PROIECTAREA ALGORITMILOR

CURS 1- 02.03.2021

Titular: Şef. Lucr. Dr. Mat. Cărbureanu Mădălina

Universitatea Petrol-Gaze din Ploieşti

Bibliografie recomandată

- **Cărbureanu. M., *Elemente de proiectarea algoritmilor. Ghid teoretic şi practic*, Editura Universităţii Petrol-Gaze din Ploieşti, Ploieşti, 2021.**
- Andonie, R., Gârbacea, I., *Algoritmi fundamentali. O perspectivă C++,* Editura Libris, Cluj-Napoca, 1995.
- Ciurea, E., Iolu, M., *Algoritmi. Teorie şi aplicaţii*, Editura Universităţii Transilvania din Braşov, 2008.
- Cormen, T., Leiserson, C., Rivest, R., Stein, C., *Introduction to Algorithm*, Third Edition, MIT Press, England, 2009.
- Even, S., *Graph Algorithms*, 2nd Edition, Cambridge University Press, USA, 2012.
- Gross, J., Yellen, J., Zhang, P., *Handbook of Graph Theory, Second Edition*, CRC Press, Taylor & Francis Group, USA, 2014.
- Jungnickel, D., *Graphs, Networks and Algorithms*, *Second Edition*, Springer Berlin Heidelberg, New York, 2005.
- Jamsa, K., Klander, L., *Totul despre C si C++. Manualul fundamental de programare în C şi C++*, Editura Teora, Bucureşti, 1999.
- Knuth, D. E., *Tratat de programarea calculatoarelor. Algoritmi fundamentali,* Editura Tehnică, Bucureşti, 1974.
- Knuth, D. E., *The Art of Computer Programming: Fundamental algorithms*, Vol.1, Addison-Wesley Publisher, Massachusetts, 1997.
- Knuth, D. E., *The Art of Computer Programming: Combinatorial Algorithms,* Volume 4A, Part 1, 1st Edition, Massachusetts, 2011.

- Kernighan, B. W., Ritchie, D. M., *The C Programming Language*, Second Edition, Prentice Hall Software Series, 1988.
- Knuth, D. E., *Tratat de programarea calculatoarelor. Sortare şi căutare*, Editura Tehnică, Bucureşti, 1976.
- Kleinberg, J., Tardos, E., *Algorithm Design*, Pearson, Addison-Wesley, USA, 2005.
- Lafore, R., *Sams Teach Yourself Data Structures and Algoritms in 24 Hours*, Sams Publishing, USA, 1999.
- Marinoiu, C., *Programarea în limbajul C,* Editura Unversităţii-Petrol Gaze din Ploieşti, 2000.
- Mueller J. P, Massaron, L., *Algorithms for Dummies,* John Wiley &Sons, New Jersey, 2017.
- Mehta, D. P., Sahni, S., *Handbook of Data Structures and Applications,* Chapman&Hall/CRC, USA, 2005.
- Năchilă, C., Duşmănescu, D., *Structuri de date şi algoritmi*, Editura Solness, Timişoara, 2009.
- Roberts, E., *Thinking Recursively*, John Wiley&Sons, Inc., USA, 1986.
- Roughgarden, T. *Algorithms Illuminated. Part 2: Graph Algorithms and Data Structures,* Soundlikeyourself Publishing, San Francisco, 2018.
- Schildt, H., *C manual complet,* Editura Teora, 2002.

- Sedgewick, R., Wayne, K., *Algorithms*, Fourth Edition, Addison-Wesley, USA, 2011.
- Schildt, H., *C. The Complete Reference*, Fourth Edition, McGraw-Hill Companies, USA, 2000.
- Skiena, S., *The algorithm Design Manual.* Second Editon, Springer-Verlag, USA, 2008.
- Shaffer, C., *Data Structures and Algorithm Analysis*, Edition 3.2 (C++ Version), Departament of Cmputer Science Virginia Tech, Blacksburg, 2011.
- Sedgewick, R., *Algorithms in C*, Addison-Wesley, USA, 1990.
- Sedgewick, R., *Algorithms in C, Third Edition, Part 5 Graph Algorithms,* Addison-Wesley, USA, 2002.
- Sedgewick, R., *Algorithms in C, Third Edition, Parts 1-4 Fundamentals Data Structures. Sorting. Searching,* Addison-Wesley, USA, 1998.
- Vlădoiu, M., Constantinescu, Z., Moise, G., *Structuri de date fundamentale,* Editura Universităţii Petrol-Gaze din Ploieşti, 2016.
- Wirth, N., *Algorithm +Data Structures=Programs*, Prentice Hall, USA, 1976.
- Wirth, N., *Algorithms and Data Structures*, Prentice Hall, USA, 1985.
- West, D., *Introduction to Graph Theory. Second Edition*, Pearson Education, India, 2002.

CURS 1- Algoritmi

- Noțiunea de algoritm;
- Exemple de algoritmi și caracteristici;
- Studiul algoritmilor;
- Reprezentarea algoritmilor;
- Metode de proiectare a algoritmilor;
- Exemple de aplicații.

Noțiunea de algoritm

- Noțiune ce stă la baza programării calculatoarelor și a proiectării algoritmilor;
- Dezvoltarea algoritmului- primul pas în realizarea oricărui program;
- Termenul **algoritm** - "al-Khowârizmî", ceea ce înseamnă din "orașul Khowârizm" (orașul Khiva din Uzbechistan);
- Abu Ja'far Mohammed ibn Musâ al-Khowârizmî (secolul VIII-IX) - cartea de matematică *Algorithmi de numero indorum*, în cadrul căreia noțiunea de algoritm era strict folosită în sens matematic;

- **Algoritm** - set finit și general (abordează toate cazurile posibile) de reguli care furnizează o secvență ordonată de operații (transformări precis definite, neambigue), operații care se aplică datelor de intrare (pentru obținerea datelor de ieșire dorite), în vederea soluționării unui tip specific de probleme [Knuth, D. E.];
- **Algoritm** - metodă generală de rezolvare a unui anumit tip de problemă, metodă ce poate fi implementată pe calculator prin intermediul unui program [Andonie, R., Gârbacea, I.];
- **Program** - modalitatea prin care un algoritm este exprimat într-un limbaj de programare de nivel mediu (C, C++) sau de nivel înalt (Java, PHP, Prolog, Python, etc.);
- **Algoritm -** șir de reguli care aplicat la o anumită clasă de probleme de același tip, conduce la soluția problemei respective, cu ajutorul unui număr finit de operații succesive [Ciurea, E., Iolu];
- **Algoritm** - număr finit de pași (fiecare pas este alcătuit la rândul lui din una sau mai multe operații sau instrucțiuni), a căror execuție trebuie să se finalizeze într-un timp rezonabil.

Exemple de algoritmi și caracteristici

- **Algoritmi de căutare**: căutare secvențială, căutarea binară (*Algoritmi de căutare și sortare*);
- **Algoritmi de sortare**: sortarea prin interschimbare (**Bubble Sort)**, sortarea prin interclasare (**Mergesort)**, sortarea rapidă (**Quick Sort)**, sortarea prin numărare (**Count Sort**), sortarea prin selecție (**Selection Sort**), sortarea prin inserție (**Insertion Sort**) (*Algoritmi de căutare și sortare*);
- **Algoritmul lui Euclid**: pentru calculul celui mai mare divizor comun a două numere întregi (*Recursivitatea-metodă principală de proiectare a algoritmilor*);

- **Algoritmul lui Kruskal** și **algoritmul lui Prim**: pentru determinarea arborelui parțial de cost minim al unui graf neorientat (*Algoritmi pentru determinarea arborelui parțial de cost minim*);

- **Algoritmul lui Dijkstra** și **algoritmul Roy-Floyd**: pentru determinarea celor mai scurte drumuri care pleacă din același vârf într-un graf (orientat/neorientat); (*Algoritmi pentru determinarea celor mai scurte drumuri*);
- **Algoritmul Roy-Warshall**: pentru determinarea matricei drumurilor plecând de la matricea de adiacență asociată unui graf orientat (*Algoritmul Roy-Warshall pentru determinarea matricei drumurilor*).

Caracteristici algoritmi [Knuth, D. E.]

- **Caracterul finit:** un algoritm trebuie întotdeauna să se termine după un număr finit de pași;
- **Caracterul determinist:** fiecare pas din componența unui algoritm trebuie să fie definit în mod precis, clar, fără ambiguități;
- **Intrarea**: un algoritm fie nu prezintă intrare, fie prezintă mai multe intrari;
- **Ieșirea:** un algoritm are una sau mai multe ieșiri;
- **Eficacitatea:** toate operațiile din cadrul unui algoritm care se doresc a fi executate trebuie să fie suficient de fundamentale, astfel încât să fie realizate exact și într-un interval finit de timp, de către programator;
- ! **un algoritm bun trebuie să fie eficient**.

Caracteristici algoritmi [Ciurea, E., Iolu, M.]

- **Generalitatea (universalitatea)**: algoritmul rezolvă o clasă de probleme de același tip și nu doar o singură problemă;
- **Finititudinea (eficacitatea)**: trebuie să existe un număr finit de operații ce se pot aplica pentru obținerea soluției problemei;
- **Unicitatea (claritatea)**: toate operațiile care se aplică pentru obținerea soluției problemei, sunt univoc determinate de regulile algoritmului.

Studiul algoritmilor

- **Studiul algoritmilor** cuprinde mai multe aspecte, precum:
- **Elaborarea algoritmilor – creativitatea umană + intuiția și experiența programatorului + tehnici fundamentale de proiectare a algoritmilor**, în vederea obținerii unor algoritmi eficienți;
  - etapă absolut necesară atunci când pentru rezolvarea unei probleme date, nu se cunosc algoritmi pentru rezolvarea acesteia sau algoritmii existenți sunt ineficienți;
- **Descrierea algoritmilor** – exprimarea în mod natural a metodei de rezolvare a problemei (a algoritmului aferent) într-un program trebuie să fie clară și concisă ➡ utilizarea unui anumit limbaj de programare (programare structurată, programare orientată pe obiect);
  - algoritmul poate fi exprimat în mod natural sub formă de schemă logică sau pseudocod;

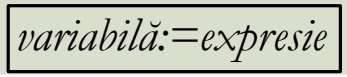
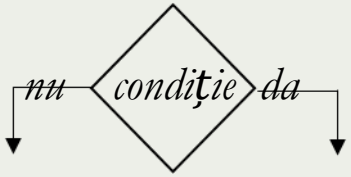
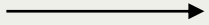
- **Validarea algoritmilor** – este necesară validarea algoritmului pentru a ne asigura că algoritmul proiectat este corect, adică furnizează rezultatul dorit pentru orice situație posibilă, indiferent de limbajul în care acesta va fi programat;

- **Analiza complexității algoritmilor** – pentru a putea stabili care dintre algoritmii care rezolvă același tip de problemă este mai bun, se utilizează diferite criterii de apreciere a eficienței unui algoritm, precum: durata de timp necesară pentru execuția algoritmului (timpul de calcul), spațiul de memorie utilizat de către acesta, adaptabilitatea algoritmului la calculatoare, simplitatea și eleganța lui;
- **Implementarea algoritmului** – aceasta constă din două subetape și anume: programarea algoritmului într-un limbaj de programare cunoscut + testarea programului elaborat;
  - **testarea programului:** depanare (debugging) + trasare (profiling);
  - **depanare** - procesul de executare a unui program pe date de test și corectarea eventualelor erori;
  - **trasare** - procesul de executare a unui program corect pe diferite date de test, cu scopul de a determina timpul de calcul și memoria utilizată.

Reprezentarea algoritmilor

- **schemele logice** – este un mod de reprezentare grafică a modului în care se execută operaţiile în cadrul unui algoritm; se utilizează blocuri cu înțeles prestabilit (tabelul 1), precum: blocul de început/sfârșit, blocul de intrare/ieșire (I/E), blocul de calcul, blocul de decizie, blocul pentru subalgoritmi, liniile de flux și blocul conector;
- **pseudocodul** (limbajul algoritmic) – este un limbaj simplificat, o cale de mijloc între limbajul de programare și limbajul natural, ce utilizează o serie cuvinte cheie (*dacă-if*, *atunci-then*, *altfel-else*, *cât timp-while*, *citește-read*, *scrie-write*, *pentru-for*, *execută-do*, etc.), aferente instrucțiunilor (cu excepția instrucțiunii de atribuire).

Tabelul 1.

Reprezentare	Descriere
	*Bloc de început/sfârșit*: desemnează operația de pornire, respectiv de oprire a procesului de calcul.
	*Bloc de I/E*: desemnează operația de introducere (citire), respectiv de afișare (scriere) a datelor.
	*Bloc de calcul (de operații):* desemnează operația de atribuire (:=/←).
	*Bloc de decizie*: desemnează o operație de decizie (se evaluează valoarea de adevăr a unei condiții funcție de care se execută instrucțiunile de pe ramura Da sau Nu.
	*Bloc pentru subalgoritmi*: grupează o serie de operații de calcul ca un algoritm independent (funcție sau procedură).
	*Linii de flux*: indică sensul de parcurgere al operațiilor succesive.
	*Bloc conector*: leagă două linii de flux.

Metode de proiectare a algoritmilor

- **Recursivitatea** – este principala și cea mai simplă metodă de proiectare a algoritmilor;
  - mecanism în programare care permite scrierea și folosirea unor funcții (funcții recursive directe și funcții recursive indirecte), care se autoapelează (ce se petrece la un nivel va avea loc și la nivelele următoare, dar cu alte valori ale parametrilor) direct sau indirect;
- **Metoda Divide et Impera** ("Împarte/dezbină și stăpânește") – constă în împărțirea repetată a unei probleme complexe de dimensiune mai mare, în mai multe subprobleme mai simple de același tip, urmată de combinarea rezultatelor subproblemelor, în vederea obținerii soluției problemei inițiale;
  - probleme rezolvabile prin Divide et Impera: căutarea binară, Turnurile din Hanoi, sortarea rapidă (Quick Sort), sortarea prin interclasare (Merge Sort), determinarea maximului/minimului dintr-un vector, problema dreptunghiului de arie maximă, problema calculului puterii unui număr, etc.;

- **Metoda Greedy** (“avidă, lacomă”) – se aplică problemelor în care pentru o mulțime dată X={x1, x2,…,xn}, se cere determinarea unei submulțimi a lui X, respectiv S cu m elemente (m≤n), care să îndeplinească anumite condiții, ce diferă de la o problemă la alta;
  - se aplică problemelor de optimizare: problema restului de plată, problema submulţimii cu valoare maximă, problema rucsacului, problema spectacolelor, probleme pentru determinarea celor mai scurte drumuri într-un graf, probleme de determinare a arborelui parţial de cost minim într-un graf, etc.;
- **Metoda Backtracking** (“Căutare cu revenire”) – se aplică problemelor în care soluția se poate reprezenta sub forma unui vector S=(x1, x2,…,xn), soluția construindu-se pas cu pas; dacă se constată că pentru o valoare aleasă, nu se poate obține soluția, se renunță la respectiva valoare și se reia căutarea din punctul în care s-a ajuns;
  - probleme rezolvabile prin această metodă sunt: problema colorării hărţilor, problema generării permutărilor unei mulțimi cu n elemente, problema reginelor, produsul cartezian a n mulțimi, generarea aranjamentelor și combinărilor, problema comis-voiajorului, problema discretă a rucsacului, etc.;

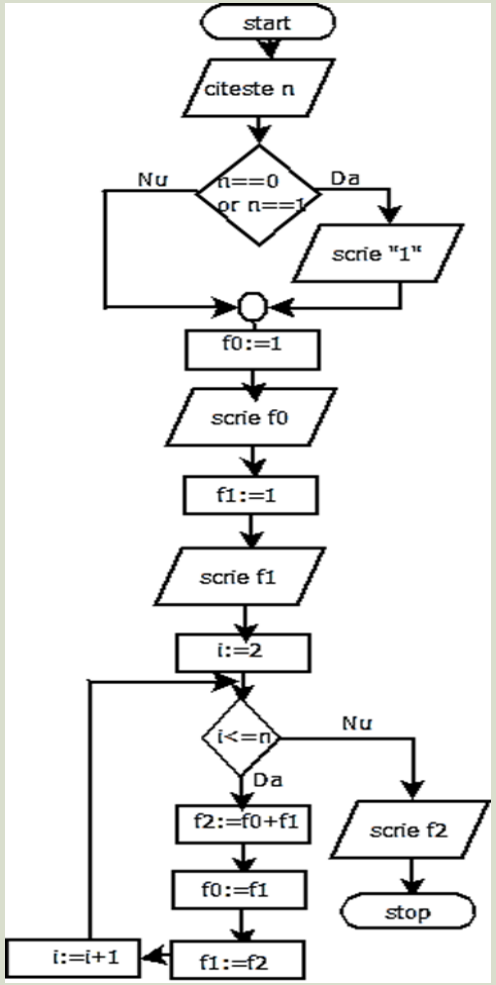
- **Metoda Branch și Bound** (“Ramifică și mărginește”) – este o metodă înrudintă cu metoda backtracking, prin faptul că se poate reprezenta pe un arbore rădăcină, numit arborele rădăcină al spațiului de stări posibile; un nod din arbore este o soluție posibilă (stare posibilă); problema constă în determinarea drumului minim de la o stare posibilă la starea admisibilă, adică la starea posibilă care verifică anumite condiţii;
  - o problemă rezolvabilă prin această metodă este așa numitul joc Perspico;
- **Metoda programării dinamice** – se aplică problemelor de optimizare în care soluția optimă este rezultatul unui șir de decizii din mulţimea deciziilor, deci se aplică problemelor pentru care optimul general implică optimul parțial; de obicei, rezolvarea unei probleme cu această metodă presupune verificarea unui principiu de optimalitate, scrierea și rezolvarea unor relații de recurență care cuantifică modul de obținere a optimului general din optime parțiale și abia apoi scrierea programului;
  - probleme rezolvabile prin această metodă sunt: problema discretă a rucsacului, problema distanţei și a drumului minim între oricare două noduri într-un graf orientat, etc.

Exemple de aplicații

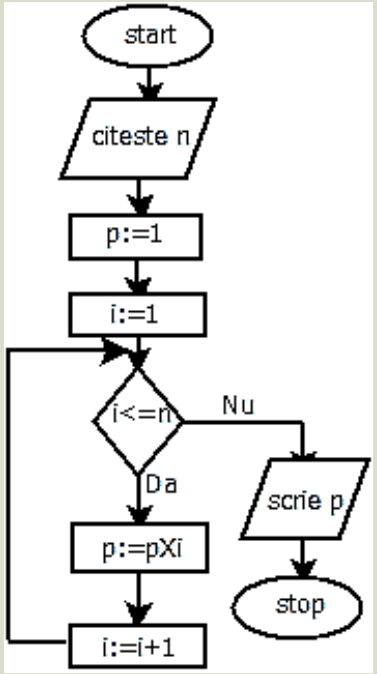
- **Schemă logică și pseudocod pentru șirul lui Fibonacci;**
- **Schemă logică și pseudocod pentru calcul *n*!;**
- **Schemă logică și pseudocod pentru calcul cmmdc(*x*, *y*).**

Schemă logică şi pseudocod pentru şirul lui Fibonacci

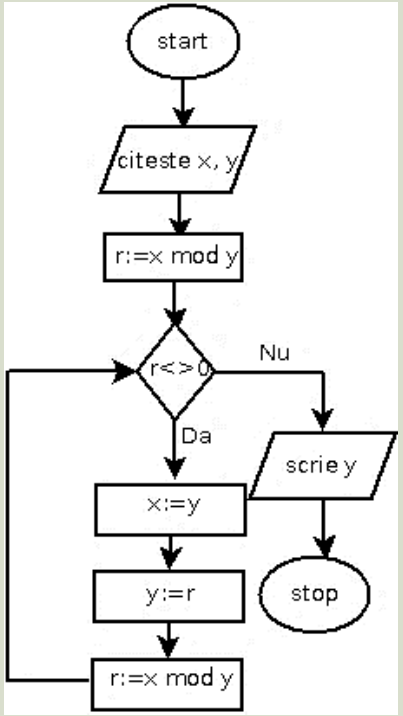
n=8 şirul este: 0, 1, 1, 2, 3, 5, 8, 13

Schemă logică	Pseudocod
	**(1)** *program Fibonacci;*
	(2) *început*
	(3) *citește n;*
	(4) *dacă n = =0 sau n = =1 atunci*
	(5) *scrie ”1”;*
	(6) *sfârșit dacă;*
	(7) *f0←1;*
	(8) *scrie f0;*
	(9) *f1←1;*
	(10) *scrie f1;*
	(11) *pentru i←2 la n execută*
	(12) *f2←f0+f1;*
	(13) *f0←f1;*
	(14) *f1←f2;*
	(15) *sfârșit pentru;*
	(16) *scrie f2;*
	(17) *sfârșit.*

Schemă logică și pseudocod pentru calcul *n*!

Schemă logică	Pseudocod
	**(1)** *program factorial;*
	(2) *început*
	(3) *citește n;*
	(4) *p←1;*
	(5) *pentru i←1 la n execută*
	(6) *p←p×i;*
	(7) *sfârșit pentru;*
	(8) *scrie p;*
	(9) *sfârșit.*

Schemă logică şi pseudocod pentru calcul cmmdc(*x*, *y*)

Schemă logică	Pseudocod
	**(1)** *program cmmdc;* **(2)** *început* **(3)** *citeşte x, y;* **(4)** *r←x mod y;* **(5)** *cât timp r <> 0 execută* **(6)** *x←y;* **(7)** *y←r;* **(8)** *r←x mod y;* **(9)** *sfârşit cât timp;* **(10)** *scrie y;* **(11)** *sfârşit.*

Să vă fie de folos!