Abordarea modernă a conceptului de algoritm

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

Abstract

Practica dezvoltării aplicațiilor/produselor software (ce necesită rezolvarea diverselor clase de probleme) scoate în evidență însuşirea de noi cunoştinţe şi cunoaşterea corespunzătoare a tuturor aspectelor privind modelul fizic, respectiv modelul virtual din interdependența Sistem de calcul-Algoritmică-Programare. Modelul virtual este determinat de gândirea obiectuală şi algoritmică, de modul de reprezentare a algoritmilor, precum şi de maşina virtuală pe care trebuie să se execute algoritmul elaborat. Conceptul de algoritm a avut o evoluție dinamică determinată de interdependența menționată şi de competența şi experiența specialiştilor informaticieni în activitatea de rezolvare a problemelor folosind calculatorul. Lucrarea prezintă o abordare modernă a conceptului de algoritm şi expune detaliat modul în care ar trebui privit (de către elevi, studenți, profesori, specialişti) algoritmul luând în considerare diferite aspecte care sunt practic desconsiderate de abordarea clasică.

1. Introducere

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

- **Modelul fizic**- acest model este dat de *sistemul de calcul* şi *sistemul de operare*, model ce trebuie luat în considerare când se proiectează şi se elaborează o aplicație; acest aspect reclamă competență în domeniul sistemelor de calcul şi perfecționare continuă pentru cel care proiectează şi elaborează aplicația;
- **Modelul virtual** – acest model este dat *de gândirea obiectuală şi algoritmică*, de modul de *reprezentare a algoritmilor*, de *maşina 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* şi de *clasa problemelor* ce urmau să fie rezolvate;
- **Modelul program** – acest model este reprezentat de o îmbinare între *modelul fizic* (SC/SO) şi *modelul virtual* (**Algoritmul**); întotdeauna un program se elaborează într-un *limbaj de programare* care trebuie să respecte *restricțiile modelului fizic* (sistemul de calcul şi sistemul de operare) şi *restricțiile modelului virtual* (algoritmul).

Toate aceste aspecte au fost şi sunt într-o *interdependența* continuă ținând seama de particularitatea informaticii care oferă *sisteme de calcu*l performante şi *produse-program* competitive în rezolvarea problemelor. Utilizarea eficientă a sistemelor de

calcul şi a produselor-program reclamă o instruire continuă, atât pentru *informaticieni-programatori*, cât şi pentru *utilizatori*.

În etapa actuală de dezvoltare ştiinţifică şi tehnică, **rezolvarea unei probleme** dintr-un domeniu (*matematică, informatică*, *fizică, chimie, etc.*) reprezintă o *activitate de creaţie, un raţionament* prin construirea, generarea, descrierea următoarelor procese:

- **proces demonstrativ** (*demonstraţia*) care să arate existenţa unei soluţii sau a mai multor soluţii şi/sau să determine efectiv *soluţiile exacte*;
- **proces computaţional** (*algoritmul*) care să codifice un *proces demonstrativ*, o metodă sau o tehnică de rezolvare în scopul *determinării (eventual aproximative)* a soluţiilor exacte.

2. Algoritmizarea

Complexitatea problemelor care necesită descrierea mai multor *procese de calcul* complexe a determinat folosirea noţiunii de *algoritm* în activitatea de rezolvare a problemelor. Multe procese naturale, multe activităţi umane, pot fi descrise într-o *formă algoritmică* prin definirea unor *informaţii* şi *acţiuni* clare şi precise, eliminându-se ambiguităţile în *interpretare* şi în *operaţii*. **Algoritmizarea** este o cerinţă fundamentală în rezolvarea oricărei probleme cu ajutorul calculatorului.

Experienţa a demonstrat că *nu orice problemă* poate fi rezolvată prin *algoritmizarea rezolvării*, adică prin descrierea unui algoritm de rezolvare. Aşa 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 şi tehnici de rezolvare care au fost descoperite sau definitivate într-un anumit moment în evoluţia ştiinţifică a domeniului respectiv. Există algoritmi ce urmează metode dezvoltate înainte de apariţia calculatoarelor, dar cele mai multe probleme cer *abordări noi*.

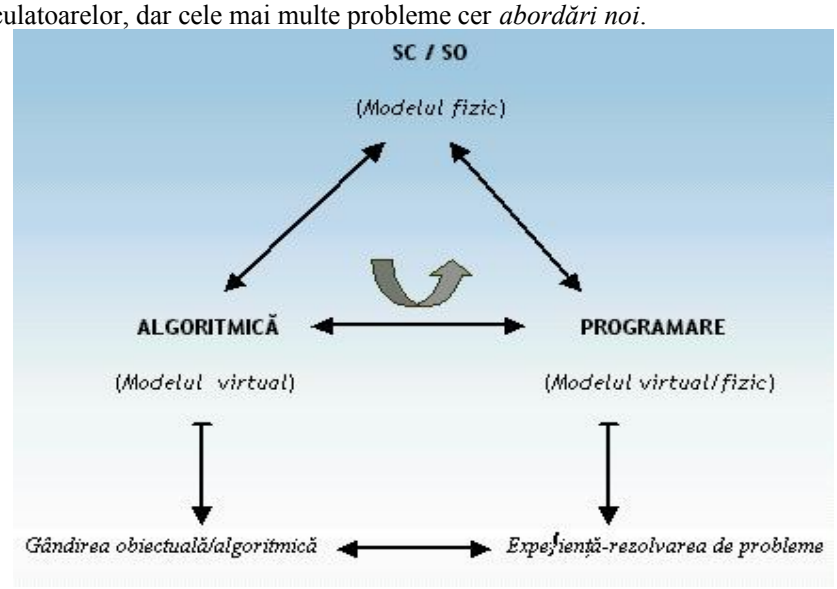

Figura 1. Triada Sistem de calcul-Algoritmică-Programare

Interdependența precizată mai sus se transmite şi între componentele de pe nivelul inferior din schema arborescentă alăturată. *Competența* şi *experiența* în *rezolvarea problemelor* se pot obține doar dacă permanent se are în vedere această interdependență şi dacă se întreprind eforturi pentru î*nsuşirea de noi cunoştințe* şi pentru *conoaşterea* corespunzătoare a tuturor aspectelor privind *modelul fizic* , respectiv *modelul virtual*, aspecte determinate de interdependența *Sistem de calcul-Algoritmică-Programare*.

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 şi nu în ultimul rând, în funcție de metodele şi tehnicile avansate privind implementarea raționamentelor pentru demonstrațiile corespunzătoare problemelor. În [4] (V. Cristea, C. Giumale, E, Kalisz, Al. Paunoiu, *Limbajul C standard*, Ed. Teora, Bucureşti, 1992) se arată ca "u*n limbaj de programare acționează ca o interfață între universul real* a*l problemei de rezolvat şi programul de rezolvare, punând la dispoziție o serie de elemente constructive (entități) şi legi de combinare a acestora, prin care elementele problemei şi acțiunile de rezolvare pot fi reprezentate şi prelucrate la nivelul programului. A construi un program de rezolvare a problemei înseamnă, în esență, a găsi modalitatea de agregare a acestor entități, în aşa fel încât rezultatul-anume programul- atunci când este executat de calculator să constituie o replică a procesului pe care un rezolvitor uman l-ar executa pentru a rezolva problema*".

Rezolvarea teoretică a unei probleme nu garantează şi *rezolvarea ei practică* cu calculatorul. În general, un limbaj de programare este menit să faciliteze rezolvarea unor *clase de probleme* şi se pretează mai bine unor anumite *genuri de algoritmi*. Este nevoie de experiență în utilizarea şi cunoaşterea calculatorului, de competență şi intuiție, este nevoie de inspirație şi creație. În astfel de situatii *este nevoie de cunoaşterea 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 totusi se ajunge să se rezolve problema, s-a făcut risipă de resurse timp / memorie / finanțe, etc., şi prin urmare eficiența şi performanța au avut de suferit.

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

În esență, rezolvarea unei probleme se va exprima printr-o *codificare* a *universului problemei* şi a *raționamentelor* pentru procesul demonstrativ.

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

1. *specificarea problemelor* – descrierea clară şi precisă a problemelor indiferent din ce domeniu provin acestea ;
2. *proiectarea soluțiilor* – includerea problemelor în clasa de probleme corespunzătoare şi alegerea modului de reprezentare a problemelor prin

234 Facultatea de Matematică şi Informatică, Bucureşti

formularea *etapelor* şi *procedeelor* corespunzătoare pentru procesele de rezolvare;
3. *implementarea soluţiilor – elaborarea algoritmilor* şi *codificarea* acestora într-un limbaj de programare modern;
4. *analiza soluţiilor – eficienţa soluţiilor* raportată la resursele utilizate: *memorie, timp, utilizarea dispozitivelor I/O*, etc.;
5. *testarea şi depanarea – verificarea execuţiei programului* cu diverse seturi de date de intrare pentru a putea răspunde rezolvării oricărei probleme pentru care aplicaţia a fost elaborată;
6. *actualizarea şi întreţinerea* – adaptarea soluţiilor implementate pentru *eliminarea erorilor* în rezolvarea unei anumite probleme şi *compatibilitatea* cu sistemul de calcul şi sistemul de operare folosite.

Conform acestor faze iese în evidenţă *interdependenţa* între urmatoarele *activităţi* importante: REPREZENTARE – ELABORARE / PROIECTARE – EXECUŢIE.

Comentarii metodologice şi pedagogice

Conform celor de mai sus *este inadecvată utilizarea schemelor logice* în reprezentarea algoritmilor. Autorii de manuale de informatică fac o mare greşeală dacă nu ţin seama de acest lucru. Inventarea şi apariţia *structurilor de control* în programare au eliminat teoretic schemele logice;

Utilizarea unui *pseudocod* în reprezentarea algoritmilor ce se exprimă cu *cuvinte-chei din limba romană nu este oportună* dacă se ţine seama că algoritmul va fi codificat într-un limbaj de programare modern: **C, C++, Java, Pascal, Foxpro, Oracle, Prolog**, etc. Încă mai există manuale care utilizează această forma de reprezentare; pe viitor astfel de manuale nu trebuie să mai existe;

Activităţile de laborator sunt esenţiale în activitatea de învaţare. Profesorii de informatică sunt obligaţi să îmbine cît mai eficient *orele de predare cu orele de laborator* prin care de fapt se încheie *ciclul* pentru rezolvarea unei probleme cu calculatorul: *problema → modelul matematic → algoritmul → programul → calculator → rezultate → verificare soluţii*;

Iese în evidenţă echivalenţa ALGORITM - PROGRAM - SC modulo operaţiile realizate de algoritm în *spaţiul virtual*, respectiv operaţiile realizate de SC prin execuţia programului corespunzător algoritmului. Algoritmul va trebui să simuleze toate operaţiile declanşate prin lansarea în execuţie a programului corespunzător algoritmului.

3. Definiţia conceptului de algoritm

Definiţie. Un *algoritm* este **sistemul virtual A = (M, V, P, R, Di, De, Mi, Me)** caracterizat de următoarele aspecte: elaborare, reprezentare, execuţie, corectitudine şi analiză.

Sistemul virtual **A** este constituit din următoarele elemente:

- **M – memorie virtuală (internă)** utilizată pentru stocarea temporară a informațiilor destinate variabilelor din mulțimea de variabile V; asupra variabilelor acționează procesul de calcul P ce are acces la memorie prin rezervarea de memorie pentru variabilele din mulțimea V; inițial memoria virtuală este folosită pentru rezervare de memorie pentru variabilele din mulțimea V, după care este utilizată pentru scrierea şi citirea de informații în locațiile corespunzătoare variabilelor utilizate în procesul de calcul P;
- **V – mulțime de variabile/structuri de date** definite conform raționamentului R corespunzător rezolvării unei probleme şi care utilizează memoria M prin locații de memorie pentru fiecare tip de variabilă din V; locațiile de memorie rezervate variabilelor din V sunt utilizate de procesul de calcul P care prin execuția instrucțiunilor ce constituie P, schimbă valorile(starea) locațiilor de memorie corespunzătoare variabilelor în conformitate cu implementarea raționamentului R;
- **P – proces de calcul** ce este reprezentat de o colecție de instrucțiuni/comenzi exprimate într-un limbaj de reprezentare(cel mai utilizat fiind pseudocod-ul); folosind memoria virtuală M şi mulțimea de variabile V, colecția de instrucțiuni implementează/codifică tehnicile şi metodele ce constituie raționamentul R conceput special pentru rezolvarea unei clase de probleme; execuția instrucțiunilor din P determină o dinamică a valorilor locațiilor de memorie corespunzătoare din V; după execuția tuturor instrucțiunilor din P, soluția/soluțiile problemei se află în anumite locații de memorie ce constituie datele de iesire De;
- **R – raționament de rezolvare** exprimat prin diverse tehnici şi metode specifice domeniului din care face parte clasa de probleme supuse rezolvării (matematică/fizică/chimie, etc.), care îmbinate cu tehnici de programare corespunzătoare realizează acțiuni/procese logice prin utilizarea memoriei virtuale M şi a mulțimii de variabile V;
- **Di – date de intrare/input** ce repezintă valori ale unor parametri ce caracterizează ipoteze de lucru/stări initiale; valorile datelor de intrare sunt stocate în memoria M prin intermediul instrucțiunilor de citire/intrare ce utilizează mediul de intrare Mi; acesta este un dispozitiv virtual pentru citirea datelor de intrare;
- **De – date de ieşire/output** ce repezintă valori ale unor parametri ce caracterizează soluția/soluțiile problemei invocate de cerințele problemei/stările finale; valorile datelor de iesire sunt obținute din valorile intermediare ale unor variabile generate de execuția instrucțiunilor din procesul de calcul P şi care în final sunt stocate în memoria M în vederea transmiterii/scrierii lor către mediul de ieşire Me; acesta este un dispozitiv virtual pentru reprezentarea/scrierea datelor de ieşire sub forma grafică sau alfanumerică;
- **Mi – mediu de intrare/input** ce este un dispozitiv virtual de intrare/citire pentru citiri virtuale ale valorilor datelor de intrare pentru a fi stocate în *memoria virtuală* M;

- **Me – mediu de ieşire/output** ce este un dispozitiv de ieşire/scriere pentru preluarea datelor de ieşire din memoria virtuală M şi care au fost obținute prin execuția procesului de calcul P; datele de ieşire sunt transmise/scrise sub formă grafică sau alfanumerică pe un *suport virtual*(ecran virtual, hârtie virtuală, disk magnetic virtual, etc.).

Elaborarea

Elaborarea / conceperea algoritmului înseamnă elaborarea unui proces demonstrativ/computațional ce va constitui raționamentul de rezolvare R şi care va îngloba metode şi tehnici eficiente pentru găsirea soluției/soluțiilor problemelor pentru care se elaborează algoritmul**;** elaborarea raționamentului de rezolvare R constă din următoarele:

- definirea datelor de intrare/input (Di) şi a datelor de ieşire/output (De)
- aplicarea metodelor şi tehnicilor utilizate pentru rezolvare
- definirea mulțimii variabilelor V utilizate în rezolvare;

codificarea raționamentului de rezolvare R într-un limbaj de reprezentare(de tip pseudocod) va constitui procesul de calcul P

Reprezentare

Prin reprezentarea algoritmului se întelege exprimarea formalizată într-un limbaj de reprezentare (în general, de tip *pseudocod*) a legăturii între memoria M, mulțimea de variabile V şi *procesul de calcul* P. Forma generală a unui algoritm:

```
algorithm  <nume_algoritm>
  <declarare_variabile>     // secțiunea de declarații
begin
  <procesul_de_calcul_P>    // corpul algoritmului
end
```

Execuția

Algoritmii se consideră executați pe **maşini abstracte / virtuale** (ale căror caracteristici le "*abstractizează*" pe cele ale maşinilor de calcul / sistemelor de calcul existente la un moment dat).

Astfel de *modele* de maşini de calcul sunt:

- **Maşina Turing sau diverse automate cu stivă** (1956-1972, a se vedea A. Aho, J. Hopcroft, J.D. Ullman, Data strutures and algorithms, Addison Wesley publishing Co.,1983) ;
- **Maşina MIX** (Knuth's MIX Language, 1973, Knuth);
- **Maşina RAM** (The Random Access Machine, 1974, Aho-Hopcroft-Ullman);
- **Maşina ASM** (Abstract State Machines - A Formal Method for Specification and Verification**);**

- **Maşina VDM** (1980, Bjorner şi Jones, Vienna Development Method, programarea logică Prolog-Logic Programming)**;**
- **Maşina WAM** (1983, Warren Abstract Machine, maşina abstractă Prolog- Logic Programming)**;**
- **Maşina pseudocod** (Pseudo-Code Language, 1994, Cormen-Leiserson-Rivest)

(a se vedea şi Michael T. Goodrich, Roberto Tomassia, *Algotithm Design - Foundation, Analysis and Internet Examples*, John Wiley & Sons, Inc., 2002; http://loki.cs.brown.edu:8081/webae/full.html).

Prin intermediul unei *maşini abstracte/virtuale* (ce simulează o maşină reală / sistem de calcul) execuția algoritmului înseamnă interpretarea reprezentării algoritmului ca un mecanism virtual de tip dinamic care are o memorie virtuală M , un proces de calcul P, mediu de intrare Mi, mediu de ieşire Me şi toate aceste componente se vor confunda cu elementele corespunzătoare maşinii virtuale, excepție procesul de calcul P ce este "*lansat în execuție*" de unitatea centrală a maşinii virtuale; prin "lansarea în execuție" a procesului de calcul P de către maşină virtuală se întelege exercitarea tuturor funcțiilor unității centrale (având memorie şi procesor) pentru a simula toate procesările invocate de instrucțiunile procesului de calcul P; execuția algoritmului va însemna generarea de procese ce se vor desfăşura în timp şi care vor folosi *memorie, dispozitive de calcul şi dispoziteve* **I/O** la fel cum se întamplă cu lansarea în execuție a formei executabile a unui program pe un sistem de calcul /calculator real. În felul acesta, algoritmul ce este **sistemul virtual**

A = (M, V, P, R, Di, De, Mi, Me)

poate fi considerat ca un "*sistem de calcul*" virtual având componentele de bază: **M**-memorie internă, **P**- procesor, **Mi**-mediu de intrare, **Me**-mediu de ieşire;

Tinând seama că un *program* este codificarea algoritmului într-un limbaj de programare, *definiția* unui program este asemănătoare cu definiția unui algoritm cu deosebirea că nu mai este necesară prezența *raționamentului* **R**, iar *reprezentarea programului* se va face conform sintaxei şi semanticii limbajului de programare ales; pentru ambele concepte "execuția" evidentiază structura unui *sistem cibernetic* ce se află la baza arhitecturii unui *sistem de calcul* (ansamblu de componente hardware (dispozitive) şi software (programe) ce oferă servicii utilizatorului pentru execuția programelor ce implementează rezolvarea unor probleme.

Maşina *abstractă/virtuală* va funcționa dupa modelul unei maşini de calcul *reale*.

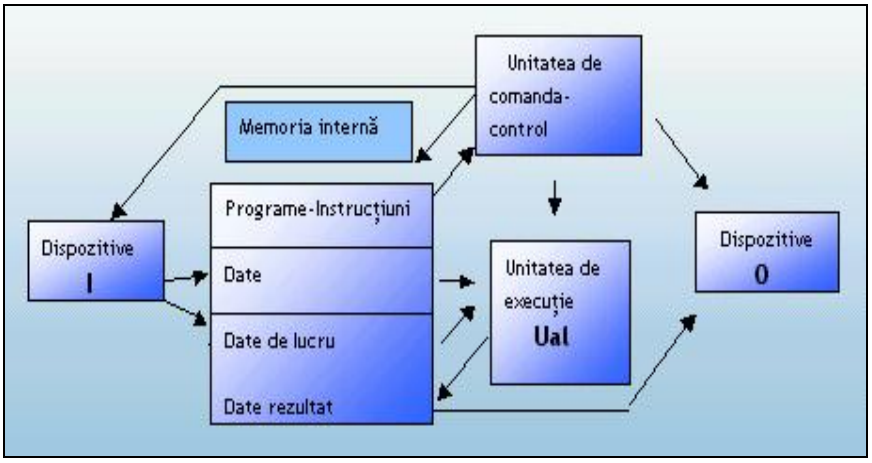

Figura 2. Schema de funcționare şi fluxul informațional

În concluzie, învățarea algoritmilor trebuie să fie precedată de învatarea elementelor de bază despre sistemele de calcul şi să preceadă învațarea programării, şi anume: *Sisteme de Calcul → Algoritmi → Programare.*

Corectitudinea şi analiza

Corectitudinea algoritmului este exprimată de *corectitudinea parțială* (procesul de calcul se termină-timpul de execuție este finit- pentru orice dată de intrare dintr-un anumit domeniu de valori) şi *corectitudinea totală* (pentru orice dată de intrare dintr-un domeniu de valori, procesul de calcul realizează valori corecte conform scopului/funcției algoritmului). Există diverse metode pentru verificarea celor două componente ale corectitudinii (de exemplu, pentru *algoritmul lui Euclid* reprezentat corect în pseudocod, *corectitudinea parțială* este verificată de faptul că şirul valorilor resturilor obținute din împărțirile succesive este un sir convergent catre 0, iar *corectitudinea totală* este verificată de metoda lui Euclid printr-o simplă demonstrație matematică). Elaborarea aplicațiilor informatice necesită *testarea programelor* care implementează diverşi algoritmi pentru ca programatorul să se convingă de corectitudinea algoritmilor concepuți.

Analiza algoritmului se referă la *spațiul de memorie* utilizat şi la timpul necesar execuției algoritmului (*timpul de execuție*); această analiză înseamnă măsurarea şi descrierea (*cantitativă*) a *performanțelor algoritmului* ce permite compararea diverselor soluții algoritmice pentru aceeaşi problemă. De regulă, resursa timp este mai critică decât resursa *spațiu de memorie*, dar sunt situații în activitatea de proiectare şi implementare a unor noi algoritmi, când se stabileşte un compromis între cerințele de spațiu de memorie şi cele referitoare la timpul de execuție.

Analiza unui algoritm cuprinde următoarele abordări:

- stabilirea unui *model de calcul* ;
- stabilirea t*impului de calcul / de execuție* ;
- stabilirea *ratei de creştere*.

Model de calcul

Acest model va specifica *operațiile fundamentale* elementare) pe care le utilizează algoritmul şi *costul* (în unitați de timp) asociat fiecărei operații elementare. În continuare vom prezenta câteva exemple :

- *algoritmii numerici* sunt analizați prin numărarea operațiilor aritmetice mai costisitoare (costul unei înmulțiri sau al unei împărțiri este mult mai mare decat al unei adunări sau al unei scăderi) ;
- *algoritmii de sortare/căutare* sunt analizați prin numărarea operațiilor de comparație;
- *algoritmii din geometria computațională* care realizează operații asupra poligoanelor, sunt analizați prin numărarea vârfurilor sau muchiilor prelucrate;

Timpul de execuție

Există aşa-numite *măsuri de complexitate* care descriu aspectul de performanță care trebuie măsurat: timpul de execuție în *cazul cel mai defavorabil*, în *cazul mediu* şi în *cazul amortizat* (marginea superioară în cazul cel mai defavorabil). Aceste masuri de complexitate depind de volumul setului de date de intrare.

Rata de creştere

Prin *analiza asimptotică* se stabileşte rata de creştere a timpului de execuție în funcție de volumul setului de date de intrare. Analiza asimptotică exprimă creşterea timpului de execuție al unui algoritm în cazul creşterii (spre infinit) a volumului setului de date de intrare. Dacă timpul de execuție este exprimat de funcția *f(n)*, n fiind numărul de elemente de intrare, atunci rata de creştere a lui *f(n)* este dată de funcția *T(n)*. De exemplu, dacă $f(n) = an^2 + bn + c$, unde *a, b, c* sunt constante, atunci când *n* creşte spre infinit, termenii de ordin *1* şi *0* sunt nesemnificativi şi astfel în acest caz rata de creştere a lui *f(n)* este funcția $T(n) = n^2$.

În funcție de variația funcției *T(n)* există următoarele categorii de algoritmi: *lineari, pătratici, cubici, exponențiali, logaritmici, liniar-logaritmici* etc.

Notațiile utilizate în analiza asimptotică a algoritmilor sunt urmatoarele:

- *O(f(n))* ce repezintă clasa funcțiilor care cresc mai puțin decât *f(n)* când *n* tinde către infinit;
- *o(f(n))* ce exprimă clasa funcțiilor care cresc strict mai lent decât *f(n)* când *n* tinde către infinit;
- *Ω(f(n))* ce exprimă clasa funcțiilor care nu cresc mai încet decât *f(n)* când *n* tinde către infinit*;*
- *Θ(f(n))* ce exprimă clasa funcțiilor care cresc cu aceeaşi rată ca şi *f(n)* când *n* tinde către infinit.

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,* Bucureşti, 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. Aplicaţii, *Ed. Teora,* Bucureşti, 2000.
[10] Vlada*,* M., Informatică*, Universitatea din Bucureşti, Ed. Ars Docendi,* Bucureşti, 1999.
[11] Vlada, M., Poligoane stelate. Problema lui Hopf şi Pannwitz*, Gazeta de matematică, nr. 8/1995, pag. 339-348.*
[12] Vlada, M., Rezolvarea problemelor folosind Eureka, *software educaţional*, www.unibuc.ro/eBooks/informatica/eureka/, *Universitatea din Bucuresti*, 2003.
[13] Vlada, M., Concepul de algoritm-abordare modernă, *Gazeta de informatică*, vol. 13/2 şi 3, pp. 25-30, pp. 35-39, *Agora*, Cluj Napoca, 2003.
[14 M. Vlada, Birotică: Tehnologii multimedia, Editura Universităţii din Bucureşti, ISBN 973-575-847-4, 2004.
[15] Zaharia*,* M. D., Structuri de date şi algoritmi. Exemple în limbajele C şi C++, *Ed. Albastră,* Cluj-Napoca, 2002.