

— Algoritmi Avansați 2023

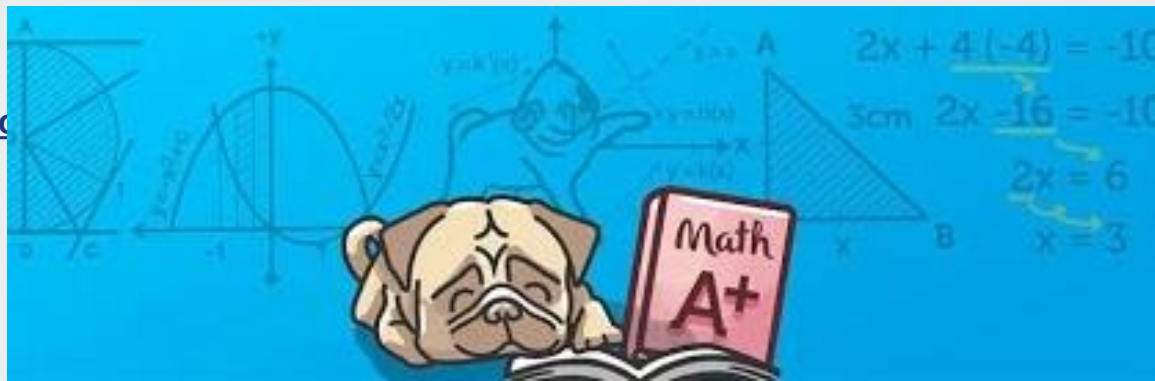
c-8

Algoritmi ρ -aproximativi

Lect. Dr. Ștefan Popescu

Email: stefan.popescu@fmi.unibuc.ro

Grup Teams:



Din cursul anterior

Recapitulare:

- reamintit ce este acela un algoritm

- complexitatea unui algoritm

- timpi determinist vs nedeterminist

- crash-course in ce inseamna P, NP, NPC



Cursul prezent

- Motivație
- Terminologie de baza
- Un prim exemplu de algoritm aproximativ
- Un exemplu mai detaliat
- Un început pt Tema 1

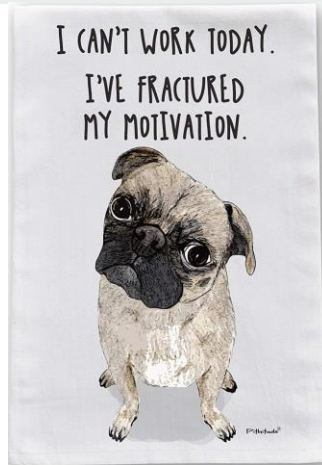


Motivație

Q: Daca avem nevoie să aflăm raspunsul la o problemă NP-hard?

A: Nu prea sunt șanse să găsim un algoritm care să ruleze în timp polinomial

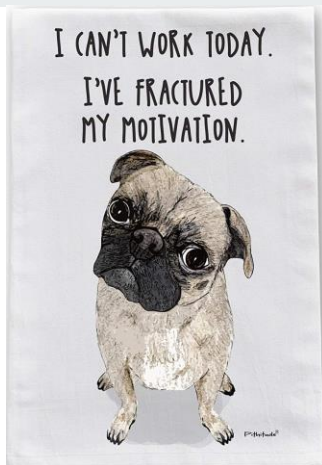
Așa că....



Motivație

Trebuie să renunțăm măcar la unul dintre următoarele 3 elemente:

1. Găsirea unui algoritm polinomial pentru problemă
2. Găsirea unui algoritm general (pentru o instanță oarecare) a problemei
3. Găsirea soluției exacte (optime) pentru problema





Basic Terminology & Notations

Problema de Optim:

Informal spus este problema in care trebuie sa gasesti o “cea mai buna” solutie/constructie fezabila.

“Cea mai buna” - poate avea doua sensuri:

Fie avem o problema de **minimizare** precum Problema Comis-voiajului.

Fie o problema de **maximizare** precum cea de a găsi o acoperire de cardinal maxim pentru multimea varfurilor unui graf



Basic Terminology & Notations



Problema de Optim:

Fie P - o problema de optim, și I o intrare pe aceasta problema. Vom nota cu $OPT(I)$ "valoarea" soluției optime.

În mod analog, atunci când propunem un algoritm care să ofere o soluție fezabilă pentru problema noastră, vom nota "valoarea" acelei soluții cu $ALG(I)$.

De cele mai multe ori, atunci când nu se crează confuzie, vom simplifica notațiile folosind termenii "OPT", respectiv "ALG"

Pe parcursul prezentării vom presupune că atât OPT , cât și ALG sunt ≥ 0 .



Basic Terminology & Notations



Problema de Optim:

Pentru a justifica ca un algoritm este *util*, acesta trebuie însoțit de o justificare că soluția oferită este fezabilă pentru problema, precum și o relație între ALG și OPT . Aceast tip de relație este descrisă astfel:

Definiție 1

- Un algoritm ALG pentru o problema de **minimizare** se numește ρ -aproximativ, pentru o valoare $\rho > 1$, dacă $ALG(I) \leq \rho \cdot OPT(I)$ pt $\forall I$ – intrare
- Un algoritm ALG pentru o problema de **maximizare** se numește ρ -aproximativ, pentru o valoare $\rho < 1$, dacă $ALG \geq \rho \cdot OPT(I)$ pt $\forall I$ – intrare



Basic Terminology & Notations

OBSERVAȚIE

(pt probleme de minim) Orice algoritm ρ -aproximativ este la rândul lui ρ' -aproximativ pentru orice $\rho' > \rho$. De aceea, în cazul unui algoritm ALG pentru o problemă de minimizare, spre exemplu, trebuie ca justificarea ce însoțește pe ALG să ofere cea mai mică valoare ρ pentru care ALG este ρ -aproximativ.

Definiție 1

- Un algoritm ALG pentru o problema de **minimizare** se numește ρ -aproximativ, pentru o valoare $\rho > 1$, dacă $ALG(I) \leq \rho \cdot OPT(I)$ pt $\forall I$ – intrare
- Un algoritm ALG pentru o problema de **maximizare** se numește ρ -aproximativ, pentru o valoare $\rho < 1$, dacă $ALG \geq \rho \cdot OPT(I)$ pt $\forall I$ – intrare



Basic Terminology & Notations

Definiție 2

Fie ALG un algoritm p -aproximativ pentru o problema de minimizare. Spunem că factorul de aproximare este "tight bounded" atunci când avem $\rho = \sup_I \frac{ALG(I)}{OPT(I)}$

Ca să arătăm că un algoritm este p -aproximativ "tight bounded", trebuie deci să justificăm următoarele 2 lucruri:

1. Trebuie să arătăm că este p -aproximativ, adică $ALG(I) \leq p \times OPT(I)$ pentru orice intrare I
2. Pentru orice $p' < p$ există un I pentru care $ALG(I) > p' \times OPT(I)$. Adesea totuși ne este mai la îndemână să arătăm ca există un I pentru care $ALG(I) = p \times OPT(I)$

O primă provocare: 1/0 Knapsack problem

Enunț pe scurt: Trebuie să găsim o submulțime de obiecte de valoare totală maximă, fără ca greutatea lor totală să depășească o capacitate dată a rucsacului. Obiectele sunt puse integral în rucsac sau sunt date deoparte. Nu pot fi fracționate!



O primă provocare: 1/0 Knapsack problem

Enunț pe scurt: Trebuie să găsim o submulțime de obiecte de valoare totală maximă, fără ca greutatea lor totală să depășească o capacitate dată a rucsacului. Obiectele sunt puse integral în rucsac sau sunt date deoparte. Nu pot fi fracționate!

Presupunere: Fiecare obiect are o greutate mai mică sau egală cu capacitatea rucsacului!



O primă provocare: 1/0 Knapsack problem

Enunț pe scurt: Trebuie să găsim o submulțime de obiecte de valoare totală maximă, fără ca greutatea lor totală să depășească o capacitate dată a rucsacului. Obiectele sunt puse integral în rucsac sau sunt date deoparte. Nu pot fi fracționate!

Rezolvare propusă:

Fie L – lista obiectelor sortate după raportul valoare/greutate

Fie O_p – obiectul cu profitul cel mai mare din lista de obiecte.

$S=0$, G =capacitatea rucsacului;

Pentru fiecare $O \in L$

Dacă $greutate(O) \leq G$, atunci $S+ = val(O)$, $G- = greutate(O)$

$$ALG(I) = \max(S, O_p)$$



O primă provocare: 1/0 Knapsack problem

Demonstrați că algoritmul de mai jos este un algoritm 1/2-aproximativ pentru problema 1/0 a Rucsacului!

Rezolvare propusă:

Fie L – lista obiectelor sortate după raportul valoare/greutate

Fie O_p – obiectul cu profitul cel mai mare din lista de obiecte.

$S=0$, G =capacitatea rucsacului;

Pentru fiecare $O \in L$

Dacă $greutate(O) \leq G$, atunci $S += val(O)$, $G -= greutate(O)$

$$ALG(I) = \max(S, O_p)$$



O primă provocare: 1/0 Knapsack problem

Demonstrați că algoritmul de mai jos este un algoritm 1/2-aproximativ pentru problema 1/0 a Rucsacului! [Justificare](#)

Rezolvare propusă:

Fie L – lista obiectelor sortate după raportul valoare/greutate

Fie O_p – obiectul cu profitul cel mai mare din lista de obiecte.

$S=0$, G =capacitatea rucsacului;

Pentru fiecare $O \in L$

Dacă $greutate(O) \leq G$, atunci $S += val(O)$, $G -= greutate(O)$

$$ALG(I) = \max(S, O_p)$$



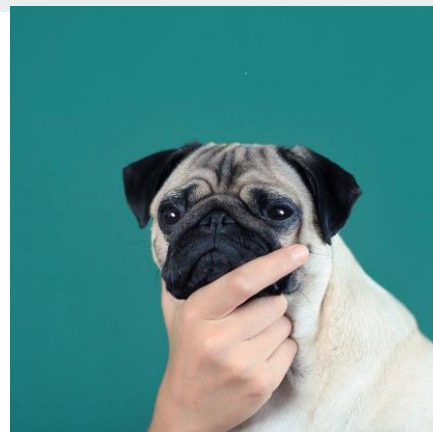
Load Balancing Problem

Input:

- m calculatoare identice; n activități ce trebuie procesate. Fiecare activitate j având nevoie de t_j unități de timp pentru execuție.
- Odată inițiată, fiecare dintre activități trebuie derulată în mod continuu pe același calculator
- Un calculator poate executa cel mult o activitate în același timp.

Scop:

Să asignăm fiecare activitate unui calculator astfel încât să minimizăm timpul până când toate activitățile sunt terminate.

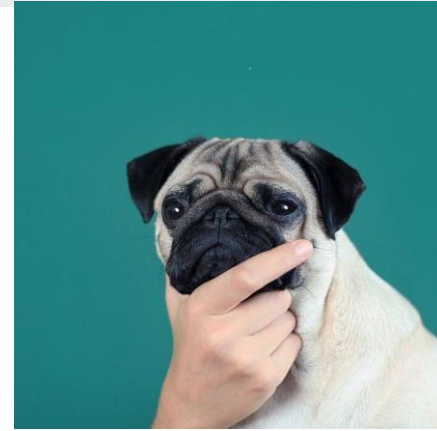


Load Balancing Problem

Notatii:

- $J(i)$ - submulțimea tuturor activităților (job-urilor) care au fost programate să se desfășoare pe mașina i .
- L_i va reprezenta "load-ul" (timpul de lucru) al mașinii i .
- $L_i = \sum_{j \in J(i)} t_j$

Scop: ???

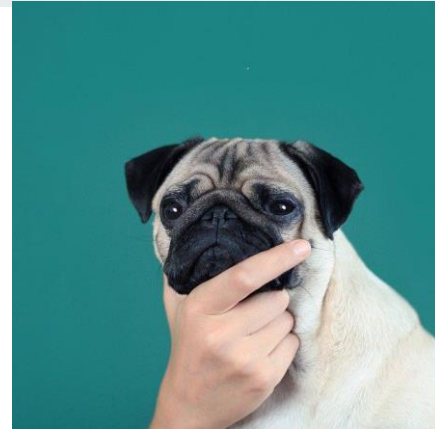


Load Balancing Problem

Notatii:

- $J(i)$ -submulțimea tuturor activităților (job-urilor) care au fost programate să se desfășoare pe mașina i .
- L_i va reprezenta "load-ul" (timpul de lucru) al mașinii i .
- $L_i = \sum_{j \in J(i)} t_j$

Scop: O asignare a activităților astfel încât L_k este minimizat, unde $k = \max_i(L_i)$, adică mașina cu cel mai mare load.



Load Balancing Problem

Pseudocodul:

Load – Balance(m, t_1, t_2, \dots, t_n)

for $i=1$ *to* m :

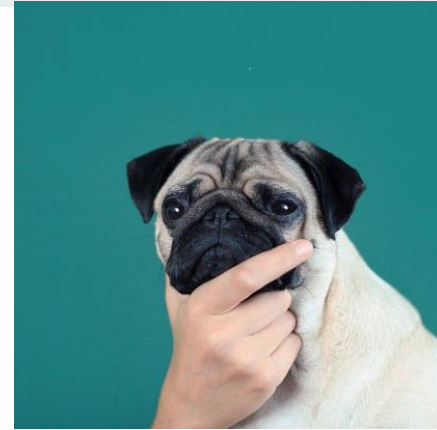
$L_i=0; J(i) = \emptyset$ # *initializare: Fiecare Load este 0 iar multimea joburilor este nula pt fiecare masina*

for $j=1$ *to* n :

$i = \arg(\min\{L_k \mid k \in \{1, \dots, m\}\})$ # *i – masina cu incarcatura cea mai mica in acest moment*

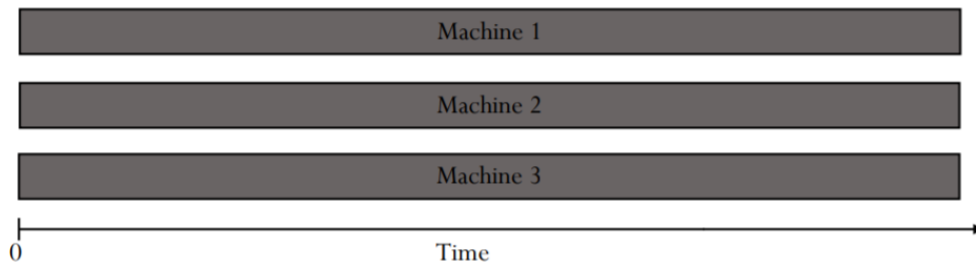
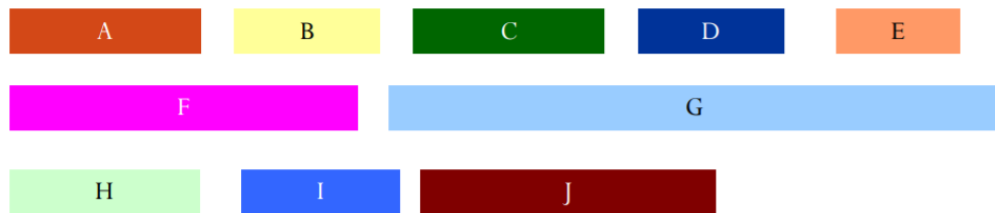
$J(i) = J(i) \cup \{j\}$

$L_i += t_j$



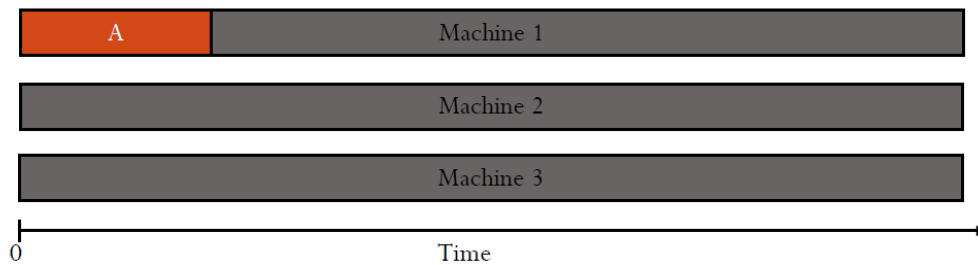
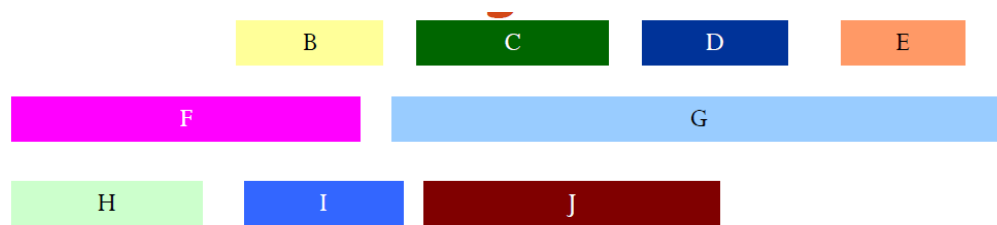


Step-by-step example



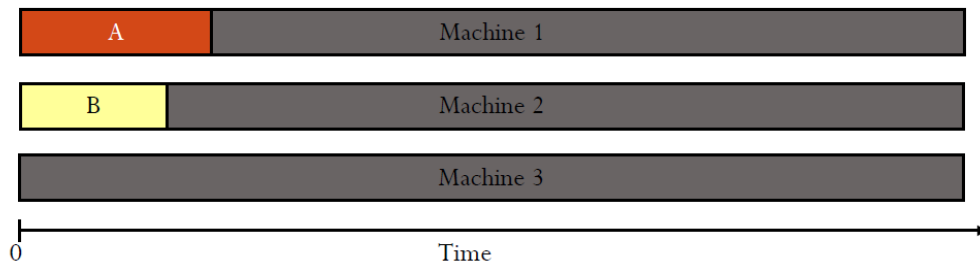
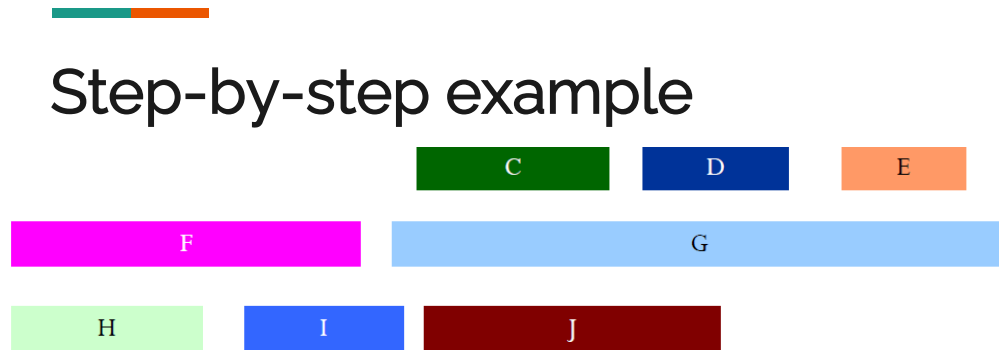


Step-by-step example



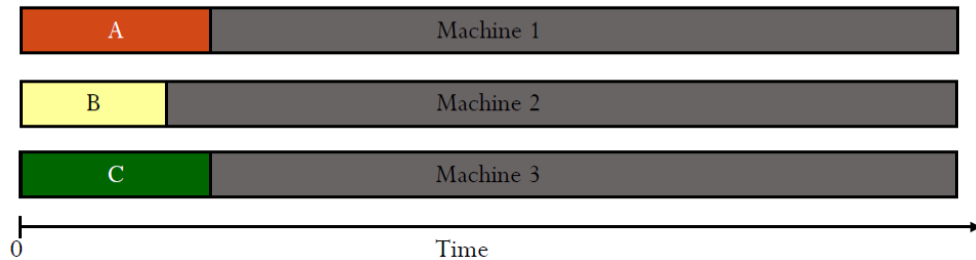
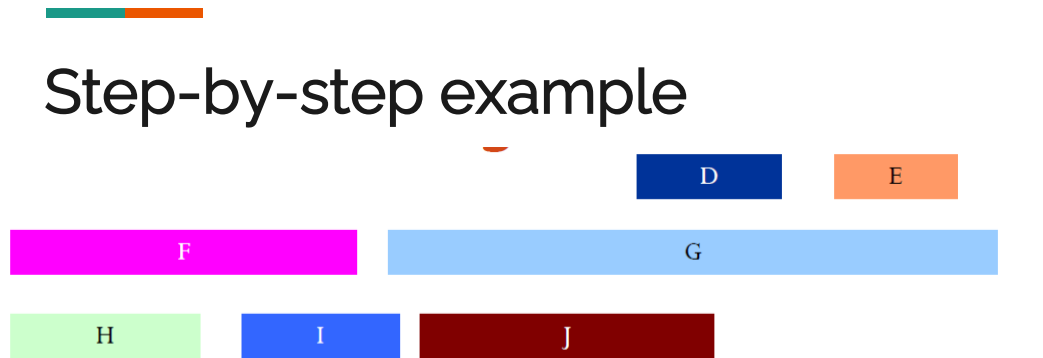


Step-by-step example



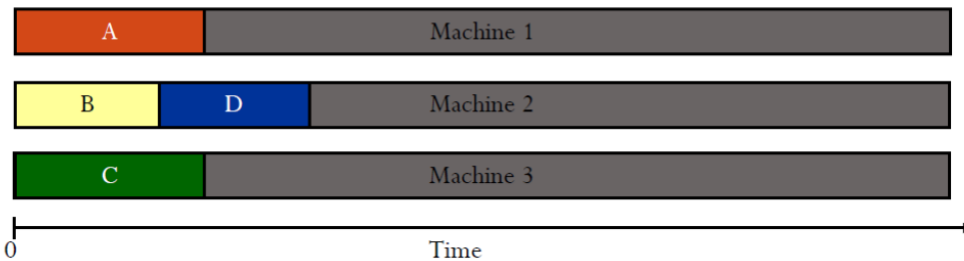
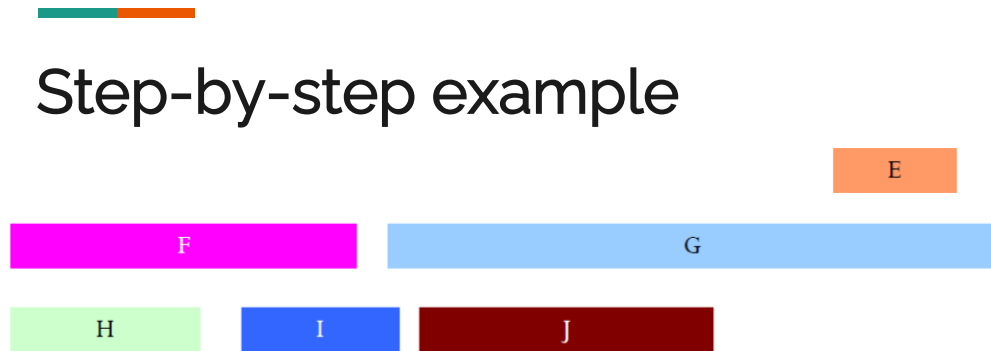


Step-by-step example



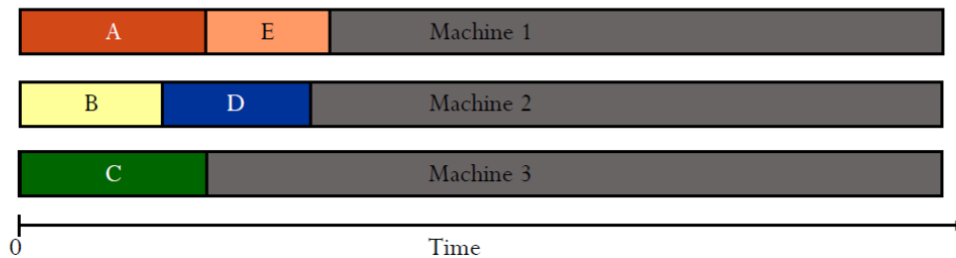


Step-by-step example



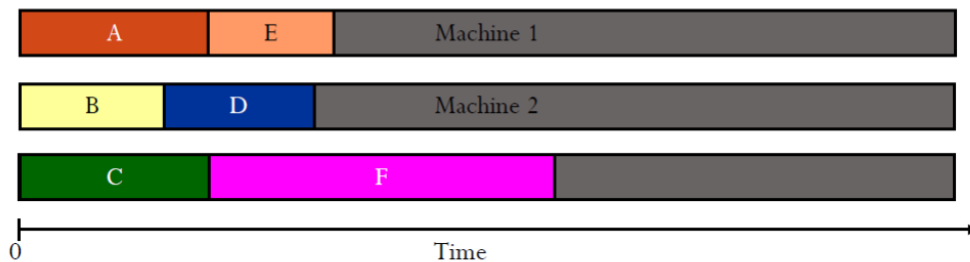


Step-by-step example



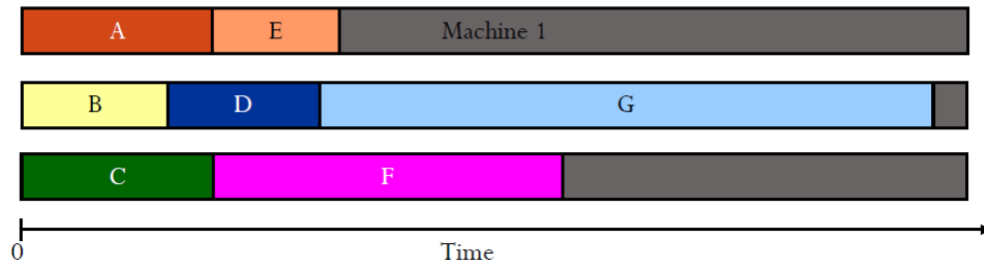


Step-by-step example



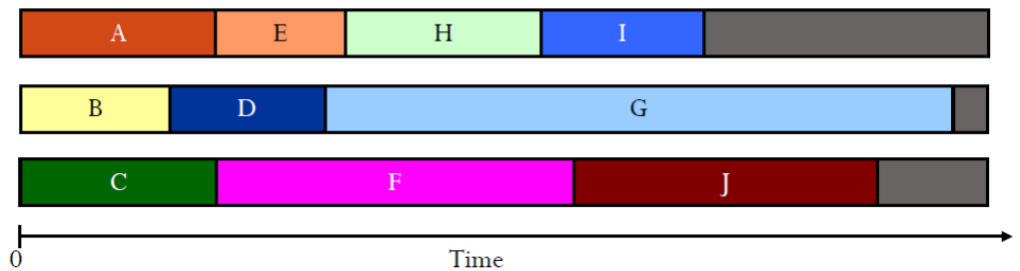


Step-by-step example

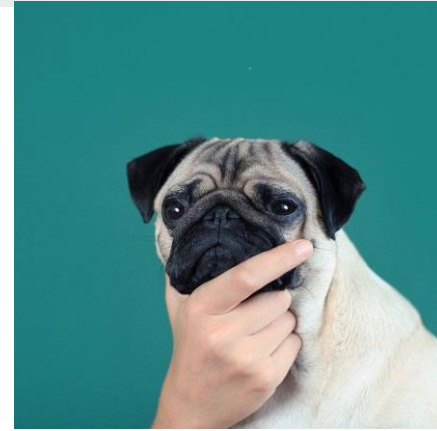




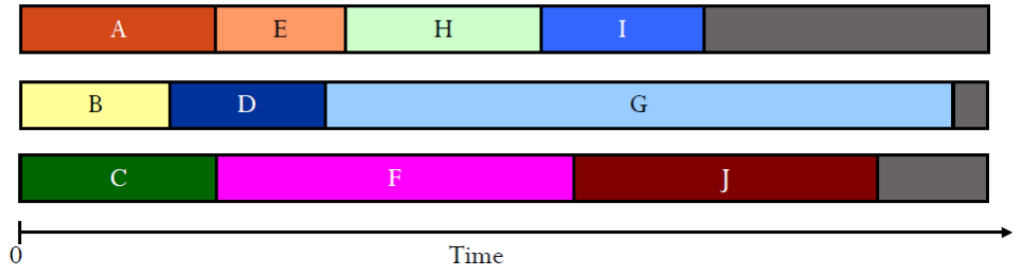
Step-by-step example (3 steps)

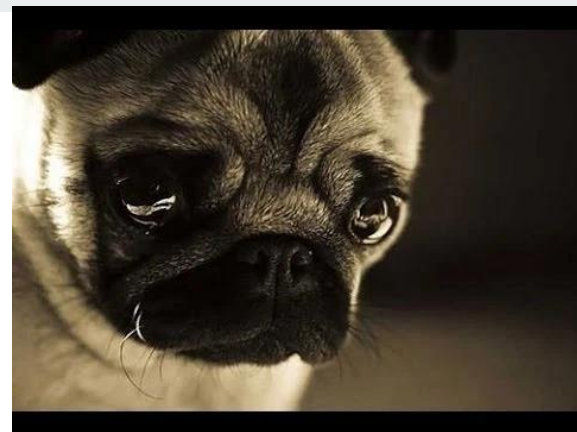
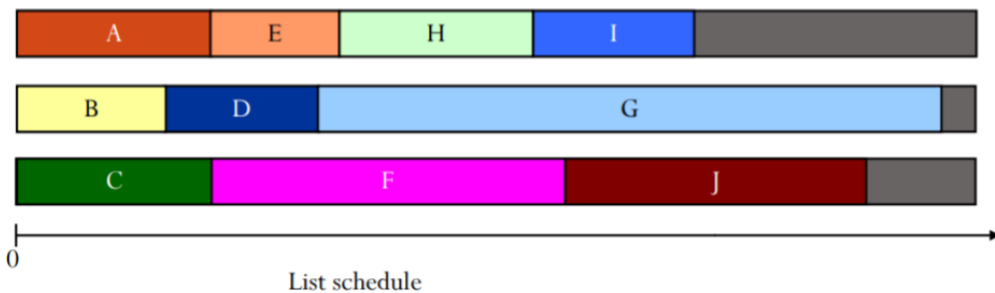
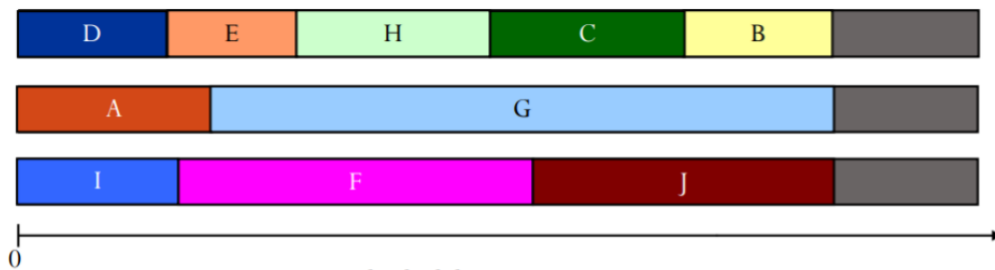


Step-by-step example (3 steps)



Este Optim?

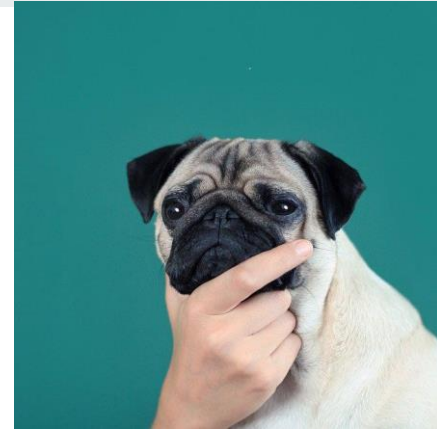




NU



Care este Factorul de Aproximare?



Care este Factorul de Aproximare?

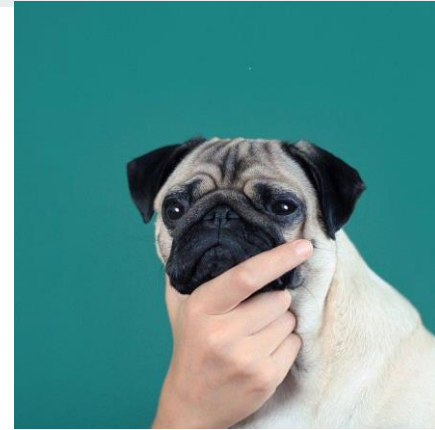
Lema 1.

$$OPT \geq \max \left\{ \frac{1}{m} \sum_{1 \leq j \leq n} t_j, \max \{ t_j | 1 \leq j \leq n \} \right\}$$

Lema 2.

Algoritmul descris anterior este un algoritm 2-Aproximativ.

Altfel spus, fie $ALG = \max(L_i | i \in \{1, \dots, m\})$ masina "cea mai incarcata". Avem de arătat că $ALG \leq 2 \times OPT$



Care este Factorul de Aproximare?

Lema 1.

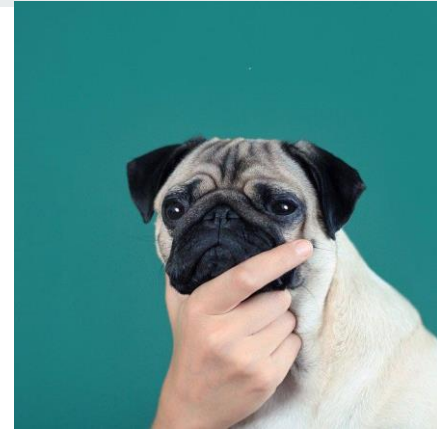
$$OPT \geq \max \left\{ \frac{1}{m} \sum_{1 \leq j \leq n} t_j, \max \{ t_j | 1 \leq j \leq n \} \right\}$$

Lema 2.

Algoritmul descris anterior este un algoritm 2-Aproximativ.

Altfel spus, fie $ALG = \max(|L_i| \mid i \in \{1, \dots, m\})$ masina "cea mai incarcata". Avem de arătat că $ALG \leq 2 \times OPT$

[Justificari](#)



Care este Factorul de Aproximare?

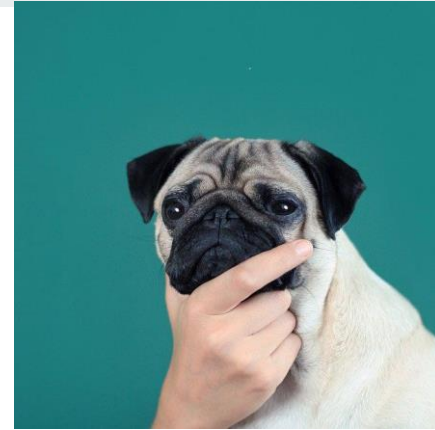
Lema 2.

Algoritmul descris anterior este un algoritm 2-Aproximativ.

Altfel spus, fie $ALG = \max(L_i | i \in \{1, \dots, m\})$ masina "cea mai incarcata". Avem de arătat că $ALG \leq 2 \times OPT$

Justificari

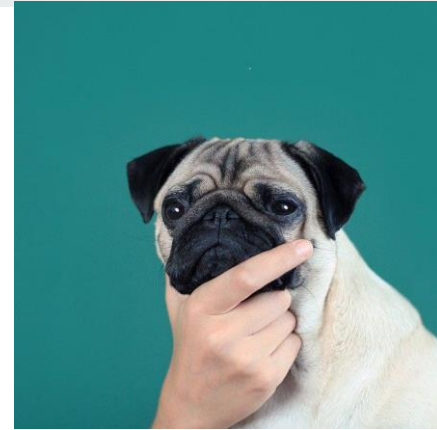
Este "tight bound"? Ce ar mai putea fi de facut?



Se poate imbunătății LB-ul?

3 abordari:

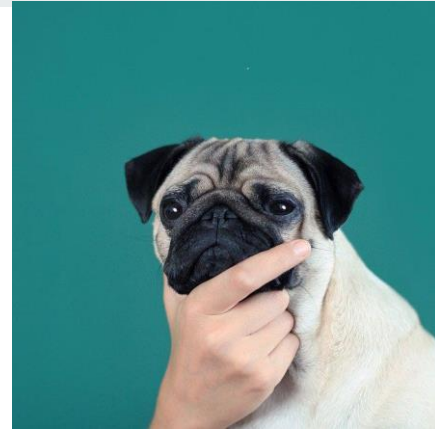
- a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit**
- b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități**
- c) Un cu totul alt Algoritm care poate da un total alt LB.**



Se poate imbunătății LB-ul?

3 abordari:

- a) **Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit**
- b) **Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități**
- c) **Un cu totul alt Algoritm care poate da un total alt LB.**



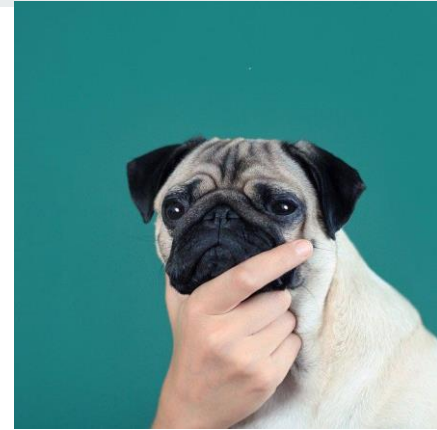
Se poate imbunătății LB-ul?

3 abordari:

a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit

Teorema: Algoritmul Greedy descris anterior este un algoritm $2-1/m$ aproximativ

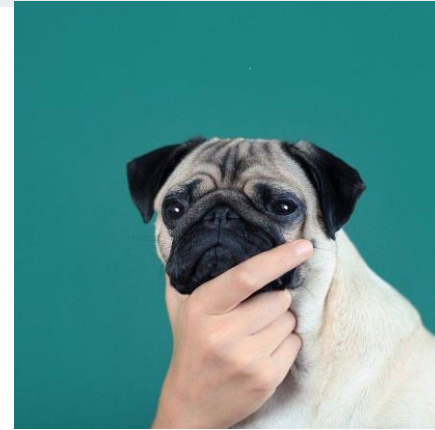
Justificari



Se poate imbunătății LB-ul?

3 abordari:

- a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit
- b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități
- c) Un cu totul alt Algoritm care poate da un total alt LB.



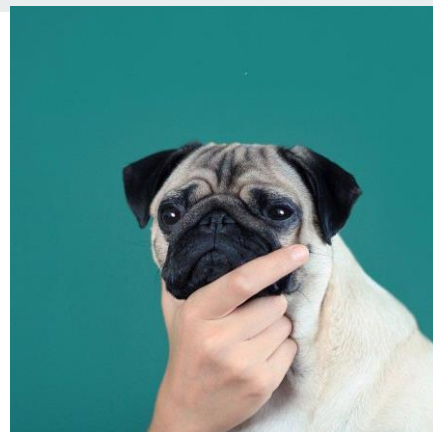
Se poate imbunătății LB-ul?

3 abordari:

Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități

Nu se poate!

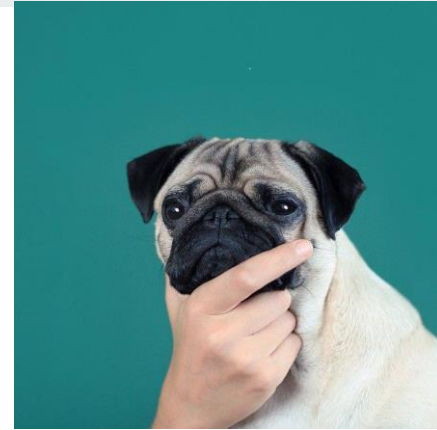
m mașini, $m(m-1)$ activitati de cost 1 si o activitate de cost m



Se poate imbunătății LB-ul?

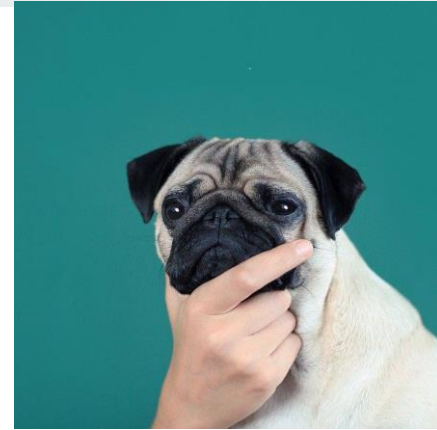
3 abordari:

- a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit
- b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități
- c) **Un cu totul alt Algoritm care poate da un total alt LB.**



Se poate imbunătății LB-ul?

Un cu totul alt Algoritm care poate da un total alt LB.





Ordered-Scheduling Algorithm

Fie algoritmul precedent la care adaugam următoarea preprocesare:

Înainte de a fi programate, activitățile sunt sortate descrescător după timpul de lucru.





Tema (preludiu)

Lema 3.

Fie o multime de n activitati cu timpul de procesare t_1, t_2, \dots, t_n astfel incat $t_1 \geq t_2 \geq \dots t_n$

Daca $n > m$, atunci $OPT \geq t_m + t_{m+1}$

TEOREMA 2

Algoritmul descris anterior (Ordered-Scheduling Algorithm) este un algoritm 3/2-aproximativ

[Justificari](#)

TEMA

- profesorul de seminar poate pune întrebări pe loc legate de rezolvările din tema
- La examenul final (în funcție de cum se va face examinarea) profesorul poate pune întrebări și din tema.





Next time:

Saptamana 3:

Curs 3: TSP & Christofides

Vom discuta a doua parte din Tema 1

