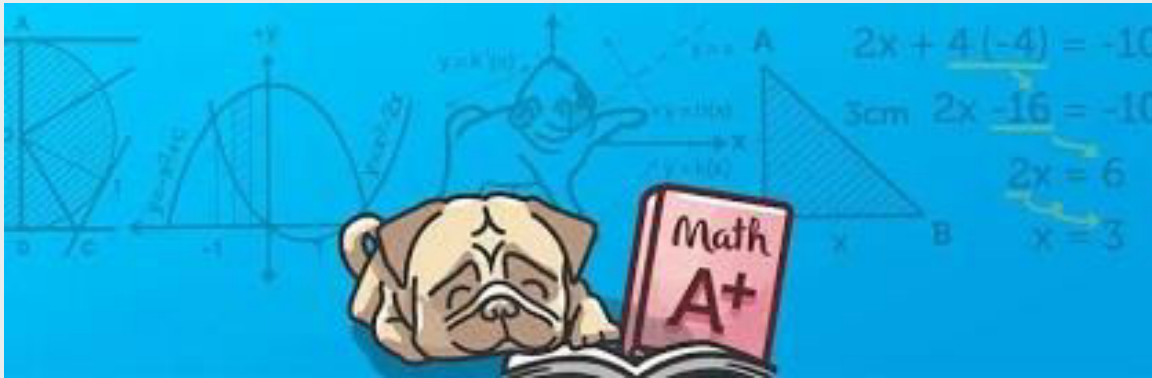
Algoritmi Avansați 2021
C-1

Lect. Dr. Ștefan Popescu

Email: stefan.popescu@fmi.unibuc.ro

Grup Teams:



Index

- Desfășurare examen & predare
- Ce este un algoritm
- Complexitatea timp a unui algoritm
- P, NP, NPC
- Ce este o problema de optim?
- Idei alternative de rezolvare (prelude)

Desfășurare examen & predare

- Curs Modular;
- Laborator 50% + examen final 50%
- Limbaj de programare: La alegere Python sau C++
- Prima jumătate a cursului [7 saptamani] va fi o continuare a cursului de AF
- Prezenta nu este obligatorie dar probabil este necesara
- Va voi fi profesor la primele 4 laboratoae si primele 4 seminarii.
- Promovarea unui mediu interactiv
- Feedback-ul este mereu de apreciat

Ce este un algoritm?

- informal: o succesiuni de pasi elementari/simpli dupa a căror execuție pe un input dat, obținem un output care este soluție pentru problema noastră

- Formal: Echivalent cu **Mașina Turing** (*to be continued*)

Complexitatea timp

Informal spus, complexitatea timp a unui algoritm este dat de *numărul de operații* efectuate până ce se ajunge la rezultat. Evident numărul de operații va depinde și de input, mai exact lungimea inputului.

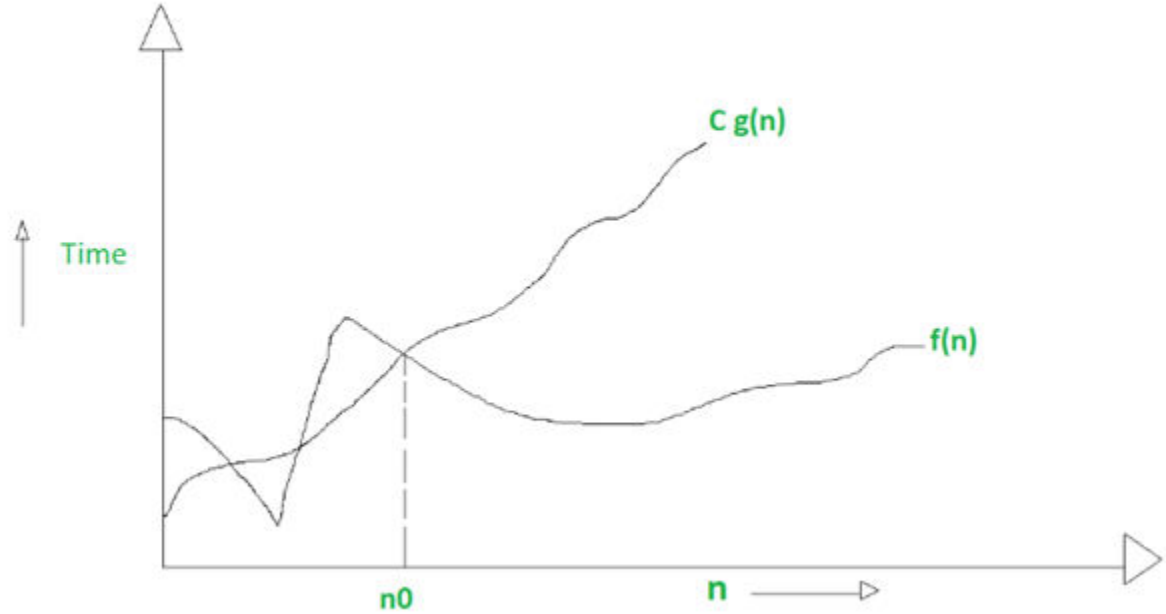
Fie un algoritm care pentru o intrare (de lungime) *n* efectuează *f(n)* operații.

Definim clasele de complexitate **O, Ω, Θ** după cum urmează

Clasa O (Big Oh)

- Descrie o **limită superioară** pentru numărul de operații efectuate de algoritm pentru orice intrare de la o lungime n_0 încolo.
- Spunem ca un algoritm rulează în timp O(g(n)) dacă există o funcție g și o valoare n_0, astfel încât să avem relația: $C \times g(n) \geq f(n) \geq 0 \mid \forall n > n_0$ (unde C este o constantă pozitivă).
- *f* este asimptotic mărginită superior de către *g* (multiplicată cu un factor constant C)
- Observăm, spre exemplu, că $O(n)$ este inclusă în $O(n^2)$

Clasa O (Big Oh)



Clasa O (Big Oh)

Definiția rigurosă este ”un pic mai complicată”:

Fie un algoritm Alg și o funcție f:N→N, astfel încât Alg se termină exact în f(n) pași pentru o intrare de lungime ”n”. Spunem că Alg rulează în O(g(n)) dacă avem relația:

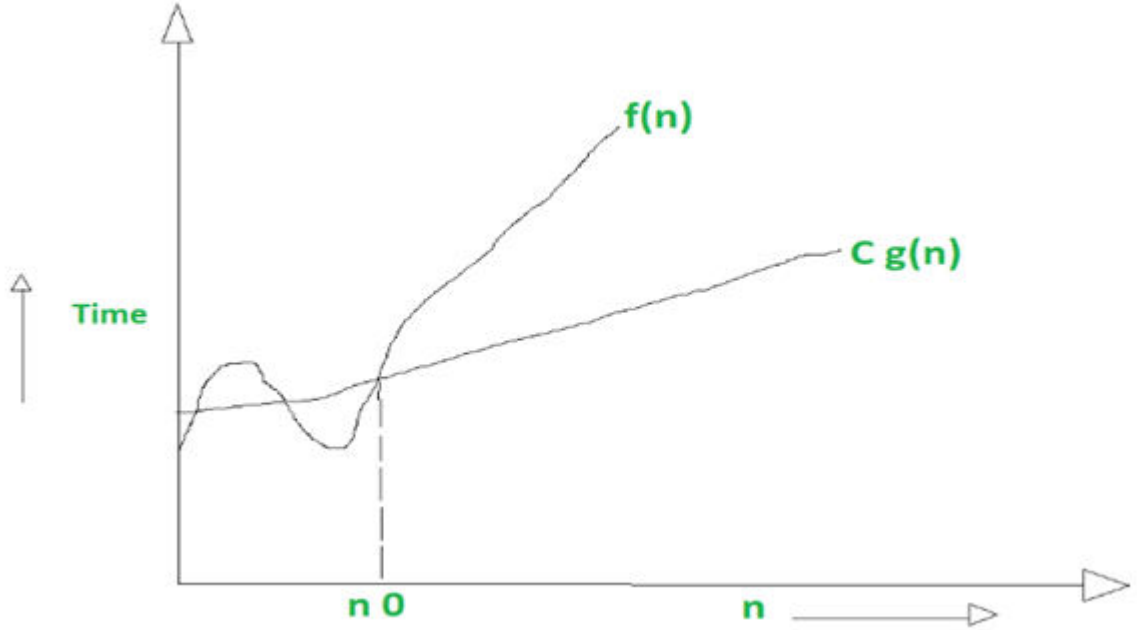
$$\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

Această definiție ne arată ca în clasa de complexitate O, factorul dominant este cel care ne dă complexitatea. Ex: $O(n^2+2n) \equiv O(n^2)$

Clasa Ω (Big Omega)

- Descrie o **limită inferioară** pentru numărul de operații efectuate de algoritm pentru orice intrare de la o lungime n_0 încolo.
- Spunem ca un algoritm rulează în timp Ω(g(n)) dacă există o funcție g și o valoare n_0, astfel încât să avem relația: $f(n) \geq C \times g(n) \geq 0 \mid \forall n > n_0$ (unde C este o constantă pozitivă).
- *f* este asimptotic mărginită inferior de către *g* (multiplicată cu un factor constant C)
- Observăm, spre exemplu, că $\Omega(n)$ este inclusă în $\Omega(n^2)$

Clasa Ω (Big Omega)



Clasa Ω (Big Omega)

Definiția rigurosă:

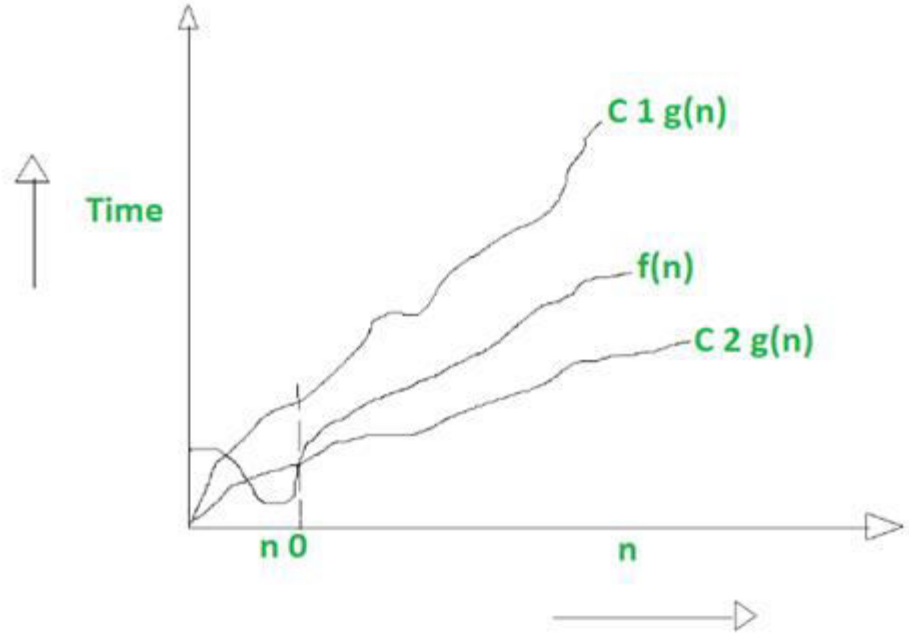
Fie un algoritm Alg și o funcție f:N→N, astfel încât Alg se termină exact în f(n) pași pentru o intrare de lungime "n". Spunem că Alg rulează în Ω(g(n)) dacă avem relația:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0$$

Clasa Θ (Big Theta)

- O combinație între cele două clase anterioare. Presupune o mărginire atât superioară cât și inferioară.
- Spunem ca un algoritm rulează în timp Θ(g(n)) dacă există o funcție ***g*** și o valoare n_0, astfel încât să avem relația: $C_1 \times g(n) \geq f(n) \geq C_2 \times g(n) \geq 0 \mid \forall n > n_0$ (unde C_1 și C_2 sunt două constante pozitive).
- *f* este asimptotic mărginită inferior de către *g* (multiplicată cu un factor constant C_2) respectiv superior tot de *g* (multiplicată cu un factor constant C_1)
- **Nu mai este valabilă observația** că **Θ(n)** ar fi inclusă în **$\Theta(n^2)$**
- Nu toți algoritmii au o complexitate Theta

Clasa Θ (Big Theta)



Clasa Θ (Big Theta)

Cum ar arăta definiția folosind limite?

Clasa Θ (Big Theta)

Fie un algoritm Alg și o funcție f:N→N, astfel încât Alg se termină exact în f(n) pași pentru o intrare de lungime "n". Spunem că Alg rulează în Θ(g(n)) dacă avem relația:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \in \mathbb{R}_+$$

Source

Discuții libere

Cel mai adesea folosim clasa O.

Evident ne interesează numitele ”tight bounds”.

Ce se întâmplă când pe o intrare de aceeași lungime ai număr de pași semnificativ diferiți?

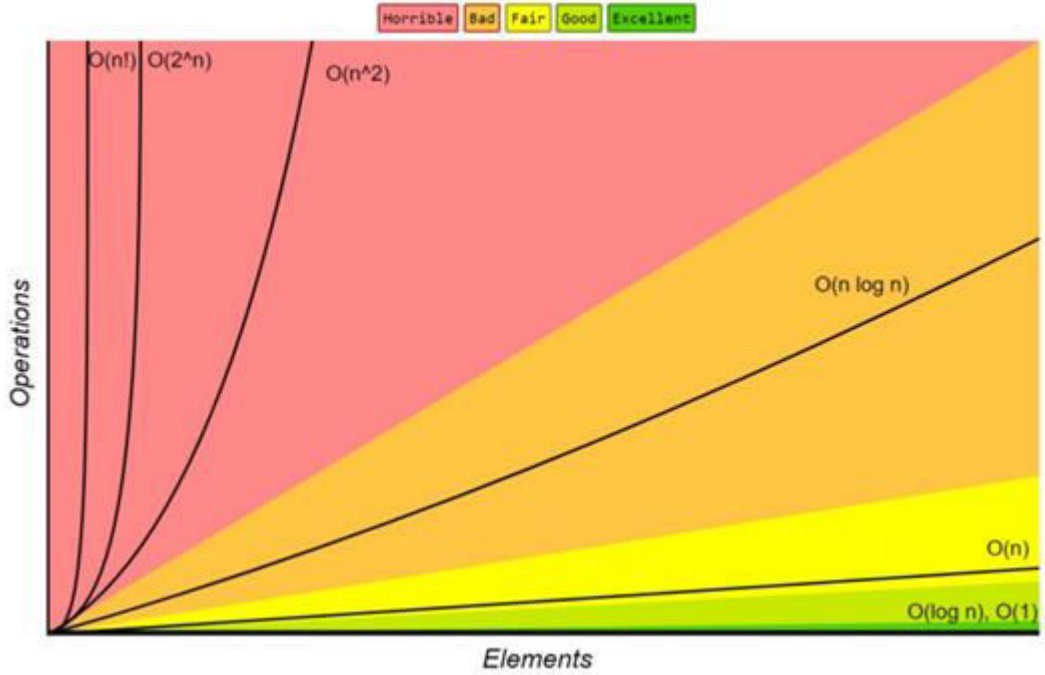
- Complexitate ”*worst case*” vs complexitate medie

Care este complexitatea următorilor algoritmi?

- Căutare binară; Bubble sort; quicksort, merge-sort;

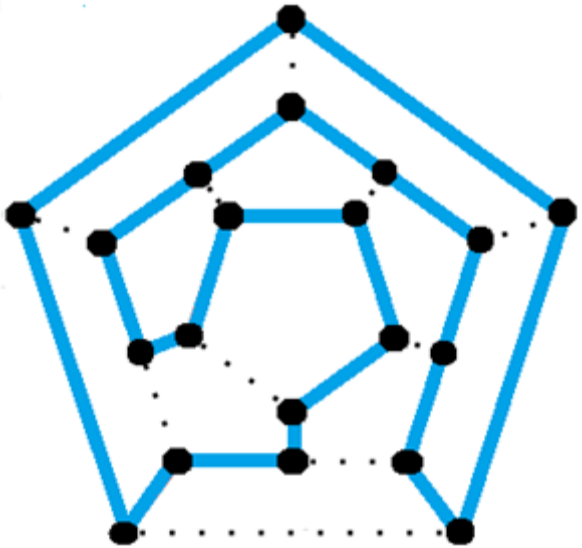
Paradoxul în care Big-O nu redă realitatea...

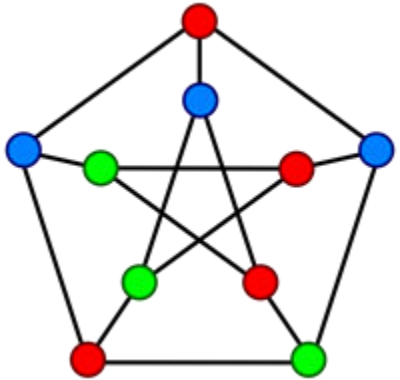
Ce inseamnă ”algoritm eficient”?



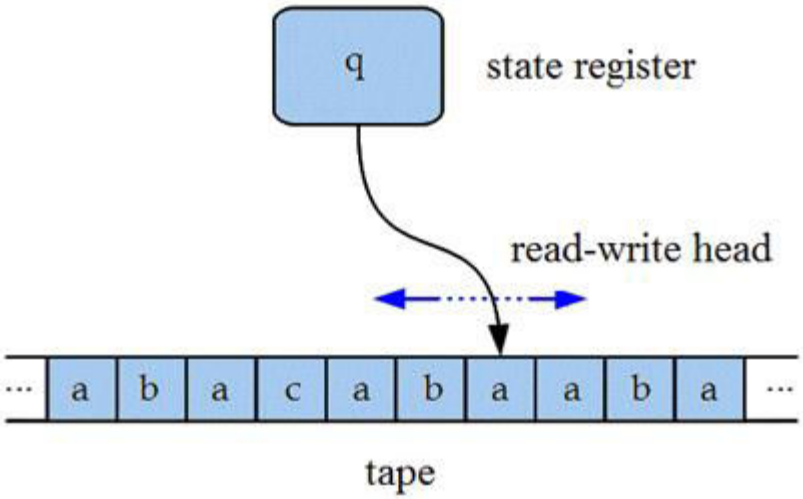
Timp polinomial:
Determinism vs nedeterminims

5		9				4		
7		8	3		4	9		
6		1				7	3	
4	6	2	5					
3	8	5	7	2		6	4	9
1		7	4		8	2		
2			1					4
		3		4			8	7
	7			5	3			6





Scurt prezentare: ”Turing Machine”



O mașină Turing M=(Q, Γ, b, Σ, δ, q_0, F) unde:

- Q - mulțimea stărilor
- Γ - alfabetul de lucru al mașinii
- b ∈ Γ - un simbol special, numit ”blank”
- Σ⊂Γ\{b} - alfabetul de intrare (alfabetul pt input)
- q_0, F - starea inițială, respectiv mulțimea stărilor finale

δ - funcția de tranziție:
cazul determinist: δ: ΓxQ → ΓxQx{left, right}

cazul nedeterminist: δ: ΓxQ → $2^{\Gamma x Q x \{left, right\}}$

Clasele de Complexitate P și NP

Formal spus, în clasa problemelor din P sunt acele probleme care pot fi rezolvate in timp polinomial, $O(n^c)$, de către un sistem determinist. (P=polynomial)

Iar cele din clasa NP sunt problemele care pot rezolvate tot în timp polinomial (!) dar de către o mașina Turing nedeterminista. (NP=nondeterministic Polynomial)

Evident ca $P \subset NP$.
Se presupune ca $P \subsetneq NP$, totuși încă nu există o demonstrație a acestui rezultat.

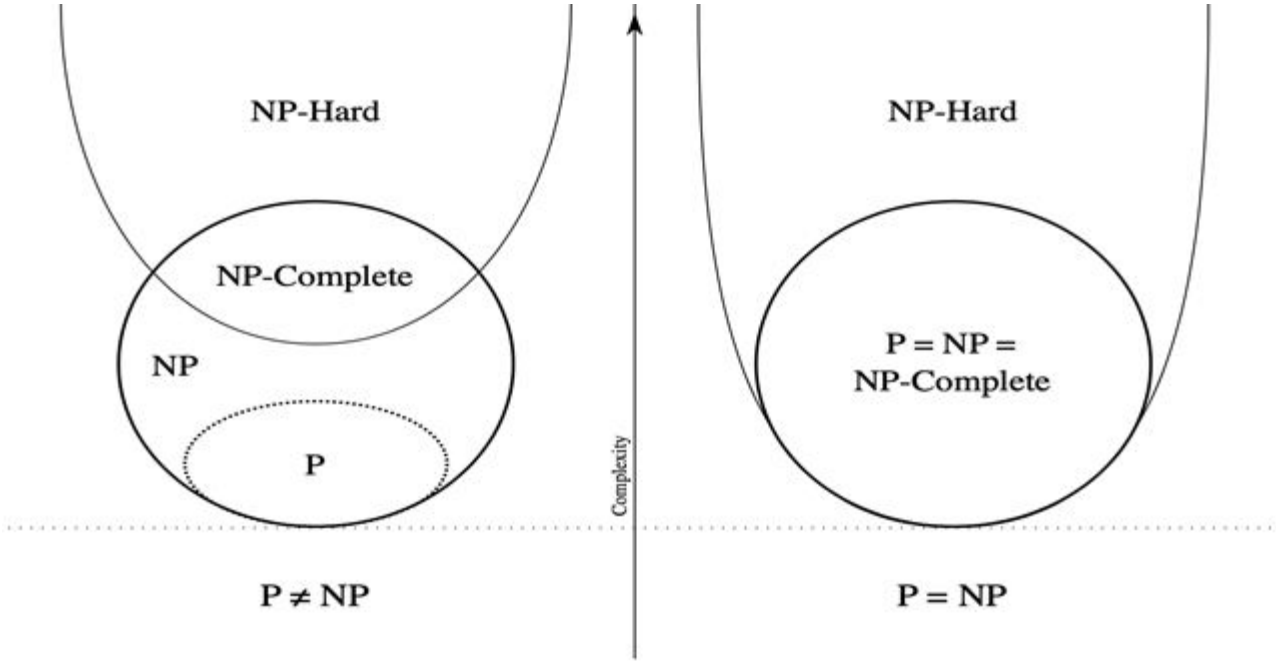
Clasele de Complexitate P și NP

Mai ușor de înțeles:
P - clasa de probleme pentru care le putem afla soluția în timp polinomial

NP - clasa de probleme pentru care **putem verifica** în timp polinomial dacă un rezultat este soluție corectă pentru problema noastră.

5		9				4		
7		8	3		4	9		
6		1				7	3	
4	6	2	5					
3	8	5	7	2		6	4	9
1		7	4		8	2		
2			1					4
		3		4			8	7
	7			5	3			6

Clasele de Complexitate P, NP, NP-C



Source for further reading

Ce ne facem cu problemele din NP

Avem la dispoziție mașini (calculatoare) nedeterministe?

În cazul problemelor de optim există două soluții: Plătim costul în timp (de multe ori nu se poate) sau ne mulțumim cu o soluție apropiată de optim, dacă nu optimă, dar ce se poate obține în timp fezabil.

Probleme de optim

O problemă de optim este de forma următoare:
Fie o mulțime de restricții. Să se construiască o soluție care nu doar îndeplinește toate restricțiile, ci minimizează/maximizeze o funcție de cost/profit.

Ex: Problema rucsacului (varianta discretă) sau probleme de acoperire minimală pentru grafuri.

Probleme de optim

Fie OPT soluția optim a problemei. Ea poate fi obținută foarte greu (practic imposibil) Două dintre căile de atac pentru astfel de probleme ar fi:
- avem un algoritm care construiește pe rand soluții la problemă, din ce în ce ”mai optime”, care converg către OPT. Lăsăm acest algoritm să ruleze un timp rezonabil, sau până când rezultatul nu se mai poate îmbunătăți si ne multumim cu ce avem.

(algoritmi evoluționiști)

Probleme de optim

Fie OPT soluția optim a problemei. Ea poate fi obținută foarte greu (practic imposibil) Două dintre căile de atac pentru astfel de probleme ar fi:
- fie cazul in care OPT trebuie să minimizeze un cost. Să reușim să contruim o soluție ALG, cu OPT<=ALG<=ρxOPT

(algoritmi ρ-aproximativi)

Aplicatie:

Algoritm aproximativ pentru 1/0 Kanspack Problem

[Whiteboard]

Next time:

Seminar & Lab - Recapitulare Fundamentele Algoritmilor

Curs 2: introducere în algoritmi aproximativi