Despre algoritmi

De ce despre algoritmi?

- numeroase aplicații
- în practică este importantă eficiența algoritmilor
- ar fi util să știm dacă algoritmii pe care îi propunem sunt corecți
  - ☺ corectitudine ≠ nu a găsit cineva încă un contraexemplu

Aspecte generale care apar la rezolvarea unei probleme

- *Teoretic*, **paşii elaborării un algoritm sunt următorii:**

1. demonstrarea faptului că este **posibilă** elaborarea unui algoritm pentru determinarea unei soluţii
2.
3.
4.
5.

Aspecte generale care apar la rezolvarea unei probleme

- *Teoretic*, **paşii elaborării un algoritm sunt următorii:**

1. demonstrarea faptului că este **posibilă** elaborarea unui algoritm pentru determinarea unei soluţii
2. **elaborarea** algoritmului
3. demonstrarea **corectitudinii** algoritmului
4.
5.

Aspecte generale care apar la rezolvarea unei probleme

- *Teoretic*, **paşii elaborării un algoritm sunt următorii:**

1. demonstrarea faptului că este **posibilă** elaborarea unui algoritm pentru determinarea unei soluţii
2. **elaborarea** algoritmului
3. demonstrarea **corectitudinii** algoritmului
4. determinarea **timpului de executare** a algoritmului
5. demonstrarea **optimalităţii** algoritmului

Existența algoritmilor

Existența algoritmilor

- **Problemă nedecidabilă** =pentru care nu poate fi elaborat un algoritm.

1. **Problema opririi programelor**: pentru orice program şi orice valori de intrare să se decidă dacă programul se termină.
2. **Problema echivalenţei programelor:** să se decidă pentru orice două programe dacă sunt echivalente (produc aceeași ieșire pentru aceleași date de intrare).

Elaborarea algoritmilor

Elaborarea algoritmilor

- **Cursurile următoare**: metode de elaborare a algoritmilor

Corectitudinea algoritmilor

Corectitudinea algoritmilor

- **Terminarea programului**

- **Corectitudinea parțială:** presupunând că algoritmul se termină, rezultatul este corect;
  - **Invarianţi** =relaţii ce trebuie îndeplinite la orice trecere a programului prin acel loc

Corectitudinea algoritmilor

- Exemplul 1 **Determinarea concomitentă a cmmdc şi cmmmc a două numere naturale** $a,b \in \mathbf{N}^*$.

Corectitudinea algoritmilor

- Exemplul 1 **Determinarea concomitentă a cmmdc şi cmmmc a două numere naturale** $a,b \in \mathbf{N}^*$.

Algoritm:

```
x ← a; y ← b;

while x≠y

    if x>y then x ← x-y;
           else y ← y-x;

write(x,(u+v)/2)
```

Corectitudinea algoritmilor

- Exemplul 1 **Determinarea concomitentă a cmmdc şi cmmmc a două numere naturale** $a,b \in \mathbf{N}^*$.

Algoritm:

```
x ← a; y ← b; u ← a; v ← b;

while x≠y

   if x>y then x ← x-y; u ← u+v
          else y ← y-x; v ← u+v

write(x,(u+v)/2)
```

Corectitudinea algoritmilor

- <u>Exemplul 1</u> **Determinarea concomitentă a cmmdc şi cmmmc a două numere naturale** $a, b \in \mathbf{N}^*$.

Algoritm:

```
x ← a; y ← b; u ← a; v ← b;

while x≠y
 {(x,y)=(a,b); xv+yu = 2ab}          (*)
   if x>y then x ← x-y; u ← u+v
          else y ← y-x; v ← u+v

write(x,(u+v)/2)
```

Corectitudinea algoritmilor

- **`{(x,y)=(a,b); xv+yu = 2ab} (*) - este invariant`**
- **Demonstrație** – **Inducție** după numărul de pași (exercițiu)

Corectitudinea algoritmilor

- **{(x,y)=(a,b); xv+yu = 2ab} (*) – este invariant**

  ⇒ **Dacă se termină avem**

  `(a,b) = (x,x) =x`

  `(u+v)/2 = ab/x = ab / (a,b) = [a,b]`

⇒ **corectitudine parțială**

- **Algoritmul se termină deoarece**

- şirul **$\{x_n+y_n\}$** este un şir ***strict* descrescător** de **numere naturale pozitive**, unde $\{x_n\}$, $\{y_n\}$ este şirul de valori succesive ale variabilelor x şi y.

Timpul de executare

Timpul de executare a algoritmilor

- se măsoară în funcţie de lungimea n a datelor de intrare

- **T(n)** = timpul de executare pentru orice set de date de intrare de lungime **n**

  – dat de numărul de operații elementare în funcție de n

Timpul de executare a algoritmilor

- Se numără operații elementare (de atribuire, aritmetice, de decizie, de citire/scriere)

- Numărare aproximativă => ordinul de mărime al numărului de operații elementare

- Pentru simplitate – se fixează operație de bază

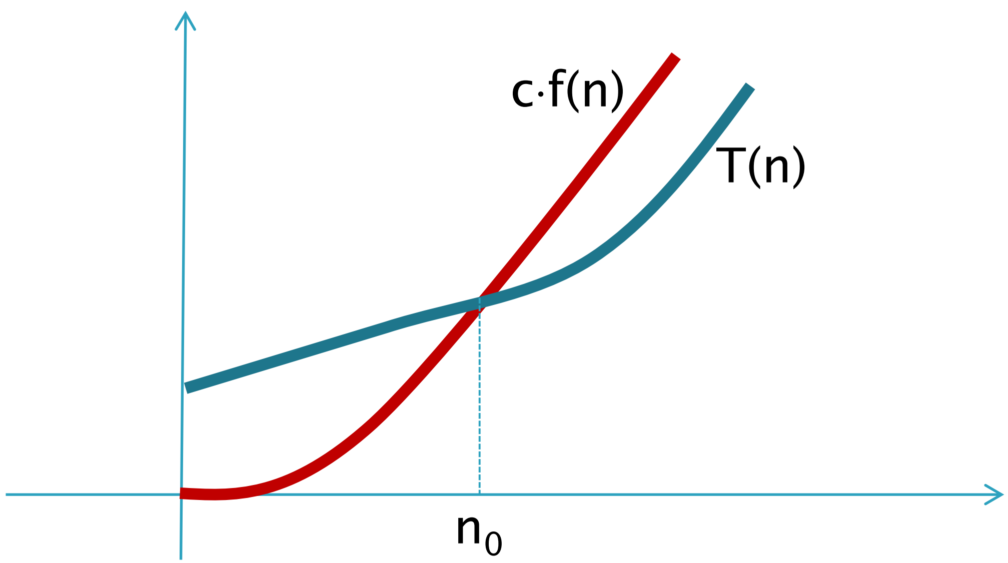
Timpul de executare a algoritmilor

- În majoritatea cazurilor ne mărginim la a evalua ordinul de mărime al timpului de executare = **ordin de complexitate al algoritmului**

$$T(n) = O(f(n))$$

$$\exists c,\ n_0 \text{ – constante a.î } \forall n \geq n_0$$

$$T(n) \leq c \cdot f(n)$$



Timpul de executare a algoritmilor

- Notație: $T(n) = O(f(n))$
- comportare **asimptotică**
- caz defavorabil
- O(expresie)= O(termen dominant)

  $O(2n) \Rightarrow O(n)$

  $O(2n^2+4n+1) \Rightarrow O(n^2)$

Timpul de executare a algoritmilor

Notație: $T(n) = O(f(n))$

Clase de complexitate uzuale (în ordine crescătoare):

- **Complexitate logaritmică** $O(\log_2(n))$, $O(\log(n))$

  Exemplu: căutarea binară

-
-

Timpul de executare a algoritmilor

Notație: `T(n) = O(f(n))`

Clase de complexitate uzuale (în ordine crescătoare):

- **Complexitate logaritmică** $O(\log_2(n))$, $O(\log(n))$

  Exemplu: căutarea binară

- **Complexitate liniară** $O(n)$

  Exemplu: minimul dintr-un vector

-

Timpul de executare a algoritmilor

Notație: `T(n) = O(f(n))`

Clase de complexitate uzuale (în ordine crescătoare):

- **Complexitate logaritmică** $O(\log_2(n))$, $O(\log(n))$

  Exemplu: căutarea binară

- **Complexitate liniară** $O(n)$

  Exemplu: minimul dintr-un vector

- **Complexitate** $O(n \log_2(n))$ **liniară logaritmică**

  Exemplu: sortarea prin interclasare MergeSort

Timpul de executare a algoritmilor

- **Complexitate pătratică** $O(n^2)$
  Exemplu: suma elementelor unei matrice, sortarea prin metoda bulelor, prin selecție

- **Complexitate polinomială** $O(n^k)$, $k \geq 3$
  Exemplu: înmulțirea a două matrice pătratice de dimensiune n

Timpul de executare a algoritmilor

- **Complexitate pătratică** $O(n^2)$
  Exemplu: suma elementelor unei matrice, sortarea prin metoda bulelor, prin selecție

- **Complexitate polinomială** $O(n^k)$, $k \geq 3$
  Exemplu: înmulțirea a două matrice pătratice de dimensiune n

- **Complexitate exponențială** $O(k^n)$, $k \geq 2$
  Exemplu: generarea submulțimilor unei mulțimi cu n elemente

- **Complexitate factorială** $O(n!)$
  Exemplu: generarea permutărilor unui vector cu n elemente

Timpul de executare a algoritmilor

- **Exemplul 1**

```
i = 0
p = 1
pentru i = 1, n executa
     pentru j = 1,p executa
          scrie j
     scrie linie noua
     p = p * 2
```

Afișare
1
1 2
1 2 3 4
1 2 3 4 5 6 7 8
1 2 .... 2^n

Timpul de executare a algoritmilor

- **Exemplul 1**

```
i = 0
p = 1
pentru i = 1, n executa
     pentru j = 1,p executa
          scrie j
     scrie linie noua
     p = p * 2
```

Afișare

1

1 2

1 2 3 4

1 2 3 4 5 6 7 8

1 2 .... 2^n

$1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$ afișări ale lui j

$O(2^n)$

Timpul de executare a algoritmilor

- Exemplul 2 – Cea mai mică putere a lui 2 mai mare ca n

```
p = 1
cat timp p<=n executa:
  p = p * 2
scrie p
```

Timpul de executare a algoritmilor

- Exemplul 2 – Cea mai mică putere a lui 2 mai mare ca n

```
p = 1
cat timp p<=n executa:
  p = p * 2
scrie p
```

$O(\log_2(n))$

Timpul de executare a algoritmilor

- **Exemplul 3 (seminar/laborator)** – 2-SUM pentru șir crescător: Se dă un vector ordonat cu n elemente întregi distincte. Să se afișeze toate perechile de elemente din vector cu suma 0

```
i = 0
j = n-1
while i<j:
    if v[i]+v[j]==0:
        print(v[i], v[j])
        i += 1
        j -= 1
    elif v[i] + v[j] < 0:
            i += 1
    else:
        j -= 1
```

Timpul de executare a algoritmilor

- **Exemplul 3 (seminar/laborator)** – 2-SUM pentru șir crescător: Se dă un vector ordonat cu n elemente întregi distincte. Să se afișeze toate perechile de elemente din vector cu suma 0

```
i = 0
j = n-1
while i<j:
    if v[i]+v[j]==0:
        print(v[i], v[j])
        i += 1
        j -= 1
    elif v[i] + v[j] < 0:
            i += 1
    else:
        j -= 1
```

O(n)

Timpul de executare a algoritmilor

Alte exemple

- Înmulțirea a două matrice A(n,m) B(m,p)
- Intersecția a două mulțimi cu n respectiv m elemente
- Reuniunea a două mulțimi ordonate cu n respectiv m elemente

Timpul de executare a algoritmilor

Alte exemple

- Înmulțirea a două matrice A(n,m) B(m,p) **O(nmp)**
- Intersecția a două mulțimi cu n respectiv m elemente **O(nm)**
- Reuniunea a două mulțimi ordonate cu n respectiv m elemente- **Interclasare O(n+m)**

Timpul de executare a algoritmilor

- Interclasare

```
i = 1; j = 1
while (i<=m) and (j<=n):
     if a[i]<=b[j]:
          scrie a[i]; i=i+1
     else:
          scrie b[j]; j=j+1
while i<=m:
      scrie a[i]; i=i+1
while j<=n:
       scrie b[j]; j=j+1
```

Timpul de executare a algoritmilor

Alte exemple

```
pentru i = 1, n executa
    pentru j= 1, m executa
          operatii O(1)
      pentru k= 1, p executa
          operatii O(1)
```

Timpul de executare a algoritmilor

Alte exemple

```
pentru i = 1, n executa
    pentru j = 1, m executa
          operatii O(1)
      pentru k = 1, p executa
          operatii O(1)
```

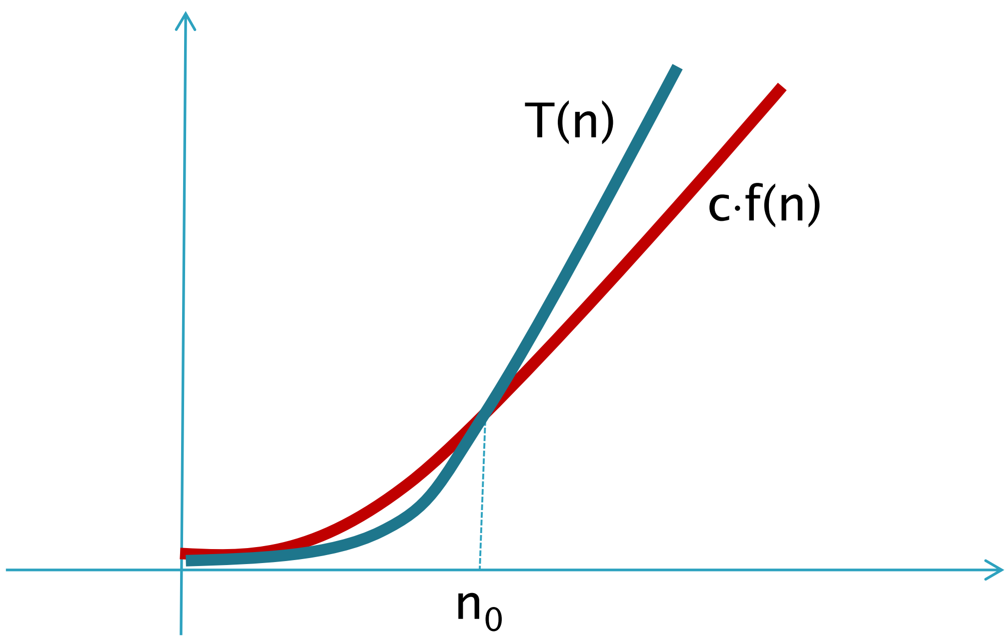
$O(n(m+p))$

Timpul de executare a algoritmilor

- **$T(n) = \Omega(f(n))$**

  $\exists c,\ n_0$ – constante a.î $\forall n \geq n_0$

  $T(n) \geq c \cdot f(n)$

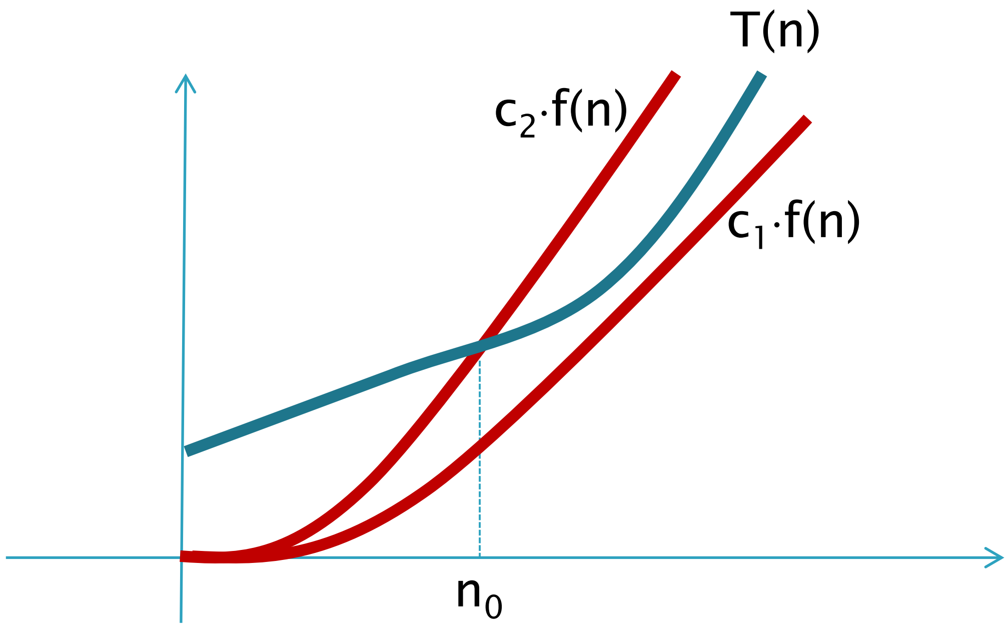


Timpul de executare a algoritmilor

- $T(n) = \Theta(f(n))$

$$\exists c_1,\ c_2,\ n_0 \text{ constante a.î } \forall n \geq n_0$$

$$c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$$



Optimalitatea algoritmilor

Optimalitatea algoritmilor

- Să presupunem că pentru o anumită problemă am elaborat un algoritm şi am putut calcula şi timpul său de executare T(n) (raportat la o operație de bază).

Algoritmul nostru este "cel mai bun" sau există un alt algoritm cu timp de executare mai mic.

Optimalitatea algoritmilor

- Exemplul 1. Să se determine minimul elementelor unui vector.

- Exemplul 2. Să se determine minimul şi maximul elementelor unui vector.

- Arătați că algoritmii propuși sunt optimali

Optimalitatea algoritmilor

- Exemplul 1. **Să se determine minimul elementelor unui vector.**

Optimalitatea algoritmilor

- Exemplul 1. **Să se determine minimul elementelor unui vector.**

```
m ← a₁
for i=2,n
    if aᵢ<m then m ← aᵢ
```

Optimalitatea algoritmilor

- Exemplul 1. **Să se determine minimul elementelor unui vector.**

$m \leftarrow a_1$

for $i=2,n$

    if $a_i<m$ then $m \leftarrow a_i$

- **n – 1 comparări între elemente ale vectorului**
- **Algoritmul este optimal?**

Optimalitatea algoritmilor

- Exemplul 1. **Să se determine minimul elementelor unui vector.**
- Pentru a demonstra că algoritmul este optimal – trebuie demonstrat că

  orice algoritm de determinare a minimului unui vector cu n elemente bazat pe comparări necesită cel puțin n-1 comparări

Optimalitatea algoritmilor

- **<u>Proprietate</u> Orice algoritm de determinare a minimului unui vector cu n elemente bazat pe comparări necesită cel puțin n-1 comparări**
- **Demonstrație – Inducție după n**
- $n=1$ – evident
- $n \Rightarrow n+1$

Optimalitatea algoritmilor

- <u>**Proprietate**</u> **Orice algoritm de determinare a minimului unui vector cu n elemente bazat pe comparări necesită cel puțin n-1 comparări**
- **Demonstrație – Inducție după n**
- $n=1$ – evident
- $n \Rightarrow n+1$ Pp. că orice algoritm care rezolvă problema pentru n numere efectuează cel puțin n-1 comparări.
  - Considerăm un algoritm oarecare care determină cel mai mic dintre n+1 numere: $m=\min\{a_1,\ldots, a_{n+1}\}$.

Optimalitatea algoritmilor

Considerăm un algoritm oarecare care determină cel mai mic dintre n+1 numere: $m=\min\{a_1,\ldots, a_{n+1}\}$.

- Considerăm **prima comparare efectuată de acest algoritm** – putem presupune că **s-au comparat a_1 cu a_2 şi că $a_1<a_2$.**

Optimalitatea algoritmilor

Considerăm un algoritm oarecare care determină cel mai mic dintre n+1 numere: $m=\min\{a_1,\dots, a_{n+1}\}$.

- Considerăm **prima comparare efectuată de acest algoritm** – putem presupune că **s-au comparat a_1 cu a_2 şi că $a_1<a_2$.**
- Atunci $m=\min(\mathbf{a_1},\mathbf{a_3},\dots,a_{n-1})$.
-

Optimalitatea algoritmilor

Considerăm un algoritm oarecare care determină cel mai mic dintre n+1 numere: $m=\min\{a_1,\dots, a_{n+1}\}$.

- Considerăm **prima comparare efectuată de acest algoritm** – putem presupune că **s-au comparat a_1 cu a_2 şi că $a_1<a_2$.**
- Atunci $m=\min(\mathbf{a_1},\mathbf{a_3},\dots,a_{n-1})$.
- **Ipoteza de inducție** $\Rightarrow$

  minim n-1 comparări pentru a determina m $\Rightarrow$

  numărul total de comparări pentru a calcula m este cel puţin egal cu $n - 1 + 1 = n$.

Optimalitatea algoritmilor

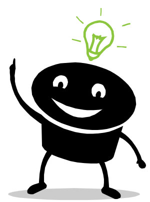
- Exemplul 2. **Să se determine minimul şi maximul elementelor unui vector.**

Idee?

Optimalitatea algoritmilor

- Exemplul 2. **Să se determine minimul şi maximul elementelor unui vector.**

Idee



- formăm **perechi** de elemente
- pe cel mai mic din pereche îl comparăm cu minimul curent, iar pe cel mai mare cu maximul

$\Rightarrow$ 3 comparații pentru fiecare pereche

Optimalitatea algoritmilor

- **Exemplul 2. Să se determine minimul şi maximul elementelor unui vector.**

```
if n impar then m ← a1; M ← a1; k ← 1
else if a1<a2 then m ← a1; M ← a2
     else m ← a2; M ← a1;
     k ← 2
```

Optimalitatea algoritmilor

- Exemplul 2. **Să se determine minimul şi maximul elementelor unui vector.**

```
if n impar then m ← a1; M ← a1; k ← 1
else if a1<a2 then m ← a1; M ← a2
     else m ← a2; M ← a1;
     k ← 2
while k ≤ n-2
   if ak+1<ak+2 then if ak+1<m then m ← ak+1
                     if ak+2>M then M ← ak+2
                else if ak+2<m then m ← ak+2
                     if ak+1>M then M ← ak+1
   k ← k+2
```

Optimalitatea algoritmilor

- Exemplul 2. **Să se determine minimul şi maximul elementelor unui vector.**

```
if n impar then m ← a1; M ← a1; k ← 1
else if a1<a2 then m ← a1; M ← a2
     else m ← a2; M ← a1;
     k ← 2
while k ≤ n-2
   if ak+1<ak+2 then if ak+1<m then m ← ak+1
                     if ak+2>M then M ← ak+2
                else if ak+2<m then m ← ak+2
                     if ak+1>M then M ← ak+1
   k ← k+2
```

- $T(n)=?$

Optimalitatea algoritmilor

- $T(n) = \lceil 3n/2 \rceil - 2$:
  - n impar :

    T(n) =

  - n par:

    T(n) =

Optimalitatea algoritmilor

- $T(n) = \lceil 3n/2 \rceil - 2$:
  - n **impar** :

    $T(n) = 3(n-1)/2 = (3n+1)/2-2 = \lceil 3n/2 \rceil-2$

  - n **par**:

    $T(n) = 1+3(n-2)/2 = 3n/2-2 = \lceil 3n/2 \rceil-2$

Optimalitatea algoritmilor

- $T(n) = \lceil 3n/2 \rceil - 2$
  - Optimal?

Optimalitatea algoritmilor

- $\mathtt{T(n)} = \lceil \mathtt{3n/2} \rceil - 2$

- <u>Proprietate</u> Orice algoritm de determinare a minimului și maximului unui vector cu n elemente bazat pe comparări necesită cel puțin $\lceil 3n/2 \rceil - 2$ comparări

- Idee de demonstrație (SUPLIMENTAR):

Optimalitatea algoritmilor

- $\mathtt{T(n)} = \lceil \mathtt{3n/2} \rceil - 2$

- <u>Proprietate</u> Orice algoritm de determinare a minimului și maximului unui vector cu n elemente bazat pe comparări necesită cel puțin $\lceil 3n/2 \rceil - 2$ comparări

- **Idee de demonstrație:**

- La un pas al algoritmului analizăm:
  - ce tipuri de elemente pot apărea în funcție de rezultatele comparărilor deja efectuate
  - între ce tipuri de elemente va face comparații un algoritm eficient

Despre algoritmi

- **Nu interesează în general demonstrarea teoretică a existenţei algoritmilor, ci accentul este pus pe elaborarea algoritmilor**

Elaborarea algoritmilor

Metode de elaborare a algoritmilor

- Metoda Greedy
- Metoda Divide et Impera
- Metoda Programării Dinamice
- Metoda Backtracking

Metoda Greedy

- **Probleme de optim**
- **Cadru posibil**:

  Se dă o mulțime finită A.

  Să se determine o submulțime finită $B \subseteq A$ care satisface anumite condiții (este **soluție posibilă**)

  $+$

  îndeplinește un criteriu de optim (este **soluție optimă**)