

Gândirea algoritmică - o filosofie modernă a matematicii și informaticii

Marin Vlada – Universitatea din București,
vlada@fmi.unibuc.ro

Abstract

Dezvoltarea informaticii actuale se datorează cercetărilor, rezultatelor și experiențelor din domeniile SISTEMELOR DE CALCUL, ALGORITMICII și PROGRAMĂRII, dar mai ales a interdependenței acestor domenii prin așa-numita triadă "SISTEM DE CALCUL - ALGORITMICĂ - PROGRAMARE". La baza acestei interdependențe se află conceptul de ALGORITM, concept ce a construit pentru om o nouă filosofie: GÂNDIREA ALGORITMICĂ. Conceptul a fost introdus în matematica veche, utilizat de matematica modernă și perfecționat de informatică în utilizarea calculatoarelor moderne. Această gândire algoritmică a făcut posibilă apariția și dezvoltarea Tehnologiei Informației (IT) ce reprezintă de fapt implementarea filosofiei procesării, gestionării și comunicării informațiilor și cunoștințelor.

1. Introducere

Întâmplător sau nu, **deceniul 7** al secolului al XX-lea a fost unul al *marilor schimbări* în domeniul informaticii și al *sistemelor de calcul*:

- **inventarea microprocesorului** ("bijuteria de bază" a actualelor sisteme de calcul) ca urmare a rezultatelor din trei domenii fundamentale : sisteme, circuite integrate și microprogramare; a urmat construirea și răspândirea pe scară largă a sistemelor de calcul de tip PC(Personal Computer), impulsivitatea dezvoltării rețelelor de calculatoare, apariția și dezvoltarea de noi sisteme de operare; performanțe sporite ale dispozitivelor I/O;
 - **supremația și răspândirea structurilor de control** în *algoritmică și programare*; apariția *limbajului pseudocod* în reprezentarea și elaborarea algoritmilor; conceperea și scrierea primelor limbaje de programare care implementează *structurile de control* (Limbajele **Pascal** și **C**), adaptarea continuă a limbajelor de programare prin implementarea *structurilor de control*, a *structurilor de date*, a *facilităților programării orientate spre obiecte (OOP- Object Oriented Programming)*;
 - **succese spectaculoase în domeniul Inteligenței Artificiale** prin construirea primelor *sisteme expert*; conceperea și scrierea primului *limbaj de programare logică* (Limbajul **Prolog**) ce oferă suportul *programării declarative*; dezvoltarea și utilizarea largă a metodelor și tehnicilor *Inteligenței Artificiale* în rezolvarea câtorva dintre cele mai dificile probleme;
 - **delimitarea problemelor** rezolvate cu calculatorul(*probleme decidabile*) în două *clase distincte*: clasa problemelor rezolvate prin *metode imperative(procedurale)* și clasa problemelor rezolvate prin *metode declarative*; delimitarea clasei *problemele nedecidabile*; apariția și dezvoltarea tehnicilor pentru comunicarea, căutarea și vizualizarea la distanță a informațiilor.
-

Toate aceste aspecte au fost și sunt într-o *interdependență* continuă ținând seama de particularitatea informaticii care oferă *sisteme de calcul* performante și *produse-program* competitive în rezolvarea problemelor. Utilizarea eficientă a sistemelor de calcul și a produselor-program reclamă o instruire continuă, atât pentru *informaticienii-programatori*, cât și pentru *utilizatori*.

Dezvoltarea *gândirii algoritmice* trebuie luată în considerare ca obiectiv în instruire, atunci când se învață *algoritmă* (metode și tehnici), dar și când se învață *programarea* (limbaje de programare). Practica instruirii elevilor și studenților a demonstrat că învățarea unui limbaj de programare este în general “*mai ușoară*” decât învățarea elaborării algoritmilor (algoritmă). Acest lucru se poate justifica prin faptul că elaborarea unui *algoritm* este echivalentă cu implementarea(reprezentarea) *raționamentelor*(*procese demonstrative*) deduse din **metode și tehnici** utilizate în rezolvarea unei probleme. **Rezolvarea problemelor** necesită nu numai *cunoștințe clare și precise*, dar și *capacitate de sinteză și control* și mai ales *capacitate de creație*. Dacă vrem să facem o analogie, un *programator* poate fi “*compozitorul*” ce realizează o “*lucrare muzicală*”. În toate etapele instruirii se va avea în vedere *interdependența*:

“SISTEM DE CALCUL – ALGORITMICĂ – PROGRAMARE”

Succesele unui *concurrent* la Olimpiadele de Informatică depind de cunoașterea utilizării unui limbaj de programare modern, dar mai ales depind de « *bogăția* » și stăpânirea cunoștințelor în *elaborarea algoritmilor*. Și mai există încă ceva: *experiența acumulată* în activitatea de rezolvare a problemelor prin formarea unei *gândiri algoritmice* solide și consistente. Același lucru se poate afirma și despre succesele unui *programator* ce lucrează într-o firmă de *produse software*.

2. Conceptul de algoritm și algoritmizarea

Complexitatea problemelor care necesită descrierea mai multor **procese de calcul** complexe a determinat folosirea noțiunii de **algoritm** în activitatea de rezolvare a problemelor. Multe procese naturale, multe activități umane, pot fi descrise într-o **formă algoritmică** prin definirea unor *informații* și *acțiuni* clare și precise, eliminându-se ambiguitățile în *interpretare* și în *operații*. **Algoritmizarea** este o cerință fundamentală în rezolvarea oricărei probleme cu ajutorul calculatorului.

Experiența a demonstrat că *nu orice problemă* poate fi rezolvată prin *algoritmizarea rezolvării*, adică prin descrierea unui algoritm de rezolvare. Așa s-a delimitat *clasa problemelor decidabile* (o problemă este decidabilă dacă există un algoritm pentru rezolvarea ei) de *clasa problemelor nedecidabile* (o problemă este nedecidabilă dacă nu există un algoritm pentru rezolvarea ei).

Un algoritm implementează diverse metode și tehnici de rezolvare care au fost descoperite sau defintivate într-un anumit moment în evoluția științifică a domeniului respectiv. Există algoritmi ce urmează metode dezvoltate înainte de apariția calculatoarelor, dar cele mai multe probleme cer **abordări noi**. Chiar dacă ne gândim numai la “**problema celor 4 culori**” , care a fost rezolvată (*în anul 1977*) doar prin utilizarea calculatorului și prin utilizarea unei metode noi (metoda *Backtracking*), facem dovada acestei afirmații.

În etapa actuală de dezvoltare științifică și tehnică, **rezolvarea unei probleme** dintr-un domeniu (*matematică, informatică, fizică, chimie, etc.*) reprezintă o *activitate de creație, un raționament* prin construirea, generarea, descrierea unui:

- **proces demonstrativ**(*demonstrația*) care să arate existența unei soluții sau a mai multor soluții și/sau să determine efectiv *soluțiile exacte*;
- **proces computațional**(*algoritmul*) care să codifice un *proces demonstrativ*, o metodă sau o tehnică de rezolvare în scopul *determinării (eventual aproximative)* a soluțiilor exacte.

În general, în procesul (activitatea) de *rezolvare a unei probleme* dintr-un anumit domeniu (științific, economic, social, etc.), este necesară evidențierea *ipotezei* (condițiile, stările inițiale, parametrii inițiali) și a *concluziei* (cerințele, obiectivele, scopurile) din analiza și studiul *enunțului* problemei. *Procesul de rezolvare* (raționamentul) constă în utilizarea selectivă a *legilor, teoremelor, propozițiilor, etc.* din domeniul problemei, pentru ca pornind de la *ipoteză(I)*, prin aplicarea succesivă a legilor, teoremelor, etc. să se obțină *concluzia(C)* problemei. Legătura dintre *ipoteză, concluzie* și *procesul de rezolvare (raționamentul - R)* determină o *structură* asemănătoare conceptului de *program*, și anume (*Ipoteză - Date de intrare; Concluzia - Date de ieșire*) :

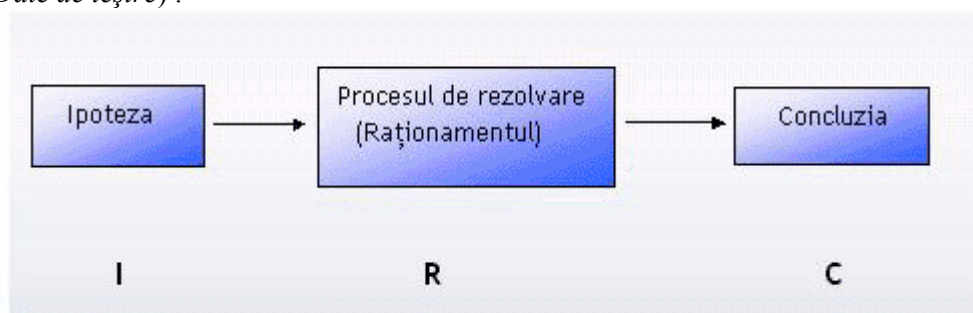


Figura 1. Rezolvarea unei probleme

De exemplu, *procesul de rezolvare* a unei probleme poate fi descris astfel: există un număr de *teoreme* T_1, \dots, T_n determinate de *ipotezele* I_1, \dots, I_n și de *concluziile* C_1, \dots, C_n ; acestea sunt selectate și apoi aplicate astfel încât să se poată realiza identificările:

$I \supseteq I_1, C_1 \supseteq I_2, \dots, C_n \supseteq C$, adică:

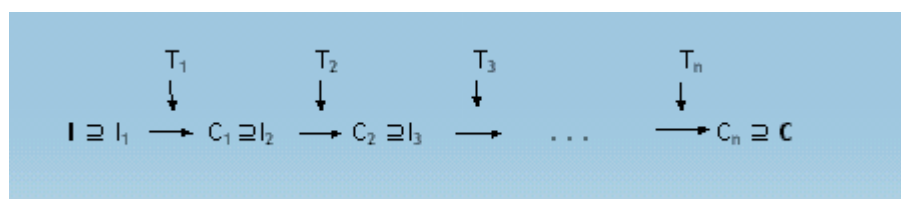


Figura 2. Procesul de rezolvare a unei probleme

Această prezentare poate fi înțeleasă dacă, de exemplu, facem analogia cu rezolvarea *problemelor de geometrie* care necesită delimitarea *ipotezei* și a *concluziei*, ca apoi să se utilizeze numai acele *teoreme, propoziții, proprietăți* care, pornind de la ipoteză, prin aplicarea succesivă, să se obțină concluzia cerută. Evident, din experiența

căpătată în *rezolvarea problemelor*, se poate afirma faptul că selectarea teoremelor și aplicarea lor se poate realiza doar prin “*stăpânirea*” domeniului respectiv la nivel de specialist sau de expert. Este cunoscut faptul că omul poate rezolva o problemă în trei moduri (metode evidențiate de *Inteligența Artificială*):

- pornind de la *ipoteză* și obținând *concluzia* (“*metoda înlănțuirii înainte*”);
- pornind de la *concluzie* și obținând *ipoteza* (“*metoda înlănțuirii înapoi*”);
- pornind simultan, de la *ipoteză* și *concluzie* (“*metoda mixtă*”).

În toate *științele* (matematică, fizică, chimie, etc.) există multe probleme care, chiar dacă au fost studiate într-o anumită etapă, abordarea lor cu *metode și tehnici moderne* poate să trezească oricând interesul tinerilor, al cercetătorilor în general. Aceasta cu atât mai mult cu cât astăzi, prin utilizarea unor metode specifice *tehnologiei informației* și prin utilizarea *calculatorului*, se pot concepe tehnici care altădată erau imposibil de imaginat.

Din punct de vedere *metodologic*, trebuie să ne obișnuim să reformulăm problemele în mod explicit și adecvat rezolvării lor matematice sau- când este posibil – cu ajutorul calculatorului. Pentru acest lucru trebuie să cunoaștem “*limitele*” **gândirii demonstrative** și în același timp “*limitele*” **gândirii algoritmice**, apoi să cunoaștem performanțele calculatoarelor.

Să amintim aici istoria problemei celor 4 culori care începe în anul 1852, când studentul englez *Francis Guthrie* a enunțat următoarea problemă pentru colorarea unei hărți:

“*Sunt suficiente 4 culori pentru a colora o hartă ce reprezintă diverse țări, cu condiția ca oricare două țări vecine (ce au frontiera comună) să fie colorate cu culori diferite*”.

Mulți matematicieni cu renume au studiat această problemă (**A. de Morgan, A. Cayley, A. B. Kempe, H. Heesch, K. Appel, W. Hakel**), dar rezolvarea ei completă nu a fost posibilă decât în anul 1977, (*K. Appel, W. Hakel, “The four color problem” în Mathematics Today, L. A. Steen(ed.), Springer Verlag, 1978*) și acest lucru numai după ce s-a utilizat calculatorul, deoarece fiind o *problemă combinatorială* cu spațiul soluțiilor de ordin foarte mare, a fost necesară conceperea unei *metode algoritmice* pentru căutarea eficientă cu ajutorul calculatorului. Teoretic, problema a fost rezolvată în cazul a *5 culori*. De altfel, rezolvarea acestei probleme a impus în Informatică o metodă devenită clasică în domeniul elaborării algoritmilor: *metoda backtracking*.

În zilele noastre apar tot mai des probleme a căror **rezolvare** (*proces demonstrativ*) este rezultatul îmbinării *metodelor pur matematice* cu **metodele algoritmice** (*compuționale*). Uneori, rezultate computaționale pot oferi surse de inspirație pentru metodele matematice. Informaticienii au și venit în întâmpinarea matematicienilor și specialiștilor prin elaborarea diverselor *sisteme de programe* care realizează *calculare numerice, logice, simbolice, reprezentări spațiale*, etc. cu ajutorul cărora pot fi dobândite cunoștințe matematice sau se pot rezolva probleme din diverse domenii. Totuși, aceste aspecte nu trebuie să conducă la neglijarea procesului demonstrativ. Cele mai utilizate astfel de *sisteme de programe* sunt: MATHEMATICA, STATISTICA, MATLAB, MATHCAD, DERIVE, MAPLE, ORIGIN, SLIDEWRITE, WORKPLACE, EUREKA, etc.

Este cunoscut faptul că în științele naturii se obțin rezultate deosebite prin îmbinarea metodelor teoretice cu metodele experimentale. Pentru *matematică* (și alte științe),

informatica (tehnologia informației) este cea care oferă rezultate experimentale prin utilizarea calculatorului în rezolvarea problemelor.

Metoda algoritmică va fi un experiment pentru justificarea rezultatelor *procesului demonstrativ* de la *metoda matematică*, dar în același timp poate să fie o metodă de rezolvare independentă. Metoda algoritmică trebuie să substituie rezultatele obținute pe cale teoretică cu metode computaționale eficiente ținând seama de limitele proceselor algoritmice și de performanțele calculatoarelor. Este adevărat că uneori metodele matematice pot conduce la simplificarea metodelor algoritmice și invers. *Elevii* care participă la olimpiadele școlare de informatică au constatat că, de multe ori, o analiză profundă a abordării matematice, poate să conducă la obținerea unei metode algoritmice eficiente. Participanții la olimpiadele de matematică pot constata același lucru la o analiză profundă a abordării algoritmice. De aici, importanța atât a **proceselor demonstrative (metode matematice)**, cât și a **proceselor computaționale (metode algoritmice)**.

În funcție de natura metodelor/tehnicilor implementate în procesele computaționale, *algoritmii* pot fi: *numerici, seminumerici, formali, combinatoriali, neuronali, de căutare, de sortare, recursivi, de rescriere, secvențiali, paraleli, determinați, nedeterminați, probabiliști, aleatori, euristici, de tip Monte Carlo, genetici, de simulare, computational geometry, etc.*

Întreaga activitate de cercetare și elaborare de software din domeniul *Tehnologiei Informației* este determinată de inventarea, conceperea, elaborarea, testarea, și implementarea de *algoritmi performanți și utili*. Marea diversitate a algoritmilor și marea aplicabilitate a acestora în toate domeniile, face ca această *temă* să fie mereu actuală și într-o continuă schimbare și perfecționare.

3. Gândirea algoritmică și gândirea obiectuală

Practica *rezolvării problemelor* folosind un *limbaj de programare* a determinat de-a lungul timpului diverse abordări în funcție de performanța limbajului de programare, performanța calculatorului și nu în ultimul rând, în funcție de metodele și tehnicile avansate privind implementarea raționamentelor pentru demonstrațiile corespunzătoare problemelor.

Rezolvarea teoretică a unei probleme nu garantează și *rezolvarea ei practică* cu calculatorul. În general, un limbaj de programare este menit să faciliteze rezolvarea unor *clase de probleme* și se pretează mai bine anumitor *tipuri de algoritmi*. Este nevoie de experiență în utilizarea și cunoașterea calculatorului, de competență și intuiție, este nevoie de inspirație și creație. În astfel de situații *este nevoie de cunoașterea mai multor limbaje de programare* pentru a alege *limbajul de programare* adecvat pentru *clasa de probleme* din care face parte problema de rezolvat. Experiența a arătat că, atunci când nu este ales limbajul de programare corespunzător, dacă totuși se ajunge să se rezolve problema, s-a făcut risipă de resurse (timp / memorie / finanțe, etc.) și, prin urmare, eficiența și performanța au avut de suferit.

Astăzi, performanța unui *informatician-programator* este determinată de experiența și competența obținută în desfășurarea celor două etape (ANALIZĂ, PROGRAMARE):

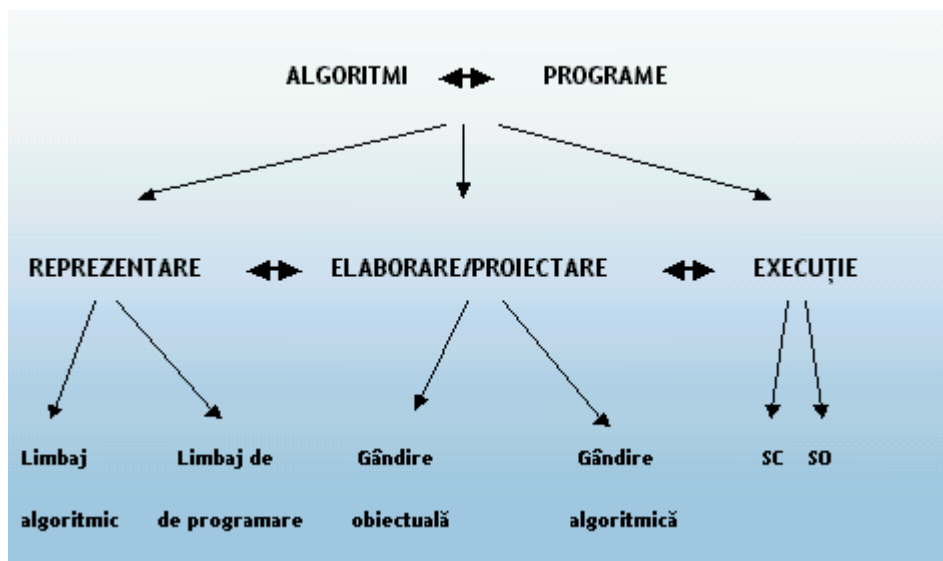
- *etapa gândirii obiectuale (ANALIZĂ - PROIECTARE)* – modul de analiză și abstractizare a problemelor prin definirea corectă a obiectelor, a tipurilor de obiecte, a relațiilor între obiecte și a operatorilor specifici (*elaborarea UAP ; etapa de concepție și analiză-proiectare*) ;
- *etapa gândirii algoritmice (PROGRAMARE - EXECUȚIE)* – alegerea și aplicarea corectă a unor metode de rezolvare prin precizarea exactă a operatorilor de prelucrare a obiectelor, reprezentarea corectă a strategiilor algoritmice, reprezentarea codificată a obiectelor și a prelucrărilor conform unui limbaj de programare (*elaborarea algoritmului și programului ; etapa de programare - codificare implementare și execuție*).

În prezent este tot mai des invocată *reprezentarea problemelor* folosind concepte **OOP (Object Oriented Programming)**. Conceptul de *obiect* (M. Minsky, *The Society of Mind*, Touchstone Books, New York, 1986) are un rol important în *știința cunoașterii și educației*. Un *obiect* modelează o *entitate* din **lumea reală** sau **virtuală**. În *activitatea de rezolvare a problemelor* trebuie să se *identifice/definească* obiectele din cadrul problemelor ce provin din diferite *domenii: științifice, economice, sociale, etc.* *Identificarea* obiectelor este echivalentă cu determinarea entităților și conceptelor care reprezintă *forme fizice/grafice, fapte, evenimente, procese, stări, etc.* Un *obiect* este caracterizat în mod unic prin *identificare, comportament* (caracteristică dinamică), și *stare* (caracteristică statică). În esență, rezolvarea unei probleme se va exprima printr-o *codificare a universului problemei și a raționamentelor* pentru procesul demonstrativ.

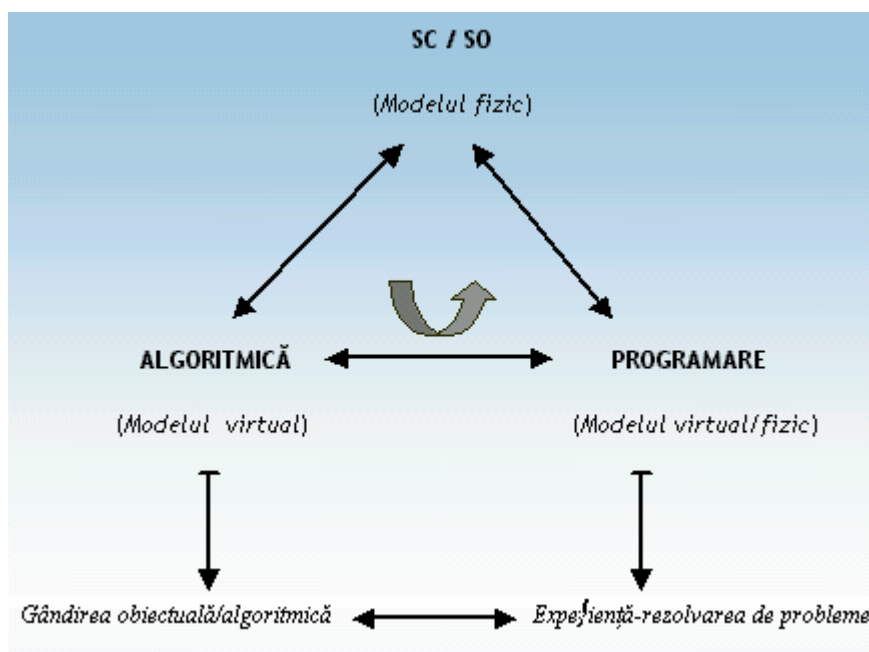
Practica *dezvoltării aplicațiilor software* (care necesită rezolvarea diverselor tipuri de probleme) a scos la iveală următoarele faze importante (*gândirea obiectuală* :fazele 1 și 2= *analiză-proiectare; gândirea algoritmică*: fazele 3-6= *programare-execuție*):

1. *specificarea problemelor* – descrierea clară și precisă a problemelor indiferent din ce domeniu provin acestea ;
2. *proiectarea soluțiilor* – includerea problemelor în clasa de probleme corespunzătoare și alegerea modului de reprezentare a problemelor prin formularea *etapelor și procedurilor* corespunzătoare pentru procesele de rezolvare;
3. *implementarea soluțiilor* – *elaborarea algoritmilor și codificarea* acestora într-un limbaj de programare modern;
4. *analiza soluțiilor* – *eficiența soluțiilor* raportată la resursele utilizate: *memorie, timp, utilizarea dispozitivelor I/O, etc.*;
5. *testarea și depanarea* – *verificarea execuției programului* cu diverse seturi de date de intrare pentru a putea răspunde rezolvării oricărei probleme pentru care aplicația a fost elaborată;
6. *actualizarea și întreținerea* – adaptarea soluțiilor implementate pentru *eliminarea erorilor* în rezolvarea unei anumite probleme și *compatibilitatea* cu sistemul de calcul și sistemul de operare folosite.

Conform acestor faze iese în evidență *interdependența* între următoarele *activități* importante: **REPREZENTARE – ELABORARE / PROIECTARE – EXECUȚIE**, și anume se poate sintetiza prin următoarea schemă (**SC=sistem de calcul, SO=sistem de operare**):



De asemenea, *interdependența* precizată mai sus se transmite și între componentele de pe nivelul inferior din schema arborescentă alăturată. *Competența și experiența în rezolvarea problemelor* se pot obține doar dacă permanent se are în vedere această interdependență și dacă se întreprind eforturi pentru *însușirea de noi cunoștințe* și pentru *conoașterea* corespunzătoare a tuturor aspectelor privind MODELUL FIZIC , respectiv MODELUL VIRTUAL, aspecte determinate de interdependența SISTEM DE CALCUL – ALGORITMICĂ – PROGRAMARE:



Practica și experiența elaborării programelor pentru rezolvarea problemelor scot în evidență următoarele aspecte foarte importante:

- **Modelul fizic**- acest model este dat de *sistemul de calcul și sistemul de operare*, model ce trebuie luat în considerare când se proiectează și se elaborează o aplicație; acest aspect reclamă competență în domeniul sistemelor de calcul și perfecționare continuă pentru cel care proiectează și elaborează aplicația;
- **Modelul virtual** – acest model este dat de *gândirea obiectuală și algoritmică*, de modul de *reprezentare a algoritmilor*, de *mașina virtuală* pe care trebuie să se execute algoritmul elaborat; în timp, acest model a suferit schimbări majore deoarece a fost tot timpul influențat de *modelul fizic* și de *clasa problemelor* ce urmau să fie rezolvate;
- **Modelul program** – acest model este reprezentat de o îmbinare între *modelul fizic*(SC/SO) și *modelul virtual* (Algoritmul); întotdeauna un program se elaborează într-un *limbaj de programare* care trebuie să respecte *restricțiunile modelului fizic* (sistemul de calcul și sistemul de operare) și *restricțiunile modelului virtual* (algoritmul)

4. Bibliografie

- [1] Appel, K. and Haken, W. "Every Planar Map is Four-Colorable, II: Reducibility." *Illinois J. Math.* 21, 91-567, 1977.
- [2] Appel, K. and Haken, W. "The Solution of the Four-Color Map Problem." *Sci. Amer.* 237, 108-121, 1977.
- [3] Brassard, G. and Bratley, P. *Fundamentals of Algorithmics*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
- [4] Cristea, V., C. Giumale, E, Kalisz, Al. Paunoiu, *Limbajul C standard*, Ed. Teora, București, 1992.
- [5] Knuth, D. E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd ed.* Reading, MA: Addison-Wesley, 1997.
- [6] Knuth, D. E. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd ed.* Reading, MA: Addison-Wesley, 1998.
- [7] Knuth, D. E. *The Art of Computer Programming, Vol. 3: Sorting and Searching, 2nd ed.* Reading, MA: Addison-Wesley, 1998.
- [8] Chabert, J.-L. (Ed.). *A History of Algorithms: From the Pebble to the Microchip*. New York: Springer-Verlag, 1999.
- [9] Popovici, M. D., Popovici, M. I., C++. *Tehnologia orientată spre obiecte. Aplicații*, Ed. Teora, București, 2000.
- [10] Vlada, M., Informatică, *Universitatea din București*, Ed. *Ars Docendi*, București, 1999.
- [11] Vlada, M., Poligoane stelate. Problema lui Hopf și Pannwitz, *Gazeta de matematică*, nr. 8/1995, pag. 339-348.
- [12] Vlada, M., Rezolvarea problemelor folosind Eureka, *software educațional*, www.unibuc.ro/eBooks/informatica/eureka/, *Universitatea din Bucuresti*, 2003.
- [13] Vlada, M., Concepul de algoritm-abordare modernă, *Gazeta de informatică*, vol. 13/2 și 3, pp. 25-30, pp. 35-39, *Agora*, Cluj Napoca, 2003.
- [14] Zaharia, M. D., Structuri de date și algoritmi. Exemple în limbajele C și C++, Ed. *Albastră*, Cluj-Napoca, 2002.
- [15] <http://www-groups.dcs.st-and.ac.uk/~history/BigPictures/Guthrie.jpeg>
-