, în ordine strict crescătoare, separate printr-un spațiu, iar dacă nu există nici o astfel de pereche, se afișează pe ecran mesajul `nu exista`. Pentru determinarea numerelor cerute utilizați un algoritm eficient din punctul de vedere al timpului de executare.

Exemplu : dacă fișierul conține numerele

5 9 0 8 10 11 12 13 15 14 6 7 40 10 0 0 5 41 95 7

atunci pe ecran se afișează, nu neapărat în această ordine, perechile

0 5

15 40

41 95

- a) Descrieți în limbaj natural algoritmul utilizat, justificând eficiența acestuia. **(4 p.)**
b) Scrieți programul C/C++ corespunzător algoritmului descris. **(6 p.)**

Examenul de bacalaureat național 2015
Proba E. d)
Informatică
Barem de evaluare și de notare
(comun pentru limbajele C/C++ și Pascal)
Varianta 9

Filiera teoretică, profilul real, specializările : matematică-informatică, matematică-informatică intensiv informatică

Filiera vocațională, profilul militar, specializarea matematică-informatică

- Se punctează oricare alte modalități de rezolvare corectă a cerințelor.
- Nu se acordă punctaje intermediare, altele decât cele precizate explicit prin barem. Nu se acordă fracțiuni de punct.
- Se acordă 10 puncte din oficiu. Nota finală se calculează prin împărțirea punctajului total acordat pentru lucrare la 10.
- Utilizarea unui tip de date care depășește domeniul de valori precizat în enunț este acceptată dacă acest lucru nu afectează corectitudinea în funcționarea programului.
- Se vor lua în considerare atât implementările concepute pentru compilatoare pe 16 biți, cât și cele pentru compilatoare pe 32 de biți.

SUBIECTUL I

(30 de puncte)

1.	d		4 p.	
2.	a)	Răspuns corect : 2	6 p.	
	b)	Răspuns corect : 125, 624	4 p.	Se acordă câte 2 p. pentru fiecare valoare conform cerinței.
	c)	Pentru algoritm pseudocod corect – echivalența prelucrării realizate, conform cerinței (*) – corectitudinea globală a algoritmului ¹	6 p. 5 p. 1 p.	(*) Se acordă numai 2 p. dacă algoritmul are o structură repetitivă de tipul indicat, principial corectă, dar nu este echivalent cu cel dat. Se va puncta orice formă corectă de structură repetitivă conform cerinței.
	d)	Pentru program corect – declararea tuturor variabilelor – citire corectă – afișare corectă – instrucțiune de decizie corectă – instrucțiuni repetitive corecte (*) – atribuirii corecte – corectitudine globală a programului ¹	10 p. 1 p. 1 p. 1 p. 2 p. 3 p. 1 p. 1 p.	(*) Se acordă numai 2 p. dacă doar una dintre instrucțiuni este corectă.

SUBIECTUL II**(30 de puncte)**

1.	a	4 p.	
2.	b	4 p.	
3.	Pentru răspuns corect	6 p.	Se acordă câte 2 p. pentru fiecare nod enumerat conform cerinței (oricare dintre nodurile 1, 2, 7, 8).
4.	Pentru rezolvare corectă	6 p.	Se acordă câte 2 p. pentru fiecare aspect al cerinței (obținerea unui prefix al șirului, lungimea prefixului, memorarea în variabila indicată).
5.	Pentru program corect – declararea variabilei de tip tablou – citirea elementelor aflate pe prima linie – memorarea valorilor elementelor conform cerinței (*) – afișarea unui tablou bidimensional – declararea și citirea variabilelor simple, corectitudine globală a programului ¹	10 p. 1 p. 1 p. 6 p. 1 p. 1 p.	(*) Se acordă câte 2 p. pentru fiecare aspect al cerinței (plasarea conform cerinței a primului element al unei linii, plasarea conform cerinței a celorlalte elemente ale unei linii, obținerea tuturor elementelor tabloului).

SUBIECTUL III**(30 de puncte)**

1.	b	4 p.	
2.	Răspuns corect : dcba	6 p.	Se acordă numai 3p. pentru răspuns parțial corect, care include secvența dcb, sau pentru șirul abcd.
3.	Pentru subprogram corect – antetul subprogram (*) – determinarea numărului cerut (**) – instrucțiunea/instrucțiunile de returnare a rezultatului – declararea tuturor variabilelor locale, corectitudinea globală a subprogramului ¹	10 p. 2 p. 6 p. 1 p. 1 p.	(*) Se acordă câte 1p. pentru fiecare aspect al antetului (structură, declararea parametrului) conform cerinței. (**) Se acordă câte 2p. pentru fiecare aspect al cerinței (termenul șirului, termenul impar, numărul de ordine – inclusiv cazul în care $n < 3$).
4.	a) Pentru răspuns corect – coerența explicării metodei (*) – justificarea unor elemente de eficiență	4 p. 2 p. 2 p.	(*) Se acordă punctajul chiar dacă metoda aleasă nu este eficientă.
	b) Pentru program corect – operații cu fișiere : declarare, pregătire în vederea citirii, citire din fișier	6 p. 1 p.	(*) Se acordă punctajul chiar dacă soluția propusă nu prezintă elemente de eficiență.

		– determinarea perechilor cerute (*, **)	3 p.	(**) Se acordă câte 1p. pentru fiecare condiție impusă perechilor x, y ($y - x \geq 2$, interval (x, y) care să nu conțină nici un termen al șirului, perechi distincte).
		– afișarea datelor conform cerinței și tratarea cazului nu exista	1 p.	
		– utilizarea unui algoritm eficient (***)	1 p.	(***) Se acordă punctajul numai pentru un algoritm liniar (de complexitate $O(n)$). O soluție posibilă utilizează un vector de apariții (în care v_i este 1 dacă valoarea i apare în șir sau 0 altfel) actualizat pe măsura citirii datelor din fișier. Vectorul de apariții este parcurs o singură dată după completarea sa, memorându-se la fiecare pas ultimii doi indici i și j ($i < j$) cu proprietatea că valorile v_i și v_j sunt nenule și $j - i \geq 2$, care reprezintă fiecare dintre perechile cerute.

1. Corectitudinea globală vizează structura, sintaxa, alte aspecte neprecizate în barem.

Examenul de bacalaureat național 2015

Proba E. d)

Informatică

Limbajul C/C++

Varianta 5

Filiera teoretică, profilul real, specializările : matematică-informatică, matematică-informatică intensiv informatică

Filiera vocațională, profilul militar, specializarea matematică-informatică

- Toate subiectele sunt obligatorii. Se acordă 10 puncte din oficiu.
- Timpul de lucru efectiv este de 3 ore.
- În rezolvările cerute, identificatorii utilizați trebuie să respecte precizările din enunț, iar în lipsa unor precizări explicite, notațiile trebuie să corespundă cu semnificațiile asociate acestora (eventual în formă prescurtată).
- În programele cerute, datele de intrare se consideră corecte, validarea acestora nefiind necesară.

SUBIECTUL I

(30 de puncte)

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Expresia `C/C++` alăturată are valoarea : **(4 p.)** 4+5%7*2
- a) 4 ; b) 8 ; c) 9 ; d) 14.

2. Se consideră algoritmul alăturat, reprezentat în pseudocod.

S-a notat cu $a \% b$ restul împărțirii numărului natural a la numărul natural nenul b și cu $[c]$ partea întreagă a numărului real c .

- Scrieți valoarea afișată dacă se citește, în această ordine, numerele 997 și 1005. **(6 p.)**
- Dacă pentru m se citește numărul 54321, scrieți cel mai mare număr care poate fi citit pentru n astfel încât, în urma executării algoritmului, valoarea afișată să fie 0. **(4 p.)**
- Scrieți în pseudocod un algoritm, echivalent cu cel dat, înlocuind structura pentru... execută cu o structură de tip cât timp... execută. **(6 p.)**
- Scrieți programul C/C++ corespunzător algoritmului dat. **(10 p.)**

```

citește m, n (numere naturale,  $m \leq n$ )
nr ← 0
pentru x ← m, n execută
    y ← 0
    z ← x
    repetă
        y ← y * 10 + z % 10
        z ← [z / 10]
    pânăcând z = 0
    dacă x = y atunci
        nr ← nr + 1
scrie nr

```

SUBIECTUL II

(30 de puncte)

Pentru fiecare dintre itemii 1 și 2 scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. În declarațiile alăturate, variabilele A și B memorează coordonatele câte unui punct în sistemul de coordonate xOy. Indicați expresia care are valoarea 1 dacă și numai dacă cele două puncte coincid. **(4 p.)**

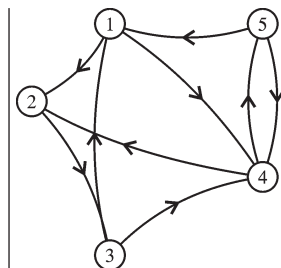
```

struct punct
{ int x, y;
} A, B;

```

- $A[x] == B[x] \ \&\& \ A[y] == B[y]$
- $A.x == B.x \ \&\& \ A.y == B.y$
- $x[A] == x[B] \ \&\& \ y[A] == y[B]$
- $x.A == x.B \ \&\& \ y.A == y.B$

2. Un graf orientat cu 5 vârfuri, numerotate de la 1 la 5, este reprezentat alăturat. Numărul maxim de arce care se pot elimina, astfel încât graful parțial obținut să fie tare conex este : **(4 p.)**



- 2 ;
- 3 ;
- 4 ;
- 5.

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare :

3. Un arbore cu 8 noduri, numerotate de la 1 la 8, este reprezentat prin vectorul de „tați” (4, 4, 0, 3, 6, 3, 2, 7). Scrieți un lanț elementar care are o extremitate în rădăcina arborelui și cealaltă extremitate în una dintre frunzele acestuia. **(6 p.)**
4. Variabilele i și j sunt de tip întreg, iar variabila a memorează un tablou bidimensional cu 5 linii și 5 coloane, numerotate de la 1 la 5, având inițial toate elementele nule.

Fără a utiliza alte variabile decât cele menționate, scrieți secvența de instrucțiuni de mai jos, înlocuind punctele de suspensie astfel încât, în urma executării secvenței obținute, variabila a să memoreze tabloul alăturat.

```
for (i=1; i<=5; i++)
    for (j=1; j<=5; j++)
```

```
10101
01010
10101
01010
10101
```

(6 p.)

5. Se consideră un text cu cel mult 100 de caractere, în care cuvintele sunt formate numai din litere mari și mici ale alfabetului englez și sunt separate prin câte un spațiu. Textul reprezintă numele unei instituții sau al unei organizații. Scrieți un program C/C++ care citește de la tastatură un text de tipul precizat și construiește în memorie, apoi afișează pe ecran un șir de caractere ce reprezintă acronimul corespunzător numelui citit. Acronimul este format din primul caracter al fiecărui cuvânt al numelui care începe cu majusculă.

Exemplu : Dacă șirul citit este

Universitatea de Arte Plastice BUCURESTI
se va obține șirul
UAPB

(10 p.)

SUBIECTUL III

(30 de puncte)

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Utilizând metoda *backtracking*, se generează toate șiragurile formate din câte 5 pietre distincte din mulțimea {opal, rubin, safir, smarald, topaz}. Două șiraguri sunt distincte dacă pietrele sunt dispuse în altă ordine. Primele patru soluții obținute sunt, în această ordine : (opal, rubin, safir, smarald, topaz), (opal, rubin, safir, topaz, smarald), (opal, rubin, smarald, safir, topaz) și (opal, rubin, smarald, topaz, safir). Indicați soluția care trebuie eliminată din enumerarea următoare, astfel încât cele rămase să apară în ordinea generării lor, pe poziții consecutive : (smarald, safir, opal, topaz, rubin) (smarald, safir, topaz, opal, rubin) (smarald, safir, topaz, rubin, opal) (smarald, topaz, opal, rubin, safir). **(4 p.)**

a) (smarald, safir, opal, topaz, rubin)

b) (smarald, safir, topaz, opal, rubin)

c) (smarald, safir, topaz, rubin, opal)

d) (smarald, topaz, opal, rubin, safir)

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

2. Subprogramul F este definit alăturat. Scrieți ce se afișează în urma apelului de mai jos.

F(154678, 3);

(6 p.)

```
void F (long a, int b)
{ if(a*b!=0)
  if(a%2==0)
  { cout<<a%10; | printf("%d",a%10);
    F(a/10,b-1);
  }
  else
  { F(a/10,b+1);
    cout<<a%10; | printf("%d",a%10);
  }
}
```

3. Subprogramul DivImpar are doi parametri, a și b, prin care primește două numere naturale din intervalul $[1, 10^5]$. Subprogramul returnează cel mai mare divizor comun impar al numerelor a și b.

Scrieți definiția completă a subprogramului.

Exemplu: dacă $a = 30$ și $b = 60$, subprogramul returnează valoarea 15.

(10p.)

4. Fișierul `date.in` conține un șir de cel mult un milion de numere naturale din intervalul $[0, 10^9]$, separate prin câte un spațiu. Șirul are cel puțin doi termeni pari și cel puțin doi termeni impari. Se cere să se afișeze pe ecran mesajul DA dacă șirul aflat în fișier are un subșir ordonat crescător, format din toți termenii pari ai săi, și un subșir ordonat descrescător, format din toți termenii impari ai săi. Dacă nu există două astfel de subșiruri, programul afișează pe ecran mesajul NU. Pentru verificarea proprietății cerute utilizați un algoritm eficient din punctul de vedere al timpului de executare și al memoriei necesare.

Exemplu: dacă fișierul `date.in` conține numerele

7 2 5 2 4 3 8

se afișează pe ecran mesajul

DA

iar dacă fișierul conține numerele

5 2 7 2 4 3 8

se afișează pe ecran mesajul

NU

- a) Descrieți în limbaj natural algoritmul utilizat, justificând eficiența acestuia.

(4p.)

- b) Scrieți programul C/C++ corespunzător algoritmului descris.

(6p.)

Examenul de bacalaureat național 2015
Proba E. d)
Informatică
Barem de evaluare și de notare
(comun pentru limbajele C/C++ și Pascal)
Varianta 5

Filiera teoretică, profilul real, specializările : matematică-informatică, matematică-informatică intensiv informatică

Filiera vocațională, profilul militar, specializarea matematică-informatică

- Se punctează oricare alte modalități de rezolvare corectă a cerințelor.
- Nu se acordă punctaje intermediare, altele decât cele precizate explicit prin barem. Nu se acordă fracțiuni de punct.
- Se acordă 10 puncte din oficiu. Nota finală se calculează prin împărțirea punctajului total acordat pentru lucrare la 10.
- Utilizarea unui tip de date care depășește domeniul de valori precizat în enunț este acceptată dacă acest lucru nu afectează corectitudinea în funcționarea programului.
- Se vor lua în considerare atât implementările concepute pentru compilatoare pe 16 biți, cât și cele pentru compilatoare pe 32 de biți.

SUBIECTUL I

(30 de puncte)

1.	d		4 p.	
2.	a)	Răspuns corect : 2	6 p.	
	b)	Răspuns corect : 54344	4 p.	
	c)	Pentru algoritm pseudocod corect – echivalența prelucrării realizate, conform cerinței (*) – corectitudinea globală a algoritmului ¹	6 p. 5 p. 1 p.	(*) Se acordă numai 2 p. dacă algoritmul are o structură repetitivă conform cerinței, principial corectă, dar nu este echivalent cu cel dat. Se va puncta orice formă corectă de structură repetitivă conform cerinței.
	d)	Pentru program corect – declarare variabile – citire date – afișare date – instrucțiune de decizie corectă – instrucțiuni repetitive corecte (*) – atribuirii corecte – corectitudine globală a programului ¹	10 p. 1 p. 1 p. 1 p. 2 p. 3 p. 1 p. 1 p.	(*) Se acordă numai 2 p. dacă doar una dintre instrucțiuni este corectă.

SUBIECTUL II**(30 de puncte)**

1.	b	4 p.	
2.	c	4 p.	
3.	Pentru rezolvare corectă	6 p.	Se acordă câte 2 p. pentru fiecare proprietate a lanțului (extremitate inițială, extremitate finală, lanț elementar) conform cerinței.
4.	Pentru rezolvare corectă – acces corect la un element al tabloului – atribuirea valorilor indicate elementelor tabloului (*)	6 p. 1 p. 5 p.	(*) Se acordă numai 2 p. dacă doar una dintre proprietățile tabloului (alternanța valorilor binare în cadrul unei linii/coloane, toate elementele suport) este conform cerinței.
5.	Pentru program corect – declararea corectă a unei variabile care să memoreze un șir de caractere – citirea șirului – accesul la un caracter al șirului – construirea șirului conform cerinței (*) – afișarea datelor – declararea variabilelor simple, corectitudinea globală a programului ¹	10 p. 1 p. 1 p. 1 p. 5 p. 1 p. 1 p.	(*) Se acordă câte 1 p. pentru fiecare aspect al cerinței (identificarea primei litere a unui cuvânt, identificarea unei majuscule, obținerea unui șir construit cu prima literă a unor cuvinte, caractere suport pentru acronim, construirea în memorie).

SUBIECTUL III**(30 de puncte)**

1.	a	4 p.	
2.	Răspuns corect : 864157	6 p.	Se acordă numai 3 p. pentru răspuns parțial corect, care include secvența 864.
3.	Pentru subprogram corect – antetul subprogramului (*) – determinarea numărului cerut (**) – instrucțiunea de returnare a rezultatului – declararea tuturor variabilelor locale, corectitudine globală a subprogramului ¹	10 p. 2 p. 6 p. 1 p. 1 p.	(*) Se acordă câte 1 p. pentru fiecare aspect al antetului (structură, declarare parametri de intrare) conform cerinței. (**) Se acordă câte 2 p. pentru fiecare proprietate a numărului cerut (divizor comun, impar, maxim).
4.	a) Pentru răspuns corect – coerența explicării metodei (*) – explicarea unor elemente de eficiență	4 p. 2 p. 2x1 p.	(*) Se acordă punctajul chiar dacă metoda aleasă nu este eficientă.
	b) Pentru program corect – operații cu fișiere : declarare, pregătire în vederea citirii, citire din fișier	6 p. 1 p.	(*) Se acordă punctajul chiar dacă soluția propusă nu prezintă elemente de eficiență.

	– verificarea proprietății indicate pentru subșiruri (*, **)	3 p.	(**) Se acordă numai 2 p. dacă s-a verificat proprietatea cerută doar pentru unul dintre subșiruri sau dacă algoritmul este principial corect, dar nu conduce la rezultatul cerut pentru orice set de date de intrare.
	– afișarea mesajului conform cerinței	1 p.	
	– utilizarea unui algoritm eficient (***)	1 p.	(***) Se acordă punctajul numai pentru un algoritm liniar (de complexitate $O(n)$), care utilizează eficient memoria. O soluție posibilă parcurge cel mult o dată fișierul, memorând atât valoarea ultimului termen par, precum și pe cea a ultimului termen impar și comparându-le, după caz, cu valoarea termenului curent.

1. Corectitudinea globală vizează structura, sintaxa, alte aspecte neprecizate în barem.

Olimpiada națională de informatică pentru gimnaziu *Clasa a V-a*

Se consideră două numerele naturale K și S și un șir de N numere naturale a_1, a_2, \dots, a_N . O secvență de lungime K este un subșir format din K elemente aflate pe poziții consecutive în șir: $a_i, a_{i+1}, \dots, a_{i+K-1}$. Parcurgând șirul de la stânga la dreapta, începând cu primul element, se elimină prima secvență de lungime K , cu suma elementelor strict mai mare decât numărul S . În urma ștergerii șirul va avea $N - K$ elemente: a_1, a_2, \dots, a_{N-K} . Operația de ștergere continuă după aceleași reguli până când nu mai există secvențe care pot fi eliminate.

Cerințe

Să se scrie un program care, citind numerele N , K , S și cele N elemente din șir, rezolvă cerințele:

1. Determină numărul secvențelor care se vor elimina respectând condiția din enunț.
2. Considerând că în șirul citit nu sunt posibile eliminări de secvențe conform condiției din enunț, programul determină numărul de elemente a_i din șir cu proprietatea următoare: ștergerea lui a_i conduce la obținerea unui șir în care se mai poate elimina cel puțin o secvență de K elemente cu sumă strict mai mare ca S .

Date de intrare

Fișierul de intrare `secv.in` conține pe prima linie un număr natural P ; numărul P poate avea doar valoarea 1 sau valoarea 2. A doua linie conține, în această ordine, separate prin câte un spațiu, numerele N , K și S . A treia linie conține, în ordine, elementele șirului, despărțite prin câte un spațiu.

Date de ieșire

Dacă valoarea lui P este 1, se va rezolva numai cerința 1. În acest caz, fișierul de ieșire `secv.out` va conține pe prima linie un număr natural reprezentând numărul secvențelor eliminate.

Dacă valoarea lui P este 2, se va rezolva numai cerința 2. În acest caz, fișierul de ieșire `secv.out` va conține pe prima linie un număr natural reprezentând numărul elementelor din

șir care au proprietatea că ștergerea fiecăruia în parte ar genera un șir în care se mai pot elimina cel puțin o secvență de K elemente cu sumă strict mai mare ca S .

Restricții și precizări

$$0 < N \leq 1\,000\,000 \text{ și } K \leq N$$

$$0 < S \leq 1\,000\,000\,000$$

$$0 \leq a_1, a_2, \dots, a_N \leq 1\,000$$

Pentru rezolvarea corectă a primei cerințe se acordă 40 de puncte, iar pentru rezolvarea corectă a celei de a doua cerințe se acordă 60 de puncte.

Exemplu

1 14 3 7 1 2 1 3 1 4 5 2 1 4 1 8 2 3	3	Prima secvență de sumă strict mai mare decât 7 începe de pe poziția 4 și este formată din elementele 3 1 4; după eliminarea ei, șirul devine: 1 2 1 5 2 1 4 1 8 2 3. A doua secvență ce va fi ștearsă începe de pe poziția 2 și este formată din 2 1 5; după eliminarea ei șirul devine: 1 2 1 4 1 8 2 3 A treia secvență ce va fi ștearsă începe de pe poziția 4 și este formată din elementele 4 1 8; după eliminarea ei, șirul devine: 1 2 1 2 3 și nu mai conține nici o secvență de 3 elemente alăturate de sumă mai mare decât 7
2 9 7 18 3 3 2 1 3 3 3 3 1	2	Două elemente au această proprietate. Dacă eliminăm elementul al treilea, de valoare 2, se poate obține șirul 3 3 1 3 3 3 3 1, care conține o secvență de 7 elemente de sumă strict mai mare ca 18, începând cu elementul de pe poziția 1. Dacă eliminăm elementul al patrulea, de valoare 1, se poate obține șirul 3 3 2 3 3 3 3 1, care conține o secvență de 7 elemente de sumă strict mai mare ca 18, începând cu elementul de pe poziția 1.

Timp maxim de executare/test: 0,5 secunde

Memorie totală: 16 MB

Dimensiunea maximă a sursei: 5KB

Descrierea soluției

Autor: prof. Dana Lica, Centrul Județean de Excelență Prahova

Fie șirul de numere A . Pentru $P = 1$, soluție $O(N)$. Vom construi soluția iterând prin lista de elemente. La fiecare pas vom menține o listă de valori încă neeliminate, la care adăugăm la final elementul curent din iterație. Dacă prin adăugarea acestui element, ultimele K elemente ale listei au suma strict mai mare ca S , atunci ștergem ultimele K elemente și apoi continuăm parcurgerea. Ștergerea se va realiza prin decrementarea cu K elemente a lungimii listei.

Verificarea sumei ultimelor K elemente se poate realiza eficient folosind un vector de sume parțiale ale listei menționate anterior. Astfel, se va construi vectorul B , în care elementul de

pe poziția i reține suma primelor i elemente citite. În felul acesta, suma dintre pozițiile x și y se obține ca $B[y] - B[x - 1]$.

Pentru $P = 2$, soluție $O(N)$. Pentru fiecare poziție X , vom determina dacă ștergerea ei generează o subsecvență de lungime K de sumă strict mai mare ca S . Acest lucru este echivalent cu verificarea existenței unei poziții Y astfel încât :

- (1) $(\sum A[i]Y + K_i = Y) - A[X] > S$ și
- (2) $Y \leq X \leq Y + K$

Cu alte cuvinte, analizăm secvențe de lungime $K + 1$ care conțin poziția X și verificăm dacă secvența rămasă în urma eliminării lui X are suma strict mai mare ca S . Pentru un Y fixat, suma dintre paranteze poate fi calculată folosind un vector de sume parțiale Sum în $O(1)$ – cu $O(N)$ precalcularea. Se observă că pentru a verifica dacă există un Y cu proprietatea menționată, este suficient și necesar să verificăm suma maximală dintre paranteze – condiționată de $Y \leq X \leq Y + K$. Astfel, vom itera cu X de la 1 la N și suntem interesați la fiecare pas de suma maximală de $K + 1$ elemente consecutive ce conține poziția X . Pentru a determina acest lucru, vom folosi un `max_dequeue` în care la fiecare pas X inserăm secvența care începe pe poziția X și o eliminăm pe cea care se termină pe poziția $X - 1$. La fiecare pas se adună poziția la soluții dacă suma maximă din `deque` respectă condiția 1.

Olimpiada națională de informatică *Clasa a IX-a*

Fie un șir de numere naturale nenule a_1, a_2, \dots, a_n și un număr natural k .

Cerință

Să se determine un grup de k numere din șir care au proprietatea că cel mai mare divizor comun al lor este maxim. Dacă există mai multe astfel de grupuri, se cere acel grup pentru care suma elementelor este maximă.

Date de intrare

Fișierul `cmmdc.in` conține pe prima linie numerele naturale n și k separate prin spațiu. Pe linia a doua se găsesc numerele naturale a_1, a_2, \dots, a_n , separate prin câte un spațiu.

Date de ieșire

Fișierul `cmmdc.out` conține pe prima linie un număr natural reprezentând cel mai mare divizor comun a exact k numere din șir, maxim posibil. Pe linia a doua, separate prin câte un spațiu și *ordonate descrescător*, se află cele k numere din șir care dau cel mai mare divizor comun maxim.

Restricții și precizări

- $1 \leq n \leq 1\,000\,000$
- $2 \leq k \leq 100\,000$
- $k \leq n$
- $1 \leq a_i \leq 1\,000\,000, i = 1..n$
- Valorile din șir se pot repeta.

Exemplu

<i>cmmdc.in</i>	<i>cmmdc.out</i>	<i>Explicație</i>
6 3 6 9 8 10 15 3	3 15 9 6	Cel mai mare divizor comun care se poate obține dintr-un grup de 3 numere este 3, iar cele 3 numere care dau suma maximă, ordonate descrescător, sunt 15, 9 și 6.

Limită de timp : 1 secundă în Windows și 1 secundă în Linux

Memorie totală disponibilă : 64 MB

Dimensiunea maximă a sursei : 15 KB

Descrierea soluției

Autori : prof. Stelian Ciurea, Liceul „Domnul Tudor”, Drobeta-Turnu Severin ;
prof. Dan Pracsiu, Liceul „Ștefan Procopiu”, Vaslui

Se construiește un vector de frecvențe ($v[i]$ = de câte ori apare i în șirul de numere dat). Se face raționamentul următor : dacă k numere au cmmdc egal cu x , atunci ele sunt fie egale cu x , fie sunt multipli ai numărului x .

Astfel se parcurge cu o variabilă x descrescător intervalul $1000000 \rightarrow 2$ și pentru fiecare valoare a lui x se determină dacă există cel puțin k numere egale cu x sau multipli de i ai lui x , cu un algoritm asemănător cu Ciurul lui Eratostene. Această determinare se face parcurgând multiplii lui x descrescător, astfel prima submulțime determinată este soluția cerută. Cel mai mare multiplu al lui x care teoretic poate să apară printre cele n numere se poate calcula în funcție de valoarea maximă din șirul a (notată $\max(a)$ și care oricum nu depășește 1000000).

Complexitate

$O(n)$ pentru construcția vectorului v . $O(\max(a) \cdot \log[\max(a)])$ pentru determinarea rezultatului (unde $\max(a)$ e maximul din șirul a).

Expresia de mai sus este aproximarea pentru :

$$\frac{\max(a)}{1} + \frac{\max(a)}{2} + \dots + \frac{\max(a)}{i} + \dots + \frac{\max(a)}{a}$$

Rezolvări alternative se pot face prin generare de submulțimi :

- una dintre surse generează o submulțime de k elemente, îi calculează cmmdc și o reține dacă are cmmdc maxim – 25 puncte ;
- cealaltă e optimizată în sensul că se calculează cmmdc după fiecare element adăugat la submulțime și, dacă acesta e mai mic decât maximul de până atunci, se trece la alegerea altui element – 40 puncte.

În ambele surse alternative se sortează elementele descrescător.

Olimpiada județeană de informatică

Clasa a X-a

Se consideră o mulțime S care conține N șiruri de caractere formate din litere mici ale alfabetului englezesc.

Un șir de caractere se numește *interesant* în raport cu celelalte șiruri ale mulțimii dacă nu există un alt șir în mulțime care să-l conțină ca subșir. De exemplu, dacă mulțimea S conține șirurile abc , bde și $abcdef$, atunci singurul șir *interesant* este $abcdef$, deoarece abc și bde nu îl conțin ca subșir. Mai mult, abc și bde sunt subșiruri în $abcdef$, deci nu sunt *interesante*.

Cerințe

Fiind dată o mulțime S formată din N șiruri de caractere, se cere :

1. Să se determine cel mai lung șir. Dacă sunt mai multe șiruri având aceeași lungime maximă, se cere cel mai mic din punct de vedere lexicografic.
2. Să se determine toate șirurile *interesante* din mulțimea S .

Date de intrare

Fișierul de intrare `interesant.in` conține pe prima linie două numere naturale p și N , despărțite prin spațiu. Pentru toate testele de intrare, numărul p poate avea doar valoarea 1 sau valoarea 2. Pe următoarele N linii, se găsesc șirurile de caractere, câte unul pe linie.

Date de ieșire

Dacă valoarea lui p este 1, se va rezolva numai cerința 1.

În acest caz, în fișierul de ieșire `interesant.out` se va scrie cel mai lung șir dintre cele citite. Dacă există mai multe șiruri de aceeași lungime, se va scrie cel mai mic din punct de vedere lexicografic.

Dacă valoarea lui p este 2, se va rezolva numai cerința 2.

În acest caz, fișierul de ieșire `interesant.out` va conține pe prima linie o valoare K ce reprezintă numărul de șiruri *interesante*, iar pe următoarele K linii, șirurile *interesante* în ordinea în care apar în fișierul de intrare.

Restricții și precizări

- $2 \leq N \leq 200$;
- lungimea unui șir va fi cuprinsă între 1 și 5000;
- un subșir al șirului de caractere $C_0 C_1 C_2 \dots C_k$ se definește ca fiind o succesiune de caractere $C_{i_1} C_{i_2} C_{i_3} \dots C_{i_k}$, unde $0 \leq i_1 < i_2 < i_3 < \dots < i_k \leq k$;
- fișierul de intrare *nu conține șiruri identice*.
- pentru rezolvarea corectă a primei cerințe se acordă 20 de puncte, iar pentru cerința a doua se acordă 80 de puncte.

Exemple

interesant.in	interesant.out	Explicație
1 5 a b c a c a a z a d a b c a c a a d a c d z y t	a b c a c a a d	p = 1 Fișierul de intrare conține 5 șiruri. a b c a c a a d este șirul de lungime maximă. Șirul a b c a c a a z are aceeași lungime, dar este mai mare din punct de vedere lexicografic. Atenție! Pentru acest test se rezolvă doar cerința 1.

interesant.in	interesant.out	Explicație
2 5 a b c a c a a d a d z a y y t a c d z y t	2 a b c a c a a d z a y y t	p = 2 a d, a c d sunt subșiruri ale lui a b c a c a a d, iar z y t este subșir al lui z a y y t. Atenție! Pentru acest test se rezolvă doar cerința 2.

Timp maxim de execuție/test : 1,5 secunde

Memorie totală disponibilă : 8 MB.

Dimensiunea maximă a sursei 10 KB

Descrierea soluției

Autor : Nicu Vlad-Laurențiu, Liceul Teoretic „Mihail Kogălniceanu”, Vaslui

Soluția propusă analizează șirurile pe parcursul citirii din fișier.

Cerința : 1-20 puncte.

Rezolvarea este clasică : determinăm lungimea maximă a unui șir, iar pentru lungimi egale se alege șirul cel mai mic lexicografic.

Cerința : 2-80 puncte.

Rezolvarea cerinței presupune :

- verificarea unui șir dacă este subșir al altui șir ;
- utilizarea unei stive care reține șirurile „distincte”.

În funcție de tipul de verificare ales, căutare secvențială (subșir, caracter), căutare asemănătoare interclasării (parcurea paralelă a șirurilor), căutare binară, frecvențele de apariție, precum și de modul de implementare utilizat, se obțin punctaje parțiale diferențiate.

B. Probleme date la admiterea la Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași (matematică și informatică)

Universitatea „Alexandru Ioan Cuza” din Iași
Facultatea de Informatică

Admitere – studii de licență
Sesiunea iulie 2015

Test la informatică *Limbajul C/C++*

Se acordă 10 puncte din oficiu. Timpul efectiv de lucru este de 3 ore.

SUBIECTUL I

(30 de puncte)

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Numerele reale x , y , z și t satisfac inegalitățile $x < y$ și $z < t$. Precizați care dintre expresiile C/C++ de mai jos este echivalentă cu faptul că intervalele închise $[x, y]$ și $[z, t]$ au intersecția nevidă ($[x, y] \cap [z, t] \neq \emptyset$). (4 p.)

- a. $!(z > y) \ || \ (t < x)$ b. $(x \leq z) \ || \ (y \geq t)$
c. $!(x < z) \ \&\& \ (t < y)$ d. $!(x > t) \ || \ (y > z)$

2. Se consideră algoritmul alăturat, descris în pseudocod.

- a) Scrieți valoarea afișată de algoritm dacă numărul n citit este 213521. (6 p.)
b) Care este cel mai mic număr natural format din patru cifre distincte care poate fi citit în variabila n astfel încât algoritmul să afișeze valoarea 1? (6 p.)
c) Scrieți o secvență de instrucțiuni care să folosească doar operații de adunare și scădere și care să fie echivalentă cu instrucțiunea $n \leftarrow [n / 10]$. (4 p.)
d) Scrieți programul C/C++ corespunzător algoritmului alăturat. (10 p.)

```

citește n      (număr natural)
x ← n % 10;   m ← 1;   s ← 1
cât timp n > 9 execută
┌ n ← [n / 10];   y ← n % 10
│ dacă (y-x)*m < 0 atunci
│ ┌ dacă m > 0 atunci m ← -1
│ │ altfel s ← 0
│ x ← y
└ scrie s

```

SUBIECTUL II

(30 de puncte)

Pentru fiecare dintre itemii 1 și 2 scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Care este numărul maxim de noduri de grad 3 într-un graf neorientat cu 5 noduri (4p.)
a) 2; b) 3; c) 4; d) 5.

2. Fie un graf neorientat cu mulțimea nodurilor $\{1, 2, \dots, 2015\}$. Două noduri i și j sunt unite printr-o muchie dacă și numai dacă $\max(i, j) = 2 \cdot \min(i, j)$ sau $\max(i, j) = 2 \cdot \min(i, j) + 1$. Care este numărul de muchii ale acestui graf? **(6 p.)**
- a) 2015 ; b) 2016 ; c) 2014 ; d) $(2014 \times 2015)/2$

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

3. Considerăm codificarea binară a caracterelor, în care fiecărui simbol îi revine reprezentarea pe 8 biți a codului său ASCII. De exemplu, caracterului 'A', având codul ASCII 65, îi va corespunde reprezentarea binară 01000001. Scrieți un program C/C++ care să conțină următoarele funcții :
- a) Funcția `convert_char` primește ca argument un caracter și construiește un tablou cu 8 elemente 0 sau 1, reprezentând codificarea binară a caracterului primit. **(2 p.)**
- b) Funcția `convert_string` primește ca argument un șir de caractere `s` și construiește o matrice cu `n` linii și 8 coloane (unde `n` este lungimea șirului `s`), linia `i` a matricii reprezentând codificarea binară a caracterului de pe poziția `i` din șir. **(2 p.)**
- c) Funcția `submatrix_size` primește ca argument o matrice `m` formată doar din elemente 0 și 1 (precum și dimensiunile sale) și determină dimensiunea celei mai mari submatrici pătratică a lui `m` conținând elemente având toate aceleași valori (fie 0, fie 1). **(5 p.)**

(*Observație:* funcțiile pot avea și alte argumente față de cele specificate mai sus.) Programul va citi de la tastatură un șir de caractere `s` și va afișa rezultatul determinat de funcția `submatrix_size` aplicată pe matricea construită de `convert_string` aplicată șirului `s`. **(1 p.)**

Exemplu: Pentru șirul de caractere `s="IDEEA"`, programul va afișa 3, matricea corespunzătoare fiind :

$$m = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Fie mulțimea $S = \{1, 2, \dots, n\}$, unde $n \geq 4$ este un număr natural multiplu de 4.

Scrieți un program C/C++ care :

- a) Citește de la tastatură numărul $n \geq 4$, precum și un număr natural p ($1 \leq p \leq n/2$). În cazul în care condițiile impuse nu sunt îndeplinite, va fi afișat mesajul „date invalide”. **(2 p.)**
- b) Partiționează mulțimea dată S în două submulțimi disjuncte A și B ($S = A \cup B$, $A \cap B = \emptyset$), astfel încât suma elementelor din A să fie egală cu suma elementelor din B . **(3 p.)**
- c) Elimină elementul p din mulțimea S și creează o nouă partiție A' , B' (eventual, modificând partiția creată la punctul b), astfel încât $S \setminus \{p\} = A' \cup B'$, $A' \cap B' = \emptyset$ și suma elementelor din A' este egală cu suma elementelor din B' . În cazul în care acest lucru nu este posibil, va fi afișat mesajul „partiție inexistentă”. **(5 p.)**

Exemplu: Pentru $n = 8$, $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$, partiția inițială este $A = \{1, 3, 6, 8\}$, $B = \{2, 4, 5, 7\}$. Dacă $p = 1$ sau $p = 3$, va afișa „partiție inexistentă”. Dacă $p = 2$, partiția modificată este $A' = \{3, 6, 8\}$, $B' = \{1, 4, 5, 7\}$. Dacă $p = 4$, partiția modificată este $A' = \{2, 6, 8\}$, $B' = \{1, 3, 5, 7\}$.

SUBIECTUL III**(30 de puncte)**

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Într-o urnă se află 4 bile de culoare albă și 3 bile de culoare neagră. Se extrag bilele pe rând și se reține secvența de 7 culori obținută. Câte astfel de secvențe distincte sunt? **(4 p.)**
- a) 210; b) 35; c) 70; d) 840

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

2. Pentru funcțiile F1 și F2 definite mai jos, ce valoare va returna apelul F1(34)? **(6 p.)**

<pre>int F2(int x); int F1(int x) { if (x < 7) { return 3 + x; } else { return 2 + F2(x - 2); } }</pre>	<pre>int F2(int x) { if (x < 10) { return 3 * x; } else { return 2 * F1(x / 2); } }</pre>
--	--

3. Un puzzle Minesweeper este o matrice de n linii și m coloane care conține la fiecare poziție numărul 0 (reprezentând un loc liber) sau -1 (reprezentând o mină). Pozițiile adiacente poziției (i, j) sunt: $\{(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1), (i, j+1), (i+1, j-1), (i+1, j), (i+1, j+1)\} \cap n\{0, \dots, n-1\} \times m\{0, \dots, m-1\}$.

O poziție (i, j) din matrice este periculoasă dacă cel puțin o poziție din cele maxim 8 poziții adiacente conține o mină. Fie (l, c) o poziție în matrice. *Zona sigură* este compusă din toate pozițiile accesibile din (l, c) urmând un drum format din poziții nepericuloase adiacente.

Zona activă conține toate pozițiile zonei sigure și pozițiile adiacente zonei sigure. Matricea rezultat are aceleași dimensiuni cu puzzle-ul și este definită astfel:

- Dacă (l, c) conține o mină, matricea rezultat va fi chiar puzzle-ul inițial.
- Dacă (l, c) nu conține o mină, dar este periculoasă, matricea rezultat conține -2 peste tot, cu excepția poziției (l, c) , care conține numărul de mine vecine.
- Altfel, matricea rezultat conține pe fiecare poziție (i, j) din zona activă numărul de mine adiacente poziției (i, j) și -2 în celelalte poziții.

<i>Exemplu</i>	(I)	(II)	(III)	(IV)
<i>Puzzle</i>	$\begin{pmatrix} -1 & 0 & 0 & -1 \\ 0 & 0 & \boxed{-1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 \\ 0 & \boxed{0} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & \boxed{0} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & -1 & 0 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \boxed{0} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
<i>Poziția (l,c)</i>	(1,2)	(2,1)	(0,2)	(1,3)
<i>Rezultat</i>	$\begin{pmatrix} -1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} -2 & -2 & -2 & -2 \\ -2 & -2 & -2 & -2 \\ -2 & 2 & -2 & -2 \\ -2 & -2 & -2 & -2 \end{pmatrix}$	$\begin{pmatrix} -2 & 1 & 0 & 0 \\ -2 & 2 & 2 & 2 \\ -2 & 2 & -2 & -2 \\ -2 & -2 & -2 & -2 \end{pmatrix}$	

- a) Scrieți matricea rezultat pentru exemplul (IV). **(5 p.)**
- b) Scrieți în limbajul C/C++ o funcție care, primind la intrare un puzzle, calculează o matrice (de aceleași dimensiuni cu puzzle-ul) care conține 0 pe pozițiile nepericuloase și 1 pe pozițiile periculoase. **(5 p.)**
- c) Scrieți în limbajul C/C++ o funcție care:
- primește ca argument o matrice reprezentând puzzle-ul Minesweeper și poziția (l, c) ;
 - construiește matricea rezultat după cum este descris mai sus. **(10 p.)**

Universitatea „Alexandru Ioan Cuza” din Iași
Facultatea de Informatică

Admitere – studii de licență
Sesiunea iulie 2015

*Test la informatică
Limbaajul Pascal*

Se acordă 10 puncte din oficiu. Timpul efectiv de lucru este de 3 ore.

SUBIECTUL I

(30 de puncte)

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Numerele reale x , y , z și t satisfac inegalitățile $x < y$ și $z < t$. Precizați care dintre expresiile Pascal de mai jos este echivalentă cu faptul că intervalele închise $[x, y]$ și $[z, t]$ au intersecția nevidă ($[x, y] \cap [z, t] \neq \emptyset$). **(4 p.)**

- a) $\text{not } ((z > y) \text{ or } (t < x))$ b) $(x \leq z) \text{ or } (y \geq t)$
c) $\text{not } ((x < z) \text{ and } (t < y))$ d) $\text{not } ((x > t) \text{ or } (y > z))$

2. Se consideră algoritmul alăturat, descris în pseudocod.

- a) Scrieți valoarea afișată de algoritmul dacă numărul n citit este 213521. **(6 p.)**
b. Care este cel mai mic număr natural format din patru cifre distincte care poate fi citit în variabila n astfel încât algoritmul să afișeze valoarea 1? **(6 p.)**
c) Scrieți o secvență de instrucțiuni care să folosească doar operații de adunare și scădere și care să fie echivalentă cu instrucțiunea $n \leftarrow [n / 10]$. **(4 p.)**
d) Scrieți programul Pascal corespunzător algoritmului alăturat. **(10 p.)**

```

citește n      (număr natural)
x ← n % 10;    m ← 1;    s ← 1
cât timp n > 9 execută
  n ← [n / 10];    y ← n % 10
  dacă (y-x)*m < 0 atunci
    dacă m > 0 atunci m ← 1
    altfel s ← 0
  x ← y
scrie s

```

SUBIECTUL II

(30 de puncte)

Pentru fiecare dintre itemii 1 și 2 scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Care este numărul maxim de noduri de grad 3 într-un graf neorientat cu 5 noduri? **(4 p.)**
a) 2; b) 3; c) 4; d) 5.
2. Fie un graf neorientat cu mulțimea nodurilor $\{1, 2, \dots, 2015\}$. Două noduri i și j sunt unite printr-o muchie dacă și numai dacă $\max(i, j) = 2 \cdot \min(i, j)$ sau $\max(i, j) = 2 \cdot \min(i, j) + 1$. Care este numărul de muchii ale acestui graf? **(6 p.)**
a) 2015; b) 2016; c) 2014; d) $(2014 \times 2015) / 2$

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

3. Considerăm codificarea binară a caracterelor, în care fiecărui simbol îi revine reprezentarea pe 8 biți a codului său ASCII. De exemplu, caracterului 'A', având codul ASCII 65, îi va corespunde reprezentarea binară 01000001. Scrieți un program Pascal care să conțină următoarele subprograme (funcții sau proceduri, la alegere):
 - a) Subprogramul `convert_char` primește ca argument un caracter și construiește un tablou cu 8 elemente 0 sau 1, reprezentând codificarea binară a caracterului primit. **(2 p.)**
 - b) Subprogramul `convert_string` primește ca argument un șir de caractere `s` și construiește o matrice cu `n` linii și 8 coloane (unde `n` este lungimea șirului `s`), linia `i` a matricii reprezentând codificarea binară a caracterului de pe poziția `i` din șir. **(2 p.)**
 - c) Subprogramul `submatrix_size` primește ca argument o matrice `m` formată doar din elemente 0 și 1 (precum și dimensiunile sale) și determină dimensiunea celei mai mari submatrici pătratică a lui `m` conținând elemente având toate aceeași valoare (fie 0, fie 1). **(5 p.)**

(*Observație*: subprogramele pot avea și alte argumente față de cele specificate mai sus.) Programul va citi de la tastatură un șir de caractere `s` și va afișa rezultatul determinat de subprogramul `submatrix_size` aplicat pe matricea construită de `convert_string` aplicată șirului `s`. **(1 p.)**

Exemplu: Pentru șirul de caractere `s="IDEEA"`, programul va afișa 3, matricea corespunzătoare fiind:

$$m = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Fie mulțimea $S = \{1, 2, \dots, n\}$, unde $n \geq 4$ este un număr natural multiplu de 4. Scrieți un program Pascal care:
 - a) Citește de la tastatură numărul $n \geq 4$, precum și un număr natural p ($1 \leq p \leq n/2$). În cazul în care condițiile impuse nu sunt îndeplinite, va fi afișat mesajul "date invalide". **(2 p.)**
 - b) Partiționează mulțimea dată S în două submulțimi disjuncte A și B ($S = A \cup B$, $A \cap B = \emptyset$) astfel încât suma elementelor din A să fie egală cu suma elementelor din B . **(3 p.)**
 - c) Elimină elementul p din mulțimea S și creează o nouă partiție A' , B' (eventual, modificând partiția creată la punctul b) astfel încât $S \setminus \{p\} = A' \cup B'$, $A' \cap B' = \emptyset$ și suma elementelor din A' este egală cu suma elementelor din B' . În cazul în care acest lucru nu este posibil, va fi afișat mesajul „partiție inexistentă”. **(5 p.)**

Exemplu: Pentru $n = 8$, $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$, partiția inițială este $A = \{1, 3, 6, 8\}$, $B = \{2, 4, 5, 7\}$. Dacă $p = 1$ sau $p = 3$, va afișa „partiție inexistentă”. Dacă $p = 2$, partiția modificată este $A' = \{3, 6, 8\}$, $B' = \{1, 4, 5, 7\}$. Dacă $p = 4$, partiția modificată este $A' = \{2, 6, 8\}$, $B' = \{1, 3, 5, 7\}$.

SUBIECTUL III

(30 de puncte)

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Într-o urnă se află 4 bile de culoare albă și 3 bile de culoare neagră. Se extrag bilele pe rând și se reține secvența de 7 culori obținută. Câte astfel de secvențe distincte sunt? **(4 p.)**
- a) 210; b) 35; c) 70; d.) 840

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

2. Pentru funcțiile F1 și F2 definite mai jos, ce valoare va returna apelul F1(34)? **(6 p.)**

```
function f2(x : integer) : integer;
  forward;
function f1(x : integer) : integer;
begin
  if (x < 7) then f1 := 3 + x
  else f1 := 2 + f2(x - 2);
end;
```

```
function f2(x : integer) : integer;
begin
  if (x < 10) then f2 := 3 * x
  else f2 := 2 * f1(x div 2);
end;
```

3. Un puzzle Minesweeper este o matrice de n linii și m coloane care conține la fiecare poziție numărul 0 (reprezentând un loc liber) sau -1 (reprezentând o mină). Pozițiile adiacente poziției (i, j) sunt: $\{(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1), (i, j+1), (i+1, j-1), (i+1, j), (i+1, j+1)\} \cap n\{1 \dots n\} \times m\{1 \dots m\}$.

O poziție (i, j) din matrice este *periculoasă* dacă cel puțin o poziție din cele maxim 8 poziții adiacente conține o mină. Fie (l, c) o poziție în matrice. *Zona sigură* este compusă din toate pozițiile accesibile din (l, c) urmând un drum format din poziții nepericuloase adiacente.

Zona activă conține toate pozițiile zonei sigure și pozițiile adiacente zonei sigure. Matricea rezultat are aceleași dimensiuni cu puzzle-ul și este definită astfel:

- Dacă (l, c) conține o mină, matricea rezultat va fi chiar puzzle-ul inițial.
- Dacă (l, c) nu conține o mină, dar este periculoasă, matricea rezultat conține -2 peste tot, cu excepția poziției (l, c) , care conține numărul de mine vecine.
- Altfel, matricea rezultat conține pe fiecare poziție (i, j) din zona activă numărul de mine adiacente poziției (i, j) și -2 în celelalte poziții.

Exemplu	(I)	(II)	(III)	(IV)
Puzzle	$\begin{pmatrix} -1 & 0 & 0 & -1 \\ 0 & 0 & \boxed{-1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 \\ 0 & \boxed{0} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & \boxed{0} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & -1 & 0 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \boxed{0} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
Poziția (l, c)	(2, 3)	(3, 2)	(1, 3)	(2, 4)
Rezultat	$\begin{pmatrix} -1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} -2 & -2 & -2 & -2 \\ -2 & -2 & -2 & -2 \\ -2 & 2 & -2 & -2 \\ -2 & -2 & -2 & -2 \end{pmatrix}$	$\begin{pmatrix} -2 & 1 & 0 & 0 \\ -2 & 2 & 2 & 2 \\ -2 & 2 & -2 & -2 \\ -2 & -2 & -2 & -2 \end{pmatrix}$	

- a) Scrieți matricea rezultat pentru exemplul (IV).

(5 p.)

- b) Scrieți în limbajul Pascal un subprogram (funcție sau procedură, la alegere) care, primind la intrare un puzzle, calculează o matrice (de aceeași dimensiuni cu puzzle-ul) care conține 0 pe pozițiile nepericuloase și 1 pe pozițiile periculoase. **(5 p.)**
- c) Scrieți în limbajul Pascal un subprogram (funcție sau procedură, la alegere) care :
- primește ca argument o matrice reprezentând puzzle-ul Minesweeper și poziția (l, c) ;
 - construiește matricea rezultat după cum este descris mai sus. **(10 p.)**

Universitatea „Alexandru Ioan Cuza” din Iași
Facultatea de Informatică

Admitere – studii de licență
Sesiunea iulie 2015

Proba scrisă la informatică

Barem de evaluare și notare comun pentru limbajele C/C++ și Pascal

- Se punctează oricare alte modalități de rezolvare corectă a cerințelor.
- Nu se acordă punctaje intermediare, altele decât cele precizate explicit prin barem. Nu se acordă fracțiuni de punct.
- Se acordă 10 puncte din oficiu. Nota finală se calculează prin împărțirea punctajului total acordat pentru lucrare la 10.
- În cazul în care răspunsul final la o întrebare care nu necesită justificare nu este corect, dar există justificări parțial corecte, punctajul se calculează conform baremului afișat.

SUBIECTUL I

(30 de puncte)

1.	a	4 p.	
2.	a) Valoarea returnată este 0	6 p.	
	1. descrierea calculului	4 p.	
	2. obținerea rezultatului corect	2 p.	
	b) Cel mai mic număr natural conform cu cerința este 1230.	6 p.	
	1. identificarea proprietății de munte	2 p.	
	2. utilizarea a patru cifre distincte	2 p.	
	3. obținerea valorii 1230	2 p.	
	c) Pentru algoritm corect	4 p.	
	1. folosirea unei structuri repetitive cât timp sau repetă	2 p.	
	2. scrierea corectă a secvenței	2 p.	
	d) Pentru program corect	10 p.	
	1. structura corectă a programului	2 p.	
	2. citirea corectă a parametrului n	1 p.	
	3. instrucțiune repetitivă corectă	3 p.	
	4. instrucțiuni de decizie corecte	3 p.	
	5. afișarea corectă a valorii s	1 p.	

SUBIECTUL II**(30 de puncte)**

1.	c	4 p.	
2.	Răspuns corect c	6 p.	
	1. stabilirea proprietății de arbore	2 p.	
	2. un arbore cu n noduri are $n - 1$ muchii	2 p.	
	3. obținerea valorii 2014	2 p.	
3.	Pentru soluție corectă	10 p.	
	1. citirea datelor de intrare	1 p.	
	2. alocarea memoriei	1 p.	
	3. lucrul cu șiruri de caractere	1 p.	
	4. structura corectă <code>convert_char</code>	1 p.	
	5. structura corectă <code>convert_string</code>	1 p.	
	6. funcția (secvența de cod) care determină cea mai mare submatrice având colțul stânga sus la o anumită poziție a matricii	3 p.	
	7. parcurgerea pozițiilor matricii date	1 p.	
	8. structura corectă <code>submatrix_size</code>	1 p.	
4.	Pentru soluție corectă	10 p.	
	1. citirea datelor de intrare	1 p.	
	2. validarea datelor de intrare	1 p.	
	3. observația că există un număr par de perechi de forma $(k, (n + 1) - k)$	1 p.	
	4. crearea corectă a partiției inițiale, mulțimea A având elemente impare pe primele $n/4$ poziții, iar B elemente pare pe primele $n/4$ poziții	2 p.	
	5. observația că dacă p este număr impar, $S \setminus \{p\}$ nu poate fi partiționată	1 p.	
	6. observația că dacă p este par, $p \in B$, $\text{suma}(B)$ scade cu $p/2$	1 p.	
	7. observația că dacă $p/2$ este impar, $p/2 \in A$, partiția modificată este $A' = A \setminus \{p/2\}$, $B' = B \cup \{p/2\}$	1 p.	
	8. observația că dacă $p/2$ este par, $1 + p/2 \in A$, $A' = A \setminus \{1 + p/2\} \setminus \{1\} \cup \{2\}$, $B' = B \cup \{1 + p/2\} \cup \{1\} \setminus \{2\}$	1 p.	
	9. crearea corectă a partițiilor A', B'	1 p.	

SUBIECTUL III

(30 de puncte)

1.	b	4 p.	
	Răspuns corect : 74	6 p.	
2.	1. observarea faptului că funcțiile F1 și F2 sunt mutual recursive	1 p.	
	2. calculul recursiv (lista tuturor apelurilor recursive)	3 p.	
	3. obținerea rezultatului corect	2 p.	
3.	a) Pentru răspuns corect – 2 10 0 110 0 001 1 0 0 1 -2	5 p. 5 p.	
	b) Pentru soluție corectă 1. parcurgerea tuturor pozițiilor matricii 2. parcurgerea tuturor vecinilor pentru poziția curentă 3. determinarea corectă a faptului că poziția curentă este periculoasă sau nu	5 p. 1 p. 2 p. 2 p.	
	c) Pentru soluție corectă 1. identificarea și tratarea corectă a cazului particular în care (l, c) conține o mină 2. identificarea și tratarea corectă a cazului particular în care (l, c) nu conține mină și este periculoasă 3. găsirea zonei sigure printr-un algoritm de tip <i>flood-fill</i> (bfs, dfs sau ad-hoc) 4. identificarea pozițiilor adiacente zonei sigure 5. calculul numărului de vecini care conțin mine pentru fiecare poziție din zona activă (sigură + poziții adiacente zonei sigure) 6. determinarea rezultatului corect pentru cazul general	10 p. 2 p. 2 p. 3 p. 1 p. 1 p. 1 p.	

Universitatea „Alexandru Ioan Cuza” din Iași
Facultatea de Informatică

Admitere – studii de licență
Sesiunea iulie 2014

Test la informatică
Limbajul C/C++

Se acordă 10 puncte din oficiu. Timpul efectiv de lucru este de 3 ore.

SUBIECTUL I

(30 de puncte)

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Precizați care dintre expresiile C/C++ de mai jos este echivalentă cu relația $a \in [-4, -1] \cup [2, 4]$. **(4p.)**

- a) $!((a < -4 \parallel a > -1) \&\& (a \geq 2 \parallel a \leq 4))$
- b) $!(a \geq -4 \&\& a \leq -1 \&\& a \geq 2 \&\& a \leq 4)$
- c) $(a \geq -4 \parallel a \leq -1) \&\& (a \geq 2 \parallel a \leq 4)$
- d) $!(a < -4 \parallel a > -1) \parallel !(a < 2 \parallel a > 4)$

2. Se consideră algoritmul alăturat, descris în pseudocod.

- a) Scrieți valoarea afișată de algoritmul dacă numărul n citit este 91. **(6 p.)**
- b) Care sunt cea mai mică, respectiv cea mai mare valoare pe care o poate lua n în intervalul $[1, 300]$ astfel încât algoritmul să afișeze valoarea 11 **(6 p.)**
- c) Înlocuiți instrucțiunea $x \leftarrow n \% y$ cu o secvență echivalentă de instrucțiuni care folosește doar adunări/scăderi repetate. **(4 p.)**
- d) Scrieți programul C/C++ corespunzător algoritmului alăturat. **(10 p.)**

```

citește n
    (număr natural nenul)
x ← 1
y ← 2
cât timp x ≠ 0 și y*y ≤ n execută
    x ← n % y
    y ← y + 1
scrie y-1

```

SUBIECTUL II

(30 de puncte)

Pentru fiecare dintre itemii 1 și 2 scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

- 1. Fie un arbore binar cu 15 noduri. Numărul nodurilor cu cel puțin un fiu este cel mult : **(4 p.)**
 - a) 14 ;
 - b) 8 ;
 - c) 7 ;
 - d) 1.
- 2. Se consideră un graf neorientat cu 8 vârfuri numerotate de la 1 la 8 și următoarele muchii : $\{1, 7\}, \{1, 8\}, \{3, 4\}, \{3, 5\}, \{3, 6\}, \{3, 7\}, \{4, 7\}, \{5, 6\}, \{5, 8\}, \{6, 7\}, \{6, 8\}, \{7, 8\}$. Care este numărul minim de culori cu care pot fi colorate vârfurile grafului astfel încât oricare două vârfuri adiacente să aibă culori diferite? **(6 p.)**
 - a) 2 ;
 - b) 3 ;
 - c) 4 ;
 - d) 8.

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

3. O matrice are proprietatea ParImpar dacă fiecare dintre liniile sale este formată fie doar din numere întregi pare, fie doar din numere întregi impare. Scrieți un program C/C++ care :
- Citește de la tastatură un număr natural $n \geq 2$ și o matrice pătratică A de dimensiune $n \times n$, calculează matricea produs $B = A \times A$ și verifică dacă B are proprietatea ParImpar , conform definiției de mai sus. **(6 p.)**
 - Justificați faptul că, dacă matricea A are proprietatea ParImpar , atunci și matricea B are proprietatea ParImpar . **(4 p.)**
4. Scrieți un program C/C++ care :
- Citește de la tastatură un număr natural $n \geq 2$ și un vector w_1, \dots, w_n de numere întregi. **(2 p.)**
 - Construiește o permutare w'_1, \dots, w'_n a vectorului inițial w_1, \dots, w_n , astfel încât oricare două elemente aflate pe poziții consecutive în permutare sunt diferite : $w'_i \neq w'_{i+1}$, $\forall i = 1..n - 1$. Dacă nu există o astfel de permutare, atunci algoritmul va afișa nu există. **(8 p.)**

Exemple :

- pentru vectorul 1, 1, 1, 2, 2, 2, 3, 3, permutarea ar putea fi 3, 2, 1, 3, 2, 1, 2, 1 ;
- dacă vectorul este 1, 1, 1, 2, atunci permutarea nu poate fi creată.

SUBIECTUL III

(30 de puncte)

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. La un concurs participă 4 elevi, iar concursul se desfășoară în două săli : sala A și sala B. Fiecare sală are capacitatea de maxim 3 locuri. În câte moduri pot fi împărțiți elevii în sălile A și B ? (ordinea elevilor în cadrul unei săli nu contează) **(4 p.)**
- a) 8 ; b) 14 ; c) 16 ; d) 18.

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

2. Pentru funcția C definită alăturat, ce valoare va returna apelul $C(5, 3)$? **(6 p.)**
- ```
int C(int n, int k) {
 if (k == 0 || n == k) return 1;
 return C(n - 1, k - 1) + C(n - 1, k);
}
```

3. O grilă SUDOKU este o matrice  $9 \times 9$  care respectă următoarele proprietăți :

- fiecare element al matricii este un număr natural între 1 și 9 ;
- fiecare linie conține toate numerele naturale de la 1 la 9 ;
- fiecare coloană conține toate numerele naturale de la 1 la 9 ;
- fiecare dintre cele 9 submatrici de dimensiune  $3 \times 3$ , evidențiate prin linii îngroșate în exemplul alăturat, conține toate numerele de la 1 la 9.

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 6 | 2 | 8 | 5 | 3 | 4 | 9 | 1 | 7 |
| 5 | 1 | 9 | 8 | 7 | 2 | 4 | 3 | 6 |
| 4 | 3 | 7 | 9 | 1 | 6 | 2 | 5 | 8 |
| 8 | 6 | 5 | 2 | 4 | 7 | 1 | 9 | 3 |
| 3 | 9 | 2 | 1 | 8 | 5 | 7 | 6 | 4 |
| 7 | 4 | 1 | 6 | 9 | 3 | 5 | 8 | 2 |
| 2 | 5 | 4 | 3 | 6 | 9 | 8 | 7 | 1 |
| 1 | 7 | 6 | 4 | 5 | 8 | 3 | 2 | 9 |
| 9 | 8 | 3 | 7 | 2 | 1 | 6 | 4 | 5 |

Un puzzle SUDOKU este o matrice  $9 \times 9$  completată parțial cu numere naturale de la 1 la 9. Mai jos este un exemplu de puzzle SUDOKU. O soluție a unui astfel de puzzle este o grilă SUDOKU care coincide cu puzzle-ul pe pozițiile precompletate.

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 8 | 7 | 3 |   | 9 | 4 | 1 |
| 6 |   | 9 | 8 | 2 | 4 | 3 |   | 7 |
| 4 |   | 7 |   | 1 | 5 | 2 | 6 |   |
| 3 | 9 | 5 | 2 | 7 |   | 4 |   | 6 |
|   | 6 | 2 | 4 |   | 8 | 1 |   | 5 |
| 8 | 4 |   | 6 | 5 |   | 7 | 2 | 9 |
| 1 | 8 | 4 | 3 | 6 | 9 | 5 | 7 | 2 |
|   | 7 |   | 1 | 4 | 2 |   | 9 | 3 |
| 9 | 2 | 3 | 5 | 8 | 7 | 6 | 1 | 4 |

- Găsiți o soluție pentru puzzle-ul SUDOKU de mai sus. **(4 p.)**
- Descrieți în limbaj natural un algoritm pentru rezolvarea unui puzzle SUDOKU. **(6 p.)**
- Scrieți în limbajul C/C++ o funcție care :
  - primește ca argument o matrice reprezentând puzzle-ul SUDOKU (celulele necompletate ale puzzle-ului sunt reprezentate în matrice de cifra 0);
  - returnează o matrice reprezentând soluția puzzle-ului. Dacă problema nu are soluție, matricea returnată va conține pe toate liniile și coloanele doar cifra 0. **(10 p.)**

Universitatea „Alexandru Ioan Cuza” din Iași  
Facultatea de Informatică

Admitere - studii de licență  
Sesiunea iulie 2014

*Test la INFORMATICA*  
*Limbajul Pascal*

Se acordă 10 puncte din oficiu. Timpul efectiv de lucru este de 3 ore.

**SUBIECTUL I**

**(30 de puncte)**

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

- Precizați care dintre expresiile Pascal de mai jos este echivalentă cu relația  $a \in [-4, -1] \cup [2, 4]$ . **(4 p.)**
  - $\text{not}((a < -4) \text{ or } (a > -1) \text{ and } (a \geq 2) \text{ or } (a \leq 4))$
  - $\text{not}((a \geq -4) \text{ and } (a \leq -1) \text{ and } (a \geq 2) \text{ and } (a \leq 4))$
  - $((a \geq -4) \text{ or } (a \leq -1)) \text{ and } ((a \geq 2) \text{ or } (a \leq 4))$
  - $\text{not}((a < -4) \text{ or } (a > -1)) \text{ or } \text{not}((a < 2) \text{ or } (a > 4))$
- Se consideră algoritmul alăturat, descris în pseudocod.
 

|                                                                                                                                                                                                                                                                                                               |                                                                                                                                                              |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none"> <li>Scrieți valoarea afișată de algoritm dacă numărul n citit este 91. <b>(6 p.)</b></li> <li>Care sunt cea mai mică, respectiv cea mai mare valoare pe care o poate lua n în intervalul [1, 300] astfel încât algoritmul să afișeze valoarea 11? <b>(6 p.)</b></li> </ol> | <pre> citește n     (număr natural nenul) x ← 1 y ← 2 cât timp x ≠ 0 și y*y ≤ n     execută         x ← n mod y         y ← y + 1 scrie y-1           </pre> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|

- c) Înlocuiți instrucțiunea  $x \leftarrow n \bmod y$  cu o secvență echivalentă de instrucțiuni care folosește doar adunări/scăderi repetate. **(4 p.)**
- d) Scrieți programul Pascal corespunzător algoritmului alăturat. **(10 p.)**

## SUBIECTUL II

**(30 de puncte)**

Pentru fiecare dintre itemii 1 și 2, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. Fie un arbore binar cu 15 noduri. Numărul nodurilor cu cel puțin un fiu este cel mult : **(4 p.)**
  - a) 14 ;
  - b) 8 ;
  - c) 7 ;
  - d) 1.
2. Se consideră un graf neorientat cu 8 vârfuri numerotate de la 1 la 8 și următoarele muchii :  $\{1, 7\}, \{1, 8\}, \{3, 4\}, \{3, 5\}, \{3, 6\}, \{3, 7\}, \{4, 7\}, \{5, 6\}, \{5, 8\}, \{6, 7\}, \{6, 8\}, \{7, 8\}$ . Care este numărul minim de culori cu care pot fi colorate vârfurile grafului astfel încât oricare două vârfuri adiacente să aibă culori diferite ? **(6 p.)**
  - a) 2 ;
  - b) 3 ;
  - c) 4 ;
  - d) 8.

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

3. O matrice are proprietatea  $\text{ParImpar}$  dacă fiecare dintre liniile sale este formată fie doar din numere întregi pare, fie doar din numere întregi impare. Scrieți un program Pascal care :
  - a) Citește de la tastatură un număr natural  $n \geq 2$  și o matrice pătratică  $A$  de dimensiune  $n \times n$ , calculează matricea produs  $B = A \times A$  și verifică dacă  $B$  are proprietatea  $\text{ParImpar}$ , conform definiției de mai sus. **(6 p.)**
  - b) Justificați faptul că, dacă matricea  $A$  are proprietatea  $\text{ParImpar}$ , atunci și matricea  $B$  are proprietatea  $\text{ParImpar}$ . **(4 p.)**
4. Scrieți un program Pascal care :
  - a) Citește de la tastatură un număr natural  $n \geq 2$  și un vector  $w'_1 \dots w'_n$  de numere întregi. **(2 p.)**
  - b) Construiește o permutare  $w'_1 \dots w'_n$  a vectorului inițial  $w'_1 \dots w'_n$  astfel încât oricare două elemente aflate pe poziții consecutive în permutare sunt diferite :  $w'_i + w'_{i+1}, \forall i = 1..n - 1$ . Dacă nu există o astfel de permutare, atunci algoritmul va afișa nu există. **(8 p.)**

*Exemple :*

- pentru vectorul 1, 1, 1, 2, 2, 2, 3, 3, permutarea ar putea fi 3, 2, 1, 3, 2, 1, 2, 1 ;
- dacă vectorul este 1, 1, 1, 2, atunci permutarea nu poate fi creată.

## SUBIECTUL III

(30 de puncte)

Pentru itemul 1, scrieți pe foaia de examen litera corespunzătoare răspunsului corect.

1. La un concurs participă 4 elevi, iar concursul se desfășoară în două săli: sala A și sala B. Fiecare sală are capacitatea de maxim 3 locuri. În câte moduri pot fi împărțiți elevii în sălile A și B? (ordinea elevilor în cadrul unei săli nu contează) **(4 p.)**

a) 8;                      b) 14;                      c) 16;                      d) 18.

Scrieți pe foaia de examen răspunsul pentru fiecare dintre cerințele următoare.

2. Pentru funcția  $C$  definită alăturat, ce valoare va returna apelul  $C(5, 3)$ ? **(6 p.)**

```
function C(n,k:integer):integer;
begin
 if (k = 0) or (n = k) then
 C := 1
 else
 C := C(n-1,k-1)+C(n-1,k);
 end;
```

3. O grilă SUDOKU este o matrice  $9 \times 9$  care respectă următoarele proprietăți:

1. fiecare element al matricii este un număr natural între 1 și 9;
2. fiecare linie conține toate numerele naturale de la 1 la 9;
3. fiecare coloană conține toate numerele naturale de la 1 la 9;
4. fiecare dintre cele 9 submatrici de dimensiune  $3 \times 3$ , evidențiate prin linii îngroșate în exemplul alăturat, conține toate numerele de la 1 la 9.

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 6 | 2 | 8 | 5 | 3 | 4 | 9 | 1 | 7 |
| 5 | 1 | 9 | 8 | 7 | 2 | 4 | 3 | 6 |
| 4 | 3 | 7 | 9 | 1 | 6 | 2 | 5 | 8 |
| 8 | 6 | 5 | 2 | 4 | 7 | 1 | 9 | 3 |
| 3 | 9 | 2 | 1 | 8 | 5 | 7 | 6 | 4 |
| 7 | 4 | 1 | 6 | 9 | 3 | 5 | 8 | 2 |
| 2 | 5 | 4 | 3 | 6 | 9 | 8 | 7 | 1 |
| 1 | 7 | 6 | 4 | 5 | 8 | 3 | 2 | 9 |
| 9 | 8 | 3 | 7 | 2 | 1 | 6 | 4 | 5 |

Un puzzle SUDOKU este o matrice  $9 \times 9$  completată parțial cu numere naturale de la 1 la 9. Mai jos este un exemplu de puzzle SUDOKU. O soluție a unui astfel de puzzle este o grilă SUDOKU care coincide cu puzzle-ul pe pozițiile precompletate.

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 8 | 7 | 3 |   | 9 | 4 | 1 |
| 6 |   | 9 | 8 | 2 | 4 | 3 |   | 7 |
| 4 |   | 7 |   | 1 | 5 | 2 | 6 |   |
| 3 | 9 | 5 | 2 | 7 |   | 4 |   | 6 |
|   | 6 | 2 | 4 |   | 8 | 1 |   | 5 |
| 8 | 4 |   | 6 | 5 |   | 7 | 2 | 9 |
| 1 | 8 | 4 | 3 | 6 | 9 | 5 | 7 | 2 |
|   | 7 |   | 1 | 4 | 2 |   | 9 | 3 |
| 9 | 2 | 3 | 5 | 8 | 7 | 6 | 1 | 4 |

- a) Găsiți o soluție pentru puzzle-ul SUDOKU de mai sus. **(4 p.)**
- b) Descrieți în limbaj natural un algoritm pentru rezolvarea unui puzzle SUDOKU. **(6 p.)**
- c) Scrieți în limbajul Pascal o funcție care:
  - primește ca argument o matrice reprezentând puzzle-ul SUDOKU (celulele necompletate ale puzzle-ului sunt reprezentate în matrice de cifra 0);
  - returnează o matrice reprezentând soluția puzzle-ului. Dacă problema nu are soluție, matricea returnată va conține pe toate liniile și coloanele doar cifra 0. **(10 p.)**

Universitatea „Alexandru Ioan Cuza” din Iași  
Facultatea de Informatică

Admitere – studii de licență  
Sesiunea iulie 2014

*Proba scrisă la informatică*

*Barem de evaluare și notare comun pentru limbajele C/C++ și Pascal*

- Se punctează oricare alte modalități de rezolvare corectă a cerințelor.
- Se acordă 10 puncte din oficiu. Nota finală se calculează prin împărțirea la 10 a punctajului total acordat pentru lucrare.
- În cazul în care răspunsul final la o întrebare care nu necesită justificare nu este corect, dar există justificări parțial corecte, punctajul se calculează conform baremului afișat.

**SUBIECTUL I**

**(30 de puncte)**

|    |                                                                                                                                                                                    |                                              |  |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|--|
| 1. | d                                                                                                                                                                                  | <b>4 p.</b>                                  |  |
| 2. | a) <b>Valoarea returnată este 7</b><br>1. descrierea calculului<br>2. obținerea rezultatului corect                                                                                | <b>6 p.</b><br>4 p.<br>2 p.                  |  |
|    | b) Cea mai mică valoare a lui n este 121<br>Cea mai mare valoare a lui n este 253                                                                                                  | <b>3 p.</b><br>3 p.                          |  |
|    | c) <b>Pentru algoritm corect</b><br>1. folosirea unei structuri repetitive cât timp sau repetă<br>2. scrierea corectă a algoritmului                                               | <b>4 p.</b><br>2 p.<br>2 p.                  |  |
|    | d) <b>Pentru program corect</b><br>1. structura corectă a programului<br>2. citirea corectă a parametrului n<br>3. instrucțiunea repetitivă corectă<br>4. afișarea expresiei y – 1 | <b>10 p.</b><br>3 p.<br>1 p.<br>5 p.<br>1 p. |  |

**SUBIECTUL II**

**(30 de puncte)**

|    |                                                                                                                                                                                                                       |                                                           |  |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|--|
| 1. | a)                                                                                                                                                                                                                    | <b>4 p.</b>                                               |  |
| 2. | b)                                                                                                                                                                                                                    | <b>6 p.</b>                                               |  |
| 3. | a) <b>Pentru soluție corectă</b><br>1. citirea datelor de intrare<br>2. calculul matricii produs B<br>3. verificarea proprietății ParImpar                                                                            | <b>6 p.</b><br>1 p.<br>2 p.<br>3 p.                       |  |
| 4. | b) <b>Pentru răspuns corect</b><br>a) <b>Pentru soluție corectă</b><br>b) <b>Pentru soluție corectă</b><br>1. crearea corectă a permutării în cazul în care există<br>2. descrierea cazului în care nu există soluție | <b>4 p.</b><br><b>2 p.</b><br><b>8 p.</b><br>5 p.<br>3 p. |  |

## SUBIECTUL III

(30 de puncte)

|    |                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                               |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
|----|----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|------|--|
| 1. | b                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 4 p.                                          |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
|    | <b>Răspuns corect : 10</b>                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 6 p.                                          |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
| 2. | 1. observarea faptului că funcția C este recursivă |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 1 p.                                          |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
|    | 2. calculul lui C(5, 3)                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 3 p.                                          |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
|    | 3. obținerea rezultatului corect                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 2 p.                                          |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
| 3. | a)                                                 | <b>Pentru răspuns corect</b><br><table border="1"><tr><td>2</td><td>5</td><td>8</td><td>7</td><td>3</td><td>6</td><td>9</td><td>4</td><td>1</td></tr><tr><td>6</td><td>1</td><td>9</td><td>8</td><td>2</td><td>4</td><td>3</td><td>5</td><td>7</td></tr><tr><td>4</td><td>3</td><td>7</td><td>9</td><td>1</td><td>5</td><td>2</td><td>6</td><td>8</td></tr><tr><td>3</td><td>9</td><td>5</td><td>2</td><td>7</td><td>1</td><td>4</td><td>8</td><td>6</td></tr><tr><td>7</td><td>6</td><td>2</td><td>4</td><td>9</td><td>8</td><td>1</td><td>3</td><td>5</td></tr><tr><td>8</td><td>4</td><td>1</td><td>6</td><td>5</td><td>3</td><td>7</td><td>2</td><td>9</td></tr><tr><td>1</td><td>8</td><td>4</td><td>3</td><td>6</td><td>9</td><td>5</td><td>7</td><td>2</td></tr><tr><td>5</td><td>7</td><td>6</td><td>1</td><td>4</td><td>2</td><td>8</td><td>9</td><td>3</td></tr><tr><td>9</td><td>2</td><td>3</td><td>5</td><td>8</td><td>7</td><td>6</td><td>1</td><td>4</td></tr></table> | 2                                             | 5 | 8 | 7 | 3 | 6 | 9 | 4 | 1 | 6 | 1 | 9 | 8 | 2 | 4 | 3 | 5 | 7 | 4 | 3 | 7 | 9 | 1 | 5 | 2 | 6 | 8 | 3 | 9 | 5 | 2 | 7 | 1 | 4 | 8 | 6 | 7 | 6 | 2 | 4 | 9 | 8 | 1 | 3 | 5 | 8 | 4 | 1 | 6 | 5 | 3 | 7 | 2 | 9 | 1 | 8 | 4 | 3 | 6 | 9 | 5 | 7 | 2 | 5 | 7 | 6 | 1 | 4 | 2 | 8 | 9 | 3 | 9 | 2 | 3 | 5 | 8 | 7 | 6 | 1 | 4 | 4 p. |  |
| 2  | 5                                                  | 8                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 7                                             | 3 | 6 | 9 | 4 | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
| 6  | 1                                                  | 9                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 8                                             | 2 | 4 | 3 | 5 | 7 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
| 4  | 3                                                  | 7                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 9                                             | 1 | 5 | 2 | 6 | 8 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
| 3  | 9                                                  | 5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 2                                             | 7 | 1 | 4 | 8 | 6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
| 7  | 6                                                  | 2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 4                                             | 9 | 8 | 1 | 3 | 5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
| 8  | 4                                                  | 1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 6                                             | 5 | 3 | 7 | 2 | 9 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
| 1  | 8                                                  | 4                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 3                                             | 6 | 9 | 5 | 7 | 2 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
| 5  | 7                                                  | 6                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 1                                             | 4 | 2 | 8 | 9 | 3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
| 9  | 2                                                  | 3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 5                                             | 8 | 7 | 6 | 1 | 4 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
|    | b)                                                 | <b>Pentru soluție corectă</b><br>1. descrierea structurii de date<br>2. descrierea structurii unui algoritm de tip <i>backtracking</i><br>3. testarea validității unei configurații parțiale                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | 6 p.<br>2 p.<br>3 p.<br>1 p.                  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |
|    | c)                                                 | <b>Pentru funcție corectă</b><br>1 structura funcției (argumente, tip returnat)<br>2. validarea unei grile parțiale<br>3. implementarea unui algoritm valid<br>4. returnarea soluției<br>5. determinarea inexistenței soluției                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 10 p.<br>1 p.<br>2 p.<br>5 p.<br>1 p.<br>1 p. |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |  |

**C. Subiecte date la concursurile de obținere  
a definitivatului și/sau gradului II în informatică,  
pentru profesorii din învățământul preuniversitar  
(din județele arondate centrului universitar și Inspectoratului  
Școlar Județean Iași).**

Începem cu observația că, până în anul 2012, subiectele se împărțeau în două mari categorii : *subiecte de specialitate* și *subiecte de didactică/metodică*. Acestea se alegeau (și se aleg și astăzi) dintr-o listă pusă în prealabil la dispoziția profesorilor, pe site-ul ministerului de resort (<http://www.edu.ro>). O teză avea astfel, în mod uzual, două subiecte, câte unul din fiecare categorie. De obicei, subiectul de metodică era cotate mai mult (două treimi din teză).

Pentru subiectele de specialitate (informatică), implementarea se cerea a fi făcută într-unul dintre limbajele C sau Pascal, la alegere.



Exemple :

- Proiectați clasele necesare pentru lucrul cu cercuri, fiecare cerc fiind caracterizat prin poziția centrului și lungimea razei. Se vor implementa următoarele metode :
  - a) Inițializarea unui cerc.
  - b) Calculul ariei unui cerc.
  - c) Calculul lungimii unui cerc.
  - d) Mutarea unui cerc dat într-o nouă poziție.
- Proiectați structurile de date necesare lucrului cu polinoame. Implementați :
  - a) Citirea unui polinom.
  - b) Afișarea unui polinom.
  - c) Adunarea a două polinoame.
  - d) Înmulțirea a două polinoame.
- Reprezentarea grafurilor neorientate cu ajutorul matricilor. Operații semnificative de implementat :
  - a) Inițializarea unui graf.
  - b) Obținerea grafului complementar unui graf dat.
  - c) Calculul gradului unui nod dintr-un graf dat.
  - d) Calculul lungimii drumului minim dintre două noduri ale unui graf dat.
- Generați toate permutările unei liste date.

Baremul pentru un asemenea subiect era cotelat cu un număr fix de puncte (de obicei, 10 sau 100), un număr constant de puncte fiind dat din oficiu (de obicei 1, respectiv 10). Dacă un subiect avea subsubiecte (să zicem 4), unul dintre acestea (cel considerat mai dificil) era cotelat cu mai multe puncte, celelalte fiind cotelate cu un număr egal de puncte (3, 2, 2, 2, respectiv 30, 20, 20, 20).

Subiectele de metodică erau axate pe ideea *elaborării unui proiect didactic* pentru o lecție dată. Punctajul (separat de cel anterior) era acordat în principal în funcție de corectitudinea elementelor de metodică abordate de candidat, anumite greșeli de natură conceptuală, științifică sau de implementare fiind ignorate. Elementele de metodică avute în vedere spre a fi analizate sau punctate erau următoarele : *obiectivele lecției, metoda/metodele de predare* propuse a fi utilizate pentru lecția respectivă, *materialele didactice, instrumentele de evaluare* considerate a fi utile pentru lecția respectivă, *modul propus pentru desfășurarea lecției*. Iată și câteva exemple :

- Elaborați un proiect didactic pentru o lecție cu subiectul „Metoda *greedy*”. Exemple. Tipul lecției este la alegere.
- Elaborați un proiect didactic pentru o lecție cu subiectul „Metoda *divide et impera*”. Exemple. Tipul lecției este la alegere.
- Elaborați un proiect didactic pentru o lecție cu subiectul „Metode de sortare”. Tipul lecției este la alegere.
- Elaborați un proiect didactic pentru o lecție cu subiectul „*Backtracking*”. Tipul lecției este la alegere.

După 2012, subiectele au fost doar de informatică și, probabil, această modalitate de testare nu va fi schimbată prea curând. Subiectele sunt înglobate în cadrul unei lecții (de una sau

două ore), trebuind să fie tratate corect atât din punct de vedere științific (cu implementări într-unul din limbajele C, C++ sau Pascal), cât și din punct de vedere metodic. Modalitatea de construire a *baremului complet* rămâne, în mare, cea descrisă mai sus, inclusiv punctele acordate din oficiu, elementele de metodică care contează și ponderea metodicii față de conținutul legat de specialitate. Dintre subiectele care s-au dat în ultimii ani (pentru zona Iași) amintim :

- Să se proiecteze o lecție care are ca scop predarea conceptelor de *algoritmi iterativi și recursivi*. Ca exemplificare practică, se va considera calculul elementelor șirului  $(x_n)_{n \geq 0}$  definit prin :

$$x_0 = 0, \quad x_1 = 2, \quad x_n = x_{n-1} \cdot 3 + x_{n-2} \cdot 2 + 4.$$

Se vor trata, științific și metodic, punctele :

- Implementarea unei funcții care primește ca parametru un indice  $n$  și calculează în mod iterativ elementul  $x_n$ .
  - Implementarea unei funcții care primește ca parametru un indice  $n$  și calculează în mod recursiv elementul  $x_n$ .
- Să se proiecteze o lecție care are ca scop predarea conceptului de *integrare numerică*. Ca exemplificare practică, se va considera calculul *integralei definite* a funcției :

$$f(x) = \begin{cases} x^3 + 4 \cdot x^2 + 6 \cdot x - 5, & , x \geq 0 \\ x^4 + 3 \cdot x \cdot \sin(x) + 9 & , x < 0 \end{cases}$$

Se vor trata, științific și metodic, punctele :

- Implementarea unei funcții care primește doi parametri  $a$  și  $b$ , numere reale, și calculează valoarea  $\int_b^a f(x)dx$  prin *metoda dreptunghiurilor*.
  - Implementarea unei funcții care primește doi parametri  $a$  și  $b$ , numere reale, și calculează valoarea  $\int_b^a f(x)dx$  prin *metoda trapezelor*.
- Să se proiecteze o lecție care are ca scop predarea noțiunii de *conectivitate* într-un graf. Ca exemplificare practică, se vor considera *determinarea componentelor conexe ale unui graf*. Se vor trata, științific și metodic, punctele :
    - Definirea structurii de date utilizate pentru reprezentarea în program a unui graf.
    - Implementarea unei funcții care primește ca parametri un graf (reprezentat printr-o structură de date de tipul definit mai sus) și un vârf al acestuia și returnează componenta conexă a grafului care conține vârful respectiv.
  - Să se proiecteze o lecție care are ca scop predarea *listelor înlănțuite* ca *structuri de date alocate dinamic*. Ca exemplificare practică, se vor considera *inserarea și eliminarea unui element* dintr-o *listă simplu înlănțuită*, ale cărei elemente sunt numere întregi. Se vor trata, științific și metodic, punctele :
    - Implementarea unei funcții care primește ca parametri o listă simplu înlănțuită, un indice  $n$  și o valoare întregă  $v$  și inserează în listă un nou element, pe poziția  $n$ , având valoarea  $v$ .
    - Implementarea unei funcții care primește ca parametri o listă simplu înlănțuită și o valoare întregă  $v$  și elimină din listă toate elementele având valoarea  $v$ .

- Să se proiecteze o lecție care are ca scop predarea tehnicii de programare *divide et impera*. Ca exemplificare practică, se va considera *sortarea crescătoare* a elementelor dintr-un tablou de numere întregi. Se va trata, științific și metodic, punctul :
  - a) Implementarea unei funcții care primește ca parametri un tablou de numere întregi și numărul de elemente ale acestuia și ordonează crescător elementele tabloului (utilizând, desigur, tehnica *divide et impera*).
- Să se proiecteze o lecție care are ca scop predarea conceptelor de bază ale *programării orientate pe obiecte*. Ca exemplificare practică, se va considera *definirea unei clase* destinată reprezentării numerelor complexe. Se va trata, științific și metodic, punctul :
  - a) Implementarea următoarelor *metode* : constructor, adunare cu un număr complex, adunare cu un număr real, înmulțire cu un număr complex, înmulțire cu un număr real.

---

### Observație

La examenele de titularizare, bibliografia a fost aproximativ aceeași cu cea de la examenele de grad și definitivat, cu subiecte axate mai puțin pe didactică/metodică și mai mult pe îndemânările practice legate de informatică. Cum întreaga bibliografie fixată de ministerul de resort datează aproximativ din anul 2000(!), credem că ea trebuie urgent actualizată de factorii în drept.

---



## Anexa 3

### Adrese web utile

I. Cărți, culegeri în format electronic, oferite online în mod gratuit :

- Cătălin Frâncu, *Psihologia concursurilor de informatică* – <http://www.infobits.ro/psihologia-concursurilor/>.
- Carmen Popescu, Vlad Tudor, *Competențe digitale* – <http://www.infobits.ro/tic-competente-digitale-volumul-1/> : <http://www.infobits.ro/tic-competente-digitale-volumul-2/>.
- Tudor Sorin, Vlad Tudor, *Bazele programării în Java* – <http://www.infobits.ro/java/>.
- Alina Boca, Doru Popescu Anastasiu, Vlad Tudor, Mișu Ionescu, *TIC. Clasa a VIII-a* – <http://www.infobits.ro/tic-informatica-manual-clasa-a-VIII-a/>.

II. Platforme online de pregătire pentru performanță :

- <http://campion.edu.ro/arhiva>.
- <http://infoarena.ro>.
- <http://pbinfo.ro>.
- <http://varena.ro>.

III. Platforme de pregătire/evaluare pentru examene de bacalaureat, atestat, admitere la facultate :

- Academia Oracle : <http://academy.oracle.com>.
- Bacalaureat la informatică : <http://www.infobits.ro/bacalaureat-la-informatica-2015/>.
- Laborator virtual de informatică : <http://lab.infobits.ro/>.
- Competențe digitale : <http://competentedigitale.ro/>.
- Advanced eLearning, portalul SEI (Sistem Educațional Informatizat) : <http://portal.edu.ro/>.

IV. Resurse online pentru elevi și profesori (inclusiv tehnici inovatoare de proiectare și predare de noi cursuri).

- Comunitatea online a cadrelor didactice : <http://didactic.ro>.
- <http://www.w3schools.ro>.
- <http://scratch.infobits.ro/scratch-curriculum-la-decizia-scolii.php>.
- <http://olimpiada.info>.
- <http://infoeducatia.ro>.
- <http://www.manuale-de-informatica.ro/>.
- <http://www.ciee.org/>.
- <http://www.rafonline.org/>.

V. Legătura cu MENCS (nu mai listăm și paginile institutelor de învățământ superior, liceelor importante sau inspectoratelor școlare, acestea fiind ușor de găsit) :

- Site-ul oficial : <http://www.edu.ro>.
- Subiecte date la diverse concursuri (grad, definitivat etc.) : [http://www.edu.ro/index.php/articles/\\*](http://www.edu.ro/index.php/articles/*) , unde \* poate fi c862, c876, c879 etc.
- <http://titularizare.edu.ro>.

VI. Legătura cu MCSI :

- Site-ul oficial : <http://www.mcsi.ro>.
- Portalul național pentru societatea informațională în România (e-România) : <http://www.romania.gov.ro>.

VII. Alte site-uri utile :

- Internet Society : <http://www.internetsociety.org/>.
- ISPS Studien, Berichte : <http://www.isps.ch/>.
- Agenția de Administrare a Rețelei Naționale de Informatică pentru Educație și Cercetare : <http://www.roedu.net/>.
- Link Academy blog : <http://www.link-academy.com/blog/> <L1> .
- Software pentru prezentări (Q & A) : <https://prezi.com/> <L3> .

## Bibliografie

- [AHU] V. Aho, J.E. Hopcroft, J.D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, 1983.
- [An] M. Anastasiei, *Metodica predării matematicii*, curs litografiat, Editura Universității „Al.I. Cuza”, Iași, 1985.
- [At] A. Atanasiu, „Algoritmi. Reprezentare și clasificare”, *Gazeta de Informatică*, nr. 1-2/1991, 1-3/1992.
- [Ba] D. Badea, „Didactica la școala competențelor – dominante și exemplificări ale scenariului didactic”, în *Revista de Pedagogie*, anul LIX, nr. 4, București, 2011.
- [Bac1] J. Backus, *Can Programming Be Liberated from the von Neumann Style ? A Functional Style and Its Algebra of Programming*, CACM, 1978.
- [Bac2] J. Backus, *The History of FORTRAN I, II and III*, ACM Sigplan Notices, 1978.
- [Bare] H.P. Barendregt, *The Lambda Calculus : Its Syntax and Semantics*, North Holland, 1985.
- [Barr] D. Barron, *Comparative Programming Languages*, American Elsevier, 1968.
- [Bi] C. Birzea, „Definirea și clasificarea competențelor”, în *Revista de Pedagogie* nr. 58 (3), București, 2010.
- [Bl] M. Blum, *A Machine-independent Theory of the Complexity of Recursive Functions*, JACM, 1976.
- [BJ] C. Bohm, G. Jacopini, *Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules*, CACM, 1966.
- [CMS] A. Catană, M. Săcuiu, O. Stănășilă, *Metodica predării analizei matematice*, Editura Didactică și Pedagogică, București, 1983.
- [CS] C. Cazacu, V. Slabu, *Logică matematică*, Editura „Ștefan Lupașcu”, Iași, 1999.
- [CCȘ] G. Ciucu, V. Craiu, A. Ștefănescu, *Statistică matematică și cercetări operaționale*, Editura Didactică și Pedagogică, București, 1974.
- [CȘ] E. Cerchez, M. Șerban, *Programarea în limbajul C/C++ pentru liceu*, Editura Polirom, Iași, 2005 (Vol. 1, 2), Iași, 2006 (Vol. 3).
- [CTT] E. Ciurea, S. Tăbărcă, T. Tăbărcă, *Algoritmi. Metode de elaborare*, Editura Universității „Transilvania”, Brașov, 1997.
- [CLR] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introducere în algoritmi*, Editura Computer Libris Agora, București, 2000.
- [Cri] S. Cristea, *Fundamentele pedagogice ale reformei învățământului*, Editura Didactică și Pedagogică RA, București, 1994.
- [Cro] C. Croitoru, *Tehnici de bază în optimizarea combinatorie*, Editura Universității „Al.I. Cuza”, Iași, 1992.
- [D1] P.J. Denning (ed.), „A Debate on Teaching Computer Science”, *Communications of the ACM*, vol. 32, nr. 12, 1989.

- [D2] P.J. Denning (ed.), „Educating a New Engineer”, *Communications of the ACM*, vol. 35, nr. 12, 1992.
- [DM] P.J. Denning, R.M. Metcalfe, *Beyond Calculation. The Next Fifty Years of Computation*, Springer Verlag, 1997.
- [DCTPCN] A. Dima, V. Cațarschi, C. Tănase, I. Porof, S. Cațarschi, M.I. Nițu, *Educație tehnologică (proiectări didactice, clasele IX-X)*, Editura Performantica, Iași, 2003.
- [DC] W.S. Dorn, D.D. McCracken, *Metode numerice cu programe în FORTRAN*, Editura Tehnică, București, 1976.
- [Fr] F. Frumos, *Didactica : Fundamente și dezvoltări cognitive*, Iași: Editura Polirom, 2008.
- [Hoa] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.
- [Hor] E. Horowitz, *Fundamentals of Programming Languages*, Computer Science Press, 1984.
- [HS] E. Horowitz, S. Sahni, *Fundamentals of Data Structures of Computer Algorithms*, Computer Science Press, 1978.
- [IP] C. Ivașc, M. Prună, *Bazele informaticii (grafuri și elemente de combinatorică). Proiect de manual pentru clasa a X-a, profil informatică*, Editura Petrion, București, 1995.
- [JT] T. Jucan, F.L. Țiplea, *Rețele Petri. Teorie și practică*, Editura Academiei Române, București, 1999.
- [K] O. Kallenberg, *Foundations of Modern Probability*, Springer Series in Statistics, Springer Verlag, Berlin, 2002.
- [K1] D.E. Knuth, *Tratat de programarea calculatoarelor. Algoritmi fundamentali*, Editura Tehnică, București, 1974.
- [K2] D.E. Knuth, *Tratat de programarea calculatoarelor. Sortare și căutare*, Editura Tehnică, București, 1976.
- [LG] L. Livovschi, H. Georgescu, *Sinteza și analiza algoritmilor*, Editura Tehnică, București, 1986.
- [L] D. Lucanu, *Bazele proiectării programelor și algoritmilor*, Editura Universității „Al.I. Cuza”, Iași, 1996.
- [MY] M. Machtey, P. Young, *An Introduction to the General Theory of Algorithms*, Elsevier, North Holland, 1978.
- [Mas1] C. Masalagiu, *Introducere în programarea logică*, Editura Universității „Al.I. Cuza”, Iași, 1996.
- [Mas2] C. Masalagiu, *Distance Learning and Its Methodical/Pedagogical and Social Implications*, TR-HH-04-99, Restructuring of the (re)Training of School Teachers in Computer Science, S\_JEP 11168-96, Editura Computer Libris Agora, București, 1999.
- [Mas3] C. Masalagiu, *Fundamentele logice ale informaticii*, Editura Universității „Al.I. Cuza”, Iași, 2004.
- [MA] C. Masalagiu, I. Asiminoaei, *Didactica predării informaticii*, Editura Polirom, Iași, 2004.
- [MAM] C. Masalagiu, I. Asiminoaei, I. Maxim, *Metodica predării informaticii*, Editura MatrixRom, București, 2001.
- [MM] E. Mateescu, I. Maxim, *Arbori*, Editura „Țara Fagilor”, Suceava, 1996.
- [Max1] I. Maxim, *Un punct de vedere asupra metodicii predării informaticii*, lucrare metodică științifică pentru obținerea gradului didactic I, Editura Universității din București, București, 1997.
- [Max2] I. Maxim, *O încercare de generalizare a algoritmilor de sortare*, Analele Universității „Ștefan cel Mare”, Suceava, anul VI, nr. 12, 1999.
- [MC] O. Mândruț, L. Catană, Suport de curs, *Program de formare - Proiectarea curriculum-ului centrat pe competențe*, Acreditat prin O.M.E.C.T.S. Nr. 4306/01.06.2012, Universitatea Craiova.
- [MCBA] O. Mândruț, L. Catană, D. Badea, A. Ardelean, *Didactica formării de competențe*, „Vasile Goldiș” University Press, Arad, 2012.
- [Me] K. Mellhorn, *Data Structures and Algorithms*, Springer Verlag, 1984.
- [MS] B.M.E. Moret, H.D. Shapiro, *Algorithms from P to NP, Design and Efficiency*, Benjamin Cummings, Redwood, 1990.



- [Po] G. Polya, *Descoperirea în matematică*, Editura Didactică și Pedagogică, București, 1971.
- [Pr] V.R. Pratt, *Semantical Considerations on Flyd-Hoare Logic*, Proceedings of the 17<sup>th</sup> IEEE Symposium on Foundations of Computer Science, 1976.
- [R] J.R. Rogers, *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967.
- [RV] I. Rus, D. Varna, *Metodica predării matematicii*, Editura Didactică și Pedagogică, București, 1983.
- [Sa] J. Sammet, *Programming Languages : History and Fundamentals*, Prentice Hall, 1969.
- [St] M. Stoica, *Sinteze de pedagogie și psihologie*, Editura Universitaria, Craiova, 1992.
- [T] F.L. Țiplea, *Introducere în teoria mulțimilor*, Editura Universității „Al.I. Cuza”, Iași, 1998.
- [WO] P.C. Wankat, F.S. Oreowicz, *Teaching Engineering*, McGraw-Hill, New York, 1993.
- [W1] N. Wirth, *The Programming Language Pascal*, Acta Informatica, 1971.
- [W2] N. Wirth, *Algorithms + Data Structures = Programmes*, Prentice Hall, 1976.
- [\*1] \*\*\*, „Computing as a Discipline, Final Report of the ACM Task Force on the Core of Computer Science”, în P.J. Denning, D.E. Comer, D. Gries, M.C. Mulder, A. Tucker, A.J. Turner, P.R. Young (eds.), *Communications of the ACM*, vol. 32, nr. 1, 1989.
- [\*2] \*\*\*, *Dicționarul explicativ al limbii române (DEX)*, Editura Academiei Române, București, 1975.
- [\*3] \*\*\*, *Documentații (Pascal, C/C++ etc.)*.
- [\*4] \*\*\*, *Ghid metodologic (Tehnologia informației și a comunicațiilor în procesul didactic. Învățământ primar)*, Curriculum Național, MECT, CNC, 2001.
- [\*5] \*\*\*, *Ghid metodologic (Tehnologia informației și a comunicațiilor în procesul didactic. Gimnaziu și liceu)*, Curriculum Național, MECT, CNC, 2002.
- [\*6] \*\*\*, *Networks for People and Their Communities. First Annual Report in the European Commission from the Information Society*, 1996.
- [\*7] \*\*\*, *Ordinul MEN nr. 3188 din 10.05.1999 privind programa școlară pentru disciplina informatică*.
- [\*8] \*\*\*, *Ordinul MEN nr. 3207 din 03.02.1999 privind aplicarea noilor Planuri-cadru de învățământ pentru învățământul primar, gimnazial și liceal, începând cu anul școlar 1999-2000*.
- [\*9] \*\*\*, *Ordinul MEN nr. 3879 din 26.05.1999 privind funcționarea claselor de matematică-informatică cu studiul intensiv al informaticii*.
- [\*10] \*\*\*, *PCWEBOPAEDIA*, Sandy Bay, 1996.



## COLLEGIUM

### Metodică

au apărut :

- Mihaela Neagu, Georgeta Beraru – *Activități matematice în grădiniță*  
Teresa Siek-Piskozub – *Jocuri și activități distractive în învățarea limbilor străine*  
Aurel Dascălu – *Educația plastică în ciclul primar*  
Ileana Dăscălescu, Ana Alexandru, Sonia Hudac – *Îndrumar în sprijinul desfășurării activităților de cunoaștere a mediului înconjurător la grupa pregătitoare în grădiniță*  
Elena Simionică, Fănica Bogdan – *Gramatica... prin joc*  
Elena Simionică, Veronica Anton – *Caietul învățătorului*  
Vasile Ghica – *Ghid de consiliere și orientare școlară*  
Elena Simionică, Florica Caraiman – *Matematica... prin joc*  
Constantin Parfene – *Metodica predării limbii și literaturii române*  
Doina Bâclea, Margareta Constantinescu (coord.) – *Chimie. Planuri de lecție (clasele VII-XII)*  
Elena Ilie – *Limba română. Evaluare formativă. Ghid metodologic. Clasele I-IV*  
Veronica Păduraru (coord.), Geta Fotea, Mariana Șenu, Maria Curcan, Angela Popa – *Activități matematice în învățământul preșcolar. Sinteze*  
Valerian Dragu (coord.), Doina-Eugenia Steva, Carmen Filoti, Corneliu Horaicu, Doru Vlasov – *Geografie – Geologie. Ghid metodologic*  
Liliana Stan (coord.), Doina-Eugenia Steva, Valerian Dragu, Doru Valentin Vlasov – *Elemente de didactica geografiei*  
Adrian Adăscăliței – *Instruire asistată de calculator. Didactică informatică*  
Mihaela Neagu, Mioara Mocanu – *Metodica predării matematicii în ciclul primar*  
Oana Iucu – *Didactica științelor juridice și administrative*  
Emanuela Ilie – *Didactica literaturii române. Fundamente teoretico-aplicative*  
Constantin Petrovici – *Didactica activităților matematice în grădiniță*  
Constantin Petrovici – *Didactica matematicii pentru învățământul primar*  
Emanuela Ilie – *Didactica limbii și literaturii române*  
Adriana Vizental – *Metodica predării limbii engleze. Strategies of teaching and testing English as a foreign language* (ediția a IV-a)  
Cristian Masalagiu, Ioan Asiminoaei, Mirela Țibu – *Didactica predării informaticii* (ediția a II-a)



*www.polirom.ro*

Redactor : Ines Simionescu  
Coperta : Carmen Parii  
Tehnoredactor : Irina Lăcătușu

Bun de tipar : iulie 2016. Apărut : 2016  
Editura Polirom, B-dul Carol I nr. 4 • P.O. BOX 266  
700506, Iași, Tel. & Fax : (0232) 21.41.00 ; (0232) 21.41.11 ;  
(0232) 21.74.40 (difuzare) ; E-mail : office@polirom.ro  
București, Splaiul Unirii nr. 6, bl. B3A, sc. 1, et. 1,  
sector 4, 040031, O.P. 53  
Tel. : (021) 313.89.78 ; E-mail : office.bucuresti@polirom.ro

---

Tiparul executat la S.C. LUMINA TIPO s.r.l.  
str. Luigi Galvani nr. 20 bis, sect. 2, București  
Tel./Fax : 211.32.60, 212.29.27, E-mail : office@luminatipo.com

---





