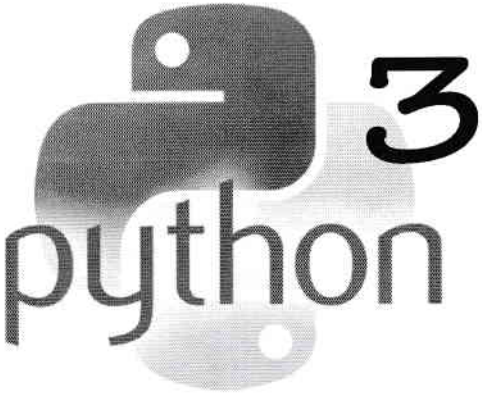
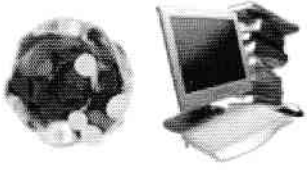
Vlad TUDOR

Curs de programare în



Volumul II

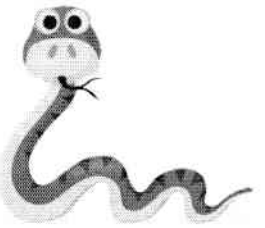
Structuri de date și algoritmi



INFOBITS.RO
L&S SOFT

Cuprins

Volumul II – Structuri de date și algoritmi (Teorie și aplicații)



Cuvânt înainte 9

Capitolul 1. Despre algoritmi 11
1.1. re(Introducere) 11
1.2. Enunțul unei probleme, date, etapele rezolvării 14
1.3. Noțiunea de algoritm, caracteristici 19
1.4. Obiectele cu care lucrează algoritmii și operațiile permise 22
1.5. Complexitatea algoritmilor 23

Capitolul 2. Tablouri 27
2.1. Introducere 27
2.2. Citirea și afișarea unui vector 31
2.2.1. Citirea unui vector 31
2.2.2. Afișarea unui vector 33
2.2.3. Citirea/afișarea vectorilor folosind fișiere text 34
Miniproiect 39
2.3. Algoritmi fundamentali care lucrează cu vectori 40
2.3.1. Maxim, minim 40
2.3.2. Elemente distincte 44
2.3.3. Metode de sortare 48
A. Sortarea prin selectarea minimului (maximului) 49
B. Sortarea prin interschimbare (Bubble Sort) 52
C. Sortarea prin inserție 54
2.3.4. Interclasare 57
2.3.5. Căutare binară 61

2.4. Aplicații cu matrice 64
2.4.1. Citirea și afișarea matricelor 64
2.4.2. Interschimbarea liniilor 66
2.4.3. Spirala 68
2.5. Biblioteca *NumPy* 70

Probleme propuse 75

Capitolul 3. Introducere în recursivitate 78
3.1. Prezentare generală 78
3.2. Modul în care se realizează autoapelul 79
3.3. Mecanismul recursivității 80
3.4. Cum gândim un algoritm recursiv? 81
3.5. Aplicații recursive 83
3.5.1. Aplicații la care se transcrie o formulă recursivă 83
3.5.2. Recursivitate indirectă 89
3.5.3. Aplicații la care nu dispunem de o formulă de recurență 90

Probleme propuse 92
Indicații / rezolvări 95

Capitolul 4. Divide et Impera 99
4.1. Prezentare generală 99
4.2. Aplicații 100
4.2.1. Valoarea maximă din vector 100
4.2.2. Sortarea prin interclasare 101
4.2.3. Turnurile din Hanoi 103
4.3. Fractali 107
4.3.1. Lucrul în mod grafic 108
4.3.2. Curba lui Koch pentru un triunghi echilateral. Fulg de nea 111
4.3.3. Curba lui Koch pentru un pătrat 114
4.3.3. Arborele 118

Aplicație practică 120

Capitolul 5. Alte structuri de date 121
5.1. Structuri studiate anterior 121
5.2. Unde este struct (din limbajul C++)? 121
5.3. Structura de tip stivă 123
5.4. Structura de tip coadă. Modulul *collections* 129

5.5. Liste liniare 132
5.5.1. Mai mult despre structurile de date 132
5.5.2. Liste liniare 133
5.5.3. Liste liniare alocate secvențial 134
5.5.4. Liste liniare alocate înlănțuit 136

Capitolul 6. Tratarea erorilor 140

Capitolul 7. Direcții de cercetare – proiecte propuse 146
7.1. Scrie un articol – ajută-i pe alții și implicit pe tine! 146
7.2. Creează mici proiecte utile pentru cei din jur 147
7.3. Studiază în detaliu biblioteca *Matplotlib* 148
7.4. Prelucrarea imaginilor 151
7.5. Machine Learning 152
7.6. Prelucrarea video 153
7.7. Ajută un meseriaș! – proiect practic și util 154

Anexa 1. Tabelul codurilor ASCII 162

Anexa 2. Baze de numerație 163

Anexa 3. Cum se memorează datele? 173
Exerciții propuse 182

Anexa 4. STEM. Exemple de utilizare a algoritmilor *(fizică și chimie)* 183
Proiecte propuse 187

Anexa 5. Modulul *turtle* – detalii 188
A5.1. Introducere 188
A5.2. Spațiul de lucru și Țestoasa 190
A5.3. Metode și proprietăți utile 192
A5.4. Trasarea liniilor curbe (Bezier: cuadratice, cubice) – *cool!* 195
Aplicație. O inimioară în Python 3? 197

Anexa 6. Un operator interesant - *walrus* 200

Anexa 7. PEP8 – Ghidul oficial al stilului de redactare a codului 203

Bibliografie selectivă 204

Capitolul 1
Despre algoritmi

1.1. (re)Introducere

Noțiunea de algoritm este primară, nu se definește, întocmai ca și noțiunea de mulțime în matematică. Cu toate acestea, ca și mulțimea, algoritmul poate fi descris. *În esență, este vorba de o succesiune de etape care se pot aplica mecanic în vederea obținerii unui anumit rezultat.*

În activitatea cotidiană întâlnim la tot pasul algoritmi. Vom da câteva exemple mai jos.

1. *Algoritmul prin care o persoană dădea un telefon în anul 1984*. **:)** Persoana ia receptorul, așteaptă tonul. Dacă nu vine tonul, închide telefonul, apoi îl redeschide. După apariția tonului formează numărul, etc. Nu îmi propun să descriu amănunțit algoritmul prin care se dă un telefon pentru că este mult prea cunoscut, dar atrag atenția asupra faptului că se poate descrie foarte bine, astfel încât persoana poate executa mecanic operațiile necesare.

2. *Algoritmul prin care o persoană poate găti un anumit fel de mâncare*. Persoana are la dispoziție făină, zahăr, ouă etc. pe care le combină în anumite proporții, după care amestecul obținut se fierbe (se prăjește sau se pune în cuptor) un anumit interval de timp. Produsului obținut i se mai poate adăuga câte ceva, după care este din nou fiert (prăjit) un interval de timp, după care se obține produsul finit, adică mâncarea dorită.

3. *Algoritmul prin care se adună două fracții*. Cele două fracții se aduc la același numitor, se fac înmulțirile, adunările, apoi fracția este simplificată.

Oricare din algoritmii de mai sus poate fi descris în termeni preciși, astfel încât cel care-l execută poate efectua operațiile fără să fie nevoit să gândească ce are de făcut la un anumit moment.

O analiză sumară a exemplelor ne conduce la următoarele observații:

- *În orice algoritm se pornește de la ceva și se dorește obținerea unui anumit rezultat.*
  - Dăm un telefon pentru ca un anumit mesaj să ajungă la destinație. Se pornește de la un mesaj și se dorește ca acesta să ajungă la o anumită persoană.
  - Dacă dorim să gătim un anumit fel de mâncare, pornim de la anumite produse și obținem mâncarea solicitată – **rețeta**.
  - Dacă adunăm două fracții, pornim de la cele două fracții și obținem suma lor - **metoda**.
- *În orice algoritm se operează cu anumite "obiecte" asupra cărora sunt permise anumite operații.*
  - Algoritmul prin care dăm un telefon operează cu telefonul. Operațiile permise sunt deschiderea, închiderea telefonului, formarea numărului, etc.
  - Algoritmul prin care se obține un anumit fel de mâncare operează cu farfurii, tăvi, aragaz, alimente, etc. Operațiile permise sunt amestecarea, fierberea, prăjirea alimentelor.
  - Algoritmul prin care se adună două fracții operează cu valori numerice. Operațiile sunt cele descrise de regulile matematice.
- *În cele mai multe cazuri cel care elaborează algoritmul este diferit de executant.*
  - Pentru a putea fi aplicat algoritmul prin care se dă un telefon a fost necesară inventarea telefonului, gândirea modului în care cineva dă ușor un telefon. O mulțime de personalități au gândit toate acestea, dar algoritmul poate fi aplicat de orice persoană care știe cifrele între **0** și **9**.
  - Pentru a putea fi aplicat algoritmul prin care se obține un anumit fel de mâncare au fost făcute numeroase experimente care au

condus până la urmă la o rețetă. Poate găti orice persoană care știe să citească și să folosească aragazul / plita / cuptorul.

- Pentru a putea aduna două fracții a fost necesar ca matematica să se dezvolte suficient de mult. Să nu uităm că oamenii au lucrat mult timp doar cu numere naturale...

Din cele de mai sus, rezultă că noțiunea de algoritm este extrem de generală, cu ea ne întâlnim tot timpul. *În această carte ne ocupăm numai de elaborarea algoritmilor pentru programarea calculatoarelor.* Aici "executantul" este calculatorul, iar cel care elaborează algoritmul poartă numele de "**programator**". Calculatorul doar execută instrucțiuni, nu gândește, dar viteza de executare a instrucțiunilor este foarte mare, imposibil de atins de om. Frecvent, mai intervine o persoană, de cele mai multe ori diferită de programator, numită "**utilizator**". Ea este cea care utilizează programul obținut și beneficiază de avantajele lui. De cele mai multe ori, o astfel de persoană are o pregătire minimă de specialitate în informatică. Din păcate, se face deseori confuzia între programator și utilizator.

Așadar, reține faptul că un program cuantifică un algoritm de calcul și are rolul de a prelua anumite **date de intrare** (precum niște stimuli) pe care le **prelucrează** și le oferă ca **date de ieșire** (un răspuns):

Acesta se numește modelul **black-box**, care abstractizează primar orice element "viu" din jurul nostru. Motorul mașinii are o turație mai mare dacă este apăsată mai tare pedala de accelerație; dacă ne este frig, mușchiul firului de păr se contractă și ni se face "pielea de găină"; dacă apăsăm pe telefon o anumită pictogramă, se deschide apoi programul asociat, ș.a.m.d.

Datele de intrare sunt "**citite**" cu ajutorul unui dispozitiv periferic de intrare, cum ar fi: tastatura, mouseul, controllerul, ecranul tactil, etc.

După prelucrare, rezultatul este "**afișat**" pe ecran ori trimis către un alt periferic de ieșire: ecran, imprimantă, etc.

1.2. Enunțul unei probleme, date de intrare și de ieșire, etapele rezolvării unei probleme

Wow, funcții matematice! Să presupunem că se consideră funcția:

$$f:\Re\to\Re,\quad f(x)=\begin{cases}x^2-2, & x<0;\\ 3, & x=0;\\ x+2, & x>0.\end{cases}$$

Se cere să se calculeze **f(x)** pentru **10** valori reale ale lui **x**. De exemplu: **-3.1, 7, 0, 2.23,** ș.a.m.d.

Considerăm prima valoare a lui **x**, și anume **-3.1**. Observăm că această valoare este mai mică decât **0**. Calculăm **(-3.1)^2-2**. A doua valoare pentru care trebuie calculată funcția este **7**. Pentru că ea este mai mare decât **0**, calculăm **x+2**, adică **7+2**. A treia valoare este **0**. Funcția ia valoarea **3**. Repetăm calculul pentru cele **7** valori rămase. Această modalitate de calcul este plicticoasă și cere mult timp. Pentru a rezolva astfel de probleme - și nu numai de tipul acesteia - a fost inventat calculatorul. Acesta va efectua calculele în locul nostru. Observați faptul că *pentru efectuarea calculelor, calculatorul trebuie să ia anumite decizii automat*. Astfel, în funcție de valoarea lui **x** (negativă, zero, sau mai mare decât 0) acesta va decide ce expresie calculează pentru a obține **f(x)**. Dacă calculele elementare pot fi făcute de un simplu calculator de buzunar, neprogramabil, deciziile automate le poate lua numai un calculator programabil. Evident, deciziile se iau în urma aplicării algoritmului.

Calculatorul rezolvă o problemă atunci când execută un anumit program, corespunzător problemei. În esență, programul este alcătuit din comenzi (instrucțiuni) pe care calculatorul le execută.

Programul se obține prin codificarea algoritmilor într-un limbaj de programare.

În continuare, prezentăm în linii mari etapele obținerii unui program.

1. Identificarea datelor de intrare și a celor de ieșire

Am văzut faptul că în orice algoritm se pornește de la ceva și se urmărește un anumit rezultat. Cu alte cuvinte, trebuie să ne fie clar de la ce plecăm și ce vrem să obținem. Aceasta înseamnă că trebuie să cunoaștem **datele de intrare** - de la ele plecăm - și **datele de ieșire** - pe ele trebuie să le obținem. Pentru exemplul considerat, datele de intrare sunt **10** valori reale $x_1, x_2, ..., x_{10}$. Datele de ieșire sunt cele **10** valori calculate $f(x_1), f(x_2), ..., f(x_{10})$.

Observație. Algoritmii operează cu date de intrare și de ieșire, chiar și atunci când acest fapt nu este atât de evident. Să presupunem că jucăm un joc pe calculator. Aceasta are loc sub controlul unui anumit program, care a fost obținut în urma codificării unui algoritm. O dată de intrare poate fi apăsarea unei taste, un clic al *mouse*-ului, etc. O dată de ieșire poate fi o anumită imagine sau o succesiune de imagini care creează impresia că un obiect se deplasează, un anumit sunet (și acestea sunt codificate numeric), etc.

2. Elaborarea algoritmului de rezolvare a problemei

În linii mari, algoritmul specifică operațiile pe care le "are de făcut" calculatorul pentru ca, pornind de la datele de intrare, să obțină datele de ieșire. Iată, de exemplu, cum arată algoritmul simplificat de rezolvare a problemei propuse:

Se parcurg următoarele etape, de **10** ori:

- se citește valoarea lui **x**;
- în funcție de valoarea proprie a lui **x**, se procedează astfel:
  - dacă este negativă, se calculează $f=x^2-2$;
  - dacă este **0**, valoarea funcției este **3**;
  - dacă este pozitivă, se calculează **f=x+2**;
- se scrie valoarea calculată pentru **f**.

Observați faptul că algoritmul a fost descris în limbaj natural. În practică se folosesc **limbaje de tip pseudocod** sau, din ce în ce mai rar, **scheme logice.**

Ce este un limbaj de tip pseudocod?

După cum știm, programele pentru calculator sunt scrise în anumite limbaje: **C++**, **Python**, **Java**, etc. Pentru a scrie programele într-un limbaj de programare este necesar să respectăm anumite reguli specifice limbajului. Atunci când elaborăm algoritmul care stă la baza programului este greu să avem în vedere și regulile specifice limbajului. Trebuie să ne concentrăm asupra problemei, nu asupra unor detalii. Atunci va trebui să folosim un limbaj de tip pseudocod. *Adică un limbaj care nu are multe reguli, dar care seamănă cu orice limbaj de programare.* Un algoritm redactat în pseudocod nu poate fi rulat pe calculator, este necesară conversia sa în limbajul de programare dorit. Însă conversia este aproape mecanică, pentru că atunci nu suntem concentrați asupra algoritmului.

Revenind la exemplul nostru, putem scrie în pseudocod:

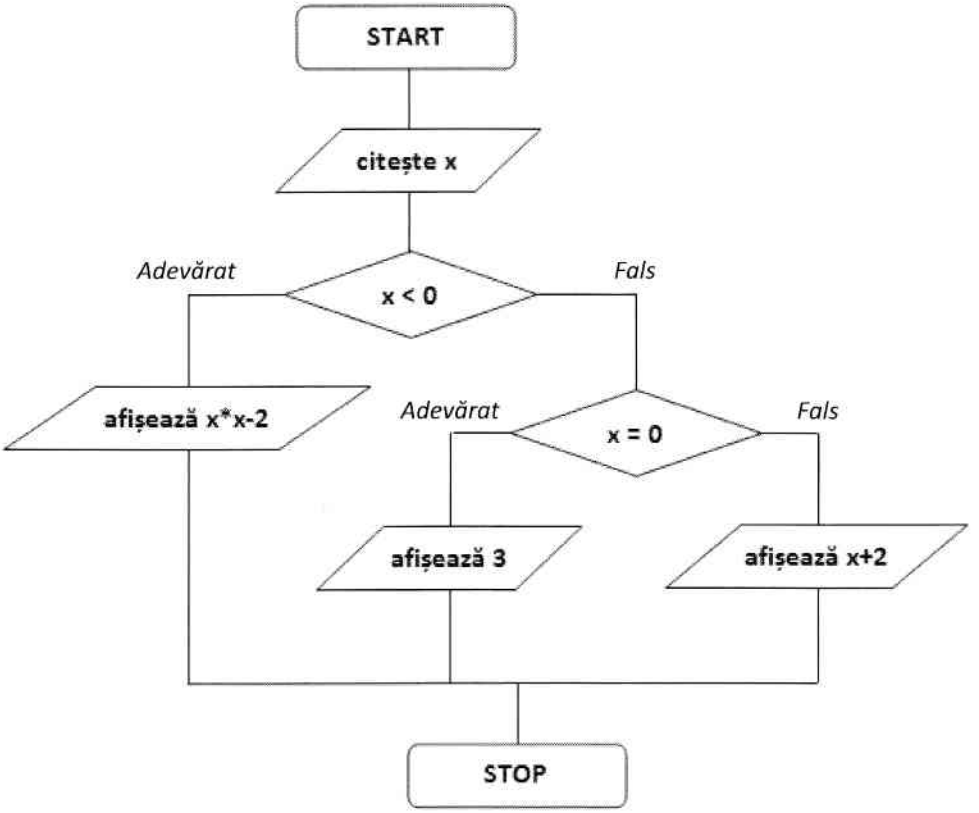
```
citește x
┌ dacă x<0 atunci
|   afișează x*x-2
| altfel
|    ┌ dacă x=0 atunci
|    |   afișează 3
|    | altfel
|    |   afișează x+2
|    └■
└■
```

Aproape de limbajul nostru natural, secvența de mai sus arată de acum precum un cod scris într-un oarecare limbaj de programare, nu? Și asta doar pentru că am folosit unele convenții și folosind câteva reguli.

Atrag atenția asupra faptului că un adevărat limbaj de tip pseudocod permite *conversia cu ușurință a algoritmului în **orice** limbaj de programare*. Există multe limbaje de tip pseudocod și aproape orice persoană poate să creeze unul personalizat.

Scheme logice

Ca și limbajele de tip pseudocod, schemele logice permit redactarea algoritmilor. Ele au fost mult folosite în trecut, dar astăzi se utilizează din ce în ce mai rar. Motivul? Ocupă mult spațiu pe hârtie și creează neplăceri privind reprezentarea grafică.

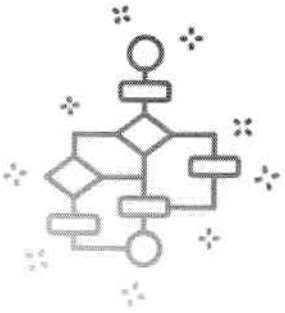


Cu toate acestea, sunt considerate *intuitive* și unii profesori le utilizează.

3. Transpunerea algoritmului într-un limbaj de programare

În această etapă se operează aproape mecanic. Odată cunoscut un limbaj de programare, transpunerea algoritmului nu este o problemă dificilă.

Esențial



Nu este greu să înveți un anumit limbaj la nivel de bază - este precum spaniola/turca reținută vizionând telenovelele de la televizor - însă este greu să știi să elaborezi algoritmi corecți și eficienți.

Un real avantaj pe piața muncii în IT.

Mulți învață doar limbajul de pe siteuri ori Yt... și hop, țop, sunt programatori în Python! Se trezesc apoi la birou că lucrurile nu stau chiar așa...

Algoritmul poate fi și codificat direct în limbajul de programare dorit. Cu timpul, când veți căpăta suficientă **experiență** și dacă problema nu este dificilă chiar așa veți proceda.

4. Testarea programului și corectarea sa până când "funcționează" corect

Atunci când programul este complex, este aproape imposibil să fie scris corect de la început – pot apărea tot felul de conflicte. Din acest motiv este necesară faza de mai sus. Nu uitați, *calculatorul este cel mai bun corector, hârtia suportă orice greșeală*.

Precum observați în piață, există deja o meserie – cea de ***tester***.

1.3. Noțiunea de algoritm, caracteristici

Noțiunea de algoritm a mai fost prezentată în primul paragraf al acestui capitol. Aici o reamintim, o particularizăm pentru programarea calculatoarelor și evidențiem caracteristicile sale. *Prin algoritm înțelegem o succesiune de etape care se pot aplica mecanic pentru ca, pornind de la datele de intrare, să se obțină datele de ieșire.* Rețineți: **pentru orice algoritm trebuie precizat în mod clar care sunt datele de intrare și care sunt cele de ieșire.** Dacă această precizare nu a fost făcută, nu se mai poate vorbi de algoritm. Iată câteva caracteristici ale algoritmilor.

1. Finititudine - *este proprietatea algoritmilor de a furniza rezultatele într-un timp finit.* Nu trebuie înțeles de aici că dacă un algoritm furnizează rezultatele în timp finit este neapărat bun. El trebuie să fie și eficient, adică să alegem calea cea mai simplă de rezolvare, înțelegând prin aceasta, cea care presupune un efort de calcul cât mai mic.

Exemplu: *Se cere să se elaboreze algoritmul prin care se listează primele 100 pătrate perfecte.* O primă idee ar fi să testăm care număr (începând cu **1**, continuând cu **2**, **3**, etc.) este pătrat perfect. Dacă acesta îndeplinește condiția este tipărit, altfel se trece mai departe. Algoritmul se termină

atunci când am listat **100** de pătrate perfecte. Altfel: se tipăresc valorile $\mathbf{1^2}$, $\mathbf{2^2}$, ..., $\mathbf{100^2}$. E mai simplu, nu? Să vedem de ce. În primul caz se analizează primele **10000** numere naturale (pentru că $\mathbf{100^2=10000}$) și se tipăresc cele care sunt pătrate perfecte. În al doilea caz se efectuează doar **100** de înmulțiri. Veți spune că nu contează, calculatorul face calculele extrem de rapid. Nu-i așa! Pentru probleme complexe, un algoritm performant rulează cu mult mai repede decât unul care nu îndeplinește această condiție. *Aceasta înseamnă că, atunci când avem de elaborat un algoritm, nu ne vom opri la prima soluție găsită.* Referitor la această proprietate, se impun anumite precizări.

Există probleme pentru care nu se cunosc algoritmi de rezolvare rapizi. *Pentru anumite seturi de date de intrare, este necesar un timp de calcul de ordinul sutelor sau chiar miilor de ani și aceasta pe calculatoarele ultramoderne*. Din acest motiv, în practică se consideră că pentru aceste probleme nu se cunosc algoritmi de rezolvare, chiar dacă timpul de lucru este finit.

2. Claritatea - *este proprietatea algoritmilor prin care procesul de calcul este descris precis, fără ambiguități.*

Dacă vrem să îi spunem robotului să ne cumpere *un tricou colorat, pe gustul nostru*, ... habar nu are! Discutăm despre **Inteligență Artificială** și **Machine Learning** deja. El trebuie să știe ce modele ne plac, ce culori ori magazine preferăm, ș.a.m.d., din experiența dobândită anterior, aplicând algoritmi complecși de calcul și analiză.

Bineînțeles, algoritmii trebuie să fie **EFICIENȚI**. Dacă suntem în Sibiu și ne hotărâm să plecăm la Constanța, cu siguranță ruta noastră nu va conține orașul Oradea, decât dacă luăm și vreo rudă de acolo cu noi. Vă imaginați ce algoritmi fantastici sunt dezvoltați pentru *Google Maps* ori *Waze* a.î. ruta optimă să fie afișată în timp real? *Adaptiv*, în funcție de trafic?