

CALCULABILITATE

Ce se poate calcula?

Calculator ideal

Limbaj de programare ideal

Teorema - multimea programelor este numărabilă.

Teorema programului universal

Există un program $Q \in P_{\text{uni}}$ a.t. $\begin{cases} Q(r, s) = v, \text{ dacă } P_r(s) = v \\ Q(r, s) = L, \text{ dacă } P_r(s) = L \end{cases}$

$P_1, P_2, P_3, \dots, P_k, \dots$

Teorema oprii (Turing)

Nu există un program $Q \in P_{\text{uni}}$ a.t.

$$\begin{cases} Q(r, s) = z, \text{ dacă } P_r(s) = v \\ Q(r, s) = o, \text{ dacă } P_r(s) = L \end{cases}$$

Demonstră:

1. Scriem programul Q care citește r și s , construiește P_r . $v \leftarrow P_r(s)$

Scrie v

2. Presupunem prin reducere la absurd că $\exists Q$ ca în ipoteză. Atunci putem scrie programul $S \in P_{\text{uni}}$:

$$S(r) = z, \text{ dacă } P_r(r) = v$$

$$S(r) = o, \text{ dacă } P_r(r) = L$$

S. citește r

$$v \leftarrow Q(r, r)$$

tipărește v

Atunci putem scrie programul $T \in P_{\text{uni}}$ a.t.:

$T(r) = 1$, dacă $P(r, r) = v$

$T(r) = 0$, dacă $P(r, r) = 1$

$T(r) \in P_n \Rightarrow \exists k \text{ a.t. } T = P_k$.

I $P_k(r) = v = T(r) = 1$ ABSURD!

II $P_k(r) = 1 = T(r) = 0$ ABSURD!

Teorema lui Gödel

În orice sistem formal de complexitatea aritmetică

- teorema care nu poate fi demonstrată.

Ideeă demonstrației:

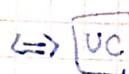
1. se face o numerațare a tuturor formulelor/expreziilor aritmetice, inclusiv teoreme;
gödelizare

2. teorema cu numărul x e falsă.

x - nr. din gödelizare.

Modele matematice ale calculului

Arhitecturi von Neumann



Problema: gătuirea.

programare funcțională - fără efecte laterale (dataflow).

Entaplement

"Can programming be liberate from the von Neumann style?"

J. Backus

$v \leftarrow x + f(x) \Leftrightarrow v \leftarrow f(x) + x$ - matematic echivalente, nu

în programare.

Ce se poate calcula? - ce probleme

↳ de decizie

$$f: A \rightarrow \{ \text{da}, \text{nu} \}$$

Program - funcție recursivă - multime recursivă - predicat

Functii aritmetice $f: \mathbb{N} \rightarrow \mathbb{N}$

totale și parțiale ($f: A \subseteq \mathbb{N}$)

$$f: A \rightarrow \mathbb{N}, \quad \text{def}(f) = A$$

$$f: \mathbb{N} \rightarrow \mathbb{N} \text{ parțială}$$

Def.: O funcție f este recursivă ($f: \mathbb{N} \rightarrow \mathbb{N}$) dacă este calculabilă $f(n) = v$, dacă $n \in \text{def}(f) = A$, $\exists P \in \mathcal{P}_{\mathbb{N}}$ a.s. $P(n) = v$ și dacă $n \notin \text{def}(f)$, $P(n) = L$.

Teoremă

Există mai multe funcții nerecursive decât recursive.
(mult mai multe)

Demonstratie:

Discutăm de funcții $f: \mathbb{N} \rightarrow \mathbb{N}$. Multimea tuturor funcțiilor aritmetice $HOM(\mathbb{N}, \mathbb{N})$ are cardinalul egal cu multimea numerelor reale.

Multimea funcțiilor recursive \rightarrow funcții parțiale pentru care există un program care o calculează.

Fie $\Phi: F_{\text{rec}} \rightarrow \boxed{P}$ are card li \mathbb{N} (e numărabilă).

$\Phi(f) = P$ (programul de dimensiune minimă care o calculează)

$\Rightarrow F_{\text{rec}} - \text{numărabilă} \Rightarrow$ mult mai mică decât multimea tuturor funcțiilor aritmetice.

TEZA Turing-Church

Multimea calculabilității sunt funcțiile recursive.

Teorema

Există și funcții totale nerecursive.

Demonstratie:

Fie $f: \mathbb{N} \rightarrow \mathbb{N}$ totală,

$$f(n) = \begin{cases} \perp, & \text{dacă } P_n(k) \text{ definită pentru } \forall k \in \mathbb{N} \\ 0, & \text{altfel.} \end{cases}$$

Construim programul

$$Q(n) = \begin{cases} P_n + 1, & \text{dacă } f(n) = \perp \\ 0, & \text{altfel.} \end{cases}$$

Atunci $Q \in \mathcal{P}_{\perp,1}$ ⇒ $\exists l \in \mathbb{N}$ a.t. $Q = P_l$.
deoarece

$$Q(l) = P_l(l) = P_l(l) + \perp$$

ABSURD! ⇒ f -nu e recursivă

Def.: O multime $A \subseteq \mathbb{N}$ este recursivă dacă:

a) funcția caracteristică a mulțimii este recursivă

$$f(n) = \begin{cases} \perp, & n \in A \\ 0, & n \notin A \end{cases}$$

Obs.: f -totală.

b) Există un program $P \in \mathcal{P}_{\perp,1}$ a.t.

$P(n)$ întoarce \perp , $n \in A$ și 0 , $n \notin A$.

(adevărat-fals, orice 2 valori)

Def.: O multime $A \subseteq \mathbb{N}$ este recursiv-enumerabilă, dacă:

a) Există un program $P \in \mathcal{P}_{\perp,\infty}$ care întoarce

\perp , dacă $n \notin A$ și nu se termină dacă $n \in A$;

b) Există un program $P \in \mathcal{P}_{\perp,\infty}$ care generează toate elementele lui A .

f : funcția caracteristică f este partită pe A .

$$A = \text{def}(f).$$

Teoremă: \forall mulțime recursivă este și recursiv enumerabilă.

A -recursivă $\Rightarrow A$ recursiv enumerabilă.

Demonstratie: $\exists P$ program care

întoarce 1 sau 0, dacă $n \in A$ sau $n \notin A$.

Construim programul:

citește n

$m \leftarrow P(n)$

dacă $m=0$, atunci sări la etichetă

etichetă:

tipărește m

Teoremă: Dacă o mulțime A și $N \setminus A$ - RE, atunci A - R.

$N \setminus A$ - complementara lui A .

Demonstratie:

Notez $P_A[n]$ - programul ce îl generează pe A

$P_{C_A}[n] = \neg P_A[n]$

citește n

găsit \leftarrow fals

$i = 1$

etichetă: dacă $P_A[i] = n$, atunci $găsit = 1$

dacă $P_A[i] \neq n$, atunci $găsit = 0$

$i \leftarrow i + 1$

dacă $găsit = \text{fals}$ atunci sări la etichetă

tipărește $găsit$

TESTUL TURING de IA

↓
Scheme Winograd

Multime recursivă și recursiv enumerabilă

Teorema: 1. } multimi recursive;

2. } multimi recursiv enumerabile care nu sunt recursive;

3. } multimi care nu sunt nici măcar recursiv enumerabile.



Demonstratie: 1. \mathbb{N}, \mathbb{N} - pare

2. Multimea programelor care se termină

$$A = \{ p \in P \mid P \in \mathcal{P}_{\text{progr}}, \forall n \in \mathbb{N}, P(n) \neq \perp \}$$

Arătăm că a) este RE și b) nu e R :

a) Arătăm (construim) programul care le generează

$n \leftarrow 1$ // nr. pasi

eticheta 2: $m \leftarrow 1$

eticheta 1:

Dacă $P_m(n)$ se termină în n pasi și nu a fost tipărit, tipărește-l.

$m \leftarrow m+1$

Dacă $m < n$ sări la eticheta 2.

$n \leftarrow n+1$

Sări la eticheta 2.

b) Presupunem prin absurd că este recursivă.

$$\Rightarrow f(n) = \begin{cases} 1, n \in A \\ 0, n \notin A \end{cases} \Rightarrow p_0(n) \neq L$$

$\Rightarrow p_0(n) = 1$ este recursivă.

$\Rightarrow \exists$ un program $P(n)$ care calculează $f(n)$.

Așa că $Q(n) = \begin{cases} P_n(n) + 1, & f(n) = 1 \\ 0, & f(n) = 0. \end{cases}$

$Q =$ al k -lea program în lista programelor

$Q = P_k, Q(k) = P_k(k) = P_k(k) + L$ REZURBII

3. Arătăm că sunt multimi nici măcar recursiv enumerabile.

Fie $C_n = \{ p \mid p \in P_{n+1}, \forall n \in \mathbb{N}, P_n(n) = 1 \}$

Dacă C_n este RE și A este RE $\Rightarrow A$ este ABSURD!

$\Rightarrow C_n$ este !RE.

- Calculator ideal
- Program ideal
- Funcții recursive
- Multimi recursive și recursiv enumerabile
- Predicte decidabile, semidecidabile
- Probleme decidabile, semidecidabile

Probleme - de decizie

- de optimizare
- de construcție
- ...

Predicat = $\pi : \mathbb{N}^k \rightarrow \{\text{aderărat, fals}\}$ este decidabil:

a) dacă \exists un program $P \in P_{\text{sem}}$ a.t.

$P(x_1, x_2, \dots, x_n) = \begin{cases} \text{aderărat, } \pi(x_1, \dots, x_n) = \text{aderărat} \\ \text{fals, } \pi(x_1, \dots, x_n) = \text{fals} \end{cases}$

b) $A = \{(x_1, x_2, \dots, x_n) \mid \pi(x_1, \dots, x_n) = \text{aderărat}\}$ este recursivă;

Definiție: Un predicat $\pi(x_1, \dots, x_n)$, $\pi : \mathbb{N}^k \rightarrow \{\text{aderărat, fals}\}$ este semidecidabil dacă:

a) $\exists P \in P_{\text{sem}}$ a.t. $P(x_1, \dots, x_n) = \begin{cases} \text{aderărat, } \pi(x_1, \dots, x_n) = \text{aderărat} \\ \perp, \text{ altfel} \end{cases}$

b) $\exists P \in \mathcal{P}_{\text{so}, \infty}$ a.t. P generează toate triplelele

(x_1, \dots, x_n) a.t. $\mathcal{S}(x_1, x_2, \dots, x_n)$ - adevarat.

c) \mathcal{A} de la def. anterioră este recursiv enumerabilă;

Teorema lui Rice

Oricare proprietate extensională, nebună, a programelor este nebanală.
nedecidabilă.

În general, o proprietate poate fi definită intensional (formula matematică) sau extensional (enumerând elementele cu proprietatea dată).

Ex.: $\cdot \{ s | s \in \text{Student} \wedge \text{notă}(s) \geq 9 \}$

$\cdot \{ \text{Ion}, \text{Vasile}, \text{Maria}, \dots \}$

O proprietate ^{extensională} prop a programelor este multimea programelor care \in prop și proprietatea că dacă

$(P \in \text{prop} \Leftrightarrow Q \in \text{prop})$ dacă $(P = Q)$.

Prop este nebanală, dacă $\text{Prop} \neq \emptyset$.

Demonstratie:

Fie programul nonstop (care nu se termină).

nonstop : etichetă ...

sau la etichetă.

Presupunem prin reducere la absurd că o proprietate ^{nebanală} prop extensională, nebună este decidabilă.

Avem 2 variante: a) nonstop \in prop sau

b) nonstop \notin prop.

a) nonstop \in prop

$\exists Q \notin \text{prop}, Q \neq \text{nonstop}$.

Potem scrie programul

R: citește n

$R = \{ \text{nonstop}, P(x) = 1 \}$

$P(x) ; Q(n)$

$\} Q(n), P(x) \neq 1$

prop - extensibilă, nebahadă $\Rightarrow \begin{cases} R \in \text{prop, dacă } P(x) = 1 \\ R \notin \text{prop, dacă } P(x) \neq 1 \end{cases}$

prop-decidabilită + (i) \Rightarrow este decidabilită oprinea lui P.

Absurdi

b) similar.

\Rightarrow problema oprinții nu e singura nedecidabilă.

De acum, discutăm de ce probleme decidabile.

Complexitatea calculului

Problema - Algoritm

- Date

		Dimensiunea înălțării					
volum de comprimare		10	20	30	40	50	60
n		0,000015	0,000025				0,000065
n^2		0,00015	0,00045				0,00365
n^3		0,0015	0,0085	0,0275	0,0645	0,1255	0,2165
n^5		0,15	3,25	24,35	177min	522min	13 min
2^n		0,001	1	17,9min	12,7zile	35,7ani	366 secole
3^n		0,059	58min	6,5 ani	3855 secole	$2 \cdot 10^8$ secole	$1,3 \cdot 10^{13}$ secole

Complexitate temporală polinomială

Nu exponential! (complexitate retractabilă)

Inceram calculatoare mai performante.

volum de date ce se poate calcula într-o oră:

complexitate	initial	100x	1000x	
n	N_1	$100N_1$	$1000N_2$	$2^b N_5 + b, 64 N_{5+3,97}$
n^2	N_2	$10N_2$	$31,6 N_2$	$3^b N_6 + 4,19 N_6 + 6,29$
n^3	N_3	$4,64 N_3$	$10 N_3$	
n^5	N_4	$2,5 N_4$	$3,98 N_4$	

Analiza complexității algoritmilor

Necesară o metrică a complexității.

O , Ω , Θ , Θ , w

Aspecte ale complexității algoritmilor.

→ cazul cel mai defavorabil;

→ cazul mediu;

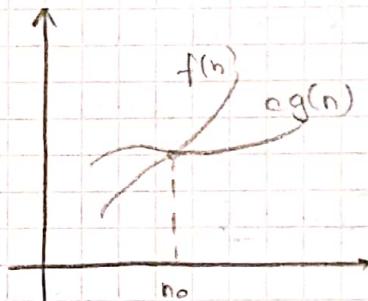
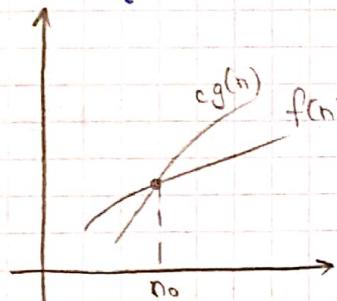
→ cazul cel mai favorabil.

Analiza amortizată

Calculul complexității - aproximativ

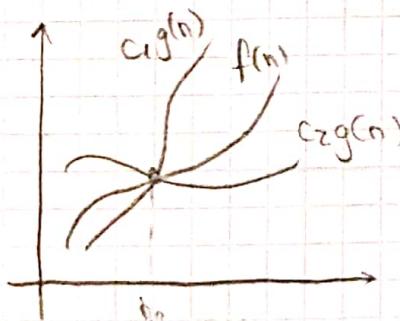
- precis

$$O(g(n)) = \{f(n) \mid \exists c > 0, n_0 \in \mathbb{N} \text{ a.t. } \forall n > n_0, f(n) \leq cg(n)\}$$



$$\Omega(g(n)) = \{f(n) \mid \exists c > 0, n_0 \in \mathbb{N} \text{ a.t. } \forall n > n_0, f(n) \geq cg(n)\}$$

$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0, n_0 \in \mathbb{N} \text{ a.t. } c_2 g(n) \leq f(n) \leq c_1 g(n)\}$$



$$\Theta(g(n)) = \{f(n) \mid \exists c > 0, \exists n_0 \in \mathbb{N} \text{ a.t. } \forall n > n_0, f(n) \sim cg(n)\}$$

$$w(g(n)) = \{f(n) \mid \forall c > 0, \exists n_0 \in \mathbb{N} \text{ a.t. } \forall n > n_0, cg(n) \sim f(n)\}$$

$$f(n) : \Theta(g(n)) \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

$$w(g(n)) \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

Proprietăți:

→ toate sunt tranzitive.

$$\begin{aligned} f(n) &= O(g(n)) \\ g(n) &= O(h(n)) \end{aligned} \quad \left\{ \Rightarrow f(n) = O(h(n)) \right.$$

Ω , Θ , \mathcal{O} , ω

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

Ω , Θ

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \omega(f(n))$$

$$f(n) = \Theta(g(n)) \Leftrightarrow \begin{cases} f(n) = O(g(n)) \\ f(n) = \omega(g(n)) \end{cases}$$

$$(c_1 n^k_1 + c_2 n^{k_2} + \dots) = O(n^k).$$

$$K = \max\{k_1, k_2, \dots\}$$

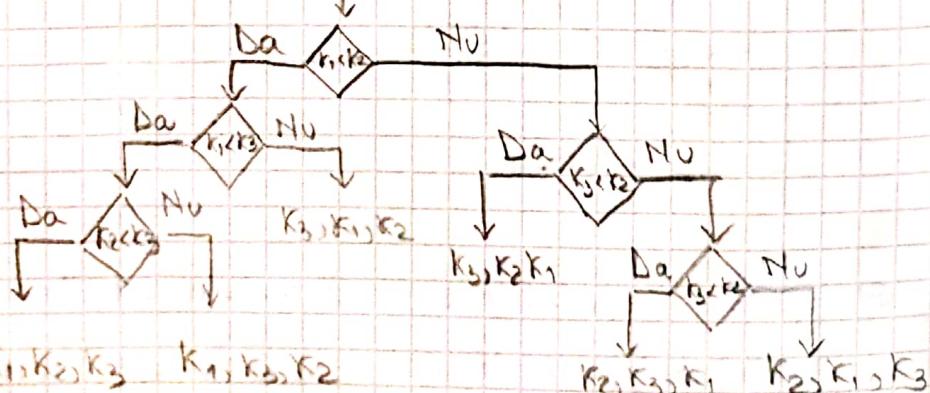
$$O(g_1(n) + g_2(n)) = O(\max\{g_1(n), g_2(n)\}).$$

Teorema: Orice sortare prin comparație de ~~tipuri~~ are chei complexitate $\Omega(n \log n)$.

Demonstratie:

Să luăm un exemplu simplu

(K_1, K_2, K_3) .



Generalizăm → orice nr. de chei și orice ordine de comparații considerăm
n chei $\Rightarrow n!$ frunze.

Complexitatea algoritmului este numărul de comparații.
→ sănătatea arborelui (h).

Nr. total posibili de frunze, dacă arborele ar fi complet ar fi 2^h .

$$h! < 2^h$$

Formula lui Stirling: $n! > \left(\frac{n}{e}\right)^n$.

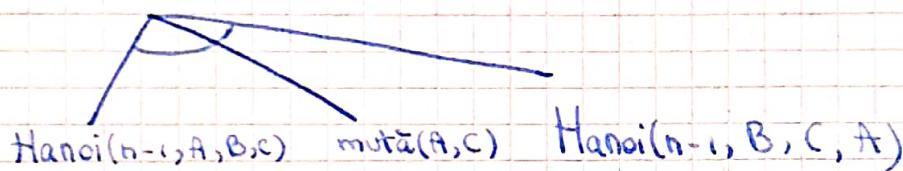
$$\left(\frac{n}{e}\right)^h < n! < 2^h$$

Logaritmomăm: $h > \log\left(\frac{n}{e}\right)^h = h \log n - h \log e$

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

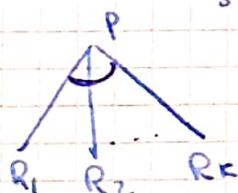
→ nr. de comparații → complexitatea.

Hanoi(n , A, C, B) B-intermediar



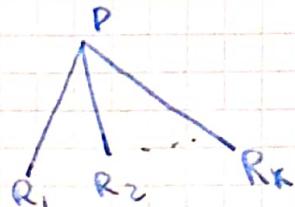
Arbore și sau

Notăm „și”



$P \rightarrow R_1 \text{ și } R_2 \text{ și } \dots \text{ și } R_k$

Notăm „sau”



$P \rightarrow R_1 \text{ sau } R_2 \text{ sau } \dots \text{ sau } R_k$

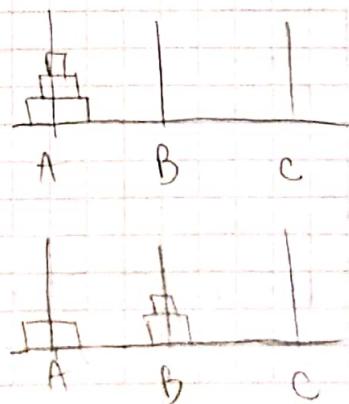
dacă $n > 1$

adună Hanoi($n-1$, A, B, C)

mută(A, C)

Hanoi($n-1$, B, C, A)

sfârșit



$$T(n) = T(n-1) + 1 + T(n-1)$$

$$T(n) = 2T(n-1) + 1$$

$$2T(n-1) = 2(2T(n-2) + 1) = 2^2 T(n-2) + 2$$

$$2^2 T(n-2) = 2^3 T(n-3) + 2^2$$

$$2^3 T(n-3) = 2^4 T(n-4) + 2^3$$

.....

$$\underline{2^k T(n-k) = 2^{k+1} T(n-k-1) + 2^k} \quad \text{Telescopare}$$

$$T(n) = 2^{k+1} T(n-k-1) + (2^k + 2^{k-1} + \dots + 1)$$

Dar $k = n-2$

$$n-k-1 = 1 \quad (T(1))$$

$$T(n) = 2^{n-2} + \sum_{i=1}^{n-2} 2^i = \sum_{i=1}^{n-1} 2^i = 2^n - 1$$

$$T(n) = O(2^n).$$

$$2^{64}-1 \text{ mutări} \approx 10^{19} \text{ mutări}$$

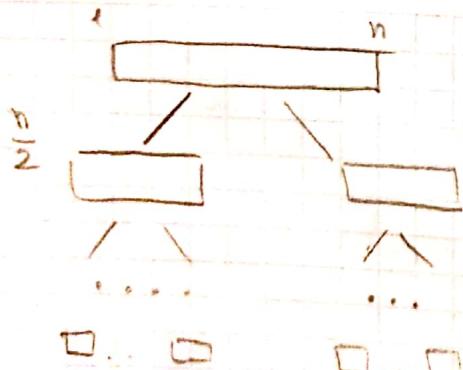
$$1 \text{ mutare} = 10^{-6} \text{ secunde}$$

$$\Rightarrow 10^{19} \text{ secunde} - \text{mai mult decât } 10^6 \text{ ani.}$$

Divide et impera

Foarte multe probleme se reduc la:

$$T(n) = a T\left(\frac{n}{b}\right) + f(n)$$



la fiecare nivel avem $O(n)$

$\log n$ niveluri

$$O(n \log n)$$

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

Teorema master - nu acoperă însă toate cazurile

În funcție de fenomenul dominant

 A domină

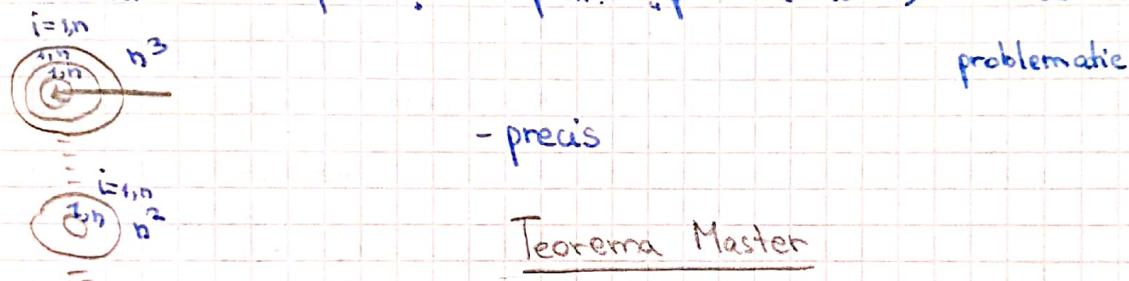
 A și B au aceeași complexitate

 B domină $\rightarrow O(f(n))$

Calculul recurențelor - Telescopare

- Evaluare pe arborele de descompunere
- Inductie matematică
- Teorema master.

Calculul complexității - aprox. \rightarrow „probe” (măsură) - în locul cel mai



este greșit, dar nu
este suficient de
precis

Teorema Master

Dacă avem o recurentă $T(n) = aT(\frac{n}{b}) + f(n)$
 $a \geq 1, b \geq 1$ - și f - asimptotică crescătoare
pozitivă, atunci:

a) $f(n) \in O(n^{\log_b a - \epsilon}), \epsilon > 0$ - atunci
 $T(n) \in \Theta(n^{\log_b a})$

b) $f(n) \in O(n^{\log_b a} \log^k n), k \geq 0$ - atunci

$$T(n) \in \Theta(n^{\log_b a + \frac{1}{2}})$$

c) $f(n) \in \Omega(n^{\log_b a + \epsilon}), \epsilon > 0$ - și

- ordonare $a \cdot f(\frac{n}{b}) \leq c \cdot f(n), c < 1$ - atunci

- politică de coadă + priorități n suficient

de mare, atunci $T(n) \in \Theta(f(n))$.

1. Specificare formală

 Tipuri de date abstracte (TDA)

 formalizare a unei structuri de date (la un nivel abstract)

 Definim ca o capsulă/cutie neagră.

Spanem daar cum o folosim.

Modularizare - reutilizare.

$$P_{\text{oo}} = TDA + \text{mostenire}_s$$

σ -algebra

Signature

Tipuri de date Q

1. Se definesc domenii

E (elemento)

R+ (priorități)

2. Signatura - operării - constructori - au Q

→ nulari (nu au domeniu de def)
= new Q

↳ non-lari - recursivi $B \dots \rightarrow B$

- recursive

$$C_{pr}: A_1 \times A_2 \times \dots \times A_n \rightarrow B$$

new: → Q

insert: $\mathbb{Q} \times E \rightarrow \mathbb{Q}$

extract: $\mathbb{Q} \rightarrow \mathbb{Q}$

- altü (generali)

topelement : $Q \rightarrow E$

`empty? : Q → {true, false}` - predicate

3. Comportament - axiome ale tcl

extract (new) envelope

$\text{extract}(\text{insert}(e, g)) = g$ \Rightarrow dacă se scoate maximul

`extract(insert(esg)) = dača es topelement(g)`

abusing

affel

Exhibit 26

insert(extract(q))

$\text{extract}(\text{insert}(e, \text{new})) = \text{new}$

$\text{topq}(\text{insert}(e, q)) =$ dacă $e > \text{topq}(q)$
atunci e
altfel
 $\text{topq}(q)$

$\text{topq}(\text{insert}(e, \text{new})) = e.$

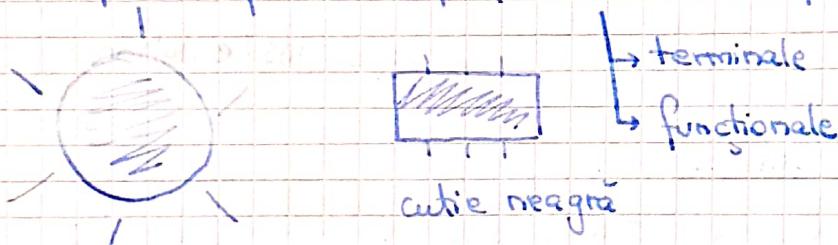
Coadă cu priorități

$\leftarrow \boxed{\dots} \leftarrow$ nu este FIFO

FIFO
 $\hookrightarrow \text{Best}(\min/\max).$

Obs.: La Dijkstra - coadă de priorități + modifică prioritatea.
pt.d.o.

Incapsularea prelucrări \rightarrow specificare abstractă \rightarrow implementare.



API

1) Coadă cu priorități ca tda

(tda) \rightarrow bazată pe Σ -algebra

- descrierea capsulei este ca o sănătăție (contract);

\rightarrow sorturi - cozi cu priorități Q - multimi/entități din acel tda.

E - elemente

{true, false}

semnătura: new: $\rightarrow Q$ constructor nular,

• ins: $\rightarrow E \times Q \rightarrow Q$ constructor recursiv

sau constructor norecursiv (ex: list: $E \times E \times \dots \times E \rightarrow L$ /
list: $E^n \rightarrow L$).

obs.. recursiv ar fi cons: $E \times L \rightarrow L$.

- max: $Q \rightarrow E$
- delmax: $Q \rightarrow Q$

PoC = tda + menținere.

- empty?: $Q \rightarrow \{\text{true}, \text{false}\}$.

axiome: (funcționarea e legată de ele)

max(new) = eroare, eventual se dă o restricție
constraint: $\max(Q) \text{ if } (Q \neq \text{new})$.

$\max(\text{ins}(e, q)) = \begin{cases} \text{dacă } q = \text{new}, \text{ atunci } e \\ \text{altfel, dacă } e > \max(q), \text{ atunci } e \\ \text{altfel } \max(q). \end{cases}$

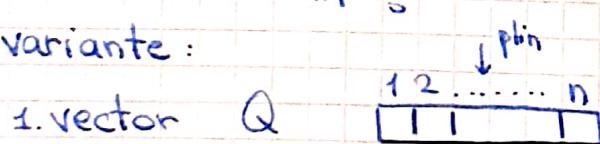
$\text{delmax}(\text{ins}(e, q)) = \begin{cases} \text{dacă } q = \text{new, atunci } q \\ \text{altfel, dacă } e > \max(q), \text{ atunci } q \\ \text{altfel } \text{ins}(e, \text{delmax}(q)). \end{cases}$

Implementare

- acum considerăm ce este în interiorul capsulei;

- ne interesează "pretul" \rightarrow complexitatea $(O(f(n)), \theta)$.

variante:



new: $Q[1:n]$

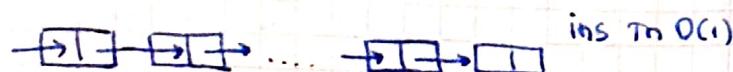
ins: $Q[\text{plin}+1] \leftarrow e$ $O(1)$ plin = plin + 1

max: $O(n)$

delmax: $O(n)$

Vor fi necesare extinderi $n \rightarrow 2n - O(n)$.

2. liste



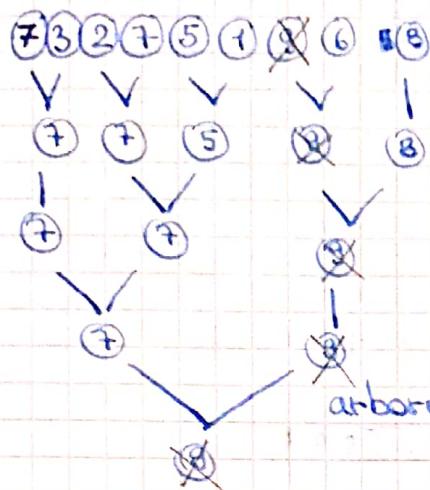
new: . \rightarrow null

delmax sterge și reface legătura $O(n)$

dezavantaj: se dublează necesarul de memorie.

complexitate apropiată

3. arbore de competiție



o propunere prenevală

→ bazată pe

/ metafore

folosirea unor elemente

din experiența

noastră în algoritmi

Probleme: - stergere maxim: rejudicare (stergerea unei ramuri din arbore + comparare)

- multă memorie suplimentară

critică
soluție imposibilitate

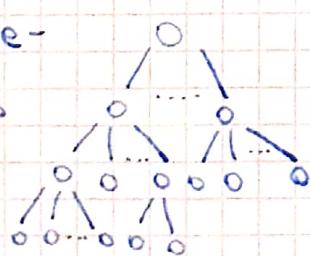
→ fără memorie suplimentară

→ rejudări minime

4.

Iterație-

întimp



Legile lui Murphy - Principiul lui Peter:

Fiecare aranzează până la nivelul său de incompetență.

Idee → initial, arborele este construit la întâmplare

→ îl punem în ordinea în care au venit;

→ îl punem în competiție local (șef și subordonat) și aranzează cel mai bun a.t. la sfârșit să avem

$v[i] \geq v[j_e]$; $i = 1, 2, \dots, R$; - proprietate de heap.

→ metaforă

Sortare
polifazică

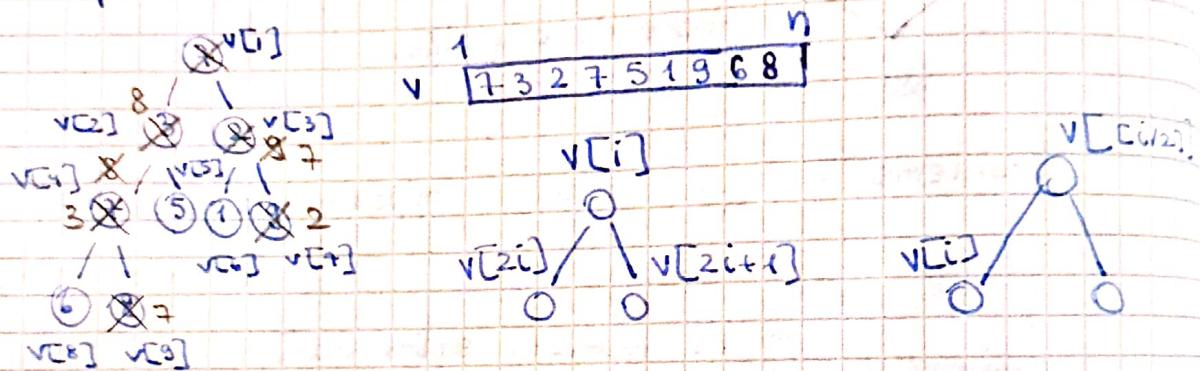
→ se repetă la toate nivelurile
↳ propagare a celor buni.

heap → binar (maxim 2 fiu)

→ multican (mai mulți de 2 fiu)

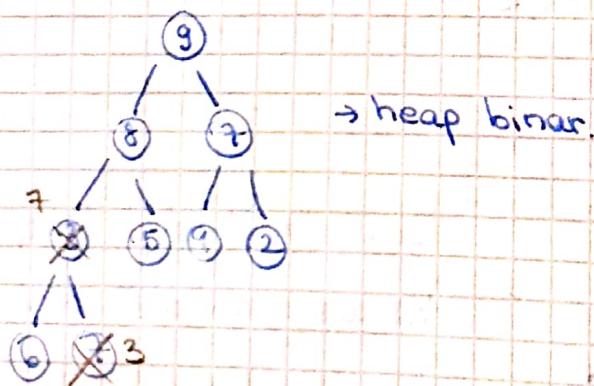
- binomial
- Fibonacci
- leftist

Optimizăm ideea arborilor de competiție.



arbore binar esențial complet

Competiția locală → tată - fiu.



iterații

Iterație sistematică:

- de sus în jos (top-down);
- de jos în sus (bottom-up);
- aleator

Procedura: Construiește heap ($V, n; V$) cu max.
 pentru $i \in [\frac{n}{2}]$ la sfârșit repeta propagă-in-jos($i, V, n; V$) $\Theta(n)$

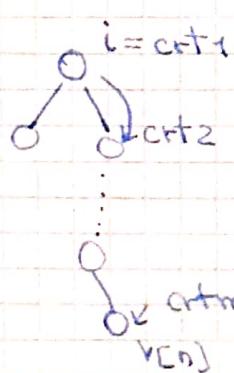
Procedura: propagă-in-jos($i, V, n; V$)
 mută $\leftarrow i;$
 repetă
 current \leftarrow mută
 dacă $v[2 \cdot \text{current}] > v[\text{current}]$ atunci mută $\leftarrow 2 \cdot \text{current}$
 mută $\leftarrow 2 \cdot \text{current} + 1$
 dacă $v[2 \cdot \text{current} + 1] > v[\text{mută}]$ atunci mută $\leftarrow 2 \cdot \text{current} + 1$
 dacă mută \neq current atunci ~~inter~~ aux $\leftarrow v[\text{current}];$
 $v[\text{current}] \leftarrow v[\text{mută}];$
 $v[\text{mută}] \leftarrow \text{aux};$
 până când mută = current $\approx O(\log n)$

Complexitatea construirii heapului:

1. evaluare aproximativă: $O(n \log n)$ - corect, dar nu suficient de precis.

2. calcul mai precis

$$n \geq \text{crt}_m \geq 2^{\text{crt}(m-1)} \geq 2^2 \text{crt}(m-2) \geq \dots \geq 2^{\text{crt}_1} = 2^{m-1}.$$



$$n \geq 2^{m-1}$$

Ne interesează m (nr. de pași/iterații în propagă-in-jos)

$$\frac{n}{i} \geq 2^{m-1}$$

$$\log \frac{n}{i} \geq m-1$$

$$m \leq \log \frac{n}{i} + 1$$

$$\text{Nr. total de iterări: } \sum_{i=1}^{\left[\frac{n}{2}\right]} \left(1 + \log \frac{n}{i}\right) = \left[\frac{n}{2}\right] + \sum_{i=1}^{\left[\frac{n}{2}\right]} \log \frac{n}{i}$$

$$\sum_{i=1}^n \log \frac{n}{i} < \sum_{i=1}^n \log \frac{n}{i} < \log \prod_{i=1}^n \frac{n}{i} = \log \frac{n^n}{n!}$$

$$n! \geq \left(\frac{n}{e}\right)^n \quad e^n \geq \frac{n^n}{n!}$$

$$n \log e \geq \log \frac{n^n}{n!}$$

$$\log \frac{n^n}{n!} \leq n \log e \Rightarrow O(n)$$

$$\log \frac{n^n}{n!} < n \log e \Rightarrow O(n) - \text{valoare mai precisă.}$$

1. Definirea problemei \rightarrow tda
 2. Implementare optimă complexitate
 3. Analiza corectitudinii
 - ~~4. Implementare în limbaj~~
 5. Testare - se cercetă dacă sunt erori
 - nu se poate spune că nu are erori
- } pași pt. proiectare și implementare

Teorema Master

«Rețetă» - ușor de aplicat

- nu se aplică tot acoperă toate cazurile chiar dacă doar la recurențe la care se aplică.

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Demonstratie:

$$\begin{aligned}
 T(n) &= aT\left(\frac{n}{b}\right) + f(n) = f(n) + aT\left(\frac{n}{b}\right) = \\
 &= f(n) + a\left(aT\left(\frac{n}{b^2}\right) + f\left(\frac{n}{b}\right)\right) = \\
 &= f(n) + a^2T\left(\frac{n}{b^2}\right) + af\left(\frac{n}{b}\right) = \\
 &= f(n) + af\left(\frac{n}{b}\right) + a^2\left[aT\left(\frac{n}{b^2}\right) + f\left(\frac{n}{b^2}\right)\right] = \\
 &= f(n) + af\left(\frac{n}{b}\right) + a^2f\left(\frac{n}{b^2}\right) + a^3T\left(\frac{n}{b^3}\right) = \\
 &= f(n) + af\left(\frac{n}{b}\right) + a^2f\left(\frac{n}{b^2}\right) + \dots + a^{k-1}f\left(\frac{n}{b^{k-1}}\right) + a^kT\left(\frac{n}{b^k}\right) =
 \end{aligned}$$

$$n = b^k \Rightarrow k = \log_b n$$

$$\Theta = a^{\log_b n} + \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) = T(n).$$

$$T(n) = n^{\log_b a} + \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right).$$

3 cazuri:

a) $|I| > |II|$

$$f(n) = O(n^{\log_b a - \epsilon}) \quad \epsilon > 0$$

$$T(n) = \Theta(n^{\log_b a})$$

b) $|I| = |II|$

$$f(n) = \Theta(n^{\log_b a})$$

$$T(n) = \Theta(n^{\log_b a} \log n)$$

$$c) \exists T < \frac{n}{4} \text{ s.t. } f(n) = \Omega(n^{\log_2 c + \epsilon}) \quad (\epsilon > 0)$$

si $af\left(\frac{n}{b}\right) \leq cf(n)$.

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

Analiza amortizată a complexității

Analiza complexității: in cazul cel mai defavorabil

in cazul mediu

in cazul cel mai favorabil.

↳ amortizate

Exemplu: Stiva cu pop multiplu.

operări: push, pop, empty?, popm.

push: $e \times S \rightarrow S$

pop: $S \rightarrow S$

popm: $S \times \mathbb{N} \rightarrow S$

Complexitatea amortizată \rightarrow pe n operații

(push, pop - $O(1)$)

a) Calcul agregat

popm - $O(n)$

n -operații \rightarrow cel mai defavorabil

$$n \cdot O(n) = O(n^2)$$

Dacă împărțit pe n operații $O(n^2) / n = O(n)$

Se face de n ori succesiv $O(n)$

\rightarrow absurd! nu se pot scoate toate elementele

Algoritm: popm(K, S, s)

cât timp $\text{not}(\text{empty}(S))$ și $K > 0$

repetă

$$S \leftarrow \text{pop}; K \leftarrow K - 1$$

nr. operații = $\min(k, \dim(S))$.

Metoda creditelor

- push \rightarrow pun \pm leu costul operației de push + \pm leu costul operației de pop.
cost amortizat = 2.
- pop \rightarrow iau leul pus la push și plătesc costul
 $\text{cost}_{\text{pop}} = 0$.
- pop m (K) \rightarrow $\min(K, \dim(S))$ elem și plătesc costurile cu ce am pus la push
 $\text{cost}_{\text{popm}} = 0$.

Metoda potentialului

- metaforă \rightarrow o structură de date \rightarrow energie potențială.

Funcție de potențial ϕ la un anumit moment dat.

$$\phi(n) \geq 0.$$

$$\text{cost amortizat} = \text{costul real} + \phi(n) - \phi(n-1)$$

alegerea?

$\phi(n)$ = dimensiunea stivei la momentul n.

$$\begin{aligned}\text{cost amortizat}_{\text{push}} &= \text{cost real push} + \phi(n) - \phi(n-1) = \\ &= 1 + K + 1 - K = 2\end{aligned}$$

$$\begin{aligned}\text{cost amortizat}_{\text{pop}} &= \text{cost real pop} + \phi(n) - \phi(n-1) = \\ &= K + K - K - 1 = 0.\end{aligned}$$

$$\begin{aligned}\text{cost amortizat}_{\text{popm}} &= \text{cost real popm} + \phi(n) - \phi(n-1) = \\ &= \min(\dim(S_{n-1}), \dim(S_n)) - \dim(S_{n-1}) + \min(\dim(S_n), \dim(S_{n-1})) - \dim(S_n) = 0\end{aligned}$$

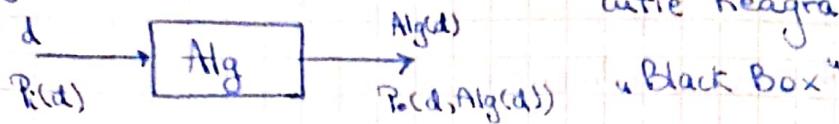
$$\left\{ \begin{array}{l} \dim < K \rightarrow \dim(S_{n-1}) \\ \dim > K \rightarrow \dim(S_n) \end{array} \right.$$

Exemplu Complex: \rightarrow complexitatea amortizată a heapului Fibonacci.

Demonstrarea corectitudinii algoritmilor

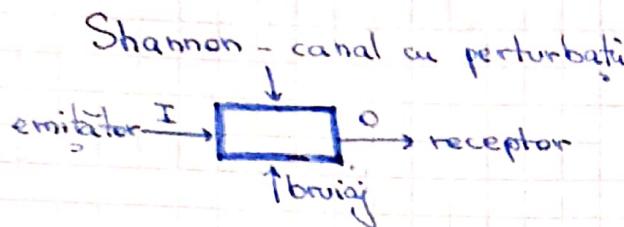
Verificarea că algoritmul proiectat (seris) se conformează cerințelor problemei specificate și problemelor formale.

Alg(d)



cutie neagră

"Black Box"



Modelle
Markov

P_i - predicat de intrare
 P_o - predicat de ieșire

Def.: Algoritmul Alg este parțial corect dacă $P_i(d)$ și $\text{Alg}(d)$ se termină $\Rightarrow P_o(d, \text{Alg}(d))$.

Def.: Algoritmul Alg este total corect dacă $P_i(d)$
 $\Rightarrow \text{Alg}(d)$ se termină și $P_o(d, \text{Alg}(d))$.

Teoremă: În general, nu se poate calcula corectitudinea totală a unui algoritm.

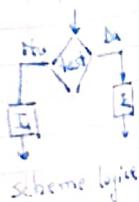
Verificarea corectitudinii - asertioni inductive (programare procedurală imparativă)

- inducție structurală

Programare procedurală - secvențe

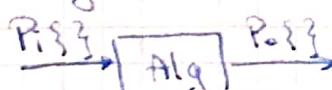
- decizii

- ciclu

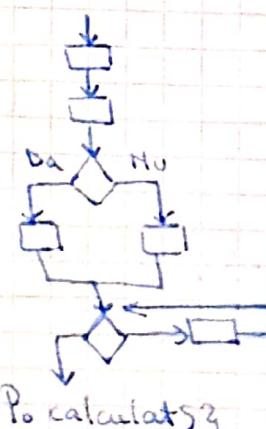


schemă logică

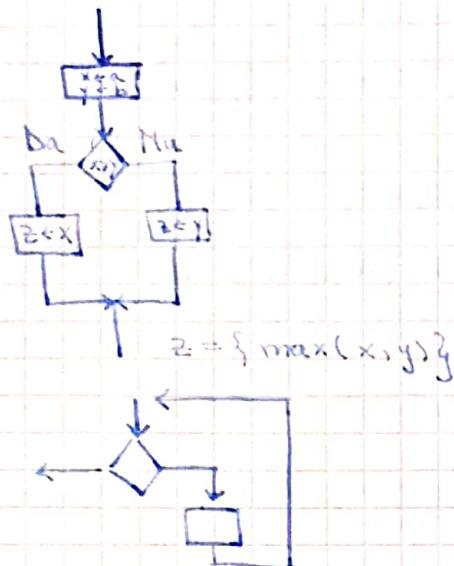
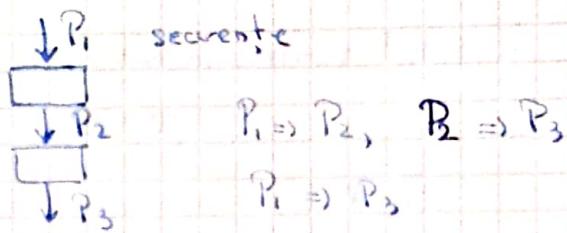
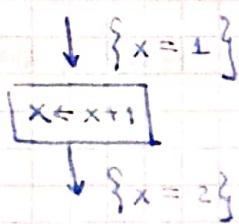
$\text{Alg} \rightarrow$ schemă logică



P_i calculat? P_o



P_o calculat?



La ciclare problema se complica
 foarte mult. Trebuie calculat invariantul
 ciclului.

Practic - definite P_i și P_o (logica matematică)

- se "trece" P_i prin program și se obține $P_{calculat}$

- se verifică dacă $(P_{calculat} \Rightarrow P_o)$ - teorema de demonstrat.

Pt. un program oricare P_i, P_o , invariant - probleme dificile.

La mijlocul anilor '70 a apărut programarea structurată și f.d.a.

Demonstrarea corectitudinii în contextul apariției f.d.a. (\mathbb{F} -algebră)

- se dă descrierea f.d.a.-orilor

- se dă un algoritm

- se verifică o proprietate - prin inducție structurală

\downarrow
 o generalizare a inducției
 matematice.

Cum putem generaliza inducția completă?

\mathbb{N} , p de demonstrat.

Demonstrăm $P(1)$

$\left. \begin{array}{l} P(1) \\ P(n) \Rightarrow P(n+1) \end{array} \right\} \Rightarrow \forall n \in \mathbb{N}, P(n).$

Multimea \mathbb{N}

$\xrightarrow{\text{generalizare}}$ A

Relație de ordine \subseteq $\xrightarrow{\text{generalizare}}$ $\# \subset$ - relație de ordine parțială.
($\forall m, n \in A$ avem $m \# n$ sau $m = n$)

Esential pentru inducție - nu sunt siruri infinit descrescătoare + celelalte, generalizare se păstrează.

(A, \leq) - multime bine ordonată (A).

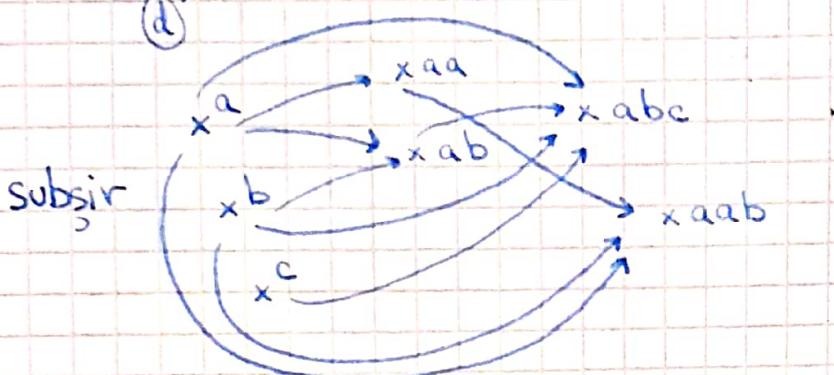
Ex.: (\mathbb{N}, \leq) , $(\{a, b, c\}^*, \text{subșir})$

Ex. negative: (\mathbb{Z}, \leq) , (\mathbb{R}, \leq) , $(\frac{1}{n}, \leq)$

a) $(\text{Arb}, \text{subarbore})$

b) c) $((\text{nil} \& \text{nil}) \& \text{ nil}) \& (\text{nil} \& \text{ nil})$

d)



$(\forall y, x, x \leq y \Rightarrow P(x) \Rightarrow P(y)) \Rightarrow \forall x P(x)$

$P(a), P(b), P(c)$

Inducție bine formată

f.d.a. op. - constructori nulari, recursivi

Verificarea corectitudini algoritmilor

Se poate face doar verificarea parțială, în general, a algoritmilor.

→ verificate prin aserționi inductive

→ inducția structurală - pleacă de la Σ algebre (stă la baza t.d.a.)

T.D.A. → sorturi

$\left\{ \begin{array}{l} \rightarrow \text{sorturi} \\ \rightarrow \text{operatori} \\ \rightarrow \text{axiome} \end{array} \right.$	$\rightarrow \text{constructori}$	naturi : $\rightarrow *$	
		$\left\{ \begin{array}{l} \text{recursivi} : x \times x \rightarrow * \\ \text{nerecursivi} : A_1 \times \dots \times A_k \rightarrow * \end{array} \right.$	*
			generali (cei alții)

t.d.a. algoritm vedem dacă verifică un predicat (P).

Inducția structurală → generalizare a inducției matematice

Inducție matematică $\xrightarrow[n_0]{}$

$$P(n_0), \forall n > n_0 \ P(n) \Rightarrow P(n+1).$$

$$\Rightarrow \forall n > n_0 \ P(n) \quad n, n_0 \in \mathbb{N}$$

- Inducția completă

$$[\forall n_1, \forall n_2 > n_1 : P(n_1) \Rightarrow P(n_2)] \Rightarrow \forall n \in \mathbb{N} \ P(n).$$

Inducția bine-formată

$[(\mathbb{N}, \leq)-\text{total ordonată}]$

(A, \preceq) , \preceq - ordine parțială pe A .

→ Nu există siruri infinit descrescătoare.

→ multime bine-ordonată

(A, \preceq) bine-ordonată

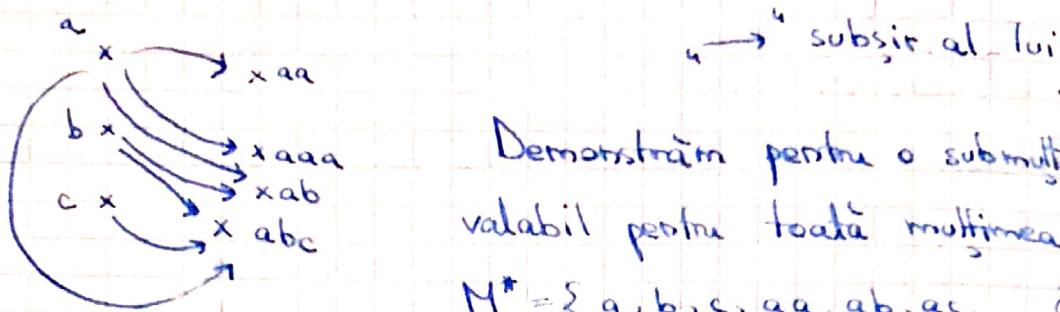
$$[\forall x \in A, \forall y > x, y \in A, x \neq y, P(x) \Rightarrow P(y)] \Rightarrow$$

$$\Rightarrow \forall x \in A, P(x).$$

În plus față de inducția completă : \mathbb{N} se întâlniește cu
A oarecare și \subset cu \preceq oarecare.

Exemplu: $M = \{a, b, c\}$

$(M^*, \text{subșir_al_lui}) \rightarrow \text{bine-ordonată}$



Demonstrăm pentru o submultime și e valabil pentru toată multimea.

$$M^* = \{a, b, c, aa, ab, ac, \dots\}.$$

Inducția structurală - caz particular al inducției bine-formate
 $A \rightarrow$ multimea instantelor a unei t.d.a.

Ex.: cozile de prioritate.

Relație de ordine - substructură

2. arbori binari

$A = \text{Arborei}$

Sorturi = arborei, elemente

operatori : $\text{noarbt} : \rightarrow \text{Arbore}$

$\text{arb} : \text{Arbore} \times \text{El} \times \text{Arbore} \rightarrow \text{Arbore}$

$\text{arivid?} : \text{Arbore} \rightarrow \{\text{adevărat}, \text{fals}\}$.

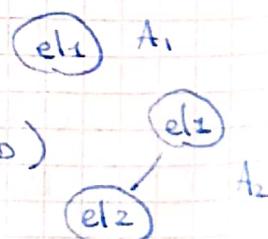
Exemple de arbori:

noarbt

$(\text{noarbt}, \text{el}_1, \text{noarbt})$

$((\text{noarbt}, \text{el}_2, \text{noarbt}), \text{el}_2, \text{noarbt})$

$A_1 < A_2$



Pentru t.d.a, în general, trebuie făcută verificare pentru toți constructorii.

Pentru t.d.a.

Fie P - predicatul de verificat

1) Pentru fiecare operator nular T se verifică P . ($P(O_n)$).

2) -ii- recursiv $O_r : A_1 \times A_2 \times \dots \times A_r \rightarrow T$ se arată că dacă $P(x) \Rightarrow P(y)$, $P(x) \Rightarrow P(O_r(a_1 \times a_2 \times \dots \times x))$.

3) — II — nerecursiv $P(O_n \cup (\dots))$ — aderărat.

1), 2), 3) — doar pentru constructori

Sinteză

notății matematice — liniare

— bidimensionale.

Inductie matematică :
 (\mathbb{N}, \leq)

$$\frac{\forall n \in \mathbb{N} \text{ s.t. } P(n_0), n_0 \in \mathbb{N}}{\forall n \in \mathbb{N}, n \geq n_0, P(n)}$$

Inductia completă:
 (\mathbb{N}, \leq)

$$\frac{\forall n \in \mathbb{N} \text{ s.t. } \forall m > n [P(m)]}{\forall n \in \mathbb{N}, P(n)}$$

Inductia bine-formată:
 (A, \leq)

$$\frac{\forall x \in A \text{ s.t. } \forall y \in A [P(x, y)]}{\forall x \in A, P(x)}$$

Inductia structurală:

(T, substructură)

T-constructori:

— nulari O_n

— recursivi O_n : $A_1 \times A_2 \times \dots \times T \times \dots \times T \rightarrow T$

— nerecursivi O_n : $A_1 \times \dots \rightarrow T$

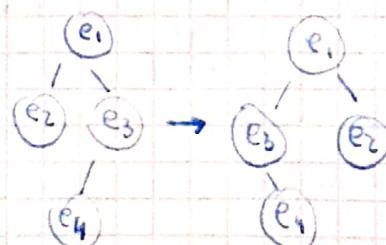
$$\frac{\begin{array}{l} \forall O_n, P(O_n) \\ \forall O_m, P(O_m) \end{array} \quad \forall i \quad \frac{[P(x_i)]}{P(O_r(a_1, a_2, \dots, x_i, \dots))}}{\forall x \in T, P(x)}$$

$x_i \rightarrow$ din domeniul de definiție al lui T

Ex. cu t.d.a. Arboresc.

Algoritm mirror(tarb, arb)

(arbore, arbore)



dacă arbvid? (arbore)

atunci arbore

altfel // arbore = arb(a₁, e, a₂)

arb(mirror(a₂), e, mirror(a₁))

sfârșit

Verific că este corect \rightarrow satisfacă proprietatea ~~de arbore~~

$$P(a) : \text{mirror}(\text{mirror}(a)) = a.$$

Verific prin inducție structurală:

Pentru operatori nulari

$$a = \text{novarb}$$

$P(\text{novarb})$ - aderărat

$$\text{mirror}(\text{mirror}(\text{novarb})) = \cancel{\text{novarb}} \text{ novarb}$$

\swarrow \searrow

novarb

Pentru constructori:

$$\text{mirror}(\text{mirror}(\text{arb}(a_1, e, a_2))) = \cancel{\text{arb}}(a_1, e, a_2)$$

$$= \text{mirror}(\text{arb}(\text{mirror}(a_2), e, \text{mirror}(a_1))) =$$

$$= \text{arb}(\text{mirror}(\text{mirror}(a_1)), e, \text{mirror}(\text{mirror}(a_2)))$$

Din ipoteza de inducție, dacă din $P(\text{mirror}(\text{mirror}(a_i))) \Rightarrow$

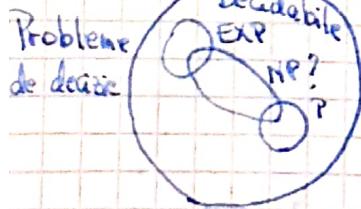
$\Rightarrow P(\cancel{\text{mirror}} \text{ arb}(a_1, e, a_2)) \Rightarrow$ \forall arbore din
Arbore: $P(\text{arbore}).$

NP-Compleitudine

↳ nedeterminist polinomial

↳ clasă de probleme de decizie

algoritmi asociați



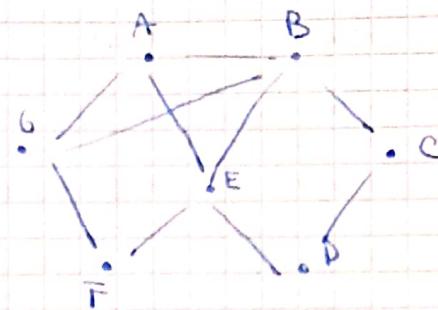
P \rightarrow probleme care se rezolvă determinist ("normal") cu complexitate polinomială

Ex: P \rightarrow ——— exponențială (ex: Hanoi).

S-a demonstrat că nu există un algoritm polinomial.

Exemplu: Ciclu Hamiltonian

Există un drum ciclic care trece prin toate nodurile grafului doar o dată?
→ {da, nu}



$$G = (N, A); A \subseteq N \times N$$

$$k = \text{card}(N) - \text{nr. de noduri din } N.$$

$$\#N \quad a_i \in N$$

$$P \left(\exists (a_1, a_2, \dots, a_n, a_1) \right)$$

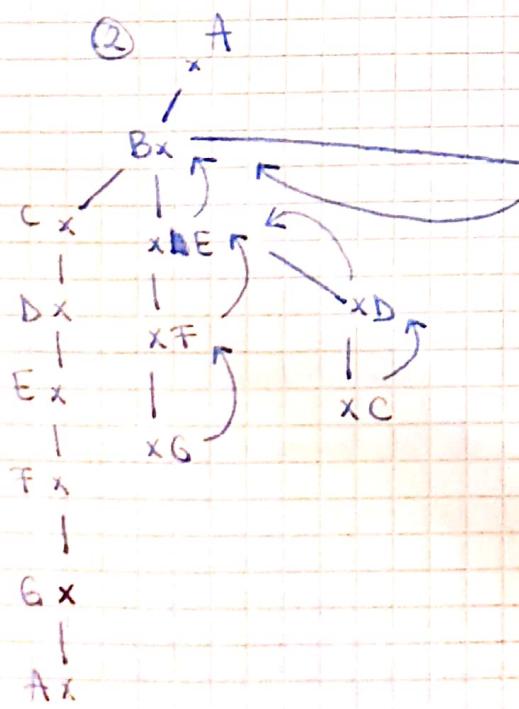
$$\forall i \in \overline{1, n-1}, (a_i, a_{i+1}) \in A, (a_n, a_1) \in A, \forall i, j, a_i \neq a_j.$$

Rezolvări: ① Generează și testează (toate permutările de noduri și testăm proprietatea P)
complexitate inaceptabilă — n!

② Backtracking

complexitate exponentială

③ ? complexitate polinomială



• foarte neficient

• complexitate exponentială
în general

Obs.: ca „noroc”, complexitate $O(\text{card}(N))$

Obs.: dacă am urea un „oracol”
complexitate $O(\#(N))$.

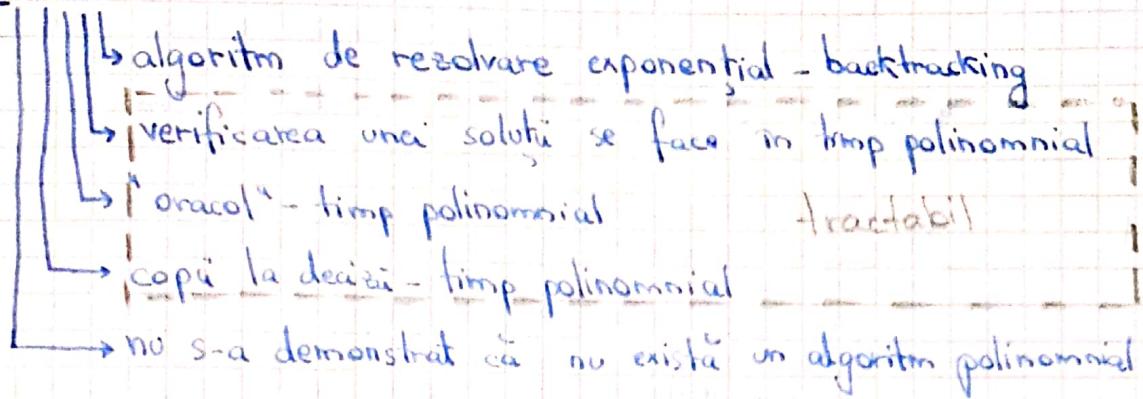
Obs.: Testarea dacă o searcă
de nr. noduri - se face
în timp polinomial.

Obs.: se lansează copy la fiecare

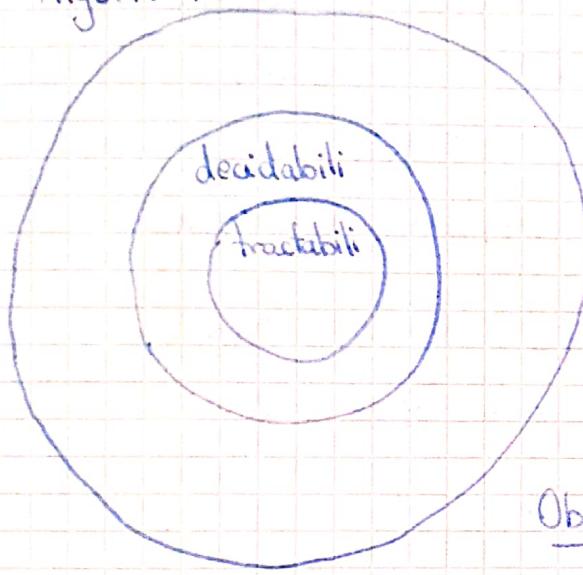
dezivie, care rulează în paralel
complexitate $O(\#(N))$.

NP - COMPLETITUDINE

Ex.: Cielo Hamiltonian



Algoritmi



Algoritm tractabil -
complexitate polinomială (n^k)
(timp constant)

Algoritm itractabil $O(n^k)$,
 k -constant.

Obs.: tractabil cu k mare, totuși
inaceptabil ($O(n^{100})$).

Algoritm nedeterminist

În plus față de algoritmi deterministi (reali):

- instrucțiunile: choice(x_1, \dots, x_k)

succes, fail

- algoritmi ideali, abstracti;

- probleme de decizie, care întorc $\{$ adevărat, fals $\}$ sau
 $\{$ succes, fail $\}$ (fail nu neapărat fals);

$y = \text{choice}(x_1, \dots, x_k)$

/ \ ... \

$y = x_1 \quad y = x_2 \quad y = x_k$

ruleată în paralel

- algoritmul continuă în k copii;

- până avem undeva fail sau succes;

- se termină în cazul în care se ajunge la fail pe toate rădăcinile;
- se termină cu fals, dacă nu s-a întâlnit succes și s-a

$$G = (N, A)$$

Algoritm nedeterminist - Ciclu Hamiltonian

N - CicluHamiltonian (N, A) {

$k \leftarrow \# N$ $y \leftarrow \text{choice}(N)$ anterior $\leftarrow y$

$i \leftarrow 2$ $ciclu \leftarrow \{y\}$ primul $\leftarrow y$

căt timp $i \leq k$ repetă $O(k-1)$

$y \leftarrow \text{choice}(A)$ // echivalent cu $y = \text{choice}(x_1, \dots, x_k)$

(dacă $(\text{anterior}, y) \notin A$ atunci fail)

pentru fiecare z din ciclu repetă

(dacă $y = z$ atunci fail $O(k)$)

ciclu \leftarrow ciclu $\cup \{y\}$ //

anterior $\leftarrow y$ $i \leftarrow i+1$

(dacă $y \in (\text{anterior}, \text{primul}) \in A$ atunci succes

atfel fail)

Complexitate angelică

↓

Complexitatea unui algoritm

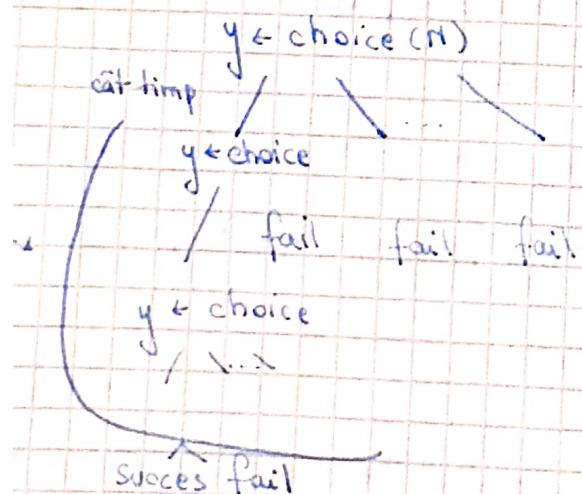
nedeterminist

- complexitatea primei rădăcini

terminată cu "succes"

- complexitatea celei mai lungi

în cazul în care întâlnește "fail";



Ansamblu $O(k^2)$ - polinomial.

Este tractabil.

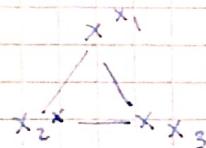
Dacă N -Circuit hamiltonian \rightarrow complexitate (angelică) polinomială

Def.: Clasa problemelor care se rezolvă cu un algoritm nedeterminist cu complexitate polinomială \rightarrow

Probleme nondeterminist polinomiale (NP)

Def.: P probleme care se rezolvă cu un algoritm determinist cu complexitate (reală) polinomială.

$$G = (\{x_1, x_2, x_3\}; \{(x_1, x_2); (x_2, x_3); (x_1, x_3)\})$$



$k=3$

$i=1$

$y \leftarrow \text{choice}$
↓ ↓ ↓
 $y=x_1$ $y=x_2$ $y=x_3$
 $\text{ant} = x_1$ $\text{ant} = x_2$ $\text{ant} = x_3$
 $\text{ciclo} = \{x_1\}$ $\text{ciclo} = \{x_2\}$ $\text{ciclo} = \{x_3\}$
 $\text{primit} = x_1$ $\text{primit} = x_2$ $\text{primit} = x_3$

↓ ↓
 $y=x_1$ $y=x_2$ $y=x_3$
fail fail $\text{ciclo} = \{x_1, x_2\}$
 $i=2$
 $\text{ant} = x_2$
 $y \leftarrow \text{choice}$
↓ ↓
 $y=x_1$ $y=x_2$ $y=x_3$
fail fail $\text{ciclo} = \{x_1, x_2, x_3\}$

$i=3$

$\text{ant} = x_3$

succes

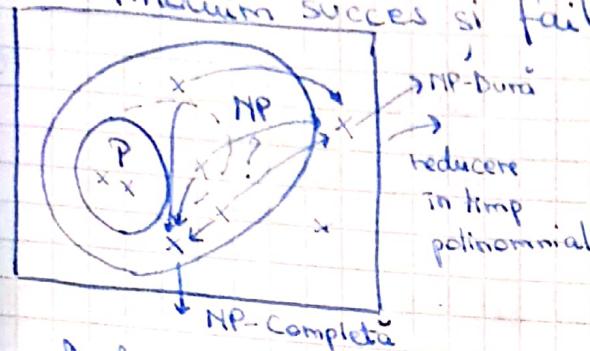
Teoremă: P ⊂ NP

Demonstratie:

Fie $\text{Alg} \in P$, Alg pentru probleme de decizie

\Rightarrow întoarce undeva return adevărat și altundeva return fals.

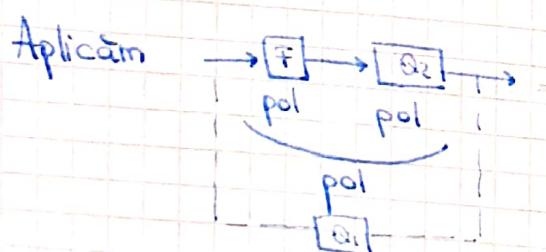
Înlocuim succes și fail. $\Rightarrow \text{Alg} \in NP$.



Def.: Fie problemele Q_1, Q_2 - două probleme spunem că Q_1 se reduce în timp polinomial la Q_2 dacă avem o transformare F cu complexitate polinomială care transformă intrările lui Q_1 și iesirile lui Q_2 sunt aceleasi ca ale lui Q_1 .
In intrările lui Q_2
 $Q_1 \leq_p Q_2$ - notație

Teoremă: Dacă $Q_2 \in P$ și $Q_1 \leq_p Q_2$, atunci $Q_1 \in P$.

Demonstratie:



Def.: O problemă este NP-dificilă (NP-dură, NP-hard), dacă la ea se reduce în timp polinomial toate problemele NP.

Def.: O problemă este NP-completă dacă este NP-dură și aparține lui NP.

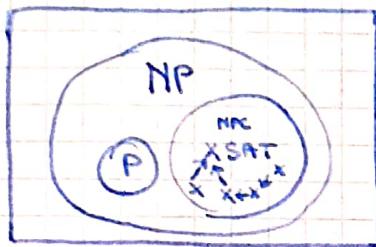
$(Q \in \text{NPC}) \Leftrightarrow (\forall R \in \text{NP}, R \leq_p Q \rightarrow Q \in \text{NP})$

Teorema (Cook)

\exists o problemă NP-Completa \rightarrow SAT (problemă satisfiabilității formulelor de calcul propositional în forma normală conjunctivă).

Demonstratie: Dacă văd căre.

\forall Algoritm $\xrightarrow{\text{NP}}$ + set de date = Problemă pot să aigneze F



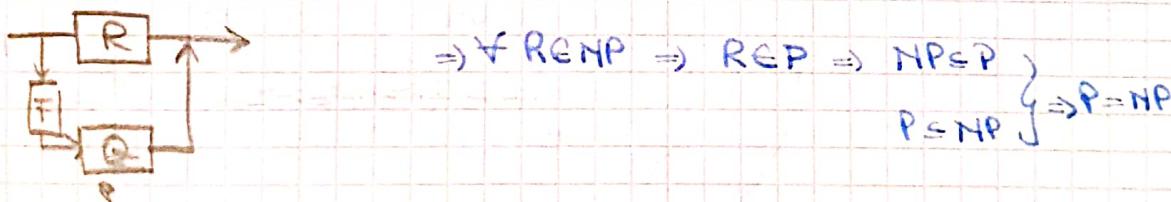
dică $\leq_p 3\text{SAT} \leq_p \text{SAT}$

Teorema

Tie $Q \in \text{NPC}$. Dacă $Q \in \text{P} \Rightarrow \text{P} = \text{NP}$.

Demonstratie:

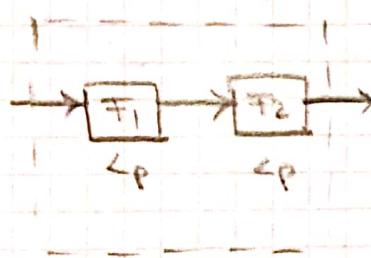
$Q \in \text{NPC} \Rightarrow \forall R \in \text{NP}, R \leq_p Q \quad \left. \begin{array}{l} \\ Q \in \text{P} \end{array} \right\} \Rightarrow \forall R \in \text{NP}$.



Practic, ca să arăt că o problemă $Q \in \text{NPC}$ trebuie să:

a) arăt că $\in \text{NP}$ - scriu un algoritm nedeterminist polinomial pentru Q ;

b) găsește $R \in \text{NP}$ și o transformare polinomială $Q \leq_p R$. (\leq_p este tranzitivă)



Arătăm că problema ciclu este NP-completă.

Def.: Fie $G = (N, A)$ un graf neorientat $A \subseteq N \times N$.

Suntem că $C = (N_1, A_1)$ este o K -clică

în G , dacă $N_1 = N$, $A_1 \subseteq A$, $\text{card}(N_1) = K$ și

C -complet ($\forall n_1, n_2 \in N_1, (n_1, n_2) \in A_1$).

K -clică = subgraf complet al lui G .

Trebuie să arătăm că G are o K -clică.

a) Algoritm: N -clica (N, A, K) {

 set $\leftarrow \{\}$

 pentru $i \leftarrow 1$ la K repeta

$y \leftarrow \text{choice}(N)$

 dacă $y \in \text{set}$ atunci fail

 altfel $\text{set} \leftarrow \text{set} \cup \{y\}$

 pentru fiecare x din set repeta

 pentru fiecare y din set repeta $O(K^2)$

 dacă $(x, y) \in A$ atunci fail

succes

b) Consider problema 3CNF (care s-a demonstrat că e NPC)

Suntem: $F(x_1, x_2, \dots, x_k) = f_1 \vee f_2 \vee \dots \vee f_m$
literale $f_i = l_1^1 \wedge l_2^2 \wedge \dots \wedge l_n^m$

Px.: $F(x_1, x_2, x_3) = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge \dots$

Satisfiabilitate = 3 o assignare de valori pt.

x_1, x_2, \dots, x_k a.t. $F(x_1, \dots, x_k) = 1$?

3CNF - cat particular SAT in care fiecare termen
ti are 3 literali.

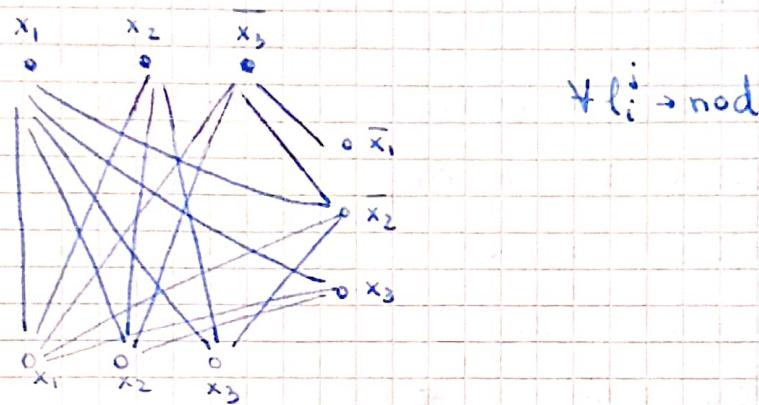
$$F(x_1, x_2, \dots, x_n) = t_1 \wedge t_2 \wedge \dots \wedge t_m$$

$$t_i = l_i^1 \vee l_i^2 \vee l_i^3, \quad l_i^j = x_p \text{ sau } \bar{x}_p$$

Ex.: $F(x_1, x_2, x_3) = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

? Transformare de la K-clica \rightarrow 3CNF.

Construiesc un graf pt. F.



Arce între termeni diferenți nu o să fie.

K-clica



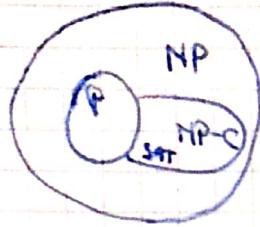
$$F(l_1^{i_1}, l_1^{i_2}, l_1^{i_3}) = 1$$

$$x_1 \rightarrow \bar{x}_2 \rightarrow x_1 \rightarrow \text{clica}$$

$$x_1 \rightarrow \bar{x}_2 \rightarrow x_3$$

$$F(1, 0, 1) = 1$$

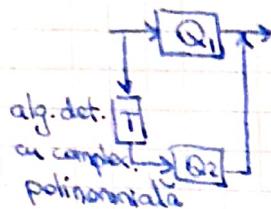
NP - Complexitate



Problema - algoritm

Probleme de decizie

← p reducere în timp polinomial



Demonstrare că o problemă este NP-Completa:

- arătăm că e în NP;
- se găsește o problemă în NP-C care se reduce în timp polinomial la ea;

Ex.: problema \Leftrightarrow clică

- a) am scris un program nedeterminist pentru clică;
- b) luăm problema 3 cîr.

\Leftrightarrow CNF

$$F(x_1, \dots, x_n) = t_1 \wedge t_2 \wedge \dots \wedge t_q$$

$$t_i = l_i^1 \vee l_i^2 \vee \dots \vee l_i^n$$

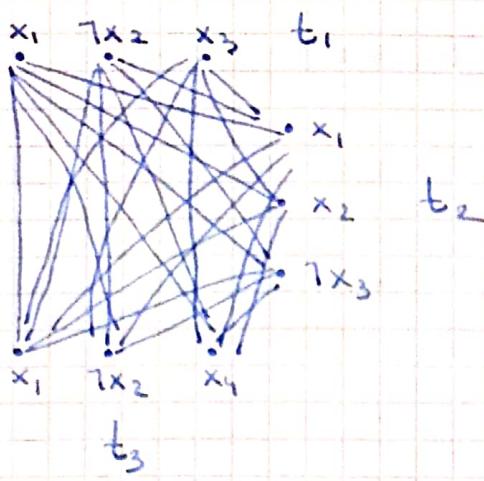
$$l_i^j = x_i \text{ sau } \overline{x}_i$$

3CNF - r=3.

$$\begin{aligned} F(x_1, x_2, x_3, x_4) &= (x_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_4) \wedge (x_1 \vee x_2 \vee \overline{x}_3) \\ &\quad (x_1 \vee \overline{x}_2 \vee x_4) \end{aligned}$$

CNF - satisfiabilitatea lui F ($\exists (x_1, x_2, x_3, x_4)$ a.i.

$$F(x_1, x_2, x_3, x_4) = 1$$



Ce trebuie demonstrat?

T este polinomială (determinist)

Algoritm

$T_{3CNF_la_dică}(F, G)$

$N \in \{\}$; $A \in \{\}$

pentru fiecare t_i din F repetă

pentru fiecare f_{ij}^k din t_i repetă $\| = 3$
 $\square \quad N \leftarrow N \cup \{var - f_{ij}^k\}$

pentru fiecare n_1 din N repetă $\| n_1 = var - f_{ij}^{j_1}$

pentru fiecare n_2 din N repetă $\| n_2 = var - f_{ij}^{j_2}$
 $\square \quad$ dacă i, j, j_1, j_2 nu sunt aceeași variabile o dată
 $\square \quad$ atunci $A \leftarrow A \cup S(n_1, n_2)\}$ negabă o dată nu

Sfârșit

$T_{3CNF_la_dică}$ are complexitate polinomială

Trebuie să arătăm că - dacă F - satisfacabilă \Rightarrow

$\Rightarrow \exists$ o dică în G .

F - satisfacibilă $\Rightarrow \exists (x_1, \dots, x_n)$ a.s. $F(x_1, \dots, x_n) = 1$.

$\Rightarrow \forall i \quad t_i = 1$ (fiecare F din CNF)

$\Rightarrow \forall i \quad \exists j$ a.t. $p_i^j = 1$

$\Rightarrow \exists j (t_1, t_2, \dots, t_k)$

x_i sau \bar{x}_i

a.t. $F(t_1, \dots, t_k) = 1$

Intre variabilele adese vom avea

sigur aree pt. că sunt în temeni diferiti

și nu există un x_i și \bar{x}_i în același timp

Teorema (Cook)

SAT $\in P \Leftrightarrow P = NP$

(este echivalent cu a spune că toate problemele din NP se reduc în timp polinomial la SAT).

$SAT \rightarrow F(x_1, \dots, x_n)$ în CNF ; $\exists (x_1, \dots, x_n)$ a.t.

$F(x_1, x_2, \dots, x_n) = 1$?

Demonstratie.

$P = NP \Rightarrow SAT \in P$

Arătăm că $SAT \in NP \Rightarrow$ scrie un algoritm nedeterminist
pt. SAT și arătăm că este polinomial

Algoritm SAT(F)

(pentru $i \leftarrow 1$ la K repeta

$v \leftarrow \text{choice}(0, 1)$

 pentru fiecare t_i din F repeta

 satisfacut = 0

 pentru fiecare p_i^j din t_i repeta

 dacă ($p_i^j = 1$ și $v - l_j^{i, p_j}$ este negativ) sau

 ($p_i^j = 0$ și $v - l_j^{i, p_j}$ este pozitiv)

(atunci satisfăcut \Leftarrow 1
 dacă satisfăcut = 0 atunci fail
 \square

succes

Demonstratie: SAT $\in P \Rightarrow P = NP$

Arătăm, de fapt, că orice problemă din NP se reduce la SAT.

Orice problemă \rightarrow $\boxed{\text{Alg; } D \xrightarrow{T} F(x_1, \dots, x_n)}$ \otimes

Dacă Alg; D se termină cu succes, atunci F este satisfabilă.

Orice algoritm (program) poate fi descris prin 2 grafuri:

- de flux de date;
- de flux de control;

\Rightarrow Alg; D \xrightarrow{T} $(g_1) \wedge (g_2) \wedge \dots \wedge (g_t)$

unde sunt flux de control și
unde flux de date.

Precizări: Considerăm ca instrucțiuni:

→ atribuire;

→ salt - neconditionat (go to etichetă)

conditionat (if \vee go to etichetă)

→ choice

→ success

→ fail

\otimes de fapt, $Q \leq_p SAT$

- nu avem matrice (vectori);

- variabilele din Alg au să fie (x_i)

în F avem $B(x, i, t)$ ca variabile \downarrow din Alg
 $B(x, i, t)$ - bitul i al var.
 \uparrow la momentul t
 $= 1$

-avem l instructiuni în alg

Algoritmii - $O(n^2)$ unde n este dimensiunea datelor de intrare D.

- în F al doilea set de variabile este

$S(i,t)$ - instrucția i este executată la momentul t,

$$\overline{S(i,t)} \rightarrow \text{--} \rightarrow h\nu \rightarrow \text{--}$$

$$F_{Alg; D} = \text{Init} \wedge \text{Start} \wedge \text{Secr} \wedge \text{Succ} \wedge \text{Eval} \wedge \text{Stop}$$

flux de
date / flux de
control

A handwritten diagram on graph paper. It features two parallel blue lines slanted upwards from left to right. A single blue curve starts below the bottom line and ends above the top line, intersecting both at approximately their midpoints.

Alte precizări: - o variabilă booleană \rightarrow bitul \neq dă valoarea

if y goto j

$$B(y_s, t)$$

- toate variabilele sunt initial 0, cu excepția unor parametri care sunt initial $\neq 0$ și corespund întrărilor;

Init \rightarrow fragment de formulă CNF care conține valoriile

futuror variabilelor la momentul $t = 2$

Start - doar instrucțiunile și se execută la momentul și

$$\text{Start} = s(1,1) \wedge \bigwedge_{i=2}^l \overline{s(i,i+1)}$$

Secr- la f moment de timp se execută doar o instrucție

$$\begin{aligned}
 & [S(1,t) \vee S(2,t) \vee \dots \vee S(e,t)] \wedge \overline{s(i,t) \wedge s(j,t)} = \\
 & = \left[\bigvee_{\substack{i=1 \\ t=1,2}}^l S(i,t) \right] \wedge \bigwedge_{\substack{i=1,2 \\ j=1,2 \\ i \neq j}} \left[\overline{s(i,t)} \vee \overline{s(j,t)} \right]
 \end{aligned}$$

Succ - fluxul de control - succesiunea pasilor execuție

- fără salt
- goto etichetă
- if(x) goto etichetă

- success
- fail

$$s(i, t) \Rightarrow s(i+1, t+1) - \text{CNF}$$

$$s(i, t) \Rightarrow s(\text{etichetă}, t+1)$$

$$[s(i, t) \wedge x \Rightarrow s(\text{etichetă}, t+1)] \vee [s(i, t) \wedge \neg x \Rightarrow \\ \Rightarrow s(i+1, t+1)]$$

$$s(i, t) \Rightarrow s(i+1, t+1) \equiv \neg \overline{s(i, t)} \vee s(i+1, t+1)$$

$$A \Rightarrow B$$

$$\neg A \vee B$$

j.v)

Teorema lui Cook

- SAT \in NPC

$P = NP \Leftrightarrow SAT \in P$

$Alg, D \rightarrow F_{Alg, D}$ CHF

$F_{Alg, D}$ satisfiabilă dacă Alg, D se termină cu succes.

$F = \text{Init} \wedge \text{Start} \wedge \text{Secr} \wedge \text{Sucr} \wedge \text{Eval} \wedge \text{Stop}$

$$\text{Sucr} = s(i, t) \Rightarrow \text{urm}(i, t) = s(i, t) \vee \text{urm}(i, t+1)$$

$$\text{urm}(i, t) =$$

a) nu este instrucțiune de control

$$s(i+1, t+1)$$

b) goto j

$$s(j, t+1)$$

c) if(x) goto j

$$[B(i, x, t) \wedge S(j, t+1)] \vee [B(i, x, t) \wedge \neg S(i+1, t+1)]$$

d) succes sau fail

$$s(i, t+1).$$

$$\text{Eval} = s(i, t) \Rightarrow \text{Ev}(i, t+1) = \overline{s(i, t)} \vee \text{Ev}(i, t+1)$$

$$\text{Ev}(i, t+1) \Leftarrow$$

a) i nu modifică vreo variabilă

Trebuie introduse axiome de cadru ("frame axiom") -

spun ce nu se modifică și ce

$$\bigwedge_{x \in \Sigma, i=1..n} [B(i, x, t) \wedge B(i, x, t+1)] \vee [\overline{B(i, x, t)} \wedge \overline{B(i, x, t+1)}$$

b) $x \leftarrow \text{choice}(y_1, y_2, \dots, y_n)$

$$\text{Ev}(i, t+1) = \bigwedge_{\substack{\forall \text{var} \\ \neq x}} \bigwedge_{j=1..n} [B(j, x, t) \wedge B(j, x, t+1)] \vee$$

$$[B(j, x, t) \wedge \overline{B(j, x, t+1)}]$$

$$\wedge \bigvee_{r=1, K} \bigwedge_{j=1, w} [B(j, y_r, t) \wedge B(j, x, t+1)] \vee \\ \vee [\overline{B(j, y_r, t)} \wedge \overline{B(j, x, t+1)}]$$

c) atribuire

- la calcule aritmetice - extrem de complicat

- luăm 2 caiuri mai simple

$$x \leftarrow y \quad x \leftarrow y \wedge z \xrightarrow{\text{si logic}}$$

$$\star \wedge \bigwedge_{j=1, w} [B(j, y, t) \wedge B(j, x, t+1)] \vee [\overline{B(j, y, t)} \wedge \overline{B(j, x, t+1)}]$$

$$x \leftarrow y \wedge z \quad x^{t+1} \Rightarrow y^t \wedge z^t \quad \frac{x^{t+1}}{(y^t \wedge z^t)} \vee x^{t+1}$$

$$y^t \wedge z^t \Rightarrow x^{t+1}$$

$$\star \wedge \bigwedge_{j=1, w} \overline{B(j, x, t+1)} \vee [\overline{B(j, y, t)} \wedge \overline{B(j, z, t)}] \wedge \\ \wedge [B(j, y, t) \vee B(j, z, t) \vee B(j, x, t+1)]$$

$$\text{stop} = \bigvee_{\substack{i=\text{toate} \\ \text{instructiunile} \\ \text{succes}}} S(i, p(n))$$

• F construit în timp polinomial

$$\text{Alg}(x, y)$$

1. x=choice(x, y)

2. if(x) goto q

3. fail

4. succès

$$D: (x = z, y = 0)$$

$$\text{INIT} = B(1, x, 1) \wedge \overline{B(1, y, 1)}$$

$$\text{START} = S(1, 1)$$

$$\text{STOP} = S(4, 3)$$

$$\text{SECV} = \bigwedge_{t=1, 2} [S(1, t) \vee S(2, t) \vee S(3, t) \vee S(4, t)] \wedge [\overline{S(1, t)} \vee \overline{S(2, t)}] \wedge$$

$$\wedge (\overline{s(1,t)} \vee \overline{s(3,t)}) \wedge (\overline{s(2,t)} \vee \overline{s(3,t)})$$

$$\text{succ} = \bigwedge_{t=1,2} [\overline{s(1,t)} \vee s(2,t+1)] \wedge [B(1,x,t) \vee \overline{S(2,t)}] \wedge \\ \wedge s(4,t+1) \vee B(1,x,t) \wedge S(3,t) \wedge \\ \wedge [\overline{s(3,t)} \vee s(3,t+1)] \wedge \\ \wedge [s(4,t) \vee s(4,t+1)]$$

$$s(1,t) \Rightarrow s(2,t+1)$$

$$s(2,t) \Rightarrow$$

$$s(3,t) \Rightarrow s(3,t+1)$$

**

$$\text{eval} = \overline{s(1,t)} \vee (B(1,x,t) \wedge \overline{B(1,x,t+1)} \vee \overline{B(1,y,t)} \wedge \\ \wedge \overline{B(1,x,t+1)} \vee B(1,y,t) \wedge B(1,y,t+1) \wedge \\ \wedge [B(1,y,t) \wedge \overline{B(1,y,t+1)}])$$

$$\wedge \overline{s(2,t)} \vee \dots$$

$$\overline{s(3,t)} \vee$$

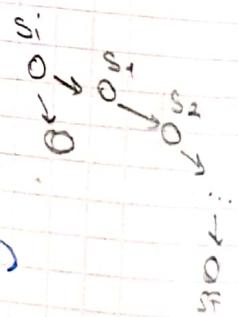
**

0 ierarhie spatiu-temp a algoritmilor

Complexitatea algoritmilor ..

Idee \rightarrow Alg.D $\xrightarrow{\text{manif}}$ spatial stabilor
 $\xrightarrow{\text{TIME(f)}}$ TIME(f)
 $\xrightarrow{\text{SPACE(f)}}$ {da, nu} \downarrow state accessibility

GAP (Graph Accessibility Problem)



executia cu succes a lui $\#$ Alg.D \Rightarrow există o cale de la s_1 la s_3

$\text{Alg}_Q^T(f) = \{ \text{Alg} \mid \text{Alg. pt. problema } Q \text{ cu complexitate temporală } O(f) \}$.

T-Timp

$\text{Alg}_Q^S(f) = \{ \text{Alg} \mid \text{spatială } O(f) \}$.

$\text{TIME}(f) = \{ Q \mid \text{Alg}_Q^T(f) \neq \emptyset \}$

$\text{SPACE}(f) = \{ Q \mid \text{Alg}_Q^S(f) \neq \emptyset \}$

$$P = \text{PTIME} = \text{TIME}(K^n)$$

PSPN...

NP TIME , NPSPACE

LOGTIME , LOGSPACE

NLOGTIME , NL LOGSPACE

$\text{P TIME} \subseteq \text{NP TIME}$

$\text{PSPACE} \subseteq \text{NPSPACE}$

$\text{LOGTIME} \subseteq \text{NLOGTIME}$

$\text{LOGSPACE} \subseteq \text{NL LOGSPACE}$

EXPTIME

EXPSPACE

Ierarhie spatiu-temp a algoritmilor

$\text{Alg}_Q^T(f)$, $\text{Alg}_Q^S(f)$, $N - \text{Alg}_Q^T(f)$, $N - \text{Alg}_Q^S(f)$.

TIME(f), SPACE(f), NTIME(f), NSPACE(f).

PSPACE, NPSPACE, NPTIME, NPSPACE

LOGTIME, LOGSPACE, NLOGTIME, NLOGSPACE

EXPTIME, EXPSPACE

PTIME \subseteq NPSPACE

PSPACE \subseteq NPSPACE

LOGTIME \subseteq NLOGTIME

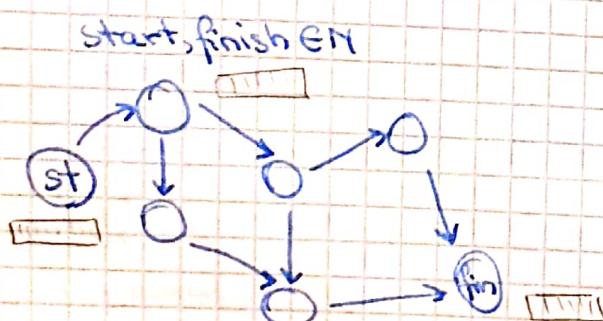
NLOGSPACE \subseteq NLOGSPACE

GAP \leq^T | determinist
 \rightarrow vom genera spațiul stăriilor
 complexitate
 să găsim cale în spațiul stăriilor
 de la start la finish

Spațiul stăriilor

graf $G = (N, A, \text{start}, \text{finish})$

$A \subseteq N \times N$



Problema accesibilității în spațiul stăriilor

Există o cale de la start la fin în ~~G = (N, A, st, fin)~~?

GAP-S → care este complexitatea spațială a lui GAP în cadrul determinist

Scriem un algoritm mai ușor de analizat din punct de vedere spațial.

Algoritm GAP-S (G)

functia verif(G, i, j, lg)

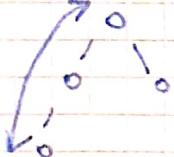
(dacă $i=j$ atunci intoarce adevărat

(dacă $lg=1$ și $(i,j) \in A$ atunci intoarce adevărat

④ (dacă ~~verif(G, i, k, $\frac{lg}{2}$) și verif(G, k, j, $\frac{lg}{2}$)~~ ^{maxim n fiecare \rightarrow spațiu ocupat $log n$} intoarce adevărat

④ pentru $k=i+1$ la j repetă

complexitatea spațială

max $\rightarrow \log n$  $\log n =$ spațiu ocupat pe o ramură

Complexitatea spațială a lui GAP-S este $O(\log^2 n)$

GAP \in SPACE $((\log n)^2)$

Algoritm GAP-T (G, st, fin) // clasic algoritm de parcurgere

în adâncime

nod. explorat \leftarrow ^{da} nu

pentru fiecare nod din N

$\Theta(n)$ $\# n = \# N$

repetă nod. explorat \leftarrow nu

explorează (st)

(dacă fin. explorat = da atunci intoarce adevărat

altfel intoarce fals

functia explorează (i)

i. explorat \leftarrow da

pentru fiecare (i, j) din A
 repeta daca j . explorat = nu
 atunci exploreaza (j) } $O(a)$, $a = n^2$

$O(a+n)$

$a = \text{maximum } n^2$

$\text{GAP} \in \text{TIME}(n^2) = \text{PTIME}$.

Cazul redeterminist

$\text{NGAP}(G, \text{st}, \text{fin}, n)$

$\text{nod} \leftarrow \text{st}$

$lg \leftarrow 0$

căt timp $lg \leq n$ repeta

$\text{urm} \leftarrow \text{choice}(n)$

daca $(\text{nod}, \text{urm}) \notin \text{dunici fail}$

$\text{nod} \leftarrow \text{urm}$

$lg \leftarrow lg + 1$

daca $\text{nod} = \text{fin}$ atunci succes altfel fail

st "

o BK

$\left\{ \begin{array}{l} \text{NGAP} \in \text{NPTIME} \\ \text{NGAP} \in \text{NLLOGSPACE} \end{array} \right.$

Alg o \rightarrow
GAP

Lemă: Fie Alg , un algoritm nedeterminist limitat (cu complexitate spațială $O(f(n))$ cu $f(n) \geq \log n$). Construirea spațială a lui Alg se poate face cu o complexitate spațială temporană $O(2^{f(n)})$, unde k - constantă.

Demonstratie:

Ce conține un nod? \rightarrow instrucțiunile - m

\rightarrow intrările (datele) - n

\rightarrow spațiu de adrese - limitat de $f(n)$

\rightarrow memoria - bitii $2^{f(n)}$.

Algoritmul construiește (Q , stofin)

pentru $i \in 1$ la m repeta $\left\{ \begin{array}{l} N \leftarrow \emptyset \\ A \leftarrow \emptyset \end{array} \right.$

pentru $j \in 1$ la n repeta

pentru $k \in 1$ la $f(n)$ repeta

pentru $e \in 1$ la $2^{f(n)}$ repeta

$N \leftarrow N \cup \{(i, j, k, e)\}$

pentru n_1 din N repeta

pentru n_2 din N repeta

dacă există în Alg tranziție între n_1 și n_2

atunci $A \leftarrow A \cup \{n_1, n_2\}$

$$st \leftarrow (1, 0, 0, 0)$$

$$fin \leftarrow (0, 0, 0, 2^{f(n)})$$

Complexitatea temporană
noduri $m \cdot n \cdot f(n) \cdot 2^{f(n)}$

$$\text{arce } (2 \cdot f(n))^2 = (2^2)^{f(n)} = k^{f(n)}$$

\Rightarrow construcția GAP are complexitate temporala $O(kf(n))$.

Teorema: Fie un algoritm care are $O(f(n))$ complexitate spațială nedeterminist. Atunci

$$N\text{-SPACE}(f) \Rightarrow \text{TIME}(cf).$$

Demonstratie:

$$\begin{array}{ccc} Q & \xrightarrow{\text{construire}} & (G, st, fin) \quad \text{Construire} \in \text{TIME}(kf(n)) \\ & \searrow & \downarrow \blacksquare \quad \text{GAP - T} \in \text{TIME}(n^2). \end{array}$$

În total avem complexitate temporala

$$Q(kf(n))^2 = O((k^2)f(n)) = O(cf(n)).$$

$$\Rightarrow Q \in N\text{-SPACE}(f) \Rightarrow Q \in \text{TIME}(c^t).$$

Cordan: $N\text{-LOGSPACE} \subseteq \text{PTIME}$

$$f(n) = k \log n$$

$$cf(n) = Ck \log n = n^{k \log e} = n^c.$$

$$N\text{-LOGSPACE} \Rightarrow \text{TIME}(n^c) = \text{PTIME}$$

Lemă: — II — spațială $\leq O(f(n))$.

Același algoritm cu diferența că $N \notin A$ se stochează într-o memorie externă.

Inlocuim $N \in N \cup \{(i, j, k, e)\}$ cu

serie (i, j, k, e)

$N \in A \cup \{(n_1, n_2)\}$

serie (n_1, n_2)

n_2 din N repetă cu cîteva n_1

Complexitatea spațială în al doilea caz va fi deată de ce variabile avem

$m, n, f(n), 2^{f(n)}$ valori maxime

\hookrightarrow nr. de instr. L dominant $O(\log_2 f(n)) = O(f(n))$.
(const)

Teorema: Fie A_{lg} cu $O(f(n))$ spațială nedeterminist

$Q \in \text{NSPACE}(f) \Rightarrow Q \in \text{SPACE}(f^2)$.
(A_{lg}, D)

Demonstratie.

$$q = k f(n)$$

$$O((\log n)^2) = O((\log k f(n))^2) = \Theta(\cancel{\log(k f(n))})$$

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

$$= O((f(n) \log K)^2) = O(f^2(n))$$

$\Rightarrow Q \in \text{NSPACE}(f) \Rightarrow Q \in \text{SPACE}(f^2)$.

Corolar: $\text{NPSPACE} = \text{PSPACE}$.

f cu complexitate polinomială $f = O(n^k)$

$\text{NPSPACE} \Rightarrow \text{PSPACE}$

$\Rightarrow \text{NPSPACE} \subseteq \text{PSPACE}$

Aren, $\text{PSPACE} \subseteq \text{NPSPACE}$

$\left. \begin{array}{l} \\ \end{array} \right\} \Rightarrow \text{NPSPACE} = \text{PSPACE}$

$$\Downarrow$$
$$O((n^k)^2) = O(n^{2k})$$

$\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{PTIME} \subseteq \text{NPSPACE}$? $\text{PSPACE} \subseteq \text{NPSPACE}$

Teorema: $\text{NPSPACE} \subseteq \text{PSPACE}$

Algoritmi de aproximare $i \in I$ NPC - probleme de decizie $O(2^n)$ NPH - probleme de optimciclu hamiltonian \leftrightarrow comis voiajorExemplu: (NPHard)

Alocarea de resurse

 n procese, p procesoare (pan) $t_1, t_2, \dots, t_n \rightarrow$ procese $d_1, d_2, \dots, d_n \rightarrow$ durata lor

cerintă: timp minim

2 variante de algoritmi de aproximare:

① la întâmplare, aleator

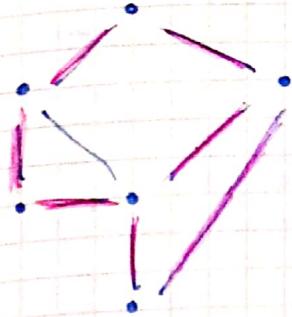
- timpul total $\leq 2 * \text{timpul optim}$ ② sortare descrescătoare după d_i și se pun în ordine- timpul total $\leq \frac{3}{2} * \text{timpul optim}$

Factor de aproximare (meren supraunitar)

Def.: Fie o problemă Q și I - domeniul valorilor de intrare. Denumim factor de aproximare

 $S(n) = \max_{i \in I} \frac{t_p(n)}{t_p^*(n)}$ n - dimensiunea intrării $S(n) = \max_{i \in I} \frac{t_p^*(n)}{t_p(n)}$ pentru probleme de minim, și $S(n) = \max_{i \in I} \frac{t_p^*(n)}{t_p(n)}$ pentru problemele de maxim.Eroarea relativă $E(n) = \frac{|t_p(n) - t_p^*(n)|}{t_p^*(n)}$.Se poate folosi și (pentru calcul când valoarea nu depinde de n): $S = \frac{t_p}{t_p^*}$ (minim), $S = \frac{t_p^*}{t_p}$ (maxim).

Exemplu: Ciclu Hamiltonian

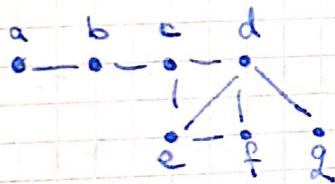


$$S = \frac{\lg k}{\lg 2} = \frac{7}{6} = 1, \dots$$

Problema acoperirii cu noduri a unui graf

Fie $G = (N, A)$ reorientat. Spunem că $M \subseteq N$ este o acoperire cu noduri a lui G dacă $V(u, v) \in A$ avem $v \in M$ sau $v \notin M$. Să se găsească M cu nr. min. de noduri.

Exemplu:



$$M = \{b, d, e\}$$

Căutăm un algoritm de aproximare.

Algoritm ac-nod-aprox(N, A)

$M \leftarrow \{\}$; $A' \leftarrow A$ cât timp $A' \neq \emptyset$ repeta

pentru fiecare (u, v) din A' repeta

dacă $v \notin M$ atunci $M \leftarrow M \cup \{v\}$

dacă $v \in M$ atunci $M \leftarrow M \setminus \{v\}$

elimină din A' arcele cu ~~extremități~~ în

o și v.

② alege (u, v) din A' , sterge (u, v) din A' .

$$M = \{a, b\}$$

$$M = \{a, b, c, d\} \quad \text{Incepem cu } (a, b)$$

$$M = \{a, b, c, d, e, f\}$$

$$M = \{b, c, d, e\} \quad \text{Incepem cu } (b, c)$$

$$\delta = \frac{\# M}{\# M^*}$$

Cât este limitarea?

$$\delta = ?$$

Cazul cel mai defavorabil este când ~~se~~ fiecare nod din M^* este eliminat singur sau alt nod care nu aparține lui M^* .

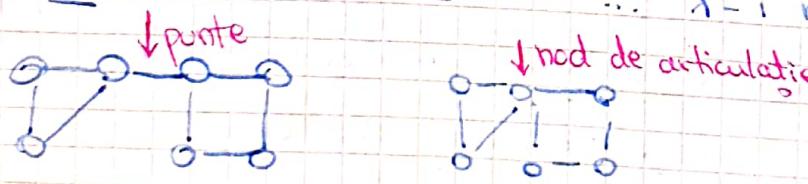
$$\Rightarrow 2 * \# M^* \Rightarrow \delta \leq 2$$

Def. ① Un graf $G = (N, A)$ este λ -arc conectat dacă după eliminarea a $\lambda - 1$ arce el este conex.

Obs.: Un graf 2-arc conectat este un graf care nu are puncti.

Un graf 2-nod conectat nu are nod de articulație (bi-conectat).

Def. ② ... λ -nod conectat ... $\lambda - 1$ noduri ...



Problema: Dându-se un graf $G = (N, A)$ λ -arc conectat, să se găsească un subgraf $G' = (N, A')$ cu $A' \subseteq A$, care este și el λ -arc conectat, dar cu număr minim de arce. (A' minim)

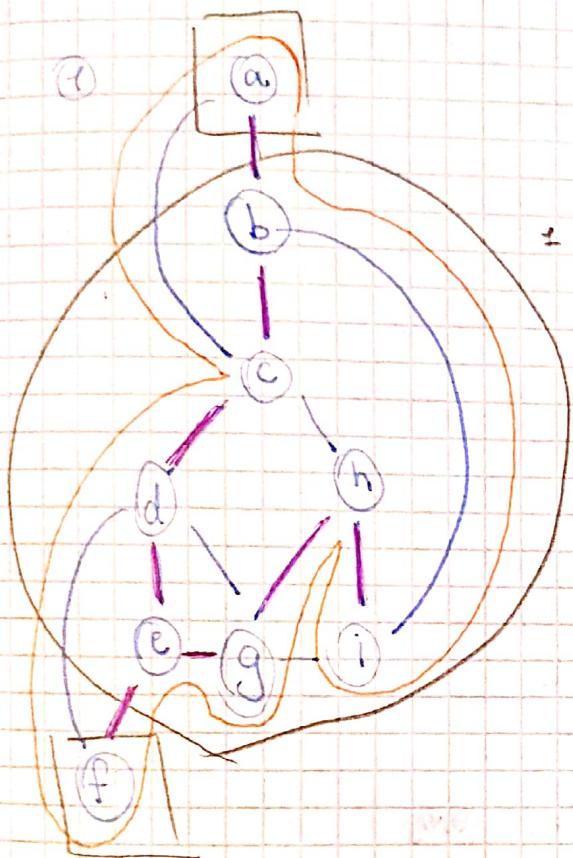
Algoritmi de aproximare

① Khuller - Vishkin

$$\delta \leq 2 - \frac{1}{\lambda}$$

② Nagamaki - Ibaraki \rightarrow face o singură trecere prin graf.

$$\delta \leq 2$$



etape: \rightarrow Construim un subgraf

2-arc conectat

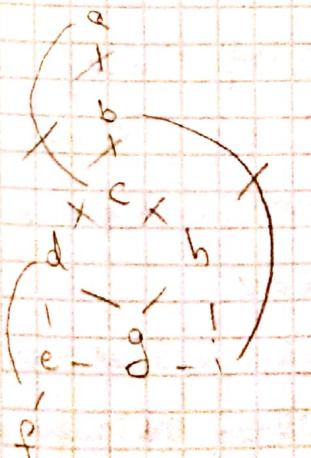
\rightarrow repetat se adaugă arce
până devine λ -arc
conectat

Un ciclu Hamiltonian este 2-arc
conectat.

a - b - i - h - g - e - f - d - c - a
ciclu hamiltonian \rightarrow optim
 \downarrow
NPC

Căutăm algoritm de aproximare.

Parcurgere în adâncime \rightarrow se construiește un arbore
de parcurgere în adâncime;
 \rightarrow se adaugă arce;



b, c

d, e

f, g

h, i

i, j

j, k

k, l

l, m

m, n

n, o

o, p

p, q

q, r

r, s

s, t

t, u

u, v

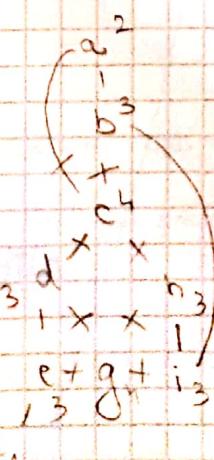
v, w

w, x

x, y

y, z

z, a



c, g, e, b, i

Algoritm arc-2. conectat (G_1, G_2)

pentru fiecare $n \in N$ repetă

n . culoare \leftarrow alb

n . tată \leftarrow nil

temp $\leftarrow 0$

$A_2 \leftarrow \emptyset$

parc. ad. ~~rec~~ rec(n) // $n \in N$

sărit

nod-culoare

- debut

- finish

- înapoi

- înapoi 2

Algoritm parc.ad. rec(u)

$v.$ debut \leftarrow temp; temp \leftarrow temp + 1;

$v.$ înapoi $\leftarrow v.$ debut

$v.$ înapoi $\leftarrow v.$ debut

$v.$ culoare \leftarrow gri

pentru fiecare v din succ(u)

dacă $v.$ culoare = alb

atunci $v.$ tată $\leftarrow u$

$A_2 \leftarrow A_2 \cup \{u, v\}$

parc. ad. rec(v)

dacă $v.$ înapoi $\leftarrow u.$ înapoi

atunci $v.$ înapoi $\leftarrow v.$ înapoi

dacă $v.$ debut $\leftarrow v.$ înapoi 2 // arc de partitie

atunci $A_2 \cup \{v.\text{anc}\}$

$v.$ înapoi 2 $\leftarrow v.$ înapoi

$v.$ înapoi 2 $\leftarrow \min \{v.$ înapoi 2, $v.$ înapoi 2}

altfel

dacă $v.$ tată $\neq v$

atunci

dacă $v.$ debut $\leftarrow v.$ înapoi

atunci $v.$ înapoi $\leftarrow v.$ înapoi

(*)

v. culoare \leftarrow negru

$G_2 = (N, A_2) \rightarrow$ 2-arc conectat

$$S = \frac{11}{10}$$

Definitie: Fie $G = (N, A)$ graf. Arboarea de partitionare de grad k este $T_G = (N_k; A_k)$ unde arboarea are proprietatile:

a) $N_k = \{N_1, N_2, \dots, N_k\}$

b) $N_1 \cup N_2 \cup \dots \cup N_k = N$

c) $N_i \cap N_j = \emptyset, \forall i, j \leq k$

d) $\forall (u, v) \in A - u \text{ si } v \in N_i$ sau

$u \text{ si } v$ sunt in N_i si N_j care sunt totdeauna frății.

$$T_G = \{N_1, N_2, N_3\}$$

$$A_k = \{(e, f); (b, a)\}$$

Teorema: Fie $G = (N, A)$ 2-arc conectat si $G_2 = (N, A_2)$ un subgraf 2-arc conectat al lui G si T_G de grad k de partitionare a lui G .

$$\text{Atunci } \#(A_2) \geq z(k-1)$$

Demonstratie:

T_G are $k-1$ arce. Pentru a fi 2-arc conectat

$$\#(A_2) \geq z(k-1)$$

Corolar: Fie $\text{Opt}(G)$ - nr. arcelor optim dintr-un subgraf de acoperire $G = (N, A)$.

T_G cu grad k . Atunci $\text{Opt}(G) \geq \max(n, z(k-1))$.

Demonstratie: $\geq 2(k-1)$ din teorema

n -ciclu hamiltonian.

Teorema: $\delta \geq \frac{3}{2}$ pt. 2-arc conectarea făcută.

Demonstratie: arborele de parcurgere

$$\delta(n) = \frac{\# A_2}{\text{opt}(G)} \leq \frac{n-L+k-L}{\max(n, 2(k-1))} \rightarrow \text{arcuri adăugate în}$$

a) $n > 2(k-1) \Rightarrow \frac{n}{2} > k-1$.

$$\leq \frac{n + \frac{n}{2}}{n} = \frac{3n}{2n} = \frac{3}{2}$$

b) $n \leq 2(k-1)$

$$\delta(n) \leq \frac{2(k-1) + (k-1)}{2(k-1)} = \frac{3}{2}$$

$$\delta < \frac{3}{2} = 2 - \frac{1}{\lambda}, \lambda = 2.$$