23.09.2014

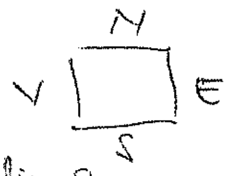
AA

Curs 1

1 Decidabilitate - Pb plăcilor lui Wang
2 Complexitate - Pb subșirurilor de sumă maximă
3 Analiza amortizată - Operații pe stivă
4 Corectitudinea algoritmilor - Pb găsirii a 2 elem de sumă x
5 Determinism. Clase de probleme - Pb acoperirii cu noduri
6 Algoritmi de aproximare - - " —

Problema plăcilor lui Wang

Input C = set finit culori
T = set finit de plăci pătrate
cu fiecare latură cu o culoare din C



Output: Se poate acoperi planul infinit cu plăci din T a.î. fiecare tip de placă se poate folosi de ∞ ori, dar nerotit și plăcile adiacente sunt colorate la fel pe latura comună.

Complexitate

-4 1 9 -4 -1 7 -3 -10 2 3 -1

Input șir de nr
Output suma maximă formată de un subșir al șirului

```
smax = v[0]
forall i in (0,n-1)
    forall j in (i,n-1)
        s = 0
        forall k in (i,j)
            s += v[k]
            smax = max(smax, s)
```

$\simeq n^3$

2 min 30 s

la fel :

```
forall j in (i, n-1)
    S+ = v[j]
    Smax = max(Smax, s)
```

$\boxed{\approx n^2}$ 6 s

SAU

```
alg(l, r)
    if (l == r)
        return v[l]
    m = (l+r)/2
    return max(alg(l,m), alg(m+1, r), central(l,m,r))
```

$\nearrow \sim n$

$T(n) = 2T(\frac{n}{2}) + c$ $\boxed{\approx n\log n}$

0,16 s

SAU

```
Smax = S = v[0]
forall i in (1, n-1)
    S = max(v[i], S+v[i])
    Smax = max(Smax, S)
```

$\boxed{\approx n}$ 0,08 s

$N=100$	t
$\log(n)$	10^{-7} s
n	10^{-6} s
$n\log(n)$	10^{-5} s
n^2	10^{-4} s
n^6	3 min
2^n	10^{14} ani
$n!$	10^{142} ani

Găsirea a 2 elemente de sumă x

input : șir de nr

Output : ∃ 2 elem distincte în șir cu suma x? $x = \underline{1}$

5 -2 3 -5 0 -6 2 8

-6 -5 -2 0 2 3 5 8

```
sort(v)
i = ~~v[0]~~ 0
j = ~~v[n-1]~~ n-1
while (i < j)
    s = v[i] + v[j]
    if s == x return true
    if s < x   i++ else j--
    return false
```

P: Dacă ∃ (i, j) unic în v pt care $v[i] + v[j] = x$ atunci în ∀ iterație k a while-ului $i_k \le i$ $j_k \ge j$

Determinism și clase de probleme

Nedeterminism

P = cls ~~alg~~ prob care se rez în timp polinomial

NP = -"- pt care o soluție se poate verifica în timp polinomial

nedeterminist polinomial

NP - complete

NP - dure

Problema acoperirii cu noduri

input graf $G = (V, E)$

Output k = nr minim a.î. ∃ $U \subseteq V$ cu $card(U) = k$ a.î. ∀ $(u, v) \in E$ are cel puțin un capăt în U ($u \in U$ sau $v \in U$)

```
forall k in (0,n-1)
    forall S ⊆ V de dim k
        if S is vertex
            return true
```

(*) PROBLEMĂ GREA

1) Fac un alg cât mai eficient
2) Caut cazuri particulare care se potrivesc cu problema
   ex: $2^k \cdot n$ pt VertexCover
3) Euristici rapide, care nu garantează o soluție optimă
   ex: alg de aproximare - garantează un factor de aproximare - de câte ori poate fi mai rea soluția găsită decât cea optimă

—

~~AA~~ 6.10.2017

Curs 2

Decidabilitate

Fie o problemă P, putem găsi un algoritm care să rezolve P? Calculabilitate

Fie A o mulțime infinită. Spunem că A este $\boxed{\text{infinit numărabilă}}$ dacă ∃ o bijecție $f: A \to \mathbb{N}$ (etichetare/numerotare a elementelor mulțimii A) $\Rightarrow A = \{a_0, a_1, a_2 \dots\}$

O mulțime pt care nu găsim $f: A \to \mathbb{N}$ bijectivă s.n. $\boxed{\text{infinit nenumărabilă}}$

$\mathbb{N}$ → numărabile

(*) $\boxed{\mathcal{P}(\mathbb{N}) = \text{mulțimea părților lui } \mathbb{N} \text{ (mulțimea tuturor subseturilor lui } \mathbb{N})}$

$\mathbb{Z} = \{0, 1, -1, 2, -2, 3, -3 \dots\}$
$z_0\ z_1\ z_2 \dots$

$\mathbb{N} \times \mathbb{N} = \{(0,0), (0,1), (1,0) \dots$
$(1,0)\ (1,1)$
$(2\ 0)\ \}$

$\mathbb{Q}$

mulțimea programelor

$\mathbb{R}$ → ~~o~~ nenumărabile

$\mathcal{P}(\mathbb{N})$ (Th Cantor)

(A → mulț finită $|A| = n$
$|\mathcal{P}(A)| = 2^n$)

$A = \{1, 2, 3\} \Rightarrow 2^A$ elem

mulțimea problemelor

Simplist, mulțimea elementelor poate fi reprezentată cu mulțimea funcțiilor $f: \mathbb{N} \to \mathbb{N} \Rightarrow \mathcal{P}(\mathbb{N})$

Să considerăm, în schimb mulțimea programelor cu o intrare și o ieșire: $P_{1,1}$ $(P_{i,1})$

Un program este definit folosind un limbaj (...)

Fie $\Sigma \to$ alfabetul de simboluri cu care scriem programele în lb nostru.

$$\Sigma = \{a \ldots z, A \ldots Z, -, |, ;, \ldots\}$$

$\Sigma = \{0, 1\}$ cod binar

$$|\Sigma^*| = \underset{\text{(cuvânt vid)}}{\Sigma_0} \cup \Sigma_2 \cup \ldots \Sigma_k \cup \ldots$$

$$f(P: <b_i \ldots b_0>) = 2^i + B10(<b_i, b_{i-1}, \ldots b_0>)$$

Concluzie: ∃ foarte multe programe pt care nu găsim un program de reprezentare

Problemă: ∃ viața extra-terestră în Univers? Probl de decizie: $P: i = \{0, 1\}$

↓

pseudoalgoritm < DA dacă ∃ extratereștrii
$\perp$ (ciclează la infinit)

(*) O problemă poate fi grea dacă are un input infinit

Teza Church-Turing

Orice funcție f este (efectiv) calculabilă dacă și numai dacă este Turing calculabilă (∃ un program care rulează pe o mașină Turing).

Turing calculabilă = o funcție recursivă

$A \subseteq \mathbb{N}$ $\longrightarrow$ I ① A mulțime recursivă $\Leftrightarrow \exists f$ ~~total~~ recursivă

$A \in \mathcal{P}(\mathbb{N})$ ② care să decidă / calculeze A

$$\exists\ P_{1,1} \text{ a.î. } P(n) = \begin{cases} 1 & n \in A \\ 0 & n \notin A,\ n \in \mathcal{C}(A) \end{cases}$$

(complement A)

ex: nr prime, pare etc.

II ②

RN

A mulțimi recursiv numărabile $\Leftrightarrow \exists f$ parțial recursivă care să genereze elementele lui A

$$\exists\ P_{(1,1)} \text{ a.î. } P(n) = \begin{cases} 1 & n \in A \\ \perp & n \notin A \end{cases}$$

Def Alternativă: Putem să construim o fcț generator care îmi întoarce un element nou din A

Proprietate

- ① A finită => A recursivă

Construim programul $P(n) \in P_{(1,1)}$ care ~~descrie~~ decide A

```
P(n)
    for (i = 1 : |A|)
        if (A(i) == n)
            return 1
    return 0;
```

- ② A recursivă => A recursiv-numărabilă

Fie $P \in P_{(1,1)}$ care decide A => $P(n) = \begin{cases} 1 & n \in A \\ 0 & n \notin A \end{cases}$

$Q \in P_{(1,1)}$

```
Q(n)
    if (P(n) == 1)
        return 1
    return ⊥
```

③ Dacă A RN și not R => A infinită

Def Alternativă (2): Putem să construim un pseudo-algorit care se termină pt ∀ element din A ($u \in A$)

Ex mulțimea universurilor cu extratereștrii
mulțimea programelor care se termină.
RN, not R

III ③ A nerecursivă (NR) - orice mulțime cai nu este RN

-④ $A, B - R \Rightarrow A \cap B, A \cup B, A \setminus B, -R$
$A, B - RN \Rightarrow$ — " — — RN

-⑤ A RN și B - RN ($B = \complement(A)$) => A R

$\complement(A) - RN \Rightarrow P_B(u) = \begin{cases} 1 & u \notin A \\ \perp & u \in A \end{cases}$

$ARN \Rightarrow P_A(u) = \begin{cases} 1 & u \in A \\ \perp & u \notin A \end{cases}$

Construim generator $Q(u) = \begin{cases} 1 & u \in A \\ 0 & u \notin A \end{cases} \Rightarrow A\ R$

```
i = 1
for( ; i++; )
    if (PA(u) se termină în i unități de timp)
        return 1;
    if (PB(u) — " — )
        return 0;
```

-⑥ A RN și not R => $\complement(A)$ nerecursiv

ex mulțimea univ fără extratereștrii
mulțimea programelor care nu se termină pt o intrare x

Fie [un predicat binar / o problemă de decizie } $\Rightarrow P : \underset{N}{i} \to \{0,1\}$

M_T - mulțimea valorilor de adevăr pt P

$M_T = \{ n \in i \text{ a.î } P(n) == 1 \}$

P { decidabilă dacă M_T recursivă
semidecidabilă M_T recursiv numărabilă
nedecidabilă M_T nerecursiv

!

PCP (Post's Correspondance Problem)

Fiind date 2 liste, fiecare cu n cuvinte din același alfabet Σ, cu prop $|\Sigma| > 1$

$X = \langle x_1, x_2 \dots x_n \rangle$

$Y = \langle y_1 \dots y_n \rangle$

∃ un șir de indici de dimensiune k cu $k \geq 1$ $(1 \leq i_k \leq n)$

a.î. $x_{i_1} x_{i_2} \dots x_{i_k} = y_{i_1} y_{i_2} \dots y_{i_k}$ (concatenare)

X = <aba, aa, acb>

Y = < ab, aaab, cb>

1 $\boxed{\begin{matrix}ab\\ab\end{matrix}}a$ 2 $\boxed{\begin{matrix}aa\\aa\end{matrix}}ab$ 3 ~~acb cb~~

1, 2 $\boxed{\begin{matrix}aba\ aa\\aba\ aa\end{matrix}}b$

HALTING PROBLEM (Problema opririi)

Fie un program P cu o intrare x $P_{1,1}$

Se termină P când primește x de la intrare?

$P(x) \neq \perp$?

Fie programul următor (consid x fixat)

```
Q(P)
    i = 1
    while(1)            -> ⊥
        if(P(x) se termină în i unități de timp)
            return 1
        i++;
```

mulțimea programelor care se termină e recursiv numărabil

$\Rightarrow$ $\boxed{\text{Semidecidabilă (HALT)}}$

Arătăm că HALT(P, x) nu este decidabilă prin reducere la absurd

Construim programul Q

```
Q(x)
    if(HALT(Q,x))
        RM
        return ⊥
    return 1
```

HALT este decidabil

$$HALT(P,x) = \begin{cases} 1, & P(x) \neq \bot \\ 0, & P(x) = \bot \end{cases}$$

Apelăm Q(Q) => 2 posibilități

1°) $HALT(Q,Q) = 1 \Rightarrow$ if(true) return $\bot$ $\Rightarrow Q(Q) = \bot \Rightarrow HALT(Q,Q),$

(CONTRADICȚIE)

↯

2°) $HALT(Q,Q) = 0 \Rightarrow$ if(false) return $\underline{1}$ $\Rightarrow Q(Q) = \underline{1} \neq \bot \Rightarrow HALT(Q,Q) =$

↯

=> HALT nu este decidabil

AA

Curs 3

Reducerea Turing

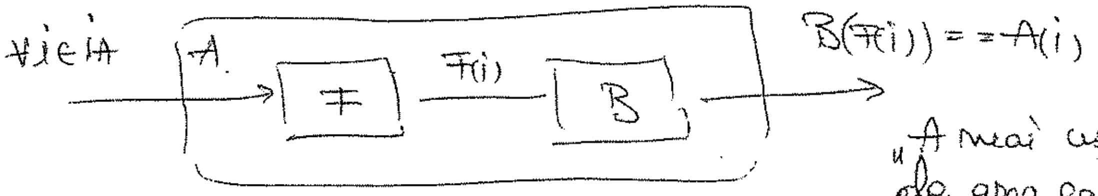
Fie A,B 2 probleme de deducere $\begin{cases} A: i_A \to \{0,1\} \\ B: i_B \to \{0,1\} \end{cases}$

Spunem că A se ~~deduce~~ reduce Turing la B dacă

1) $\exists$ $F: i_A \to i_B$ program care transformă datele de intrare ale problemei A în date de intrare $\overset{A}{}$ B

2) $\exists$ $i \in i_A$ $\left(\begin{array}{l} A(i) == 1 \Leftrightarrow B(F(i)) == 1 \\ A(i) == 1 \Leftrightarrow B(F(i)) == 1 \\ \quad\;\; == 0 \Leftrightarrow B(F(i)) == 0 \end{array} \right)$

$\boxed{A \leq B}$ → A se reduce Turing la B



„A mai ușoară sau la fel de grea ca B"

$A \leq B \Rightarrow \exists$

```
alg-A-rd(i)
    i2 = F(i)
    return alg-B(i2)
```

pt a rezolva pb A

decidabil / nedecidabil (semi + ne)

Știm $A \leq_T B$

a) A - nedecidabilă (exemplu probl opririi) => B - nedecidabilă

b) B - decidabil => A decidabil

c) B - nedecidabil => A? ⌐ decidabil
nedecidabil

Teorema lui Rice

Orice proprietate care este extensională și nebanală asupra programelor este nedecidabilă

ex: Prog pătrate perfecte

```
pătrat-perfect (n)
    return n*n;
```

$\text{HALT } P: P(x) \neq \perp\ ?$

Prog pătrat-perfect: R-program

```
halts (P,x)
    Q (n)
        P(x)
        return n*n;
    return pătrat-perf (Q)
```

$\text{HALT} \leq \text{is-pătrat-perfect}$

$\text{HALT} \leq_T \text{PCP}$

Complexitatea algoritmilor

Orice problemă decidabilă are cel mai bun algoritm de rezolvare a problemei date P?

Variante de rezolvare:

→ implementare algoritm în C / Java etc.

→ rulare pe mai multe date de intrare

Care algoritm este mai rapid

Depinde de:

- limbaj de programare
- calculator
- programator
- set de date

Problemă: Fie un vector $A[1 \ldots n]$. Vrem să determinăm subsecv de sumă maximă

$O(n^3)$ $O(n \log n)$

$O(n^2)$ $O(n)$

Vrem să determinăm cât de bun este un algoritm fără a implementa un program

Ne trebuie o modalitate de a calcula eficiența algoritmilor doar din pseudocod => complexitate:

{ temporală (cât de rapid)
{ spațială (câtă memorie e necesară / mem suplimentară

Vom măsura complexitatea unor algoritmi doar în fct de pseudocod și de datele de intrare (de dimensiunea lor) → talia problemei

$T(n) : \mathbb{N} \to \mathbb{R}_+$ fct de complexitate (timp de rulare)

Se poate extinde discuția pt probleme cu date de intrare cu mai mult de o dimensiune

$T(n, m) \to$ grafuri $G(V, E)$
$|V| \to n$
$|E| \to m$

$T(l_1, c_1, c_2) \to$ înmulțire matrici

Problema Sortării

Se dă un vector $A[1 \ldots n]$. Vrem ~~să~~ o permutare sortată a elementelor din vector: $A[i] < A[i+1] \ \forall \ 1 \leq i < n$

```
Insertion_Sort (A[1...n])
    for (j=2...n)
        x = A[j]
        i = j-1
        while (i>0 and A[i] > x)
            A[i+1] = A[i]
            i--;
        A[i+1] = x
    return A
```

Sortare „inplace"

Când ieşim din while?

1. $i == 0$
2. $x \geq A[i]$

Simplificari

(1) orice instrucţiune simplă (asignare, op, arit-logice, simplificări) durează un timp constant

Program	Complexitate instrucţiune	De câte ori este instrucţiunea executabilă?
linia 1	c_1	n
2	c_2	$n-1$
3	c_3	$n-1$
4	c_4	$T_1(n) = \sum_2^n t_j$
5	c_5	$T_2(n) = \sum (t_j - 1)$
6	c_6	$T_3(n)$
7	c_7	$n-1$
8	c_8	1

$$T(n) = \sum_{i=1}^{nr\ instr} (\text{complex instruct } i * \text{nr de ori instruct } i)$$

$$= c_1 n + (c_2+c_3+c_7)(n-1) + c_4 T_1(n) + c_5 T_2(n) + c_6 T_3(n) + c_8$$

Deoarece în cazul general timpul de execuție $T(n)$ nu poate fi calculat precis, ne uităm la cazuri particulare

-> cazul cel mai puțin favorabil (worst case)

-> cazul cel mai favorabil (least case)

-> cazul medie

② Ne interesează doar complexitate worst case

$T_{wc}(n) = an^2+bn+c$

Vector sortat invers

$t_j = j \Rightarrow T_1(n) = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$

$(j \neq i-1)$

$T_2(n) = \sum_{j=2}^{n-1} (t_j - 1) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$

$T_{bc}(n) = an+b$

Vector deja sortat

$t_j = 1 \Rightarrow \begin{cases} T_1(n) = \sum_{j=2}^{n} 1 = n-1 \\ T_2(n) = 0 \end{cases}$

$T_{med}(n) = \frac{a_{wc}}{2} n^2 + \dots$

$j/2 \sim j \Rightarrow T_1(n) = \sum j/2 = \frac{1}{2} T_{1\,wc}(n)$

③ Ne interesează complexitatea pt $n \to \infty$

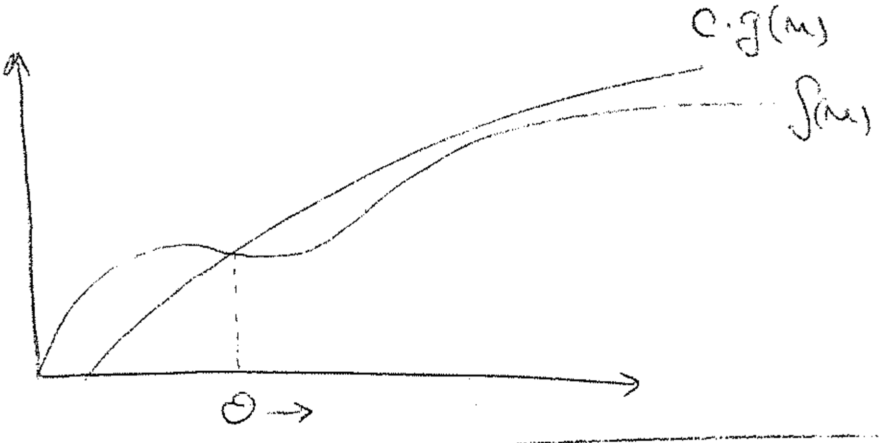
$T(n)$ $n \to \infty$ → notații asimptotice → O, Θ, Ω

↳ folosite în matematică pt a compara fcții $f : \mathbb{N} \to \mathbb{R}_+$ → $f(n) = \frac{1}{n}$, $f(n) = \sin n$, $f(n) = n^{1+\sin n}$

Margine asimptotică superioară (O)

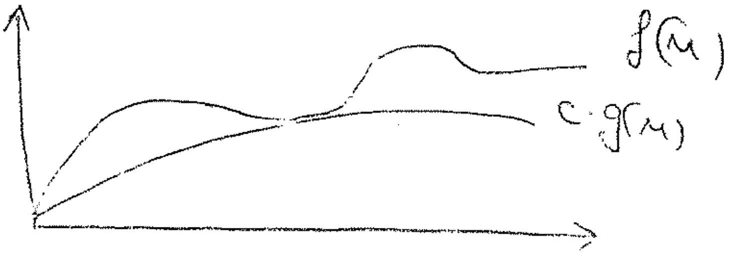
$$\boxed{O(g(n)) = \{ f(n) : \mathbb{N} \to \mathbb{R}_+ \mid \exists\ c > 0, n_0 \in \mathbb{N}_+ \text{ a.î. }} \ 0 \leq f(n) \leq c \cdot g(n) \text{ marg sup} \}$$

seturi de fcț



Mărginite inferior (Ω) asimptotic

$$\boxed{\Omega(g(n)) = \{ \ \ldots \ 0 \leq c \cdot g(n) \leq f(n) \ \forall \ n \geq 0 \}}$$



Ordinul de crestere Θ

$$\boxed{\Theta(g(n)) = \left\{ f \mid \exists\ c_1, c_2 > 0,\ n_0 \in \mathbb{N} \text{ a.î } c_1 \cdot g(n) \le f(n) \le c_2 g(n)\ \forall n > n_0 \right\}}$$

Proprietate

$$\boxed{1)\ f(n) \in \Theta(g(n)) \Leftrightarrow \begin{cases} f(n) \in O(g(n)) \\ f(n) \in \Omega(g(n)) \end{cases}}$$

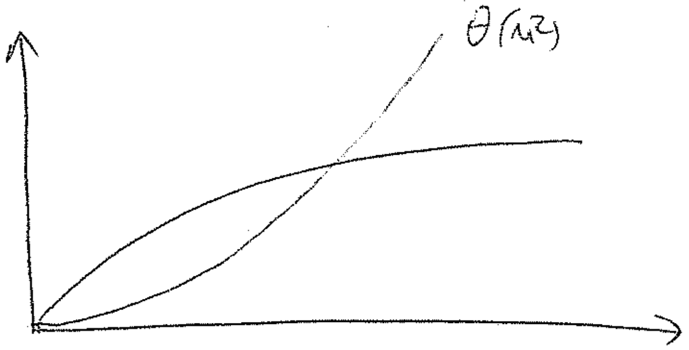
$16n^2 \in O(n^2)$? Adevarat

$7n + 100\,000 \in O(n^2)$? Adevarat

$\frac{n^2}{1000} + n^2 \in \Omega(n^2)$? Adevarat

$\in \Omega(n^3)$? Adevarat

$765n^2 + 16n + 5 \in \Theta(n^2)$? Adevarat



O, Ω → relații de ordine parțială

a) reflexivitate : $\forall f : \mathbb{N} \to \mathbb{R}_+\ f(n) \in O(f(n))$

b) Antisimetrie : $\forall f, g\ \ f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$

c) tranzitivitate: $\forall f, g, h\ \ \left.\begin{matrix} f(n) \in O(g(n)) \\ g(n) \in O(h(n)) \end{matrix}\right\} \Rightarrow f(n) \in O(h(n))$

Θ – relație de echivalență (clase de echivalență)

↳ folosite pt a defini complexitatea algoritmilor

1) reflexivitate

2) simetrie $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$

3) tranzitivitate

$\Theta(1)$, $\Theta(\log\log n)$, $\Theta(\log n)$, $\Theta(n)$, $\Theta(n\log n)$, $\Theta(n^2)$, $\Theta(n^3)$, $\Theta(n^k)$, $\Theta(2^n)$, $\Theta(n!)$, $\Theta(n^n)$

4) Dacă $T(n) = an^2+bn+c \in \Theta(n^2)$

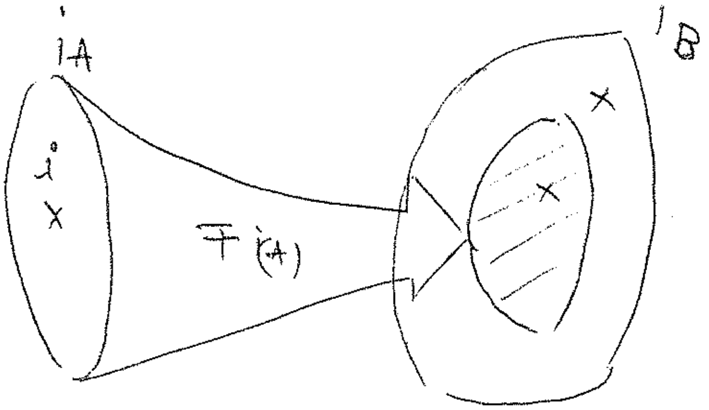
AA

Curs 4

A, B → 2 probleme de decizie

$A \leq_T B$ Probl A e mai ușoară sau cel mult egală cu B

$A: i_A \to \{0,1\}$; $B: i_B \to \{0,1\}$



$F(i_A) \leq i_B$

Problema rucsacului: $\Theta(n \cdot G)$

Complexitatea algoritmilor recursivi

Alg recursivi = alg care se apelează pe ei înșiși pt a rezolva o problemă de dimensiune n.

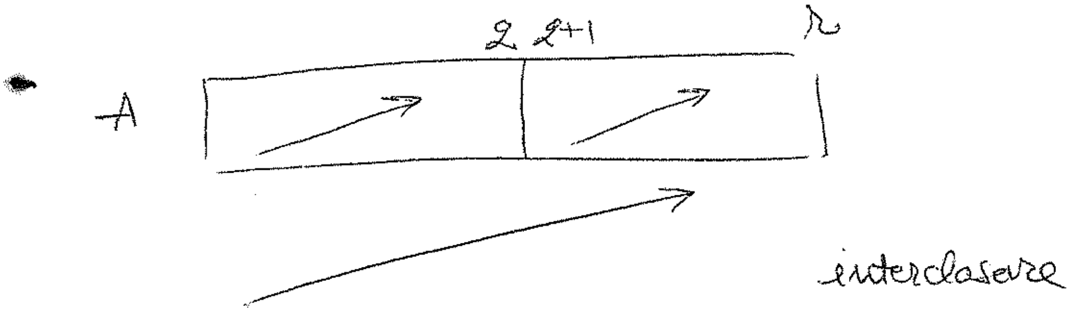
Recursivitate mutuală

```
alg = 1()          alg... 2()
   alg 2()            alg 1()
```

Important este ca apelul recursiv să fie efectuat pt probleme de dimensiune mai mică ca n. (sub-probleme)

Mergesort (Sortare prin interclasare)

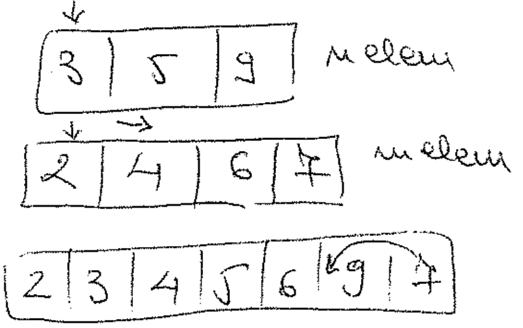
Pentru a sorta un subvector ~~vector~~ A[p...r], calculăm mijlocul, sortăm recursiv prima jumătate A[p...q] și a 2 jumătate A[q+1...r]. Apoi interclasăm cei 2 subvectori



```
Mergesort (A, p, r)  → „divide et impera"
    if (p >= r)          // A[p...r] → dim 0 sau 1
        return
    $q = \left[\frac{p+r}{2}\right]$      // $p + \frac{r-p}{2} = \frac{2p+r-p}{2} = \frac{r+p}{2}$

    Mergesort (A, p, q)
    Mergesort (A, q+1, r)
    Merge~~sort~~ (A, p, q, r)
```

Merge (A, p, q, r)



=> Complexitate $\Theta(n+m)$

Folositor pt motoarele de căutare (IR) information retrieval

analiza - 13 → 20 → 27 → 40 → 100 → ...

algoritmilor - 5 → 12 → 13 → 40 → 41 → ... → 13, 14 ....

$q = \frac{p+r}{2}$

```
Merge (A, p, q, r)
    n1 = p - q + 2
    n2 = r - (q+1) + 2 = r - q + 1
    B = new array(n1)
    C = new array(n2)
    copy(A(p...q), B(1...n1-1))
    copy (A(q+1...r), C(1...n2-1))
    B{n1} = C{n2} = iNF        // Santinela
    i = 1, j = 1, k = p
    while (i < n1 && j < n2)
        if (B(i) < C(j))
            A(k++) = B(i++)
        else
            A(k++) = C(j++)
```

↑ foloseste memorie suplimentara **Complexitate spatiala $\Theta(n)$**

Complexitate temporala

$$\underbrace{T(n)}_{\text{complexitate nrgs pt o probl dim } n} = \underbrace{2T(n/2)}_{\text{etapa impera}} + \underbrace{\Theta(1)}_{\text{etapa divide}} + \underbrace{\Theta(n)}_{\text{etapa combina}}$$

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{pt } n > 1 \\ \Theta_1 & \text{pt } n \le 1 \ (n == 1) \end{cases}$$

(relatie de recurenta)

Cum rezolvăm o relație de recurență

① Metoda iterației (metoda algebrică)

$$T(n) = 2\cancel{T(n/2)} + \Theta(n)$$

→ inducție incompletă

h → înălțimea unui arbore

$$\cancel{T(n/2)} = 2T(n/2^2) + \Theta(n/2) \quad |\cdot 2$$

$$\cancel{T(n/2^2)} = 2T(n/2^3) + \Theta(n/2^2) \quad |\cdot 2^2$$

— — — — — — — — —

$$T(n/2^i) = 2T(n/2^{i+1}) + \Theta(n/2^i) \quad |\cdot 2^i$$

$$T(n/2^h) = T(1) = \Theta(1) \qquad \cdot 2^h$$

$$T(n) = \Theta_n + 2\Theta(\tfrac{n}{2}) + 2^2\Theta(n/2^2) + 2^i\Theta(n/2^i) + \dots 2^h\Theta(n/2^h)$$

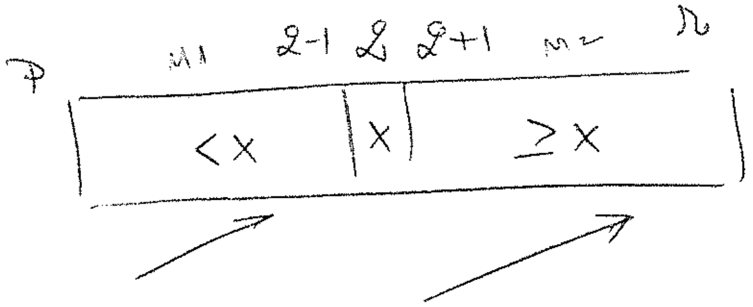
$$\Rightarrow T(n) = \Theta(n) + \Theta(n) + \dots \Theta(2^h) \Rightarrow T(n) = (h+1)\Theta(n) \Rightarrow$$

(*) $\frac{n}{2^h} = 1 \Rightarrow h = \log_2 n = \log n = \lg n$

$$\Rightarrow \boxed{T_n = \Theta(n\lg n + n) = \Theta(n\log n)}$$

Quicksort (A, p, n)

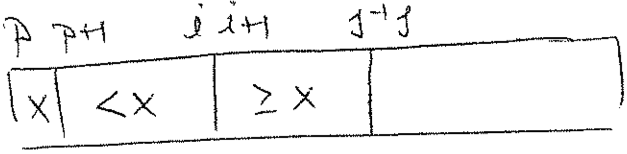
```
if (p ≥= n)
    return
q = partition (A, p, n)
Quicksort (A, p, q-1)
Quicksort (A, q+1, r)
```



$\forall y \in A[p \dots q+1] \quad y < x$

$x = A[q]$ - pivot

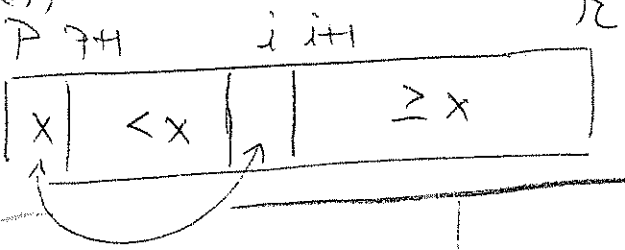
```
partition (A, p, r)
    x = A[p]
    i = p
    for j = p+1, r
        if (A(j) < x)
            i++
            swap (A(i), A(j))
    swap (A(p), A(i))
    return i
```



$A[p+1 \dots i] < x$

$i = p \quad A[p+1, \dots p] = \emptyset < x$

la final:



Complexitate:

$$T(n) = T(n_1) + T(n_2) + \Theta(n)$$

$$\left(\begin{array}{l} n_1, n_2 < n \\ n_1 + n_2 = n-1 \end{array} \right)$$

Worst case

A - ~~sortat~~ invers $\quad n_1 = n-1$, $n_2 = 0$

A → ~~sortat~~ $\quad n_1 = 0$, $n_2 = n-1$

$$T(n) = T(n-1) + \Theta(n)$$
$$T(n-1) = T(n-2) + \Theta(n+1)$$
$$\text{---}\quad \text{---}\quad \text{---}\quad \text{---}$$
$$T(1) = \Theta(1)$$

$$T(n) = \Theta(1+2+\ldots+n) = \Theta\left(\frac{n(n+1)}{2}\right) = \boxed{\Theta(n^2)}$$ ~~ac~~

Best case: De fiecare dată, $\quad n_1 = n_2 = \frac{n}{2}$

$$\boxed{T(n) = 2T(n/2) + \Theta(n)} \Rightarrow T(n) \in \boxed{\Theta(n \log n)} \quad \text{ac}$$

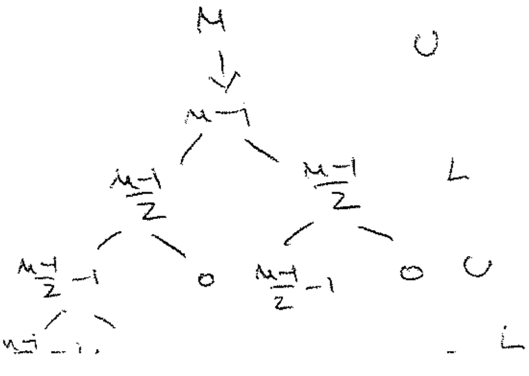
↳ cazul cel mai favorabil

„average case"

wc → Bc → wc → BC

$$U(n) = L(n-1) + \Theta(n) = 2U\left(\frac{n-1}{2}\right) + \Theta(n-1) + \Theta(n)$$
$$L(n\cancel{-1}) = 2U\left(\frac{n\cancel{-1}}{2}\right) + \Theta(n)$$

$$\le 2U\left(\frac{n}{2}\right) + \Theta(n)$$
$$\downarrow$$
$$\Theta(n \log n)$$



Cum facem să scăpăm de worstcase

① Alegem pivotul x să fie elem median din $A(p \dots r)$
$\hookrightarrow \Theta_n$

→ ② Înainte de a apela partition, aleg un pivot aleatoriu

```
i = random(p..r)
swap(A(p), A(i))
partition(A, p, r)
```

$$\frac{1}{n} + \frac{1}{n} = \frac{2}{n}$$

$$\frac{2}{n} \cdot \frac{2}{n-1} \cdots \frac{2}{2} \simeq \frac{2^n}{n!} \xrightarrow[n\to\infty]{} 0$$

AA

27.10.2017

Curs 5

Misc

1) Avem un vector A(1...n) și știm că trb să răsp la multe întrebări de forma min(A[i...j])?

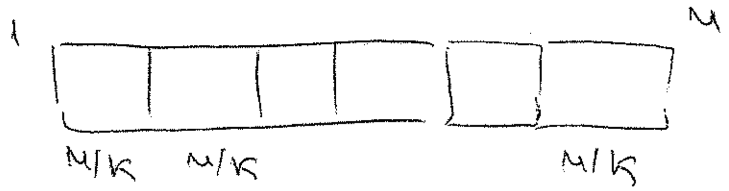
→ soluția banală ne duce în worstcase (for(k=i; k<=j, k++))

⇒ complexitate O(n)

→ precalcularea poate fi utilă

→ avem o complexitate mai mare o dată (precalculare)

→



k = subvector dim $\frac{n}{k}$

→ precalculăm minimul pt fiecare subvector de dim $\frac{n}{k}$ (subsecvență) $\Theta(n)$

→ la fiecare interogare pentru subvectorii conținuți complet în interogare folosind minimele precalc și mai rămân 2 capete pt care trb să calculăm minimul. Folosind minimele precalc, mai rămân încă 2 hașurate în desen

$j-1 < \, < n/k$

$$\text{fiecare dim} < \frac{n}{k} < 2 \cdot \frac{n}{k} + k - 2 < 2\frac{n}{\sqrt{n}} + \sqrt{n} - 2 = 3\sqrt{n} - 2$$

$$\Downarrow$$

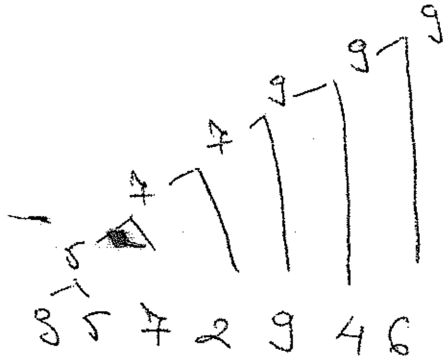
$$\Theta(\sqrt{n})$$

Câte k optim?

$$f(k) = \frac{2n}{k} + k - 2$$

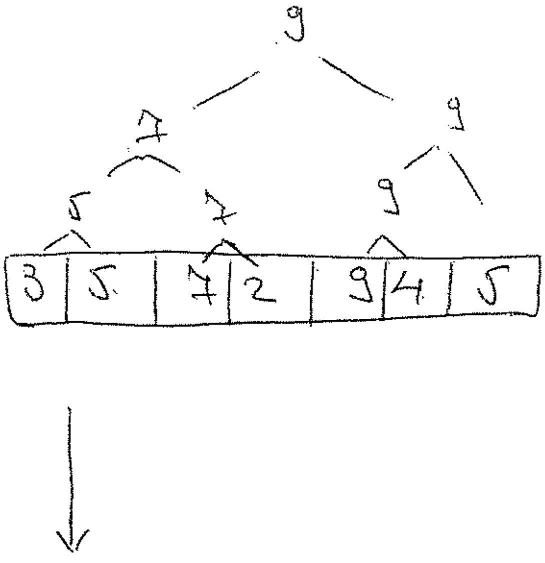
$$\frac{\partial f}{\partial k} = -\frac{2n}{k^2} + 1 = 0 \Rightarrow k^2 = 2n \Rightarrow k = \sqrt{2n} \Rightarrow k = \sqrt{n}$$

2) Care este nr minim de comparații pt a calc al 2 cel mai mare element dintr-un vector A[1...n]? (cazul cel mai nefavorabil)



(n-1) comparații pentru elementul maxim

h comparații



arbore de tip turneu (tournment tree)

h de (log n)

(n-1) comparații pt elem max

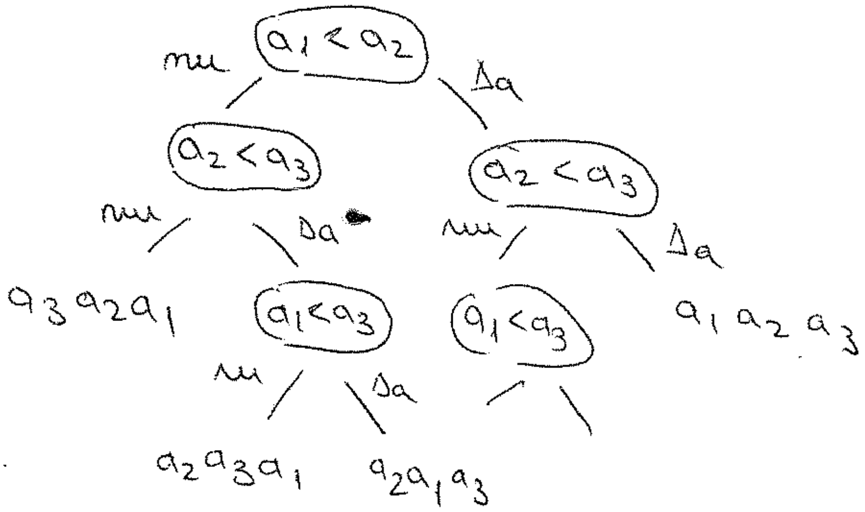
(log n - 1) comparații pt al 2 max

n + log n - 2 comparații

al 2 max ar trb să fie un contracandidat (5 sau 7)

3) Orice algoritm de sortare, prin comparaţie de chei are complexitatea $\Omega(n\log n)$

Presupunem un vector cu 3 elem $<a_1, a_2, a_3>$



În arbore avem nr minim de comparaţii de chei necesare să luăm o decizie. Frunzele sunt cele 6 permutări sortate ce se pot obţine pt un vector cu 3 elemente oarecare $(3! = 6)$

Nr de comparaţii de chei (în cazul cel mai defavorabil) e dat de înălţimea arborelui (h)

Proprietate pt arbori binari

$$\left.\begin{array}{l}\boxed{\text{nr frunze} <= 2^h} \\ \text{nr frunze} = n!\end{array}\right\} \quad \boxed{n! <= 2^h}$$

logaritmăm în baza 2 =>

$$\log_2 n! \le h \Rightarrow$$

=> Varianta I (Matematic) Thr Stirling

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\log n! \sim \log\sqrt{2\pi} + \log\sqrt{n} + \log n^n - \log e^n$$

$$\sim \frac{1}{2}\log n + n\log n - n\log_2 e$$

$$\Rightarrow \underbrace{n\log n - n^{+}\log n}_{\Theta(n\log n)} \le h \Rightarrow \boxed{h \in \Omega(n\log n)}$$

Varianta 2 (inginerească)

$$h \ge \log n! \ge \sum_{i=\frac{n}{2}}^{n} \log i = \log\left(\prod_{i=\frac{n}{2}}^{n} i\right) \ge \log\frac{n}{2}\cdot\frac{n}{2} = \frac{n}{2}(\log n - 1) =$$

$$\log\frac{n}{2} + \log\left(\frac{n}{2}+1\right) + \ldots \log n \ge n\log\frac{n}{2}$$

$$\Rightarrow \in \Theta(n\log n)$$

Complexitatea algoritmilor recursivi

Reduce et impera $T(n) = aT(n-b) + f(n)$

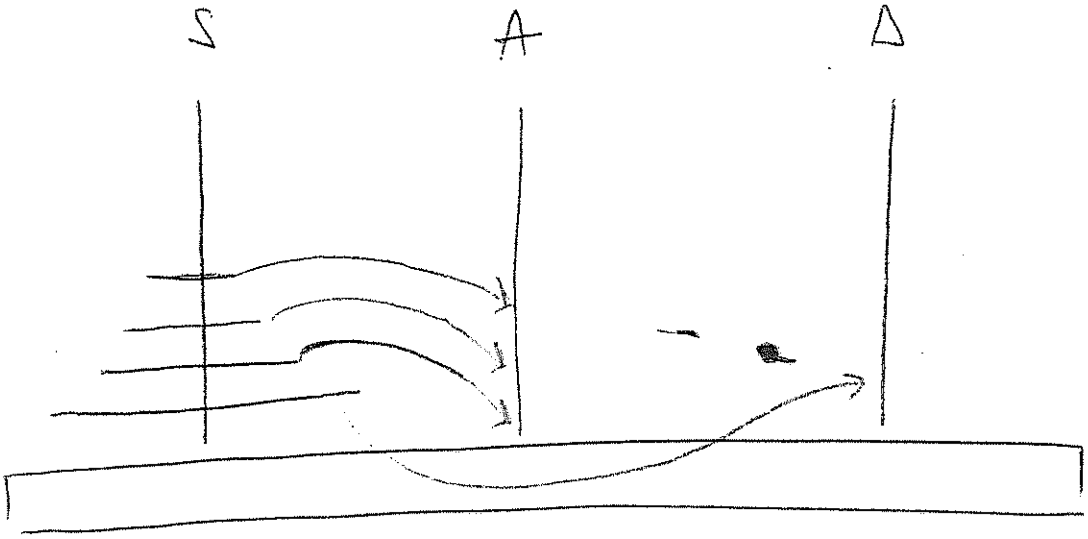
Divide et impera $T(n) = aT(n/b) + f(n)$

Problema turnurilor din Hanoi

Într-un templu tibetan există N discuri şi 3 stâlpi. Iniţial toate cele n discuri sunt plasate pe stâlpul S, iar călugării din templu vor să mute toate discurile pe stâlpul D (destinaţie).

Știind că discurile au dim diferite și întotdeauna un disc mai mare nu poate sta pe un disc mai mic, ajutați călugării să mute discurile de pe S pe Δ folosind al 3 lea stâlp ca stâlp auxiliar. (Cică N = 64)

Care e nr de pași (mutare de disc) de pe un stâlp pe altul?



```
Hanoi (S, Δ, A, N)
    if (N==0)
        return
    Hanoi (S, A, Δ, N-1)
    Print („Mută discul" + N + „de pe" + S + „pe" + Δ)
    Hanoi (A, Δ, S, N-1)
```

$T(n) = 2T(N-1) + \Theta(1)$ (Recurența)

$T(n-1) = 2T(N-2) + \Theta(1) \quad | \cdot 2$

$T(n-i) = 2T(N-i) + \Theta(1) \quad | \cdot 2^i$

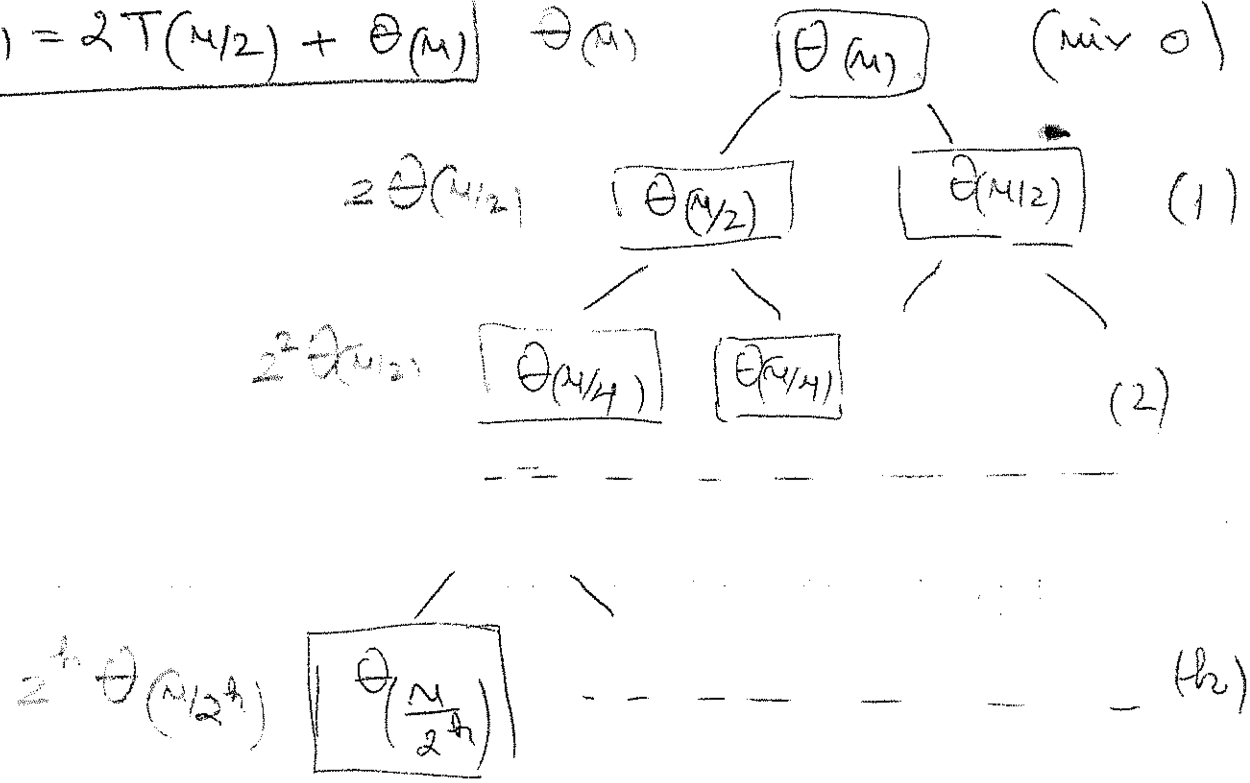
$T(0) = \Theta(1) \quad | \cdot 2^n$

$T(n) = \Theta(1) + \Theta(2^1) + \dots + \Theta(2^n) = \frac{2^{n+1}-1}{2-1} = 2^{n+1} - 1 \in \Theta(2^n)$

Metode de rezolvare a recurențelor

1. metoda iterației (algebrică)
2. metoda arborilor de recurență

$$\boxed{T(n) = 2T(n/2) + \theta(n)}$$



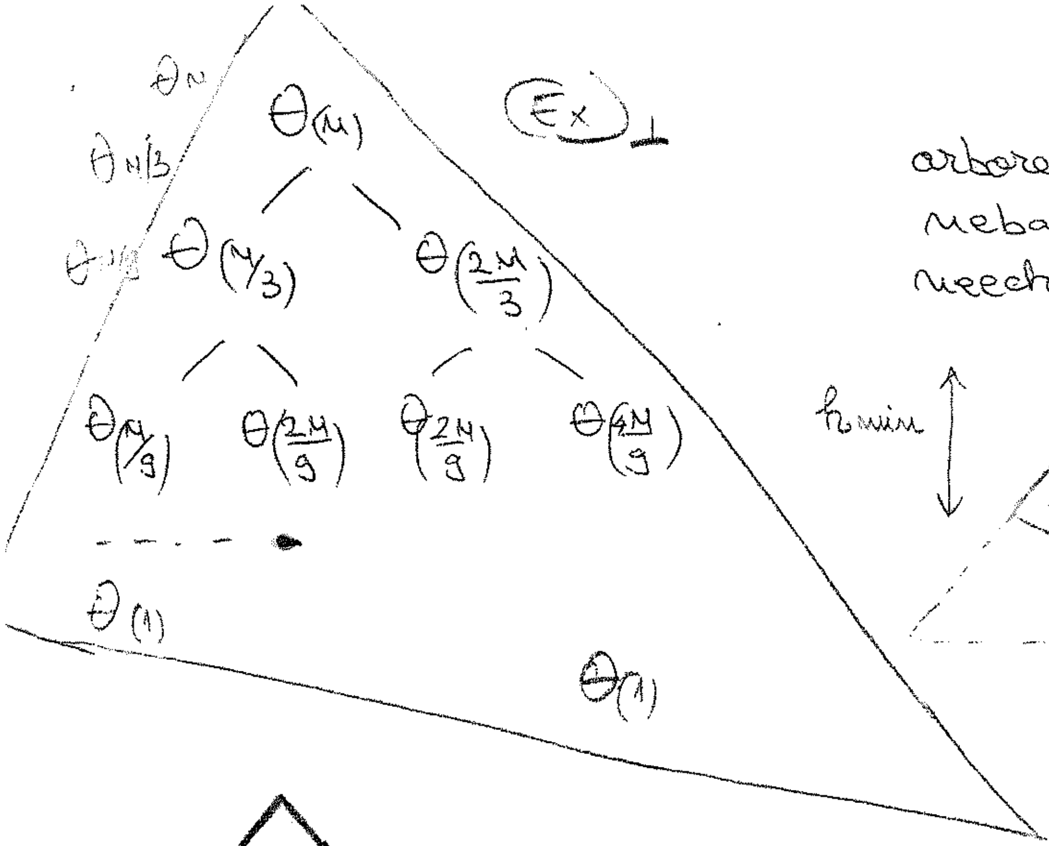
$\frac{n}{2^h} = 1 \Rightarrow h = \log_2 n$

$\theta\left(\frac{n}{2^h}\right) = \theta(1)$

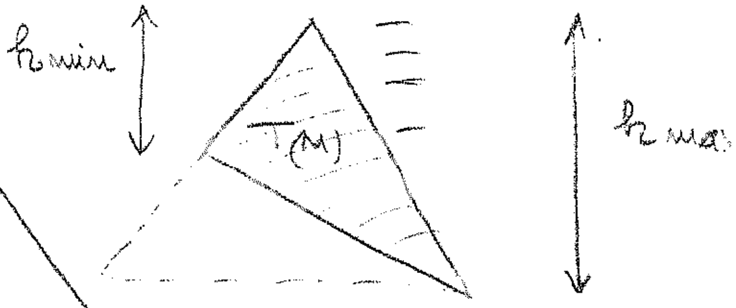
$\theta(n) \cdot (h+1)$

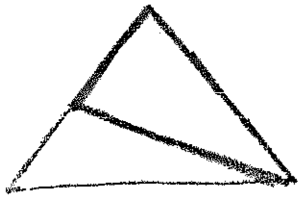
mergesort: $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \theta(n)$

Ex 1


Θ_n
$\Theta_{n/3}$
$\Theta_{n/9}$
$\Theta(n)$
$\Theta(n/3)$
$\Theta\left(\frac{2n}{3}\right)$
$\Theta\left(\frac{n}{9}\right)$
$\Theta\left(\frac{2n}{9}\right)$
$\Theta\left(\frac{2n}{9}\right)$
$\Theta\left(\frac{4n}{9}\right)$
$\Theta(1)$
$\Theta(1)$

arbore de recurență
nebalansat
neechilibrat:





$$T_1(n) = (h_{min}+1)\,\Theta(n) = \left(\frac{\log n}{\log_2 n}+1\right)\Theta(n) \in \Theta(n\log n)$$

$$\frac{n}{3^{h_{min}}} = 1 \Rightarrow h_{min} = \log_3 n = \frac{\log n}{\log_2 3}$$

$$\Rightarrow \boxed{T(n) = \Theta(n\log n)}$$

Ex 2

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + \Theta(n^2)$$

$\Theta\, n^2$ ——— $\Theta(n^2)$

$\Theta(n^2/2)$ $\Theta(n^2/2)$ ——— $\Theta\left(\frac{n^2}{4}+\frac{n^2}{16}\right) = \Theta\left(\frac{5}{16}n^2\right)$

$\Theta\left(\left(\frac{n}{4}\right)^2\right)$ $\Theta\left(\left(\frac{n}{8}\right)^2\right)$ $\Theta\left(\left(\frac{n}{8}\right)^2\right)$ $\Theta\left(\left(\frac{n}{16}\right)^2\right)$ ——— $\Theta\left(\frac{n^2}{16}+\frac{n^2}{64}+\frac{n^2}{64}+\frac{n^2}{256}\right) = \Theta\left(\left(\frac{5}{16}\right)^2 n^2\right)$

$\Theta\left(\left(5/16\right)^i n^2\right)$

i

$$T(n) = \sum_{i=0}^{?}\left(\frac{5}{16}\right)^i n^2 = n^2\sum_{i=0}^{?}\left(\frac{5}{16}\right)^i$$

progresie

$$\Rightarrow T(n) = n^2\sum_{i=0}^{?}\left(\frac{5}{16}\right)^i \le n^2\sum_{i=0}^{\infty}\left(\frac{5}{16}\right)^i = n^2\cdot\frac{1}{1-\frac{5}{16}} = n^2\frac{16}{11} \Rightarrow \Theta(n^2)$$

$$\begin{tikzcd}[column sep=tiny]
\Theta(n^2) \le T(n) \le \Theta(n^2) \arrow[d] & & \\
\Theta(n^2) \quad (\text{clește}) & &
\end{tikzcd}$$

3. metoda substituției

Spre deosebire de met 1. și 2. care folosesc o inducție incompletă (observații), 3. folosește inducție completă fiind astfel riguroasă dpv matematic.

Schema inducției complete

$P(n)\ \forall n \in \mathbb{N}$

$$\frac{P(k_0) \qquad k=0,1\ldots \quad \ldots \quad \dfrac{\text{Pp } P(k)\ \forall k<n}{P(n)}}{P(m)\ \forall m \in \mathbb{N}}$$

Ex $T(n) = 2T(n/2) + n$

Pp $T(n) = n \log n + n$

$n=1$: $T(1) = 1 \log 1 + 1 = 1$ (A)

Pp $P(k)$: $T(k) = k \log k + k \quad \forall k < n$

$T(n) = 2T\left(\frac{n}{2}\right) + n = 2\left(\frac{n}{2} + \log \frac{n}{2} + \frac{n}{2}\right) + n =$

$= n \log n - n \log 2 + 2n = n \log n + n$ (A)

Putem folosi MS și ~~puterea~~ pt a dem margini asimptotice

$$T(n) \in O(n \log n)$$

$\exists$ c a.î. $T(n) \le c \cdot n \cdot \log n \quad \forall\, n \ge n_0$

CB : $n = 1 \quad T(1) = 1 \le c \cdot 1 \log 1$

: $n = 2 \quad T(2) = 2T(1) + 2 = 4 \le c\, 2 \log 2 \Rightarrow 2 \le \frac{4}{2}$

Pi Pp $T(k) \le c \cdot k \log k$ pt $\forall\, k < n$

$$\Rightarrow T(n) = 2T\left(\frac{n}{2}\right) + n \le \cancel{2} c \frac{n}{\cancel{2}} \log \frac{n}{2} + n$$

$$T(n) \le c \cdot n \log n \underbrace{- c n \cancel{\log 2} + n}_{- \le 0} \overset{?}{\le} c n \log n$$

$$n(1 - c) \le 0 \Rightarrow 1 - c \le 0 \Rightarrow c \ge 1 \Rightarrow \exists\, c = 2$$

$$\begin{tikzcd} & O \\ T(n) \in \Theta(n \log n) \arrow[ur] \arrow[dr] & \\ & \Omega \end{tikzcd}$$

---

4. Metoda master

Folosește ~~Au~~ Th Master care rezolvă

$$T(n) = aT(n/b) + f(n) \quad \left|\begin{array}{l} a \ge 1 \\ b > 1 \\ f(n) > 0 \end{array}\right.$$

Dacă $\exists$ o dif polinomială între funcții $T(n) \in \Theta$(de cea mai mare)

Not $\alpha = \log_b a$

$n^\alpha \ ? \ f(n)$

3 cazuri

I $\boxed{f(n) \in O(n^{\alpha-\varepsilon});\ \varepsilon>0 \text{ const},\ T(n)=\Theta(n^{\alpha})}$

II $\boxed{f(n) \in \Theta(n^{\alpha}\log^{k} n),\ k\geq 0,\ \Rightarrow T(n)\in\Theta(n^{\alpha}\log^{k+1} n)}$

III $f(n)\in\Omega(n^{\alpha+\varepsilon})$, $\varepsilon>0$ ct și $\left\{\begin{array}{l}\exists \text{ ct e } \begin{array}{l} c<1 \\ c>0\end{array} \\ a f(n/b) \leq c f(n)\end{array}\right. \Rightarrow T(n)=\Theta(f(n))$

f(n) polinomială, condiția e adevărată

condiție de regularitate

Ex $T(n) = 2T(n/2) + n$

$n^{\log_2 2} = n$? $f(n) = n$

$f(n) = n \in \Theta(n^{\log_2 2}\cdot \log^0 n) \xRightarrow[\text{caz b}]{TM} T(n) = \Theta(n \log n)$

Ex

$T(n) = 2T(n/2) + n\log n$

$f(n) = n\log n = \Theta(n\log n \log n) \Rightarrow T_n = \Theta(n\log^2 n)$

$\frac{n^{\varepsilon}}{\log n} = \frac{n^{\varepsilon-1}}{\frac{1}{n}} = n^{e} \xrightarrow[n\to\infty]{} \infty$

$\varepsilon = 0{,}0003$

Obs! Teorema Master are goluri (nu poate rezolva ∀ recurența).

ex: $T(n) = 2T(n/2) + n/\log n$

$f(n) = \frac{n}{\log n} \in \Theta(n\log^{-1} n)$ nu se poate

A~~A~~

Curs 6

Misc

(1)

```
func (A, p, r)
    if (A[p] > A[r])
        A[p] <-> A[r]
    if (r - p - 1 >= 3)
        t = (r - p)/3
        func (A, p, r-1)   // primele 2 treimi
        func (A, p+t, r)   // ultimele 2 treimi
        func (A, p, r-t)   // primele 2 treimi
```

$$\boxed{T(n) = 3T\left(\frac{2n}{3}\right) + \Theta(1)}$$

$a = 3$

$b = 3/2$

$\log_{1,5} 3 =$

$f(n) \in \Theta(1) \in O(\quad\quad) \ \exists\ \varepsilon = \underline{1} > 0 \Rightarrow T(n) = \Theta(n^{2,75})$

(2)

$T(n) = 8T(n/2) + n^2 + \log^{1000} n$

$n^{\log_2 8} = n^3$

$f(n) = n^2 + \log^{1000} n$

$$\lim_{n\to\infty} \frac{n^2}{\log^{1000} n} = \lim_{n\to\infty} \frac{2n}{1000 \log^{999} n \cdot \frac{1}{n}} = \frac{2n^2}{1000 \cdot \log^{999} n}\Big|_{n\to\infty} \ldots$$

$$\ldots = \frac{4n^2}{1000 \cdot 999 \log^{999} n} = \ldots = \frac{2^{1000} n^2}{1000! \log^2 n} = \frac{2^{1001} n}{1000!} \longrightarrow \infty$$

$\log^{1000} n \in O(n^2)$

$o(n^2)$

$f(n) = O(n^2) + \Theta(n^2) = \Theta(n^2) \Rightarrow \left(\Theta(n^2) + \Omega(n^2) = \Omega(n^2) \right) \geq n^2$

$\Rightarrow O(n^2) \in \Theta(n^3) \xrightarrow{TM} T(n) = \Theta(n^3)$

Teorema Master Dem

① Dacă $f(n) \in \mathcal{O}(n^{\log_b a-\varepsilon}), \varepsilon>0 \Rightarrow T(n)\in\Theta(n^{\log_b a})$

$\exists c_1>0$ a.î. $f(n)\le c_1 n^{\log_b a-\varepsilon}$ $\forall n\ge n_0$

Vrem să dem prin metoda substituției

$T(n)\in\mathcal{O}(n^{\log_b a})$ (similar pt Ω)

Vrem să arătăm ~~că~~ $\exists c>0$ a.î. $T(n)\le c n^{\log_b a}$

ii $T(k)$ (A) pt $\forall k<n$

$\swarrow$

$T(n)$

iP $T(k)\le c\cdot k^{\log_b a}$ și $k=\frac{n}{b}<n$ $(b>1)$ $\Rightarrow T(\frac{n}{b})\le c(\frac{n}{b})^{\log_b a}$

$T(n)=aT(\frac{n}{b})+f(n)\overset{ii}{\le} a\cdot c\cdot(\frac{n}{b})^{\log_b a}+f(n)\le a\cdot c\frac{n^{\log_b a}}{b^{\log_b a}}+$

$+c_1 n^{\log_b a-\varepsilon}$ $\le c\cdot n^{\log_b a}+c_1 n^{\log_b a-\varepsilon}\overset{?}{\le} c\cdot n^{\log_b a}$

$\Rightarrow c_1 n^{\log_b a-\varepsilon}<0$ (contradicție)

$T(n)\in\mathcal{O}(n^{\log_b a})\Rightarrow\exists c>0, d>0$ a.î. $T(n)\le c\cdot n^{\log_b a}-d\cdot$

$\cdot n^{\log_b a-\varepsilon}\le c\cdot n^{\log_b a}$ (def $\mathcal{O}$)

ii $T(k)$ (A)

$k=\frac{n}{b}<n\Rightarrow T(\frac{n}{b})\le c\cdot(\frac{n}{b})^{\log_b a}-d(\frac{n}{b})^{\log_b a-\varepsilon}$

$T(n)=aT(\frac{n}{b})+f(n)\overset{ii}{\le} a\cdot c\frac{n^{\log_b a}}{a}-ad\frac{n^{\log_b a-\varepsilon}}{\frac{b^{\log_b a}}{b^2}}+f(n)\le$

$\le cn^{\alpha}-(b^{\varepsilon}d-c_1)n^{\alpha-\varepsilon}\overset{?}{\le} cn^{\alpha}-d^{\alpha-\varepsilon}$

$\boxed{b^{\varepsilon} \cdot d - c_1 > d} \Rightarrow d(b^{\varepsilon}-1) \geq c_1 \Rightarrow \boxed{d = \frac{c_1}{b^{\varepsilon}-1}} \rightarrow$

$d > 0$

→ Proprietatea de adevăr pt $\forall\ n \in \mathbb{N}$

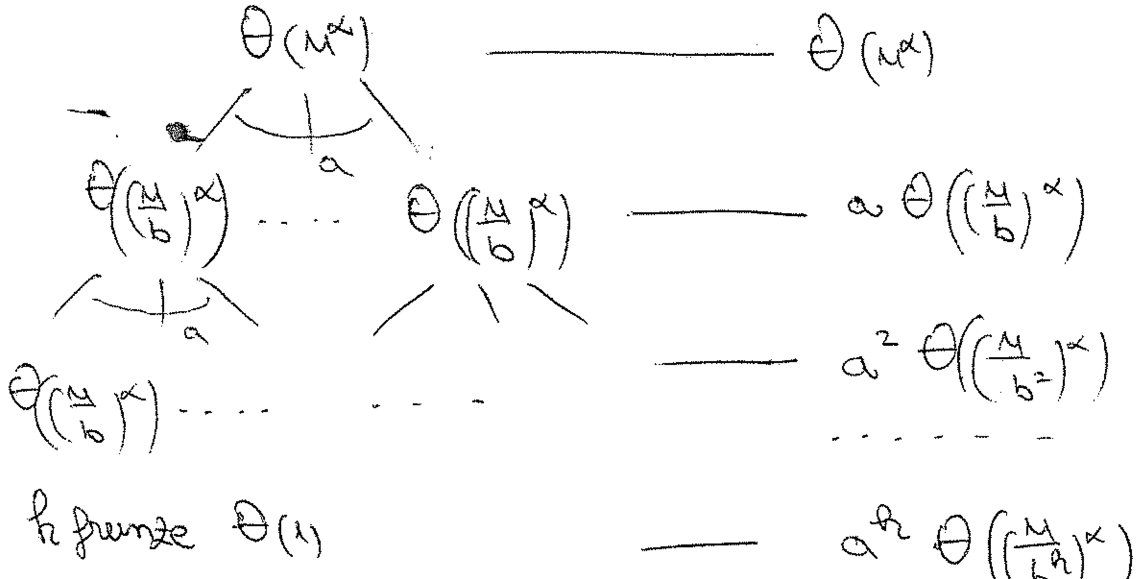
$T(n) \in \Theta(n^{\alpha})$

② Dacă $f(n) \in \Theta(n^{\alpha} \log^{k} n)$ $\Rightarrow T(n) \in \Theta(n^{\alpha} \log^{k+1} n)$

$k \geq 0$

Dem pt $k=0$ $f(n) = \Theta(n^{\alpha}) \Rightarrow T(n) = \Theta(n^{\alpha} \log n)$

Met arborelui de recurență


$\Theta(n^{\alpha})$
$\Theta(n^{\alpha})$
a
$\Theta\left(\left(\frac{n}{b}\right)^{\alpha}\right)$ .... $\Theta\left(\left(\frac{n}{b}\right)^{\alpha}\right)$
$a\,\Theta\left(\left(\frac{n}{b}\right)^{\alpha}\right)$
a
$\Theta\left(\left(\frac{n}{b}\right)^{\alpha}\right)$ ....
$a^2\,\Theta\left(\left(\frac{n}{b^2}\right)^{\alpha}\right)$
h frunze $\Theta(1)$
$a^h\,\Theta\left(\left(\frac{n}{b^h}\right)^{\alpha}\right)$

$\frac{n}{b^h} = 1 \Rightarrow \boxed{h = \log_b n}$

$$T(n) = \sum_{i=0}^{[\log_b n]} a^i\, \Theta\left(\left(\frac{n}{b^i}\right)^{\alpha}\right) + a^{\log_b n}$$

$$T(n) = \sum_{0}^{-i-} a^i\, \Theta\left(\frac{n^{\alpha}}{b^{\log_b a}}\right)^i + n^{\log_b a}$$

$$= \sum_{0}^{-i-} a^i \frac{1}{a^i} \Theta(n^{\alpha}) + n^{\alpha} = \Theta(n^{\alpha})(\log_b n + 1) + n^2$$

$$= \Theta(n^2)\left(\frac{\log n}{\log b} + 1\right) + n^{\alpha} \in \Theta(n^{\alpha} \log n)$$

```
Stivă <T>                              |S| = dim stivă
void push (T, M)                         Θ (1)
    this.vârf ++
    this v[ this.vârf ] = x

T pop ()
    if (this.vârf > 0)                       Θ (1)
        return this.v[ this.vârf-- ]

T [ ] multipop(int k)
    T [ ] result = new T(min (this.vârf, k))       Θ(min|S|, k)
    for (i = 1 ; i <= (min this.vârf) k) ; i++)
        result[i] = this.pop();
    return result;
```

Analiză amortizată

Studiază complexitatea în cazul cel mai defavorabil a unei secvențe de n operații efectuate asupra unei instanțe a unei structuri de date mai simplă sau mai complexă.

Dându-se o instanță a stivei de mai sus, după n operații oarecare de tipul push, pop, multipop; care e complx worst case a operației?

→ 3 metode de analiză amortizată

I Metoda agregării

Trebuie să luăm în consid orice secvență de n operații pt structura de date analizată și calculăm timpul mediu p operație

---

① push, pop ... , push, pop ... $\Rightarrow \boxed{n\,\Theta(1) = \Theta(n)}$

complx pe operație $\frac{\Theta(n)}{n} = \Theta(1)$

---

② $\underbrace{\text{push} \ldots \text{push}}_{n-1}, \underbrace{\text{multipop}}_{n-1} \Rightarrow \boxed{\Theta(n-1) + \Theta(n-1)}$

complx pe op $\frac{\Theta(2n-2)}{n} = \Theta(2) = \Theta(1)$

$\underbrace{\text{push}, \text{push} \ldots}_{p}, \underbrace{\text{multipop}}_{m-1}, \underbrace{\text{push} \ldots}_{q} \underbrace{\text{multipop}}_{m-1}$ $\quad p+q = m-2$

$p+q+2 = n \Rightarrow \Theta(p) + \Theta(p) + \Theta(q) + \Theta(q) = \Theta(2p+2q) =$
$= \Theta(2n-4)$

cmplx pe op $\frac{\Theta(2n-4)}{n} = \Theta(1)$

II Metoda creditelor

$(n-k)$ push ... k multipop... $\Rightarrow \Theta(2n-2k)$

pus mpop, push mpop... $\Rightarrow \Theta\left(2n - 2\frac{n}{2}\right)$

$\Theta(n)$

cmplx pe op (1)

Folosind met agregării am arătat că pt o secvență de n operații, costul mediu per op (cost amortizat) este mereu $\Theta(1)$

obs: Met agregării nu determină un cost amortizat pt o op specifică (de ex multipop)

II' Met creditelor (pușculiței, contabilului)

~~Asigurăm un cost amortizat pt fiecare op~~ ($\hat{c}$op)

(cost real c op)

	c	$\hat{c}$	$\Delta c = (\hat{c} - c)$
push	1	2	1
pop	1	0	−1
m pop	min (|S|, k)	0 $\in$ $\Theta(1)$	− min (|S|, k)

$\hat{c}$ op push = 2

$\hat{c}$ op pop = $\hat{c}$ mpop = 0

(obs): pt ∀ secv de n operații efectuate asupra structurii date

$$\sum_{i=1}^{n} \hat{c}_{op_i} \geq \sum_{i=1}^{n} c_{op_i}$$

$$\sum_{1}^{n} (\Delta c)_{op_i} \geq 0$$ ⇒ suma creditelor acumulate (pușculița)

III Metoda potențialului

Asigurăm SD o fct care trebuie să descrie starea internă a SD

$\phi(D_i)$ = starea SD după execuția operației i

$$\boxed{\hat{C}_{op} = C_{op} + \Delta\phi_{op}}$$

$$\hat{c}_{push} = c_{push} + \underbrace{\phi(D_{i+1}) - \phi(D_i)}_{(\Delta\phi)_{push}}$$

~~AA~~ .

curs 7

Misc

```
class contor
    int i = n/2
    int n
```

1) Fie un contor modificat cu analiză amortiz

$\hat{c} = 2$ $\Theta(2) = \Theta(1)$

```
increment ( )
    i++
    if (i == n+1)
        for (i > n/2)
            i --
```

fără an amortizată

$O\left(\frac{n}{2}\right) = O(n)$

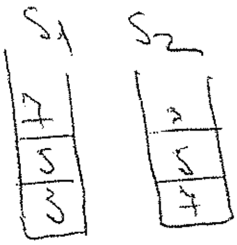
```
decrement
    i --
    if (i == 0)
        for (i < n/2)
            i ++
```

2) Fie o coadă impl cu 2 stive $O(|S|)$

```
add (x)
    S1.push(x)
```

$\Theta(1)$ $\hat{c}$ add = 4

$\hat{c}$ remove = 0



```
remove ( )
    if ( S2.size() == 0)
        while (!S1.~~pop~~ empty())
            S2 . push ( S1.pop())
    return S2.pop
```

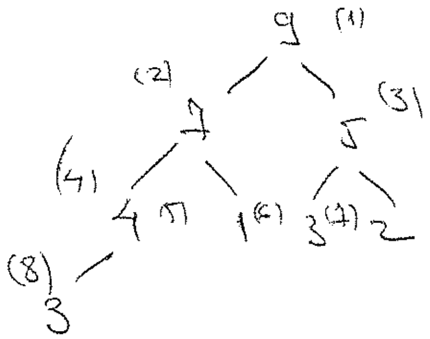
Heapuri binare

Un heap bin e o SD folosită când vrem să aflăm min sau max dintr-un vect în mod repetat

Aflare /eleminirare min/ max mai rapid ca $\Theta(n)$

→ Implementarea e un array în memorie

| 9 7 5 4 1 3 2 3 | ex de heap maximal



- aproape complet
- A[1] root
- left(i) = 2*i
- right(i) = 2*i +1
- părinte [i] = i/2

} indici

- satisface propr de heap

Prop de heap

⊖ heap maximal ∀ i=1:n A[i] <= A[parent(i)]

⊖ heap minimal ∀ i=1:n A[i] >= A[parent(i)]

$A(1) = \max\{A[i], i=1:n\}$

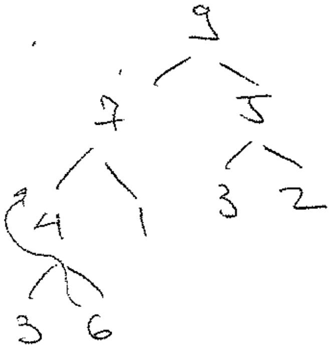
$A(1) = \min\{A[i]; i=1:n\}$

Operații cu heapuri binare

- construire heap
- inserare elem
- stergere (aflare) elem < max / min

Inserare elem

- heapul e vector A[1...n]
- A[n+1] = x => A mai e heap binar?
- În general nu (pt că s-a pierdut propr de heap)

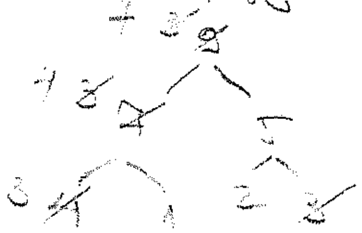


sift | heapify up
cernere în sus

Complexitate $\boxed{O(h) = O(\log n)}$

Ștergerea (aflarea) elem maxim

- Aflare max: return A[1] $\Theta(1)$ top(), peak()
  pop()
- Ștergere max
  - copiem A[n] în locul lui A[1]
  - A e heap corect? - Nu să păstrează prop de heap
  - refacem heapul prin cernere în jos / heapify down
  - n--



(A, 1, n)

```
HEAPIFY-DOWN(A, i, n)
    l = 2*i
    r = 2*i + 1
    if (l <= n && A(l) > A[i])
        max = l
    else
        max = i
    if (r <= n && A[r] > A[max])
        max = r
    if (max != i)
        A[i] <-> A[max]
        HEAPIFY-DOWN(A, max, n)
```

$O(h) = O(\log n)$

Construcție heap

Fiind dat un array $A[1 \ldots n]$, cum îl transf în heap?

1)
```
BUILD-HEAP-INEF (A[1...n])
    for (i = 2 : n)
        HEAPIFY-UP (A,i,i)
```

complex: $(n-1) \cdot$ HEAPIFY-UB $\Rightarrow O(\log n) = O(n \log n)$

Analiză amortizată:

pt $i = 2 \Rightarrow O(\log 2)$

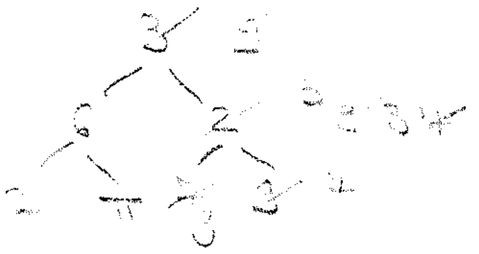
$i = j \Rightarrow O(\log j)$

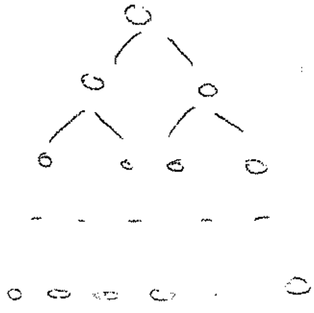
$$O\left(\sum_{i=2}^{n} \log i\right) > O\left(\sum_{n/2}^{n} \log i\right) \geq O\left(\sum_{n/2}^{n} \log n/2\right) =$$

$$= O\left(n/2 \log n/2\right) = O(n \log n)$$

2)
```
BUILD-HEAP-EFF (A[1...n])
    for (i = [n/2] : 1)
        HEAPIFY-DOWN (A,i,n)
```

→ arborii cu răd în $j > i$ sunt heapuri corecte





$\log(n+1) = h$

arbore complet cu $\boxed{n = 2^h - 1}$ noduri

h $\frac{n}{2}$ frunze ---- nr op swap → 0

$h-1$ $n/4$ frunze ---- → 1

---

k $n/2^{i-1}$ ---- → $i-1$

---

0 $\frac{n}{2^h} = \frac{n}{n} = 1$ ---- → $h-1$

① + ② ⟹ ③ $\boxed{\text{Terminare}}$: invariantul va fi true imediat după ce s-a ieșit din buclă

Ex:

```
Algoritm (A[1...n])
    BUILD-HEAP(A)   // max heap
    for (j = n:2)
        A[1] <-> A[j]
        HEAPIFY-DOWN(A,1,j-1)
```

I array int
O array out
$P_I(I) \overset{def}{=}$ true

Invarianți la ciclare

$P(j) \overset{def}{=}$ {A[j+1 ... n] sortat și conține cele mai ~~multe~~ mari n−j elem din vectorul A}

1. $P_{init}(j)$ = {A[n+1 ... n] sortat și conține ... 0 elem din A}

2. $\frac{P(j)}{P(j-1)}$ Avem un heap A[1...j] ⟹ A[1] = max{A[1...j]}
⟹ după swap : A[j] = max{A[1]...[j]} ⟹

⟹ A[j...n] conține cele mai mari n−j+1 elem din A
A[j+1 ... n] sortat și A[j] ≤ A[j+1 ... n] (heap) ⟹ A[j...n] sortat

3. P(j==1) = {A[2...n] e sortat și conține cele mai mari n−1 elem din A}

A[1] <= ∀ x ∈ A[2...n] ⟹ A[1...n] sortat

Ind matematică: $(i = \mathbb{N})$

$$\boxed{\dfrac{P(0) = \dfrac{i \in \mathbb{N} : P(i)}{P(i+1)}}{P(n) \ \forall \ n \in \mathbb{N}}}$$

Vrem să dem că

$P_i(i) \wedge$ Se termină $(Alg, i) \rightarrow \left(r == \prod_1^n k\right)$ $r = Alg(i)$
(true)

CB: $n == 0$ true $\wedge$ Se termină $(Alg, 0)$ $\rightarrow \left(\cancel{r == \prod_1^n k}\right)$ $Alg(0) =$ true
true
$r = Alg(1)$

$n == 1$ ... —ıı— .... —ıı— $Alg(1) = 1 = \prod_1^n k$

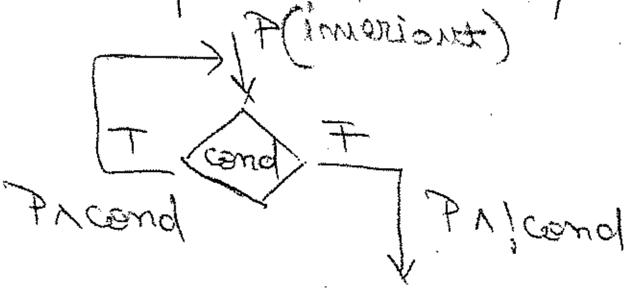
$\Rightarrow P_i$ $\dfrac{P(i)}{P(i+1)}$ Se termină $(Alg, i) \rightarrow (r = Alg(i)) == \prod_1^i k$

Se termină $(Alg, i+1)$ = Se term $(Alg, i) \overset{ii}{=}$ true

$r = Alg(i+1) = (i+1) * fact(i) = (i+1) * Alg(i) \overset{ii}{=} (i+1)\prod_1^i k =$
$= \prod_1^{i+1} k$ true

Invarianți la ciclare

- Propr adev în timpul exec unei bucle



P = imeriabil la ciclare
imeriant
(este ~~tot~~ timpul adev)

CB: $\boxed{\text{Inițializare}}$ ① - imeriantul (prop) treb să fie true înainte de a intra pt prima oară în buclă

Pi: $\boxed{\text{Menținerea}}$ ② - dacă prop e true la începutul exec unui ciclu (pas) al buclei, atunci ea trb să fie true la finalul pasului respectiv tablou dinamic

$\phi D_i = 2 * S(i) - C(i)$

Not $\boxed{S(i) = n = 2 * n - 2^{[\log n]}}$

$\hat{c}$ insert = c insert $-(\Delta\phi)$ insert

capacitatea

I avem locuri libere în tablou $(S(i) < C(i))$

$\hat{c}$ insert $= 1 + 2*(n+1) =$ capacity $- 2*n +$ capacity $= 3$

II nu mai avem locuri libere în tablou $(S(i) == C(i))$
înainte de insert

$\hat{c}$ insert $= 1 +$ capacity $+ 2*($capacity$+1) - 2*$capacity $-$
$- 2 *$capacity $+$ capacity $= 3$

$n \log m + 2^{\log m} \log m - \log m.$

$$T(m) = m + 1 + 2 + \dots + 2^{\log m} =$$
$$= m + \sum_{i=0}^{\log m} 2^i \leq m + \sum_{i=0}^{\log m} m + 2^{\log m + 1} - 1$$
$$= m + 2 \cdot \underline{2^{\log m}} - 1 = m + 2m - 1 = 3m - 1$$

$\sum_{i=0}^{\log m} m = m(\log m + 1)$

$\sum_{i=0}^{\log m} 2^{\log m + 1} = (\log m + 1)(2^{\log m} \cdot 2)$

$\sum_{i=0}^{\log m} 1 = \log m + 1$

$\log m + 1 \left(m - 1 + 2^{\log m + 1}\right) -$

$2^0 + 2$

17.11.2018

AA

Curs 8 Corectitudine

```
func [A(1..n)]
    m = - ~~INF~~ A[1]
    for (i = 1 : n)
        m = max (A[i], m)
    return m
```

→ invariații de ciclare $m = \max\{A[1\ldots i-1]\}$

Metode de obț a corectitudinii

① Inducția matematică (funcț care prelucrează nr naturale)

② Invariații de ciclare (loop invariant)

③ Inducție structurală

④ Inducție bine formată

Etape

1. Inițializare: $m = -INF = \max(A[1..0] = \max\{\emptyset\})$ (A)

2. Menținere

$P(i)$ $\quad m = \max A[1\ldots i-1]$

$P(i+1)$ $\quad m = \max\{A[i], m\}$

$= \max\{A[i], \max A[1\ldots i-1]\}$

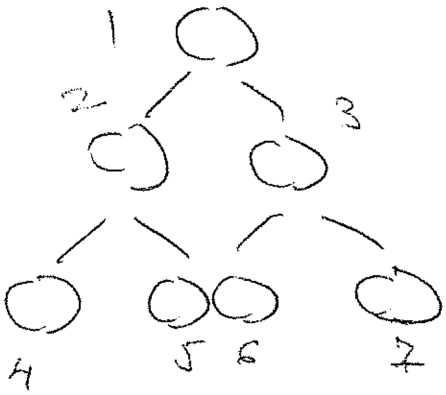
$= \max(A[1\ldots i]) \rightarrow P(i+1)$ (A)

3. Terminare $\quad m = \max A(1\ldots n+1-1) = \max\{A[1\ldots n]\}$

$i ==$

Ex

```
Build-Heap (A[1...n])
    for (j = ⌊n/2⌋ ... 1)
        Heapify-down (A, j, n)
```



Invariantul

~~Heapuri cu~~

Arborii cu rădăcina în elem $i = j+1 \dots n$ $\forall i = j = 1 : n$ sunt heapuri corecte

① Inițializare: Arborii cu rădăcina în elem $\left[\frac{n}{2}\right]+1 \dots n$ sunt heapuri corecte (conțin un singur elem) (și orice arbore cu un singur ~~heap~~ elem e corect)

② Menținerea $P(j) \longrightarrow P(j-1)$ (arătăm că e A)

↳ Arborii dominați de la $j+1 \dots n$ sunt heapuri

↳ Arborele dominat de elem j e heap?

$$\left.\begin{aligned} \rightarrow A[j] &\geq A[2 \cdot j] \\ &\geq [A\ \ 2 \cdot j + 1] \end{aligned}\right\} \text{Da, este un heap (copiii nuli)}$$

$$\begin{aligned} \hookrightarrow A[j] &< A(2j) \\ &< A[2j+1] \end{aligned}$$

Apelând heapify-down (A, j, n) elem $A[j]$ va fi cernut până la poz corectă și arborele dominat de $A[j]$ va fi un heap

③ Terminarea: Arborii dominați de $i = \begin{cases} 0+1 \dots n \\ 1 \dots n \end{cases}$ sunt heapuri corecte

↳ Arborele dominat de elem 1 va fi un heap

Misc

①

```
func (n)
    i = 0
    x = n
    s = 0.
    while (x > 0)
        x--
        s = s + x + i
        i++
    return s
```

0 : $s = 0$
1 : $s = n-1$
2 : $s = 2n-2$
$s = 3n-3$
i : $s = i(n-1)$

La pasul i
$(s == (n-1) * i)$
AND
$(i + x == n)$

Terminare: $i = n \Rightarrow s = n * (n-1)$

---

②

```
func (#R, #B)
    dându-se o urnă cu #R bile și
    #B bile albastre
    while (#B + #R > 1)   cât timp bilele din urnă > 1
        extragem aleator 2 bile din urnă
        dacă (bilele au aceeași culoare)
            le aruncăm și adăugăm o bilă roșie
        altfel
            aruncăm bila roșie și punem înapoi
            bila albastră
    (#b = 0, #r = 1) sau (#b = 1, #r = 0)
```

Ce culoare are bila rămasă în urnă

Invariant:

nr de bile albastre își păstrează paritatea

(#r, #b) → nr de bile la un pas oarecare

$\#b \% 2 = \#B \% 2$

#b → BB → #b − 2
par → RR → #b
sau imp → RB → #b

par sau impar

Inducție structurală

Unele tipuri de date pot fi definite prin recursivitate structurală și printre acestea putem aplica inducția structurală.

$\boxed{\text{Nat}}$ → tipuri de date
echivalent d.p.v.
al puterii de
repr cu nr naturale

constr pt Nat
| zero() : → Nat → (Σ_0)
| succesor(x) Nat → Nat   repr simbolică
| Succesor (zero())
| Suc(Succesor(succesor(succesor (zero)))) → 4

List<T>
constr
| empty() → list<T>   ([] Σ_0)
| simple(a).T → list<T>   ({a] Σ_e)
| cons(a,x).T × List<T> → List<T>   (a : x Σ_f)

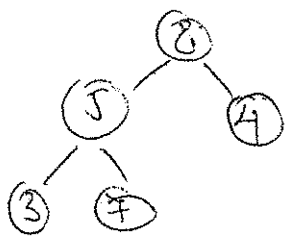
Cons(7, cons(3 , Cons(6, empty ())))   [7, 3, 6]

7 : 3 : 6 : [ ]

Tree <T> $\overset{\Sigma_0}{\rightarrow}$ arbori binari ale căror noduri interne au tot timpul 2 copii

constr
| leaf (a).T → Tree <T> Σ_e
| node (t_1, a, t_2), Tree <T> × T × Tree <T> Σ_f

node(node (leaf (3), 5, leaf (7), 8, leaf(4))



· make left (t_1, a)
make right (a, t_2)
make both (t_1, a, t_2)

Definim operații (funcții) pt aceste tipuri de date abstracte

---

Ex)

Nat : $\mathrm{sum}(n_1, n_2) : Nat \times Nat \to Nat$

$n_1 == \mathrm{zero}() \Rightarrow n_2$

$n_1 == \mathrm{succ}(n_3) \Rightarrow \mathrm{succ}(\mathrm{sum}(n_3, n_2))$

$\mathrm{sum}(\mathrm{succ}(\overset{n_3}{\mathrm{succ}(\mathrm{zero}())}), n_2) \quad \forall n_2 \in Nat$

$\overset{s_2}{=} \mathrm{succ}(\mathrm{sum}(\mathrm{succ}(\overset{n_3}{\mathrm{zero}()}), n_2))$

$\overset{s_2}{=} \mathrm{succ}(\mathrm{succ}(\mathrm{sum}(\mathrm{zero}()), n_2)) \overset{s_1}{=} \mathrm{succ}(\mathrm{succ}(n_2))$

Ex 2

Tree <T>

$\mathrm{mirror}(t) . \mathrm{Tree} <T> \to \mathrm{Tree} <T>$

m_1 if empty ret empty

$m_2 \; t = \mathrm{leaf}(a) \Rightarrow \mathrm{leaf}(a)$

$m_3 \; A = \mathrm{node}(t_1, a, t_2) \Rightarrow \mathrm{node}(\mathrm{mirror}(t_2), a, \mathrm{mirror}(t_1))$

$P(t) \overset{def}{=} (\mathrm{mirror}(\mathrm{mirror}(t)) == t) \quad \forall t \in \mathrm{Tree} <t>$

(*) Folosim ind structurală pt a dem propr legate de tipuri de date abstracte def prin recursivitate structurală

t = tip de date abstract

Σ → mulțimea constructorilor tipului t

operații/funcții pt tipul t (sum() / mirror())

Separăm constructorii în 3 cls diferite

① Σ_0 -> constr nulari

$\tau \in \Sigma_0 .\ \tau \to t$ (nu are domeniu, dar construiește)

② Σ_e -> constr extern

$\tau \in \Sigma_e .\ \tau\ Dom\tau \to t$

(t nu aparține $Dom\tau$)

③ Σ_t -> constr interni

$\tau \in \Sigma_{(t)} .\ \tau\ Dom\tau \to t$

(t aparține $Dom\tau$)

Schema/tiparul ind structurale

Caz de bază – t

$\forall \tau \in \Sigma_0 .\ P(\boxed{\tau c})$

$\forall \tau \in \Sigma_e\ \ x \in Dom\tau .\ P(\boxed{\tau(x)})$ $\forall x \in t$

(x nu e din tipul t)

Pas de inducție

$P(x)$ adev => Arăt $P(\tau(\dots x \dots))$ adev pt $\forall \tau \in \Sigma_t$

$$\frac{\text{PP } P(x_1),\ P(x_2) \text{ adev}}{P(\tau(\dots x_1 \dots x_2 \dots))}$$

$P(x)\ \forall x \in t$

M = mirror

Caz bază

Σ_0 $t = empty() \Rightarrow$ $M(M(e())) \overset{m_1}{=} M(empty()) \overset{m_1}{=} empty()$ (A)

Σ_e $t = leaf(a)$ ~~t~~ $\Rightarrow$ $M(M(leaf(a))) \overset{m_2}{=} M(leaf(a)) \overset{m_2}{=} leaf(a)$ (A)

Pi/ii: P_P $P(t_1)$ (A) $t_1 \in tree<t>$

$P(t_2)$ (A) $t_2 \in$ — " —

$t = node(t_1, a, t_2)$ $P(t) = ?$

$M(M(node(t_1, a, t_2))) \overset{m_3}{=} M(node(\overbrace{M(t_2)}^{t_1'}, a, \overbrace{M(t_1)}^{t_2'})) \overset{m_3}{=}$

$= node(M(M(t_1)), a, M(M(t_2))) \overset{ip}{=} node(t_1, a, t_2)$ (A)

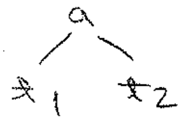
$$\boxed{\begin{aligned} ip = M(M(t_1)) &= t_1 \\ M(M(t_2)) &= t_2 \end{aligned}}$$

AA
Curs 9

Inducție structurală

Ex Tree <T>

Constructori
$empty() \to Tree<T>$
$leaf(a) \to Tree<T>$
$node(t_1, a, t_2) :: Tree<T> Tree<T> \to Tree<T>$



List <T>

$[\,] : \to List<T>$
$[a] : T \to List<T>$
$a : x : T \times List<T> \to List<T$

operații

$memberT(a,t)$
$T \times Tree<T> \to boolean$

mt_1 $t == empty() \Rightarrow false$
mt_2 $t = leaf(b) \Rightarrow (a == b)$
mt_3 $t = node(t_1, b, t_2) \Rightarrow$
$\Rightarrow (a == b) \,||\, memberT(a, t_1) \,||\, memberT(a, t_2)$

$member(a, x)$ $T \times List<T> \to boolean$

m_1 $x == [\,] \Rightarrow false$
m_2 $x == [b] \Rightarrow a == b$
m_3 $x == b : x_s \Rightarrow (a == b) \,||\, member(a, x_s)$

$flathen(t) : Tree<T> \to List<T>$

F_1 $t = empty \Rightarrow [\,]$
F_2 $t == leaf(a) \Rightarrow [a]$
F_3 $t == node(t_1, a, t_2) \Rightarrow$
$\Rightarrow append(flathen(t_1), a : flathen(t_2))$

$append(l_1, l_2) :$
$List<T> \times List<T> \to List<T>$

$A_1 : l_1 == [\,] \Rightarrow l_2$
$A_2 : l_1 == [a] \Rightarrow a : l_2$
$A_3 : l_1 == a : x_s \Rightarrow$
$a : append(x_s, l_2$

$P(t) \overset{def}{=} (\text{member } T(a,t)) \to \text{member } (e, \text{flathen } (t)))$

$\forall\ t \in \text{Tree} \langle T \rangle$

$A \to B \Rightarrow \neg A \vee B$

CB (cond de bază)

1) $t == \text{empty} ( )$

$\text{memberT} (e, \text{empty} ( )) = \text{false} \Rightarrow P(t)$ adevărat

2) $t == \text{leaf} (a) , a \in T$

$\text{member } T(e, \text{leaf}(a)) \overset{M2}{\Rightarrow} (e == a)$

$\text{member } (e, \text{flathen } (\text{leaf} a)) \overset{F2}{=} \text{member } (e, [a]) \overset{M2}{=} (e == a)$

$(e == a) \to (e == a)$ adevărat

Pi (pas de inducție)

$A = \text{node} (t_1, a, t_2)$

ii

$P(t_1) = \text{member } T(e, t_1) = \text{member } (e, \text{flathen } (t_1))$

$P(t_2) =$ — " — t_2 — " — t_2

$A = \text{node} (t_1, a, t_2)$

$\text{member } T (e, \text{node} (t_1, a, t_2)) \overset{MT3}{=} (e == a) \ || \ \text{member } T(e, t_1) \ || \ \text{member } T(e, t_2)$

ii (t_1) ii (t_2)

$\text{member } (e, \text{flathen } (\text{node } (t_1, a, t_2))) \overset{F3}{=} \text{member } (e, \text{append } (\text{flathen}(t_1)), a : \text{flathen}(t_2))$

$P1(x) = M(e, A(x_1, x_2)) = M(e, x_1) \ || \ M(e, x_2) \ \forall \ x_1 \in \text{List}\langle T \rangle$

$\overset{P1}{=} \text{member } (e, \text{flathen}(t_1)) \ || \ \text{member } (e, a : \text{flathen}(t_2)) \overset{M3}{=}$

member $(e, \text{flatten}(t_1)) \,||\, (a == e) \,||$ member $(e, \text{flatten}(t_2))$

$(e == a) \Rightarrow$ true

$(e \,!= a) \;\&\& \; MT(e, t_1) \Rightarrow$ OK

$(e \,!= a) \;\&\& \; !MT(e, t_1) \;\&\& \; MT(e, t_2) \Rightarrow$ OK

$\Rightarrow P(t)$ adevărat

Dem P_1

CB $x_1 == [\,] \Rightarrow M(e, A([\,], x_2)) \overset{M_1}{=} M(e, x_2) \rightarrow$ membru stâng MS

$M(e, [\,]) \,||\, M(e, x) \overset{M_1}{=}$ false $||\, M(e, x_2) == M(e, x_2) \rightarrow$ membru drept MD

$x == [a]$: $MS = M(e, A\,[a], x_2) \overset{A_2}{=} M(e, a : x_2) \overset{M_3}{=} (e == a) \,||\, M(e, x_2)$

$MD = M(e, [a] \,||\, M(e, x_2)) \overset{M_2}{=} (e == a) \,||\, M(e, x_2)$

MS → MD adevărat

Pi

ii $P_1(x_1)$ adevărat $\forall \, P_1(a, x_1)$

$MS = M(e, A(a : x_1, x_2)) \overset{A_3}{=} M(e, a : A(x_1, x_2)) \overset{M_3}{=} (e == a) \,|\, M(e, A(x_1, x_2))$

$MD = M(e, a : x_1) \,||\, M(e, x_2) \overset{M_3}{=} (e == a) \,||\, M(e, x_1) \,||\, M(e, x_2$

$(e == a) \Rightarrow MS(T) \rightarrow MD(T)$

$(e \,!= a) \;\&\& \; M(e, A(x_1, x_2)) \xrightarrow{ii} M(e, x_1) \,||\, M(e, x_2)$ adev

$\Rightarrow P_1(x_1)$ adev $\forall \, x \in List\langle T \rangle$

Inducție bine formată

Este o generalizare a schemelor de inducție discutate până acum (matematică, completă, structurală)

ÎBF funcționează peste seturi, mulțimi de elemente care acceptă o relație de ordine bine formată.

A mulțime (de ex mulțimea tuturor elem obținute pt un tip de date folosind recursivitate structurală) $\not\mathbb{R}$ $\mathbb{N}$

α ("mai mic rond" $< \alpha\alpha$)

↳ relația de ordine peste ~~ordine~~ elem din A

$\alpha : A \times A \to$ boolean

↳ parțială sau ~~totală~~ (definită pt $\forall$ $x, y \in A \times A$)

↳ def pe un subset de elem din $A \times A$

Exemple $(<, A)$ ~~totală~~

1) $A = \mathbb{N}$ $\alpha = <$

2) $\{f : \mathbb{N} \to \mathbb{R}\}$, $\alpha = \Theta$ parțială

(A, α) se numește bine formată dacă $\forall$ $x \in A$

$\nexists$ niciun șir infinit (ne)mărginit la stânga folosind relația de ordine $<\alpha$

$$\exists\ \underbrace{x_0 \alpha x_1 \ldots \alpha x_{n-2} \alpha x_{n-1} \alpha x = x_n}_{\text{finit}}$$

1°) $(A, <)$ e bine formată

$\exists\ 0 < 1 < . < n-2 < n-1 < n$

2°) $A = \mathbb{Z}$, $\alpha = <$ nu e bine formată

$\ldots < -1 < 0 < \ldots < n-1 < n$

$z_1 \alpha z_2$

$|z_1| < |z_2|$

$\Rightarrow \nexists\ 0 < -2 < 3 < -4 < \ldots 4$

3°) $A = \{f : \mathbb{N} \to \mathbb{R}_+\}$ $\propto = \emptyset$ nu e bine format

4°) $A = [0, \infty)$, $\propto = <$ nu e bine format

$4,99\ldots8 < 4,99\ldots9 < 5$ (şir infinit)

5°) $A = [0,\infty) \cap \mathbb{Z}$, $\propto = <$ e bine format

6°) $A = \Sigma^*$, $\propto \Rightarrow$ ordine lexicografică Σ = alfabetul lb engleze (a-z)

mulţimea tuturor cuv formate cu alfabetul Σ

$\propto a \ldots ab \ldots \propto aaab \propto aab \propto ab < b$ şir infinit

$\propto^? : A \times A \to$ Bool

$S_1 \propto S_2$ $|S_1| < |S_2|$

$\propto (S_1, S_2)$

$\propto$ abc~~d~~ede $\propto$ abceder

Schema inducţiei bine formate

Fie $(A, \propto)$ bine formată

Vrem să dem că $P(x)\ \forall\ x \in A$

$$\frac{\text{CB}\quad \begin{matrix} P(x_0) \\ x_0 \propto \end{matrix} \qquad \text{ii}\ \dfrac{P(y)\ \forall\ y \propto x}{P(x)}}{P(\propto)\ \forall\ x \in A}$$

Particularizări

1°) $A = \mathbb{N}$ $\propto = <$ inducţie completă

2°) $A = \mathbb{N}$ $x \propto y \Leftrightarrow y = x + 1$ inducţie matematică

$y = \operatorname{succ}(x)$

3°) $A = t$

t = tipul de date def prin recursivitate structurală

$x \propto y \Leftrightarrow y\,T(\dots,x,\dots)$ $\exists\, T \in \Sigma_{+}$ inducție structurală

Nedeterminism

Începem cu o paranteză

În practică nedeterminism = problemă, algoritm nu are un comportament univoc pe ~~aceleași~~ aceeași dată de intrare

Noi discutăm despre MT (mașina Turing) nedeterministă și de algoritmi pt această mașină

MT → deterministă: la fiecare moment de timp se execută o singură instrucțiune din program

$f: \text{Timp} \to \text{Instrucțiune}$

$t_0 \to i_0$
$t_1 \to i_1$
$t_n \to i_n$

MT → nedeterministă: la fiecare moment de timp se pot executa mai multe instrucțiuni diferite

$f: \text{Timp} \times \text{Instruct} \to \text{Boolean}$ $(t_1 \dots i_1)$
$(t_2 \dots i_2)$

Pt mașinile nedeterministe programul / algoritmul va avea mai multe copii care se execută în paralel, independent una de cealaltă

Introducem 3 noi instrucțiuni pt algoritmii nedeterminiști

CHOICE(A) → copiază |A| copii ale algoritmului (sau ale copiei curente)
A mulțime finită
→ aceste copii se vor executa în paralel în continuare

→ instrucțiunea întoarce elem A[i] fiecărei copii i $1 \leq i \leq |A|$

→ fiecare copie moștenește spațiul de memorie al copiei părinte

Fiecare copie e ~~[illegible]~~ independentă de celelalte !

→ propriul spațiu de memorie

→ flux de execuție și control propriu

FAIL → oprește copia curentă cu insucces (nu a ajuns la solu

SUCCES → —"— succes (a ajuns la soluția programului)

Un alg nedeterminist se oprește:

1) Toate copiile întorc FAIL ⇒ false (pt pr de decizie)

2) O copie întoarce SUCCES ⇒ true

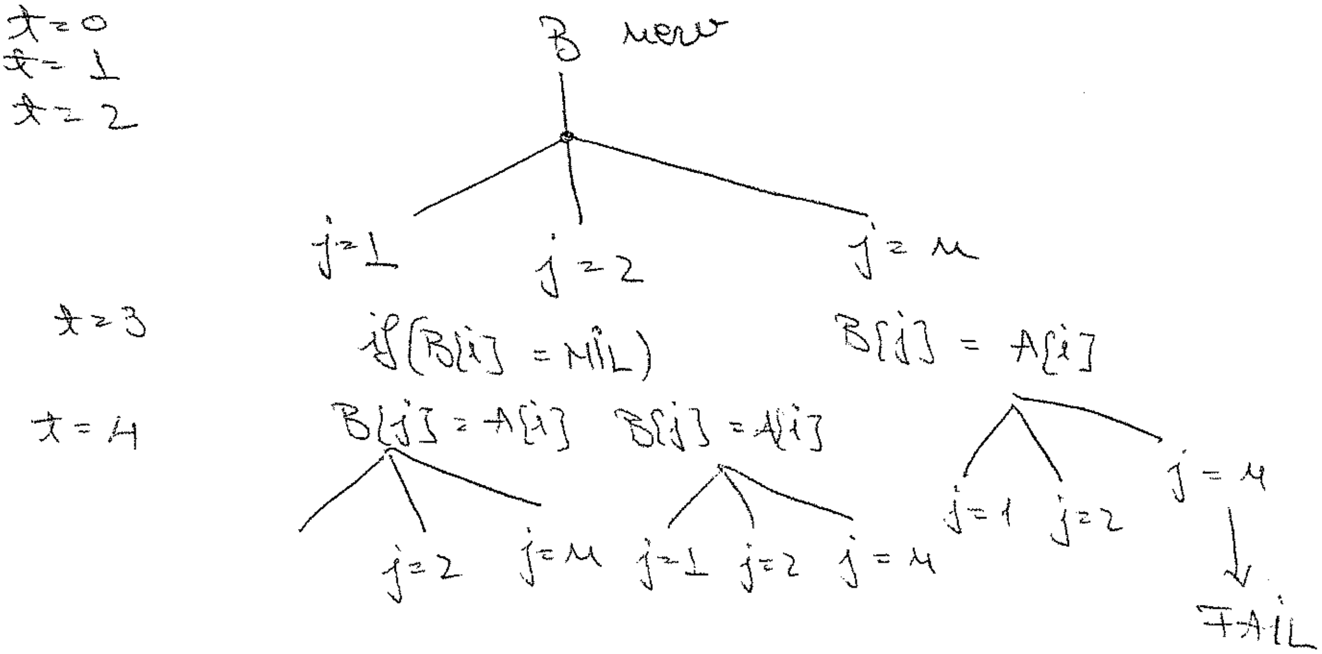
Complexitatea unui alg nedeterminist e dată de <u>complexitatea secvenței de instrucțiuni pe calea cea mai rapidă către un apel de succes (caz cel mai defavorabil)</u>

→ complexitate angelică

```
N_SORT (A[1...n])
  B = new array[n] {null, null ..., null}
  for (i = 1 ... n)
      j = CHOICE {1 ... n}
      if (B[j] != NIL)
          FAIL
      B[j] = A[i]
  for (i = 1 ... n-1)
      if B[i] > B[i+1]
          FAIL
  SUCCES  → permutările sortate
```

n! copii
câte una pt fiecare
permutare a vectorului

Θ(n) → complx algebrică



Un alg nedeterminist are 2 etape importante –

1) generează ~~toate~~ ~~soluțiile~~ candidat – câte o soluție pt fiecare copie

(CHOICE)

2) ~~testează~~ dacă fiecare sol candidat e exactă pt problema ~~necesară~~ noastră

(FAIL, SUCCES)

→ deterministă

~~AF~~ 8. 12. 2017

Curs 11

Clase de probleme

P = { problemelor care acceptă o rezolvare deterministă și polinomială }

SAU

{ Q · : i → {0, 1} | ∃ Alg $\overset{\text{polinomial}}{\text{care}}$ rezolvă Q }

$O(n^k)$ cu k = cst, n = dim datelor de intrare pt Q

NP = { Q · i → {0,1} | ∃ Alg nedet + polinom care rezolvă Q }

Def alternativă pt clasa NP

NP = { Q · i → {0,1} | ∃ un Alg det + polin care verifică dacă o soluție candidat pt ~~un alg~~ o probl Q e corectă }

Exerciții alg nedeterministe}

① Problema k - clică

Având un graf neorientat (V, E). Dacă există un subset de vârfuri $\underset{\text{(clică)}}{V'} \subseteq V$, |V'| == k a.î. V' subset?

$\forall$ u, v ∈ V', u ≠ v (u, v) ∈ E → Probl de optim coresp unei k clici este clica maximală a unui graf

```
CLICA (G (V,E))
V' = ∅
for (i = 1, k)
    u = choiche(V)
    if (u ∈ V')
        fail
    V' = V' ∪ {u}
```

→ generează subgrafuri cu k vârfuri candidate să fie clică testează dacă un subgraf candidate e clică:

```
foreach (u = V')
    foreach (v = V')
        if (u != v && (u,v) ∈ ∅)
            fail
success
```

Cmplx $\Theta(K + K^2) \Rightarrow \Theta(K^2)$

generare testarea

② Problema Q-sume (Subset sum)

Având S un set de numere reale și Q - un nr oarecare $\exists\ S' \subseteq S$ subset al lui S cu $\sum_{s \in S'} = Q$?

```
N-Sume (S, Q)
    if (Q == 0)
        Success                 // Σ ~~{}~~ ∅ == 0 == Q
    sum = 0                     // S' = ∅
    while (S ≠ ∅)
        x = choice (S)
        S = S \ {x}
        sum += x                // S' = S' ∪ {x}
        if (sum == Q)
            Success
    fail                        // S' = S ∪ {x}
                                // suma = 0
                                   for (s = S')  sum =.s
```

$\Theta(|S|)$

(*) și în anumite situații, generarea tuturor soluții lor candidat nu este foarte bine delimitată de testarea corectitudinii unei soluții

③ Fie $\{S_i\}_{1 \le i \le n}$ n - stringuri construite folosind un alfabet Σ (finit). ∃ un substring (subsequence) de dim k care să fie conținut de $\forall S_i, 1 \le i \le n$?

a~~A~~$\boxed{l}$g$\boxed{o}$ri$\boxed{t}$$\boxed{m}$ → l o t m

~~Problema~~

Problema de optim; care este substringul maximal pt n șiruri de caractere date

Sol;

```
N_Subsequence ( {S_i}, n, k, Σ )
    SS = [ ]        // new array[k]          $O(k + n(k + |S_i|))$
    for (i = 1, k)
        SS[i] = Choice(Σ)
    // verificare ~~deterministă~~ SS apare în toate cele n
       stringuri
```

Clase de probleme

P, NP, ? NPC, NP_hard

Reducerea polinomială $(\le_P)$

Fie 2 probl de decizie P, Q. Spunem că P se reduce polinomial la Q și notăm $P \le Q$ dacă

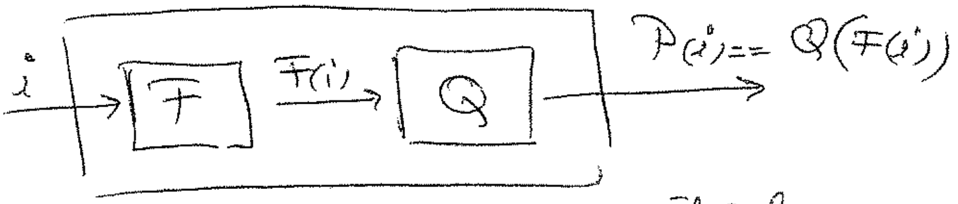
a) ∃ un Alg determinist + pol $F: i_P \to i_Q$

$P \quad i_P \to \{0,1\}$

$Q : i_Q \to \{0,1\}$

b) ~~Pt~~ $\forall i_P \quad P(i) = 1 \Leftrightarrow Q(F(i)) == 1$ $\left(\begin{array}{l} P(i) = 1 \to Q(F(i)) = \\ \quad = 0 \to \text{-"-} = \end{array} \right.$

3



Dacă $P \le_p Q \Rightarrow \exists$ următorul Alg pt probl P: alg_P_rp(i)

$i_2 = F(i)$

return alg_Q(i_2)

↓

putem folosi ∀ alg care rezolvă probl Q

Proprietăți:

① Reflexivitatea $P \le_p P$

$\Rightarrow F(i) = i$

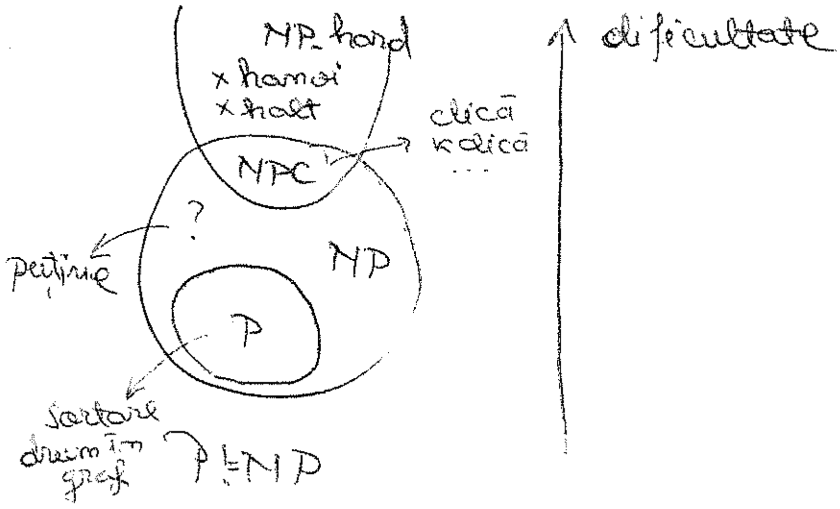
② Tranzitivitate $P \le_p Q,\ Q \le_p R \Rightarrow P \le_p R$

- $Q \in$ NP_hard (NP_dificile, NP_greu)
  dacă $(\forall)\ Q' \in NP \quad Q' \le_p Q$

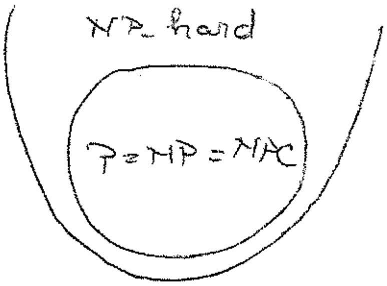
- $Q \in$ NP_complete (NPC)
  dacă i) $Q \in NP$
  ii) $Q \in$ NP_hard

$P \subseteq NP$

$NP \overset{?}{\supseteq} P \not\Rightarrow P \subsetneq NP$



$P = NP \Rightarrow$



$P = PTIME$
$NP = NPTIME$

Proprietate:
Pt NPC, $\leq_p$ este și simetrică

$Q \in NP$
$Q' \in NP\_hard$

$(\forall)\ Q, Q' \in NPC$

$$\underbrace{Q \leq_p Q'}_{\substack{Q \in NP \\ Q' \in NP\_hard}} \Leftrightarrow \underbrace{Q' \leq_p Q}_{\substack{Q \in NP\_hard \\ Q' \in NP}}$$

$\hookrightarrow (\leq_p, \underset{\boxed{P?}}{NPc}) \rightarrow$ relație de echivalență (simetrie + reflex + tranzitivitate)

Pp $P != NP$ și $Q \leq Q'$

a) $Q \in NP \setminus P \Rightarrow Q' \in NP \setminus P$ sau mai grea, sau $Q' \notin NP$
$(Q \notin P)$

b) $Q' \in P \Rightarrow Q \in P$

Cum demonstrăm că o problemă nouă Q e într-o anumită clasă?

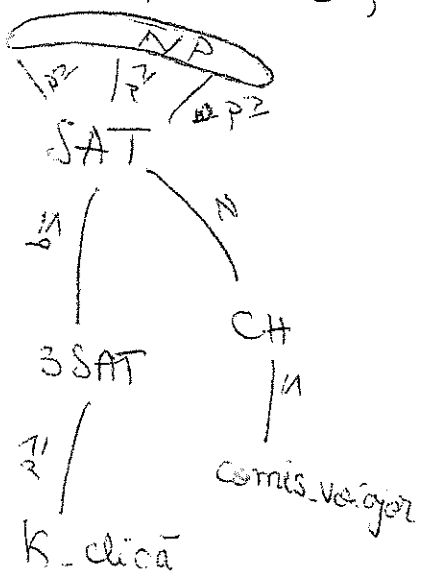
1) $Q \in P$? $\rightarrow$ a) Construim un alg det + pol care rezolvă Q
$\rightarrow$ b) $(\exists)\ Q' \in P$ a.î. $Q \leq_p Q'$

2) $Q \in NP$ $\rightarrow$ construim un alg nedeterminist + pol care rezolvă Q
$\Downarrow$
– " – det + pol care verifică Q

3) $Q \in NPC$ $\rightarrow$ 1 $Q \in NP$
și
$\rightarrow$ 2 $\forall Q' \in NP$, $Q' \leq_p Q$ (ii)

sau

ii) $\exists\, Q' \in NPC,\ Q' \leq_p Q$ | $Q \in NP\_hard$



Problema SATIASFIABILITĂȚII (SAT)

Fie-$F_{FNC}(x_1 \dots x_n)$ o formulă logică booleană cu n variabile în forma normală conjunctivă (FNC). ∃ o asignare a variabilelor la valori boolene a.î formula cu acele variabile să fie true/1?

$\exists\ x_1 \dots x_n \in \{0,1\}$ a.î $F_{FNC} = 1$?

$F_{FNC}(x_1 x_2 x_3) = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$

Th Cook

$\boxed{SAT \in P \Leftrightarrow P == NP \quad (SAT \in NPC)}$

$\forall\, Q' \in NPC,\ Q' \in P \Leftrightarrow P == NP$

$\hookrightarrow \left.\begin{array}{l} \forall\, Q'' \in NP,\ Q'' \leq Q' \\ Q' \in P \end{array}\right| \Rightarrow Q'' \in P \Rightarrow P == NP$

a) SAT ∈ NP

```
N_SAT ( F FNC(x_1 ... x_n) )
    for (i = 1 ... n)
        x_i = choice({0,1})
    foreach (T_j ∈ F FNC)
        termen_true = false
        for i = 1..n
            if (x_1 == 1 AND x_1 ∈ T_j) OR
               (x_1 == 0 AND x̄_i ∈ T_j)
                termen_true = true
                break;
        if (!termen_true)
            FAIL
    SUCCESS
```

→ 2^n generat toate interpretările posibile

$F_{FNC}(x_1 \ldots x_n) = \bigwedge_{j=1}^{m} T_j(x_1 \ldots$

$m \in O(n^k)$ k cst

→ conjuncție de disjuncții

$\Theta(n + m \cdot n)$ polinomial

① K-SAT → FNC, nr literarilor per termen == K
<= K

2-SAT: $(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_3)$

k == 2 => 2-SAT ∈ P

3-SAT NPC

⋮

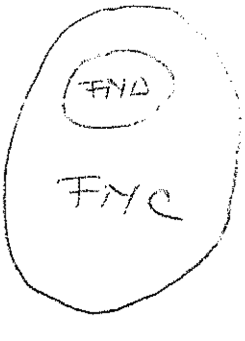
$A \vee B : \bar{A} \to B$

② $F_{FND}(x_1 \ldots x_n) = \bigvee_{j=1}^{m} F_j(x_1 \ldots x_n)$

F_j = un factor = $\bar{x}_3 \wedge \bar{x}_4 \wedge x_5 \wedge \ldots$

$(\exists)\ x_k \wedge \ldots \wedge \bar{x}_k$

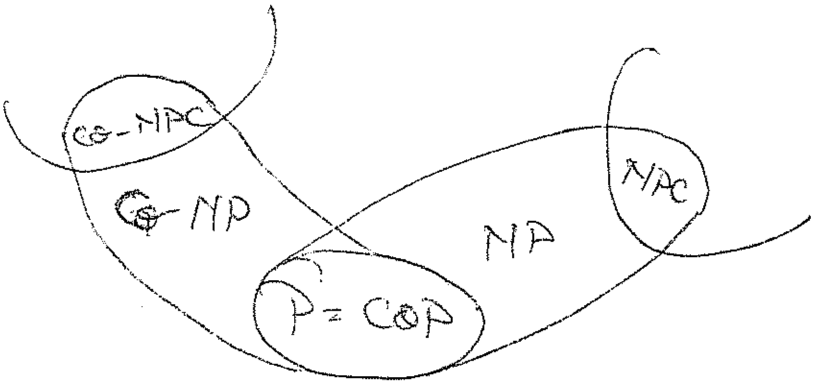
→ det + pol $\Theta(\underset{\text{factori}}{m} \cdot \underset{\text{variabile}}{n})$



③ $Q \in P$

~~ℰ~~ ~~ℰ~~ $\mathcal{E}(Q) \in P$

$$\left.\begin{array}{l} Q \in NPC \\ \mathcal{E}(Q) \notin P \end{array}\right\} Co\_NPC$$



AA

Curs 12

Clase de probleme

Reduceri polinomiale

$\forall\, Q \in NP \leq_p SAT$ (Cook)

$$\begin{tikzcd}
\boxed{SAT} \\
\boxed{3SAT} \arrow[u] \\
\boxed{K\text{-cover}} \arrow[u] \\
\boxed{SubsetSum} \arrow[u]
\end{tikzcd}$$

K cover $\leq_p$ subsetsum

SubsetSum $\in$ NP hard

K cover

$G(V,E)$ neorientat

$k \in \mathbb{N}^*$

$\exists\, V' \subseteq V, |V'| == k$ a.î. $\forall\, (u,v) \in E$

$u \in V'$ sau/și V'

Subset Sum

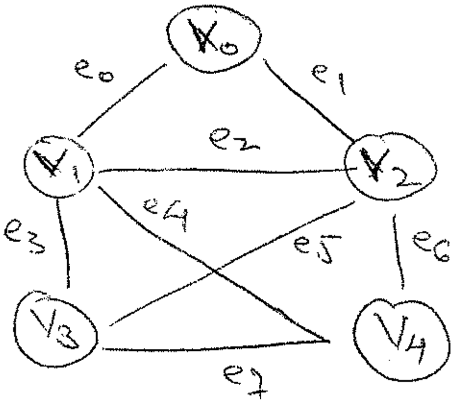
A - set de numere (nenule) și $s \in \mathbb{R}$

$(\exists)\ A' \subseteq A$ a.î. $\sum_{x \in A'} x == s$?

K cover $\leq_p$ subsetsum

1) $\exists\ F \quad i_{K cover} \to i_{subsetsum}$

$\forall\, (G,k) \longrightarrow (A,s)$



Pornim de la matricea de incidență a grafului G (B)

B ∉ {0,1} și

$$B_{i,j} = \begin{cases} 1 & \text{dacă muchia ei e adiacentă} \\ 0 & \text{altfel} \end{cases}$$

	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	→ $\vec{e}$
1	0	0	0	0	0	0	1	1	V_0
1	0	0	0	1	1	1	0	1	V_1
1	0	1	1	0	0	1	1	0	V_2
			1	0	1	0	0	0	V_3
1	1	0							V_4
1	1	1	0	1	0	0	0	0	V_5

Mulțimea A va conține |V| numere în B și |E| numere

0 0 1

0 0 . . . . . 1 0 $\quad i=(0 .\ \ n-1)$

. . . . $\quad g_i = 4^i$

$j=(0 \ldots n-1)$

0 1 0 0

În baza$_4$ modificată (cifra cea mai semnif poate fi modificată cifra Cn

cifrele de la C_0 . C_{n-1} vor fi în baza$_4$, dar C_n în oricare

$$\sum_{x\in A} x = n \underbrace{3 \dots 3}_{m \text{ ori}}$$

$$\boxed{x_i = 4^m + \sum_{k=0}^{m-1} [B(i,k)]\, 4^k \quad i=0\dots m-1}$$

$$q = k\ 2 \dots 2$$

Pas 2) Arătăm că $\forall (G,k) \in i_{kcover}$ a.î. $kcover(G,k) == 1$ $\Leftrightarrow$ SubsetSum $F(G,k) == 1$

Caz ⓐ

"⇒" Știm $kcover(G,k) == 1 \Rightarrow \exists$ o k acoperire $V' \subseteq V$ pt G

$V' = \{v_{i_1}, v_{i_2} \dots v_{i_k}\}$? $\exists A' \subseteq A$ a.î. $\sum_{x\in A'} x == q$

Alegem acele numere $x_{i_1} \dots x_{i_k}$ din A coresp vf din k acoperire V'

$$x_{i_1} + \dots + x_{i_k} = k \underbrace{\{1,2\}\{1,2\} \dots \{1,2\}}_{}$$

n cifre (de ce? pt că vârfurile din k acoperire acoperă cel puțin un capăt al fiecărei muchii, cu noroc, le acoperă pe amândouă

↳ Dacă ale $\overset{3 \text{ acoperire pt}}{V_1 V_2 V_4}$ ⇒ 3|1|2|2|2|1

$\{x_{i1}, \dots x_{ik}\} \subseteq A'$

Pt fiecare cifră c_j de mai sus care e egală cu 1, trb să adăugăm la $A' \cup \{g_j\} \Rightarrow \sum_{x \in A'} x == g$

" <= " Dacă subsetSum$(A, g) == 1$; $(A, g) == F(G, k) \Rightarrow$ kcover$(G, k) == 1$

Știm că subsetSum$(A, g) == 1 \Rightarrow \exists$ un subset $A' \subseteq A$ a.î.
$\sum_{x \in A'} x == j = (k\ 2 \dots 2)$

$\downarrow$

$\exists$ k nr de tip x_j în A
$\{x_{i1},$ ~~x_{i2}~~ $\dots x_{ik}\} \subseteq A'$

$\sum_{j=1}^{k} x_{ij} = k$ $\underbrace{\{1,2\}\ \{1,2\} \dots \{1,2\}}_{n\ ori}$ deoarece restul nr din A' sunt de

Un nr de forma g_j poate adăuga cel mult o unitate fiecărei cifre $\Rightarrow$ nu putem să avem decât 1 și 2.

Dacă alegem varf coresp în V_{ik}, $|V'| = k$, ele acoperă cel puțin un capăt
$\downarrow$
V_{ik}

Ierarhia spațiu-timp a problemelor

Momentan știm că ∃ o clasă limitată de probleme P, NP, NPC, NP dure (Co-NP...)

Vrem să introducem un nr nelimitat de clase în felul urm.

$TIME(f(n)) = \{$ Q probl de decizie | Q acce~~ptă~~ o altă probl de $O(f(n))$ temporal $\}$

$TIME(n)$

$Time(n^2)$

$NTIME(f(n)) = \{$ - " - $\}$

$$P = \bigcup_{k \geq 1} TIME(n^k)$$

$$NP = \bigcup_{k \geq 1} NTIME(n^k)$$

$SPACE(f(n)) = \{$ Q | Q acceptă o rezolv. deterministă ~~probl dintr-o~~ clasă de complex spațial $O(f(n))\}$

$NSPACE(f(n)) = \{$ - " - nedet... $\}$

$NL = NLOGSPACE = NSPACE(\log(n))$

$L = LOGSPACE = SPACE(\log n)$

$$\frac{P \leq NP}{\boxed{L \leq NL} \leq P \leq NP}$$

GAP = Graph Accessability Probl = st conn

Fiind dat un graf $G(V,E)$ orientat. Există $s \rightsquigarrow t$ (o cale) între vârfurile s start şi t dest?

Construim alg eficienţi dpv al cmplx spaţiale care pot fi ineficienţi dpv temporal sau logic

GAP ∈ NL

→ NSPACE-OL

```
N_GAP (G, u, t, lung)
    if (u == t)
        Succes
    if (lung < 0)
        fail
    u = choice (Adj[S])
    N_GAP (G, u, t, lung - 1)
```

lista de adiacenţa a nodului s („vecinii" lui s)

$\boxed{\log(n)}$

$\in \Theta(\log(n))$ ~~[illegible]~~ $\log(n$[illegible]

Apel iniţial N GAP $(G, s, t, n-1)$ unde $n = |V|$

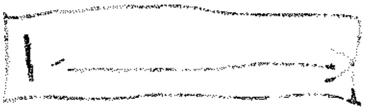
129

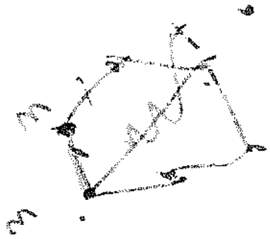
Obs - Într-un graf, cel mai lung drum fără cicluri are max $n-1$ muchii între ∀ 2 vârfuri $s, t \in V$

Care e cmplx spaţială?

- Cmplx spaţială măsoară nr de biţi suplimentari faţă de datele de intrare
- Avem 2 variabile suplimentare: n, lung $\Rightarrow 2\log n \in O(\log n)$

GAP ∈ NL





ust conn (undirected s-t conn) $\in L$

GAP $\in$ SPACE $(f(n))$ $f(n) = ?$

BFS, DFS $\to$ $O(n+m)$ temporală

~~$O(n)$? $O(\overset{n}{\log n})$~~ spațială

$O(n \log n)$

```
GAP recursiv ( G, s, t, lung )
    if (lung == 0)
        return (s == t)
    if (lung == 1)
        return (t ∈ Adj[s])
    for ( u = v1 ... vn )
        return Gaprecursiv( G, s, u, [lung/2] ) &&
               Gap recursiv ( G, u, t, [lung/2] )
```

∃ drum între s și t care să aibă max lungimea lung

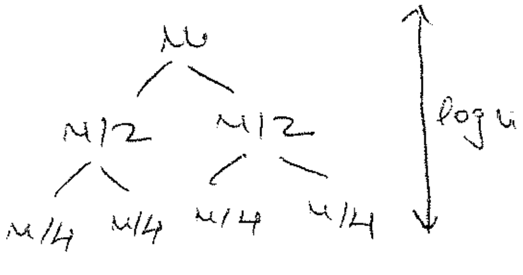
$\log(n) \cdot \log(n)$

$O(\log^2(n))$

Câte apeluri recursive avem maximal pe stivă?

Depinde de apelul inițial

GAP_return $(G, s, t, n-1)$



$\Rightarrow$ $4 \times \log n$ per apel recursiv

$\Rightarrow$ complx spațială $\Rightarrow$ $O(\log^2 n)$

$\Rightarrow$ $\boxed{\text{GAP} \in \text{SPACE}(\log^2 n)}$

Teorema Sevitch

$\forall f(n) \in \Omega(\log n)$

$\underline{NSPACE(f(n)) = SPACE(f^2(n))}$

AA

15.12.2014

Curs 12

Clase de probleme

P, NP, NP_hard, NPC

Th Cook:

SAT $\in$ P <=> P == NP (Sat $\in$ NP_completă)

a) SAT $\in$ NP

b) SAT $\in$ NP_hard (dificil, nu știm nicio altă problemă NP_hard)

$\forall$ problemă $A \in NP \Rightarrow A \leq_p SAT$

$A \in NP \Rightarrow \exists$ Alg nedet+pol care rezolvă A. Încercăm și reușim să transformăm (în timp pol și det) orice Alg nedet + pol la o formulă F_{FNC}

Red pol: $(A \leq_p B)$

i) $\exists$ $F: i_A \rightarrow i_B$ det + pol

$\forall$ $A \in NP$, $A \leq_p SAT \Rightarrow \exists$ $F: i_A \rightarrow i_{SAT}$

$\downarrow$

$F_{FNC}(v_1 \dots v_n)$

cu $T_j \dots F_{mn}$ $(j = 1, m)$

$(\forall)$ Alg nedet + pol

$(\forall)$ date de intrare pt Alg

$Alg(\Delta) == SUCCESS$ după $p(n)$ pași $\Leftrightarrow F_{FNC} = F(Alg, \Delta)$ este satisfiabilă

Ex $B(x, i, t)$ $x \in Var$

dacă bitul i $B[i] \begin{cases} == 1 & \text{la un moment de timp} \\ == 0 & \text{dacă } x[i] == 0 \text{ — " —} \end{cases}$

$|Var| \cdot w \cdot p(n)$ câte variabile

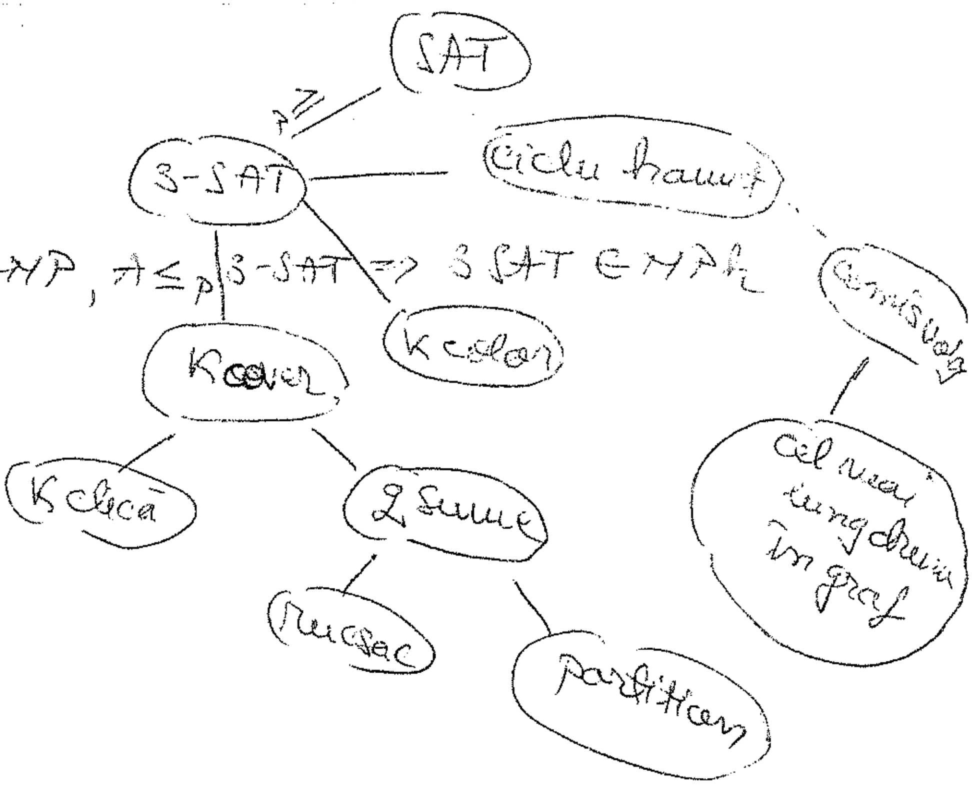
$|Var| < c \cdot p(n) \Rightarrow O(p^2(n))$

Probleme NP_complete (NP_hard)

$SAT \leq_p 3SAT$

$\forall A \in NP$

$A \leq_p SAT \xrightarrow[\equiv p]{tranz} \forall A \in NP, A \leq_p 3\text{-}SAT \Rightarrow 3SAT \in NPh$



Teste: reducere pol

K-SAT : fiecare termen F_{FNC} are k literali

Probl săptămâna trecută :

$\hookrightarrow$ 2 SAT $\in P$

$\hookrightarrow$ 3 SAT $\in NPC$

SAT $\le_p$ 3-SAT

~~SAT $\le_p$ 3-SAT~~

$Co(SAT)$ $\to$ variabile bool

$\neg \exists\ x_1 \dots x_n$ a.î. $F_{FNC}(x_1 \dots x_n) = 1$?

$\downarrow$

$\forall\ x_1 \dots x_n$; $\overline{F_{FNC}} = \overline{0}$? =>

=> — " — ; $\overline{F'_{ND}} = 1$

$\swarrow$

$\overline{F_{ND}}$ - tautologie

$\in Co\text{-}NP$

$Co(FND\text{-}SAT) = FNC$ tautologic $\in P$

i) $\exists F \cdot i_{SAT} \to i_{3SAT}$ $\forall i \in i_{SAT}$

$F_{FNC}\begin{cases} x_1 \dots x_n - \text{variabile} \\ (T_j)_{1 \le j \le n} - \text{termeni} \end{cases} \to$

$\to F'_{FNC}\begin{cases} X' - \text{set variabile} \\ T' - \text{termeni} \end{cases}$

Construcție : $X' = \underbrace{\{x_1 \dots x_n\}}_{\text{variabile din } F_{FNC}} \cup \underbrace{\{y_1 \dots\}}_{\text{variabile adiționale}}$

(T_j') vor fi construiți din (T_j) după cum urmează :

Fie $T_j = < z_{i1} \dots z_{ix} >$ $z_{ij} \to$ literali

$1 < |k| \le 2n$

$\boxed{1}$ $k = 1$ (1 literal) => z_{ij}'

$\overline{F_{FND}} = \bigwedge_{i=1} \overline{T_j'}$ conjuncție termeni

$\overline{T_j'}$ = [illegible] disjuncție [illegible]

$\Rightarrow T_j' = < z_{i1}, y_{j1}, y_{j2} > \wedge$

$< z_{i1}, y_{j1}, y_{j2} > \wedge$

$< z_{i1}, \overline{y_{j1}}, \overline{y_{j2}} >$ $\to z_{i1} \vee z_{i2}$ (A) => T_j' (A) ☺

Variabile adiționale

fiecare T_j' cu 1 literal se adaugă 2 Var y_{j1}

$k > 3$ literali se adaugă $k-3$ Var

(2) $k == 2 \Rightarrow T_j' = \langle z_{i1}, z_{i2}, y_{j1} \rangle \wedge \langle z_{j1}, z_{i2}, \overline{y_{j1}} \rangle$

(ii)

$z_{i1}(A) \Rightarrow T_j'(A)$

(3) ~~T~~ $k == 3 \Rightarrow T_j' = T_j'$

(4) $k > 3 \Rightarrow T_j' = \langle x_{i1}, x_{i2}, y_{j1} \rangle \wedge$

$\langle \overline{y_{j1}}, x_{i3}, y_{j2} \rangle \wedge$

- - - - - - - - - -

$\langle \overline{y_{j,l-2}}, x_{il}, y_{j,l-1} \rangle \wedge$

- - - - - - - - - -

$\langle \overline{y_{jk-3}}, x_{ik-1}, x_{ik} \rangle$

(ii)

→ cel puțin un x_{i_z} - true $(1 \le z \le k)$

$z = 1, 2 \Rightarrow$

$y_{j1} \cdot\; y_{jk-3}$ sunt false

F pol + det

ii) Arătăm că pt ∀ i_{SAT}, $SAT(i) == 1 \Leftrightarrow 3SAT(F(i)) = 1$

$i_{SAT} = \begin{cases} x_1 \dots x_n \\ F_{FNC} = \bigwedge_{j=1}^{n} T_j \end{cases}$

$\downarrow$ $n \Longrightarrow$ "

Satisfiabilă $\Rightarrow \exists x_1 \dots x_n \in \{0,1\}$ a.î. $F_{FNC} == 1$

$F'_{3SAT} == 1$

Pt fiecare $T_j' \rightarrow$ adev → ~~[illegible]~~ arătăm că T_j' adevărat

$1 \begin{cases} z = k-1, k \Rightarrow y_{j,1} \dots y_{j,k-3} \text{ (true)} \\ z \in \{3 \dots k-2\} \Rightarrow y_{j1} \dots y_{j,l-2} \text{ (true)} + \end{cases}$

+ [illegible]

„$\Leftarrow$" $\exists\ x'_{FFNC}$ a.î $F'_{FFNC}(x) \Rightarrow \exists\ x_1 \ldots x_n <$ $F_{FFNC}(x_1 \ldots x_n) == 1$

Știm T_j true $\Rightarrow T_j$ true

Reducerea la 3SAT la K cover $\quad$ 3SAT $\leq_p$ Kcover

Kcover : Dându-se un graf neorientat $G(V,E)$ $\exists\ V' \subseteq V$ $|V'| == k$ a.î $\forall (u,v) \in E$, $u \in V'$ și/sau $v \in V'$?

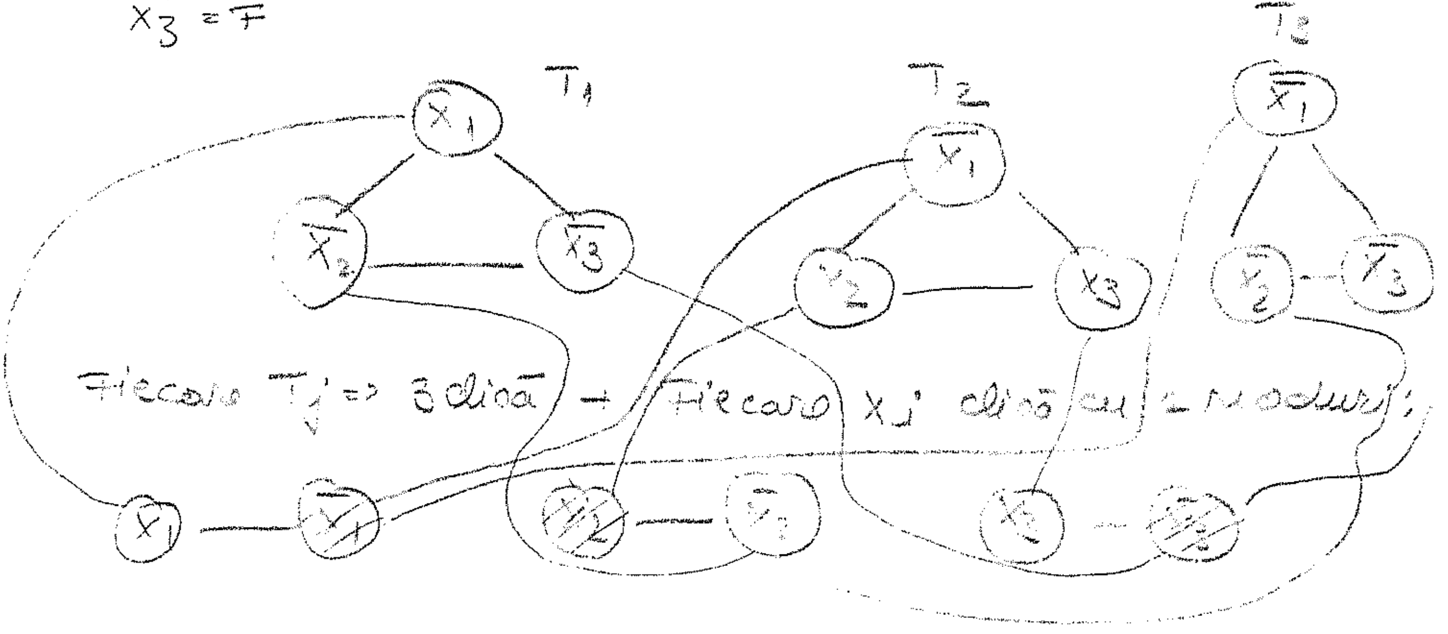
i) $\boxed{\exists\ F: i_{3SAT} \longrightarrow i_{Kcover}}$

$\forall\ x_1 \ldots x_n$

$$F_{3FNC}(x_1 \ldots x_n) = \bigwedge_{j=1}^{m} T_j \xrightarrow[?]{F} G(V,E)$$

$$(x_1, \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

$x_1 = F$
$x_3 = F$



Fiecare $T_j \Rightarrow$ 3 clică + Fiecare x_i clică cu 2 noduri;

Graful $G \longrightarrow |V| = 3m + 2n$
$|E| = 6m + n$
$K = n + 2m$ acoperire

~~iii)~~
ii)

$\boxed{\forall i \in I_{3SAT} \text{ a.î } 3SAT(i) == 1 \Leftrightarrow Kcover(F(i)) == 1}$

" $\Rightarrow$ "

Fie $F_{FNC}(x_1 \dots x_n)$ $\exists\, x_1 \dots x_n \in \{0,1\}$ a.î $F_{FNC}(x_1 \dots x_n) ==$

$\overset{}{\rule{0pt}{0pt}}$———— $\Downarrow$

$F(~~F_{3SAT}~~ F_{FNC}(x_1 \dots x_n)) \Rightarrow \begin{cases} G(V,E) \\ K = m+2n \end{cases}$

$\exists\, V' \subseteq V$, $|V'| == 2m+n$ aî $(u,v) \in E$ $u \in V'$ sau/și $v \in V'$

Alegem din vârfurile de jos cele n vârfuri coresp valorilor de adevăr pt variabilele $x_1 \dots x_n$ care asigură satisfiabilitatea formulei F_{3FNC}

Aceste vârfuri acoperă ~~toate~~ muchiile din partea de jos (n, + n muchii între nodurile de sus (coresp termenilor) + de jos (coresp variabilelor)

Pt fiecare 3 clică (gadget), aleg 2 noduri în acoperire: acelea coresp literalilor care sunt false $\Rightarrow$ adaug $2*m$ noduri în acoperirea V'. Cele 2 alese nu acoperă doar muchiile celor 3 falși ci toate muchiile din 3 clică

" $\Leftarrow$ " $\underset{F(i)}{\overset{}{G(V,E)}}$ are o $K = n+2m$ acoperire $\Rightarrow \exists\, x_1 \dots x_n$ ~~$\in$~~ $\in \{0,$

a.î $F_{3FNC}(x_1 \dots x_n) == 1$

———————— $\Downarrow$

Fie $V' \subseteq V$, $|V'| = 2m+n$ acoperire a lui G

Din fiecare 3 clică trb să aleg câte 2 noduri $\Rightarrow 2*m$ noduri

=> Pt fiecare nod din 3 clică neales, trb să aleg nodul coresp din nodurile de jos

=> Aleg n noduri din nodurile aflate în partea de jos. Dacă am ales $\boxed{x_i}$, nu aleg $\boxed{\overline{x_i}}$: $(x_i, \overline{x_i})$ este acoperit de x_i

=> Pt fiecare termen, avem în partea de jos un literal true (în V') => T_j satisfăcut $\forall\, j = 1 \ldots m$ =>

=> FFNC satisfăcută