MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
ROMÂNIA
UNIVERSITATEA DE MEDICINĂ,
FARMACIE, ȘTIINȚE ȘI TEHNOLOGIE
„GEORGE EMIL PALADE"
DIN TÂRGU MUREȘ

FACULTATEA DE INGINERIE ȘI TEHNOLOGIA INFORMAȚIEI

Algoritmi fundamentali

Curs 5

Analiza algoritmilor

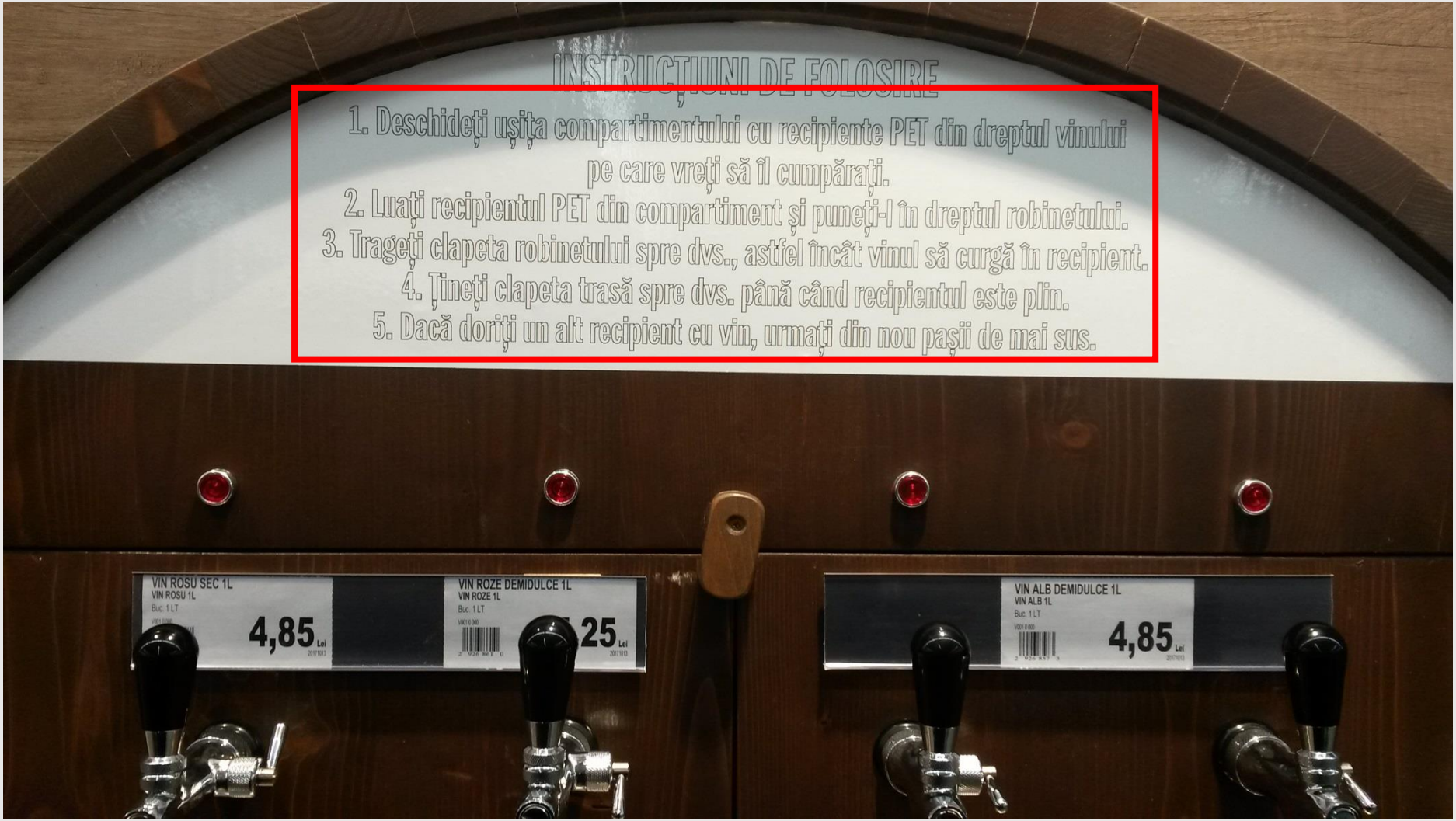
Dr. ing. Kiss Istvan

istvan.kiss@umfst.ro

Cuprins

1. Etapele rezolvării problemelor
2. Testarea solutiei
3. Complexitatea algoritmilor
    1. Operatia de baza
    2. Funcții de complexitate
    3. Clase de complexitate
    4. Determinarea complexității unui algoritm dat
4. Exemple…

Algoritm…


INSTRUCȚIUNI DE FOLOSIRE
1. Deschideți ușița compartimentului cu recipiente PET din dreptul vinului pe care vreți să îl cumpărați.
2. Luați recipientul PET din compartiment și puneți-l în dreptul robinetului.
3. Trageți clapeta robinetului spre dvs., astfel încât vinul să curgă în recipient.
4. Țineți clapeta trasă spre dvs. până când recipientul este plin.
5. Dacă doriți un alt recipient cu vin, urmați din nou pașii de mai sus.
VIN ROSU SEC 1L
VIN ROSU 1L
Buc. 1 LT
4,85 Lei
VIN ROZE DEMIDULCE 1L
VIN ROZE 1L
Buc. 1 LT
VIN ALB DEMIDULCE 1L
VIN ALB 1L
Buc. 1 LT
4,85 Lei

1. Etapele rezolvării problemelor

1. **Analiza teoretica a problemei**
   - *Pune accent pe corectitudinea solutiei*
2. **Proiectarea algoritmului de solutionare**
   - *Pune accent pe eficienta de rezolvare (timp,spatiu)*
3. **Implementarea si testarea solutiei in practica**
   - *Utilitate, siguranta in functionare, robustete, performanta*

- ***Complexitatea algoritmilor**: disciplina care se ocupa cu studiul teoretic al eficientei algoritmilor.*

2. Testarea solutiei

- Un program este **corect** dacă el satisface specificaţiile problemei.
- **Testarea programelor** este activitatea prin care programatorul observă comportarea programului în urma execuţiei lui cu date de test.
  - La programe complexe se practica testarea modulelelor
- **Robustetea** programului inseamna comportare adecvata la date de intrare intenţionat greşite, pentru care problema nu are sens.

2. Testarea solutiei

- **Testarea dupa specificatia problemei**: stabilirea datelor de test se face analizând specificaţia problemei.
- **Testarea după textul programului**: stabilirea datelor de test pe baza instrucţiunilor din program.

DACĂ *n<2* ***ATUNCI***

. . .

DACĂ *n>3* ***ATUNCI***

A

…

SFDACĂ

. . .

SFDACĂ

2. Testarea solutiei

- **Claritatea algoritmului/programului**
- Gries sugerează următoarele reguli:
  - instrucţiunile unei secvenţe se vor scrie aliniate, începând toate în aceeaşi coloană;
  - instrucţiunile unei structuri de calcul (instrucţiuni compuse) se vor scrie începând toate din aceeaşi coloană, aflată cu 2-4 caractere la dreapta faţă de începutul instrucţiunii compuse;
  - pe o linie pot fi scrise mai multe instrucţiuni, cu condiţia ca ele să aibă ceva comun. Astfel, 2-4 instrucţiuni scurte de atribuire pot fi scrise pe acelaşi rând. Acest lucru se recomandă în vederea unei scrieri compacte a programului. E bine ca un program ce se poate scrie pe o pagină, cu respectarea structurii lui, să nu fie întins pe două pagini !
- Se recomandă ca denumirile variabilelor să fie astfel alese încât să reflecte **semnificaţia acestor variabile**.

2. Testarea solutiei

- **Claritatea algoritmului/programului**

PROGRAMUL *CLASAMENT* ***ESTE:***

DATE *m,n,NUME$_i$, i=1,n, NOTE$_{i,j}$, j=1,m, i=1,n;*

PENTRU *i:=1,n* ***EXECUTĂ*** *{calculează media generală a elevului i}*

FIE *S:=0;*

PENTRU *j:=1,m* ***EXECUTĂ*** *S:=S+NOTE$_{i,j}$* ***SFPENTRU***

FIE *MEDII$_i$:=S/M*

SFPENTRU

PENTRU *j:=1,m* ***EXECUTĂ***

CHEAMĂ *ORDINE(n,NOTE$_j$,O);*

CHEAMĂ *TIPAR(n, NUME, O)*

SFPENTRU

CHEAMĂ *ORDINE(n,MEDII,O);*

CHEAMĂ *TIPAR(n,NUME,O)*

SFALGORITM

3. Complexitatea algoritmilor

- Se refera la eficienta unui algoritm
- Putem compara doi algoritmi în raport cu:
  - cantitatea de memorie necesară;
  - viteza de lucru, deci timpul necesar rezolvării problemei.
- Timpul necesar execuţiei unui program depinde de numărul operaţiilor executate.
- Numărul operaţiilor executate depinde de datele de intrare, deci se schimbă de la o execuţie la alta.
- *Complexitate de spatiu: reprezinta spatiul de memorie de lucru necesar in functie de date de intrare...*

3. Complexitatea algoritmilor

- Din punctual de vedere a timpului de executie distingem trei cazuri:
  - Cel mai rau caz: numarul operatiilor de baza efectuate este maxim. W – worst case
  - Cel mai favorabil caz. B – best case
  - Complexitatea medie: un numar mediu de operatii de baza. A – average case
- Pentru compararea algoritmilor este suficient daca se determina ordinul de marime al complexitatii.

3.1. Operatia de baza

- Operatia sau operatiile repetitive, identificate in corpul descrierii algoritmului.
- Contorizarea operatiilor de baza permite estimarea timpului de executie.
- Operatia de baza este operatia *inevitabila* in rezolvarea problemei.
- **Ex**.: in cazul sortarii unui sir de *n* numere prin comparatii este clar ca operatia de baza este operatia de comparare a numerelor.

3.2. Functii de complexitate

- Numarul de repetitii al operatiei de baza se exprima matematic printr-o *functie de complexitate* individuala asociata algoritmului, functie ce depinde in mod inevitabil de marimea vectorului datelor de intrare.
- **Ex**.: Algoritmul de determinare a maximului dintr-un sir de *n* elemente prin comparatie.
  - Numarul de comparatii effectuate va fi o functie de *n*.

3.2. Functii de complexitate

Citeste n, a_1, a_2,…,a_n
Max:=a_1;
Pentru i:=2;n executa
    daca max < a_i atunci
        max:=a_i;
    sf.daca
Sf.pentru

Deci, functia care descrie numarul operatiilor de baza este f(n)=n-1

3.3. Clase de complexitate

$f \in O(g(n))$

- O(1) clasa algoritmilor constanti
- O(log n) algoritmi logaritmici
- O(n) liniari
- O(nlog n) liniarlogaritmici
- $O(n^2)$ patratici
- $O(n^k \log n)$ polilogaritmici
- $O(n^k)$ polinomiali
- $O(a^n)$ clasa algoritmilor exponentiali

3.3. Clase de complexitate

$f \in O(g(n))$

- Exemple de algoritmi:
  - O(n): afisarea vectorului
  - O(n^2): doua cicluri imbracate: afisarea matricei, sortare prin selectie directa
  - O(n^3): trei cicluri imbracate: produsul matricelor
  - O(log n): cautare binara
  - O(nlog n): quick sort
  - O(a^n): backtracking

3.3. Clase de complexitate

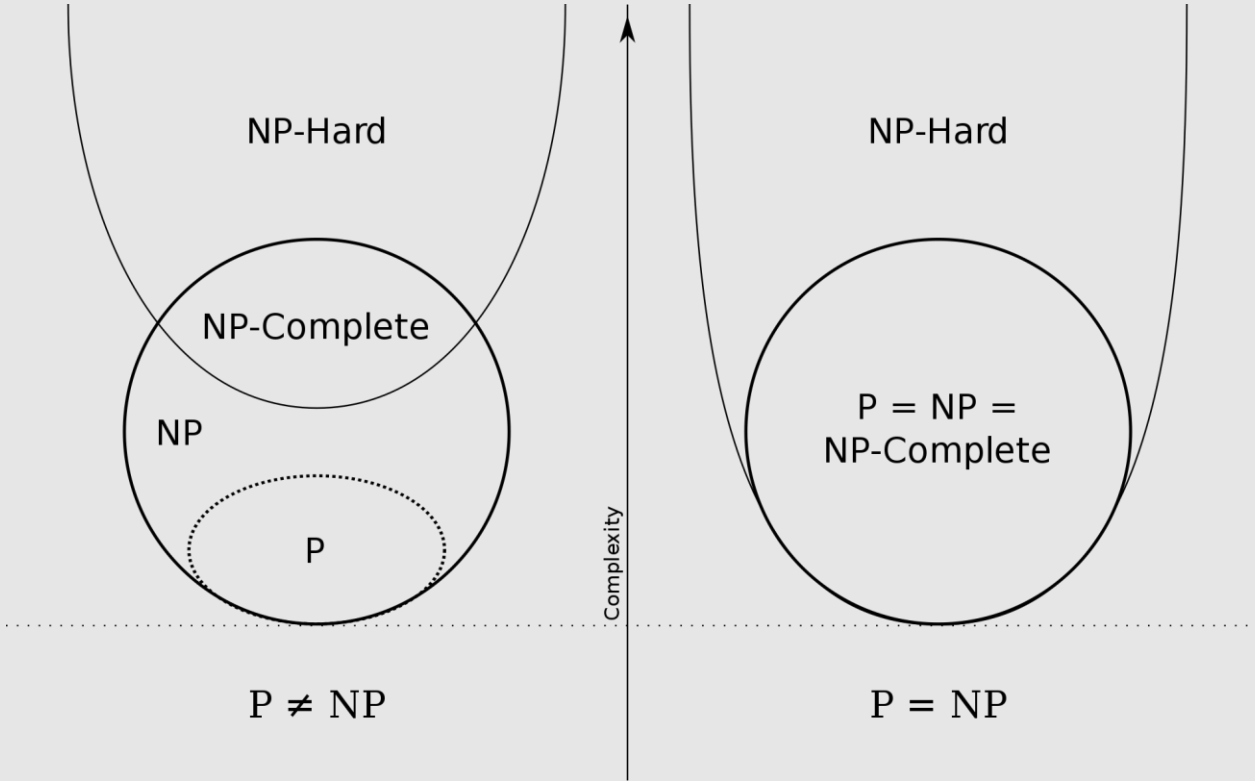
Asymptotic comparison operator	Numeric comparison operator
Our algorithm is **o**(something)	A number is **<** something
Our algorithm is **O**(something)	A number is **≤** something
Our algorithm is **Θ**(something)	A number is **=** something
Our algorithm is **Ω**(something)	A number is **≥** something
Our algorithm is **ω**(something)	A number is **>** something

3.3. Clase de complexitate

- **Clasa problemelor NP-complete** (Non-deterministic polinomial):
  - Nu pot fi solutionate prin algoritmi eficienti (de complexitate polinomiala)
- **Ex**.: Se da o expresie logica in forma normala conjunctiva cu *n* variabile, sa se determine daca pot fi atribuite valori logice variabilelor astfel incat expresia sa fie adevarata [Coo,1970].

3.3. Clase de complexitate

- **Clasa problemelor NP-complete** (Non-deterministic polinomial):
  - Nu pot fi solutionate prin algoritmi eficienti (de complexitate polinomiala)



3.3. Clasificare algoritmi

- Clasificare funcţie de implementare:
  - recursivi (se invocă pe ei înşişi) / iterativi (au construcţii repetitive);
  - logici (controlează deducţii logice);
  - seriali (un singur procesor şi o singură instrucţiune executată la un moment dat) / paraleli (mai multe procesoare care executa instrucţiuni în acelaşi timp) ;
  - deterministici (o decizie exactă la fiecare pas)/ nedeterministici (rezolvă problema plecând de la presupuneri);
- Clasificare funcţie de "design" (metodă), ex.:
  - împarte şi stăpâneşte (împarte problema în una sau mai multe instanţe mai mici);
  - programare dinamică (caută structuri optimale - o soluţie optimală a problemei poate fi construită plecând de la soluţiile optimale ale subproblemelor);
  - metoda ”Greedy” (caută tot structuri optimale cu deosebirea că soluţiile la subprobleme nu trebuie să fie cunoscute la fiecare pas; alegerea ”Greedy” se referă la ”ce arată” mai bine la un moment dat).
- Clasificare funcţie de domeniul de studiu – algoritmii sunt:
  - de căutare, de sortare, numerici, algoritmi de grafuri, de geometrie computaţională, de învăţare automată, de criptografie, etc.

3.4. Determinarea complexitatii algoritmilor

- 1.

```
CITEŞTE a, b, c
DACĂ a>b ATUNCI
        max=a ALTFEL
        max=b
Sf.daca
DACĂ c>max ATUNCI
         max=c
Sf.daca
AFIŞEAZĂ max
```

3.4. Determinarea complexitatii algoritmilor

- 2.

```
CITEŞTE n
suma=0; i=0
CÂT TIMP i<n
    i++
    suma=suma+i
Sf.cattimp
AFIŞEAZĂ suma
```

3.4. Determinarea complexitatii algoritmilor

- 3. Descompunerea unui număr în factori primi
  1. CITEŞTE n
  2. n=abs(n)
  3. f=2 //factorul prim de testat
  4. CÂT TIMP (f<=n)
     4.1.e=0 //exponentul la care apare factorul în descompunere
     4.2.CÂT TIMP (n%f==0) //n se divide la f
        4.2.1.n=n/f
        4.2.2.e=e+1
     4.3.DACĂ (e≠0) SCRIE f," la ",e
     4.4.DACĂ (f=2) f=f+1 ALTFEL f=f+2

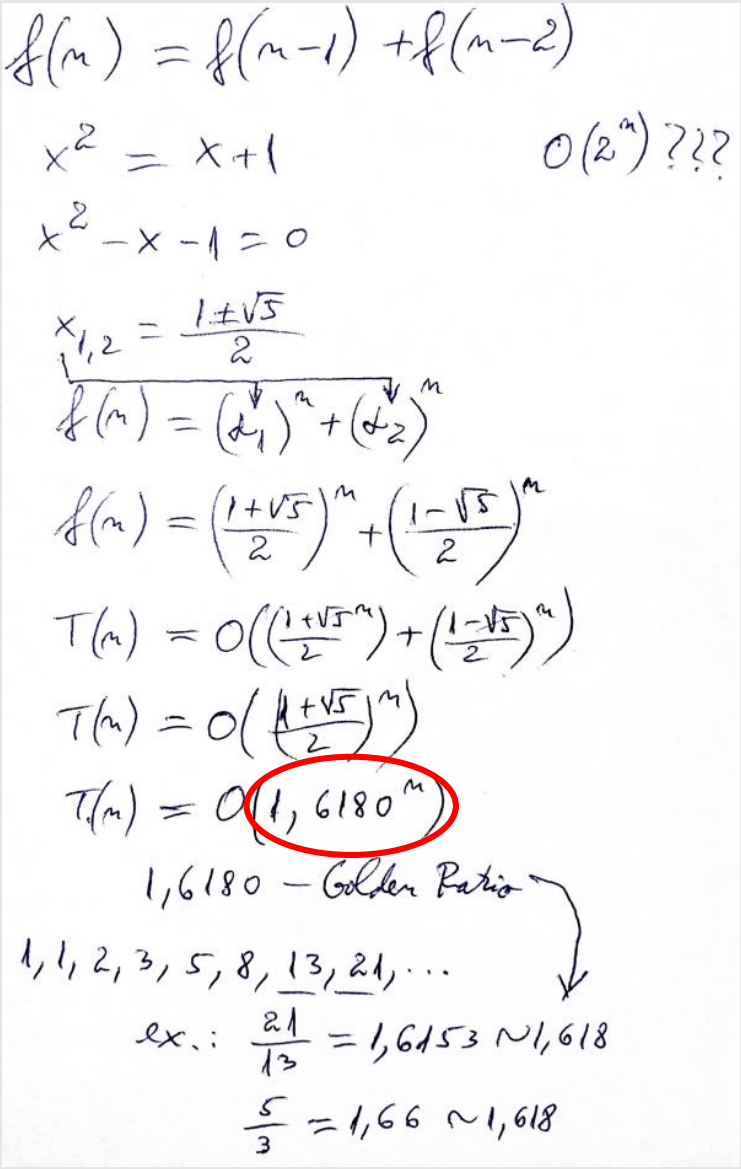
3.4. Determinarea complexitatii algoritmilor

- 4. Calculul unei sume duble

1.CITEŞTE a,b,m,n
2.i=1 3.j=1 4.s=0
5.CÂT TIMP (i<=m)
    5.1.CÂT TIMP (j<=n)
        5.1.1.s=s+ sin(a+3*i)*cos(b+5*j)
        5.1.2.j=j+1
    5.2.i=i+1
6.SCRIE s

3.4. Determinarea complexitatii algoritmilor

- 5. Genereaza numerele Fibonacci
  1. CITEŞTE n
  2. DACĂ n>=2
     - 2.1. PENTRU f0=0,f1=1, f1<n, f0=f1, f1=fi, execut
       - 2.1.1. fi=f0 + f1
       - 2.1.2. SCRIE fi



3.4. Determinarea complexitatii algoritmilor

- 6. Valoarea unui polinom intr-un punct

```
1 CITEŞTE g,x
2 i=0
3 CÂT TIMP i<n
      3.1 CITEŞTE p[i]
      3.2 i=i+1
4 v=p[0], xi=x, i=1 // v reprezinta valoarea, iar xi reprezinta x^i
5 CÂT TIMP i<n
      5.1 xi=xi*x
      5.2 v=v+xi*p[i]
      5.3 i=i+1
6 SCRIE v
```