Metoda GREEDY

Metoda Greedy

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

Se dă o mulțime finită A.

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

+

îndeplinește un **criteriu de optim** (este **soluție optimă**), adică minimizează/maximizează o funcție obiectiv f

Metoda Greedy

- **Probleme de optim**

- **Exemplu:** Dată o mulțime de intervale, să se determine o submulțime de cardinal maxim de intervale care nu se suprapun (activități cu intervale de desfășurare compatibile)

    - Soluție posibilă = o submulțime de intervale disjuncte
    - Soluție optimă = soluție posibilă de **cardinal maxim**

Metoda Greedy

- **Strategie**: Se încearcă o construire directă a **<u>unei</u>** soluții optime, **<u>element cu element</u>**

- Elementul ales la un pas pentru a se adăuga în soluție – ***cel care*** *pare cel mai "bun" la acel pas*, conform criteriului de optim

Metoda Greedy

- **Strategie**: Se încearcă o construire directă a **unei** soluții optime, **element cu element**

- Elementul ales la un pas pentru a se adăuga în soluție – ***cel care pare cel mai "bun" la acel pas***, conform criteriului de optim

- Nu este garantată obținerea unei soluții optime ⇒ **aplicarea metodei trebuie însoțită de demonstrația corectitudinii**

Metoda Greedy

- **Pentru o** mulțime finită $A=\{a_1,...,a_n\}$ notăm

  $\mathcal{P}(A)$ = mulțimea submulțimilor lui A

Metoda Greedy

- **Cadru formal**:
  - O mulțime finită $A=\{a_1,\ldots,a_n\}$
  - O funcție $f:\mathcal{P}(A)\rightarrow\mathbf{R}$ – **trebuie minimizată/maximizată**

Metoda Greedy

- **Cadru formal**:
  - O mulțime finită $A=\{a_1,\ldots,a_n\}$
  - O funcție $f:\mathcal{P}(A)\rightarrow\mathbf{R}$ – **trebuie minimizată/maximizată**
  - O **proprietate** definită pe mulțimea submulțimilor lui A

    $p:\mathcal{P}(A)\rightarrow\{0,1\}$ => **criteriul pentru ca un element să poată fi adăugat la soluția construită până la pasul curent**

Metoda Greedy

- **Cadru formal**:
  - O mulțime finită $A=\{a_1,\ldots,a_n\}$
  - O funcție $f:\mathcal{P}(A)\rightarrow\mathbf{R}$ – **trebuie minimizată/maximizată**
  - O **proprietate** definită pe mulțimea submulțimilor lui A

    $p:\mathcal{P}(A) \rightarrow \{0,1\}$ => **criteriul pentru ca un element să poată fi adăugat la soluția construită până la pasul curent**

O submulțime $S \subseteq A$ sn **soluție posibilă** dacă $p(S)=1$.

Metoda Greedy

- **Exemplu:** Dată o mulțime de intervale, să se determine o submulțime de cardinal maxim de intervale care nu se suprapun (activități cu intervale de desfășurare compatibile)

  - A – mulțimea de intervale dată
  - $X \in \mathcal{P}(A)$: $p(X) = 1$ (X este soluție posibilă)

    $\Leftrightarrow$ X este submulțime de intervale disjuncte
  - $f(X) = |X|$ – maximizat

Metoda Greedy

- **Două variante** (modalități de abordare)

  - Varianta 1 – **fără prelucrare inițială**: **elementul** care se adaugă la soluție se stabilește **la fiecare pas**, în funcție de alegerile anterioare

  - Varianta 2 – **cu prelucrare inițială:** ordinea în care sunt considerate elementele se stabilește de la început

Metoda Greedy

- **Două variante** (modalități de abordare)

  - **Varianta 1** – **fără prelucrare inițială**:

```
S←ϕ
for i=1,n
    x ← alege(A);
    A ← A - {x}
    if p(S∪{x})=1
        S ← S ∪ {x}
```

Metoda Greedy

- **Două variante** (modalități de abordare)

  - **Varianta 2** – **cu prelucrare inițială:**

```
prelucreaza(A)
S←ϕ
for i=1,n
    if p(S∪{a_i})=1
        S ← S ∪ {a_i}
```

Metoda Greedy

- **Două variante** (modalități de abordare)

  - **Varianta 2 – cu prelucrare inițială:**

```
prelucreaza(A)
S←ϕ
for i=1,n
    if p(S∪{ai})=1
        S ← S ∪ {ai}
```

În algoritm nu apare funcţia f ⇒ **nevoie de a demonstra corectitudinea**

Metoda Greedy

- <u>**Exemplul 1**</u> Se consideră mulțimea de valori reale $A=\{a_1,...,a_n\}$. Să se determine o submulțime a lui A a cărei sumă a elementelor este maximă.

- <u>**Exemplul 2**</u> **(Contraexemplu)** Se consideră mulțimea $A=\{a_1,...,a_n\}$ cu elemente **pozitive și un număr natural M**. Să se determine o submulțime a lui A de sumă maximă, dar cel mult egală cu o valoare M dată.

Metoda Greedy

- **<u>Exemplul 1</u>** Se consideră mulțimea de valori reale $A=\{a_1,\ldots,a_n\}$. Să se determine o submulțime a lui A a cărei sumă a elementelor este maximă.

Metoda Greedy

- **<u>Exemplul 3</u>** (Seminar). Se dă o sumă S și avem la dispoziție monede cu valorile: 1, 5, 10, 25 (un număr nelimitat de monede). Să se determine o modalitate de a plăti suma S folosind un număr minim de monede.

Algoritmul propus este corect și dacă aveam bancnote cu valorile 1, 10, 30, 40? Justificați.

Exemple

Minimizarea timpului mediu de acces pentru acces secvențial

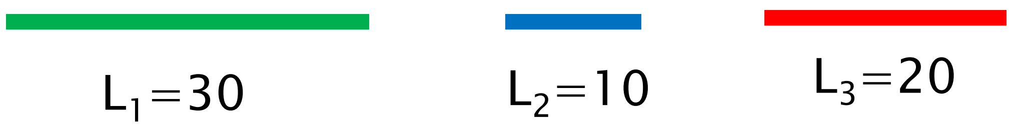
- n texte cu lungimile L(1),...,L(n) urmează a fi așezate pe o bandă cu acces secvențial

  Acces secvențial = pentru a citi textul de pe poziția k, trebuie citite textele de pe pozițiile 1,2,...,k

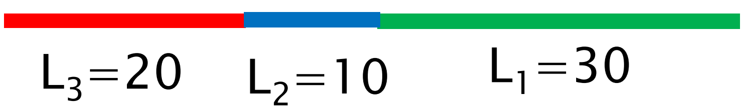
  Să se determine o modalitate de așezare a textelor pe bandă astfel încât timpul mediu de acces să fie minimizat.

Minimizarea timpului mediu de acces pentru acces secvențial

- Exemplu



Pentru așezarea



timpul mediu de acces este...

Minimizarea timpului mediu de acces pentru acces secvențial

- **Variantă de enunț:** Un evaluator de proiecte (bucătar) urmează să evalueze n proiecte (să efectueze n comenzi) depuse de directorii de proiect. Pentru fiecare proiect se știe durata de evaluare.

  Care este ordinea în care se va face evaluarea astfel încât să se minimizeze timpul total de așteptare al directorilor de proiect (clienților)?

  - Răspunsul este dat celui care a depus proiectul imediat după terminarea evaluării acestuia (nu a tuturor proiectelor)

Minimizarea timpului mediu de acces pentru acces secvențial

- Reprezentarea soluției

- Timpul mediu de acces (funcția de optimizat)

Minimizarea timpului mediu de acces pentru acces secvențial

- Reprezentarea soluției

  O permutare σ a mulțimii {1, …, n}

- Timpul mediu de acces (funcția de optimizat)

$$T(\sigma) = \frac{1}{n}\sum_{k=1}^{n}(L_{\sigma(1)} + \ldots + L_{\sigma(k)}) =$$

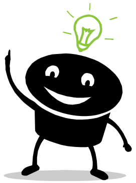
$$= \frac{1}{n}\sum_{k=1}^{n}(n-k+1)L_{\sigma(k)}$$

Minimizarea timpului mediu de acces pentru acces secvențial

Care este primul text pe care îl așezăm pe bandă?

Minimizarea timpului mediu de acces pentru acces secvențial

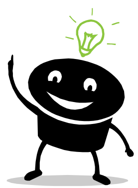
Care este primul text pe care îl așezăm pe bandă?



Cel cu lungimea cea mai mică, pentru că el va fi accesat de câte ori accesăm un alt text

Minimizarea timpului mediu de acces pentru acces secvențial

Care este primul text pe care îl așezăm pe bandă?



Cel cu lungimea cea mai mică, pentru că el va fi accesat de câte ori accesăm un alt text

$\Rightarrow$ Așezăm textele pe bandă în **ordine crescătoare** în raport cu lungimea lor

Complexitate O(nlogn)

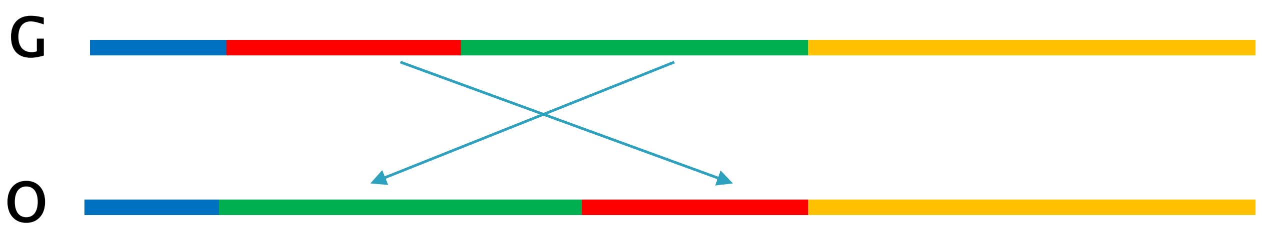
Minimizarea timpului mediu de acces pentru acces secvențial


Este corect algoritmul?

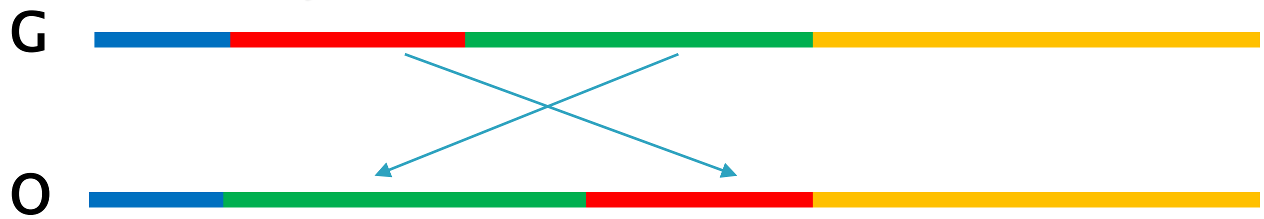
Minimizarea timpului mediu de acces pentru acces secvențial

Este corect algoritmul?

Comparăm soluția dată de algoritmul de tip Greedy cu una optimă



Minimizarea timpului mediu de acces pentru acces secvențial



Greedy – permutarea identică

Soluția optimă – permutarea p – are minim o inversiune

⇒ o nouă permutare p' cu mai puține inversiuni

p: ...., p_i ,............, p_j ,........

p: ...., p_j ,............, p_i ,........

p' este optimă?

Detalii – la curs + pdf greedy

Minimizarea timpului mediu de acces pentru acces secvențial

- **Alte tipuri de probleme**

1. Fiecare text are asociată o frecvență de citire
2. Avem la dispoziție p benzi (p>=1)
3. Activitățile au și termen limită, profit...

Teme laborator Greedy

Probleme de planificare activități cu intervale de desfășurare date

Problema spectacolelor (selecția unui număr maxim de intervale disjuncte)

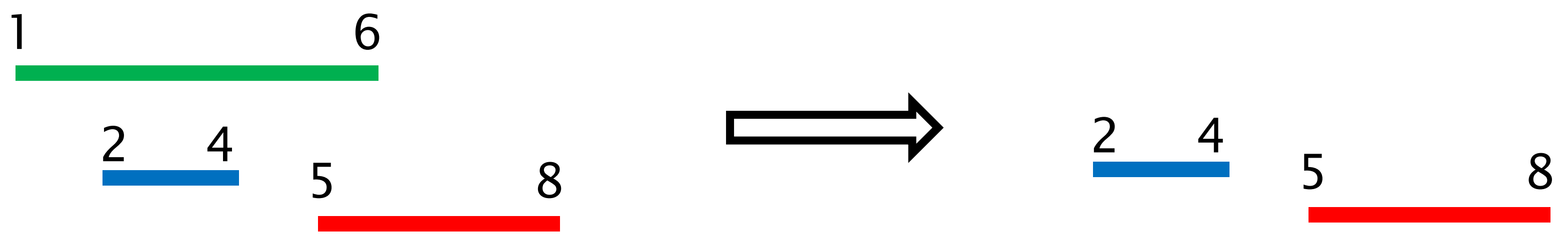
Se dă o mulțime de n intervale disjuncte (reprezentând intervalele de desfășurare a n activități, spre exemplu spectacole)

Să se determine o submulțime de cardinal maxim de intervale disjuncte două câte două (de activități compatibile, care se pot desfășura folosind o singură resursă)

Probleme de planificare activități cu intervale de desfășurare date

Problema spectacolelor (selecția unui număr maxim de intervale disjuncte)

Exemplu

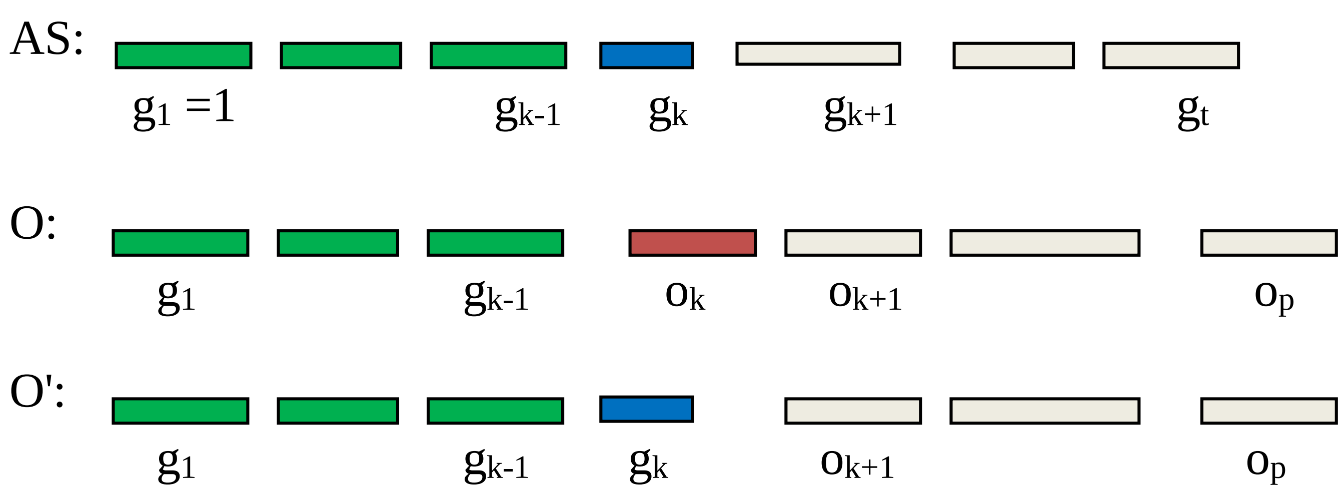


Soluție optimă

Probleme de planificare activități cu intervale de desfășurare date

Problema spectacolelor (selecția unui număr maxim de intervale disjuncte)

Soluție + corectitudine – la curs + pdf greedy



Probleme de planificare activități cu intervale de desfășurare date

Problema spectacolelor (selecția unui număr maxim de intervale disjuncte)

Variante

- Activitățile au asociate și **profituri**. Să se determine o submulțime de activități compatibile având profitul total maxim (un algoritm de tip Greedy similar celui pentru selectarea submulțimii de cardinal maxim nu mai este corect)

Probleme de planificare activități cu intervale de desfășurare date

Problema spectacolelor (selecția unui număr maxim de intervale disjuncte)

Variante

- **Problema partiționării intervalelor**: De câte resurse este nevoie pentru a putea planifica toate activitățile date + o astfel de planificare (laborator Greedy)

Probleme de planificare activități cu intervale de desfășurare date

Problema partiționării intervalelor: De câte resurse este nevoie pentru a putea planifica toate activitățile date + o astfel de planificare (laborator Greedy)

= să se împartă (partiţioneze) o mulţime de intervale date într-un număr minim de submulţimi cu proprietatea că oricare două intervale dintr-o submulţime nu se intersectează şi să se afişeze aceste submulţimi

Problema partiționării intervalelor

[2,3], [5,8], [1, 4], [7,12], [1,6]

=> 3 săli

Sala 1: [1,4] [5,8]

Sala 2: [2,3] [7,12]

Sala 3: [1,6]

Problema partiționării intervalelor

Soluție Greedy:

- Se sortează intervalele crescător după extremitatea inițială

- Pentru fiecare interval (de desfășurare) I în această ordine execută:

– se adaugă I **la o sală deja existentă** (submulţime deja construită) , **dacă se poate** = dacă nu se intersectează cu niciun interval din ea (!! nu contează la care se adaugă, dacă există mai multe variante)

– altfel **se creează o nouă sală (submulţime)** cu intervalul I

Problema partiționării intervalelor

[1, 4], [1,6], [2,3], [5,8], [7,12],

Sala 1: [1,4] [5,8]

Sala 2: [1,6] [7,12]

Sala 3: [2,3]

Problema partiționării intervalelor

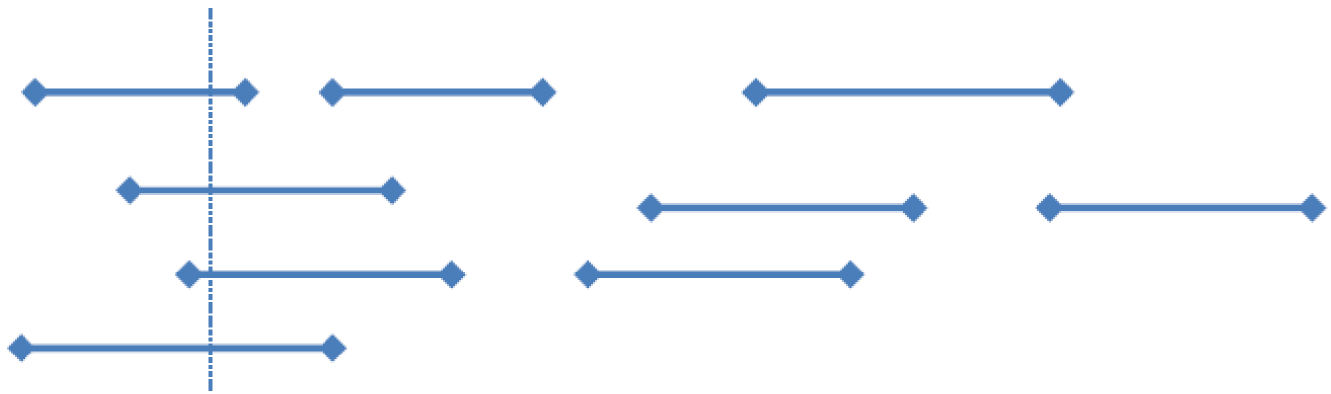
Corectitudine

- **Observații**

Numărul de săli (de submulțimi) $\geq$

numărul maxim de intervale care au un punct comun

(pentru că acestea trebuie obligatoriu programate în săli diferite)



Problema partiționării intervalelor

Corectitudine

- **Observații**

Numărul de săli (de submulțimi) $\geq$

numărul maxim de intervale care au un punct comun

Dacă are loc egalitate, atunci soluția este optimă

Problema partiționării intervalelor

Corectitudine

- Fie k numărul de săli (submulțimi) determinat de algoritmul Greedy.
- **Este suficient să demonstrăm că există k intervale care au un punct comun**

Problema partiționării intervalelor

Corectitudine

- Fie I = [s, t] primul interval adăugat în sala k.

- :

Problema partiționării intervalelor

Corectitudine

- Fie I = [s, t] primul interval adăugat în sala k.
- I a fost adăugat într-o sală nouă pentru că se intersecta cu cel puțin un interval $I_i=[s_i, t_i]$ din fiecare sală i = 1, 2, ..., k-1

Problema partiționării intervalelor

Corectitudine

- Deoarece intervalele au fost considerate în ordine crescătoare după timpul de început avem $s_i \leq s$ pentru orice $i = 1, \ldots, k-1$

Dar $I_i \cap I = [s_i, t_i] \cap [s, t] \neq \varnothing \quad \Rightarrow \quad s \in I_i$

Rezultă că cele k intervale $I, I_1, \ldots, I_{k-1}$ conțin punctul s.

Problema partiționării intervalelor

Complexitate

- Este suficient să testăm dacă intervalul curent I se poate adăuga la sala (submulțimea) care are **timpul maxim de terminare** al unui interval programat în ea **cel mai mic => O(n log(n))**

```
[1, 4], [1,6], [2,3], [5,8], [7,12]

Sala 1: [1,4]
Sala 2: [1,6]
Sala 3: [2,3]
```

Problema partiționării intervalelor

Complexitate

- Este suficient să testăm dacă intervalul curent I se poate adăuga la sala (submulțimea) care are **timpul maxim de terminare** al unui interval programat în ea **cel mai mic** => **O(n log(n))**

```
[1, 4], [1,6], [2,3], [5,8], [7,12],

Sala 1: [1,4]
Sala 2: [1,6]
Sala 3: [2,3] [5,8]
```

Problema partiționării intervalelor

Complexitate

- Este suficient să testăm dacă intervalul curent I se poate adăuga la sala (submulțimea) care are **timpul maxim de terminare** al unui interval programat în ea **cel mai mic => O(n log(n))**

```
[1, 4], [1,6], [2,3], [5,8], [7,12],

Sala 1: [1,4]
Sala 2: [1,6]
Sala 3: [2,3] [5,8]
```

Problema partiționării intervalelor

Complexitate

- Este suficient să testăm dacă intervalul curent I se poate adăuga la sala (submulțimea) care are **timpul maxim de terminare** al unui interval programat în ea **cel mai mic** => **O(n log(n))**

```
[1, 4], [1,6], [2,3], [5,8], [7,12]

Sala 1: [1,4] [7,12]
Sala 2: [1,6]
Sala 3: [2,3] [5,8]
```

Problema rucsacului

- **Varianta continuă (fracționară)**:

Se consideră un rucsac de capacitate (greutate) maximă G și n obiecte caracterizate prin următoarele **numere reale pozitive**:

- greutățile lor $g_1,\ldots,g_n$ – numere **reale pozitive**
- câștigurile $c_1,\ldots,c_n$ obținute la încărcarea lor **<u>în totalitate</u>** în rucsac.

Problema rucsacului

- **Varianta continuă (fracționară)**:

Se consideră un rucsac de capacitate (greutate) maximă G și n obiecte caracterizate prin următoarele **numere reale pozitive**:

- greutățile lor $g_1,\ldots,g_n$ – numere **reale pozitive**
- câștigurile $c_1,\ldots,c_n$ obținute la încărcarea lor **în totalitate** în rucsac.

Din fiecare obiect poate fi încărcată orice fracțiune a sa.

Să se determine o modalitate de încărcare de (fracțiuni de) obiecte în rucsac, astfel încât **câștigul total** să fie maxim.

Presupunem $g_1 + \ldots + g_n > G$

Problema rucsacului

- Exemplu

```
G = 8
n =  4
c  =  8    12    6    10
g  =  4     8    3    10
```

Ob.1 Ob.2 Ob.3 Ob.4

Problema rucsacului

- **Variantă de enunț:**

Se dau

T=timp total de funcționare a unei resurse

n activități cu **durata** d_i și **profitul** p_i (dacă activitatea se execută în totalitate) care necesită resursa

O activitate începută poate fi **întreruptă obținându-se un profit parțial.**

Se cere: profitul maxim care se poate obține

Problema rucsacului

- Reprezentarea soluției

- Câștigul total corespunzător unei soluții posibile (funcția de optimizat)

Problema rucsacului

- Reprezentarea soluției

  $x = (x_1, \ldots, x_n)$ cu x_i = fracțiunea încărcată din obiectul i

  $$\sum_{i=1}^{n} x_i g_i \leq G$$

- Câștigul total corespunzător unei soluții posibile (funcția de optimizat)

$$f(x) = \sum_{i=1}^{n} x_i c_i$$

Problema rucsacului

- Exemplu

```
G = 8
n =  4
c  =  8    12    6    10
g  =  4     8    3    10
```

	Ob.1	Ob.2	Ob.3	Ob.4
c	8	12	6	10
g	4	8	3	10

Soluție:

`x =  (1, 1/8,  1,  0)`

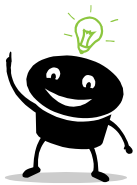
Câștig total?

Problema rucsacului

- Care este primul obiect pe care îl încărcăm în rucsac + ce fracțiune din el?

Problema rucsacului

Care este primul obiect pe care îl încărcăm în rucsac + ce fracțiune din el?



- cel cu câștigul **pe unitate** $\mathbf{c_i/g_i}$ cel mai mare (!!!! nu cel cu câștigul c_i cel mai mare)
- încărcăm o fracțiune cât mai mare, fără a depăși greutatea

Problema rucsacului

- Exemplu

G = 8

n = 4

c	=	8	12	6	10
g	=	4	8	3	10
		↑	↑	↑	↑
		Ob.1	Ob.2	Ob.3	Ob.4
r	=	2	1,5	2	1

câştigul pe unitatea de greutate

Obiectul 1 – întreg -> rămâne G=4

Obiectul 3 – întreg -> rămâne G=1

Obiectul 2 – 1/8 -> rămâne G=0

Problema rucsacului

Pseudocod

- Presupunem $g_1+...+g_n>G$
- ordonăm obiectele descrescător după câștigul pe unitatea de greutate – presupunem obiectele renotate astfel încât

$$\frac{c_1}{g_1} \geq \frac{c_2}{g_2} \geq \ldots \geq \frac{c_n}{g_n}$$

Problema rucsacului

Pseudocod

```
Gr ← G { Gr = greutatea ramasa }
for i=1,n
   if g_i ≤ Gr
       x_i←1; Gr ← Gr-g_i
   else
```

Problema rucsacului

Pseudocod

```
Gr ← G { Gr = greutatea ramasa }
for i=1,n
   if g_i ≤ Gr
      x_i←1; Gr ← Gr-g_i
   else
      x_i←Gr/g_i;
      for j=i+1,n
          x_j ← 0
      stop
write(x)
```

Problema rucsacului

Pseudocod

```
Gr ← G { Gr = greutatea ramasa }
for i=1,n
    if g_i ≤ Gr
        x_i←1; Gr ← Gr-g_i
    else
        x_i←Gr/g_i;
        for j=i+1,n
            x_j ← 0
        stop
write(x)
```

Complexitate O(nlogn)

Problema rucsacului

Observație:

Soluția obținută de algoritm – **cel mult un obiect este fracționat**

$$\mathbf{x}=(1,...,1,\mathbf{x}_j,0,...,0) \text{ cu } \mathbf{x}_j \in [0,1)$$

Problema rucsacului

- Este corect algoritmul?

Problema rucsacului

- Este corect algoritmul?
- Dar dacă obiectele nu se pot fracționa?