

## Metoda Greedy

### PREZENTARE GENERALĂ

Metoda „Greedy”:

- Este aplicabilă problemelor de optim;
- Alege soluția optimă sau o aproximare a soluției optime
- Presupune o alegere optimă pentru fiecare element adăugat soluției (optim local).
- Este aplicabilă unor probleme pentru care nu sunt cunoscuți algoritmi polinomiali, evită generarea tuturor soluțiilor posibile în timp exponențial.

### ALGORITMUL GENERAL

```
S = ∅
for i = 1,n
    x = Alege(A)
    A = A \ { x }
    If check ( S ∪ {x} )
        S = S ∪ {x}
```

Variantă:

```
S = ∅
prel(A)
for i = 1,n
    if check ( S ∪ { ai } )
        S = S ∪ { ai }
```

### EXEMPLE DE PROBLEME

1. **(Problema continuă a rucsacului)** Fiind dat un rucsac care are capacitatea  $G$  (greutatea maximă ce poate fi încărcată în rucsac) și  $n$  obiecte, pentru fiecare obiect fiind cunoscute greutatea  $g_i$  și profitul total obținut prin încărcarea obiectului în întregime  $c_i$ , să se obțină profitul maxim obținut prin încărcarea rucsacului. Un obiect poate fi încărcat parțial.

**Soluție:** Se ordonează obiectele în funcție de profitul obținut pe unitate  $c_i/g_i$ . Se vor încarca obiectele  $1, 2, \dots, u-1$  în întregime iar din ultimul obiect  $u \leq n$ , se va încarca greutatea

$$a_u = G - (g_1 + g_2 + \dots + g_{u-1}).$$

$u$  este cel mai mare indice pentru care este adevărată inegalitatea

$$g_1 + g_2 + \dots + g_{u-1} < G.$$

Obiectele  $u+1, \dots, n$  nu vor fi încărcate.

Astfel solutia data de algoritmul Greedy se poate nota astfel, in vectorul  $a$  componenta  $i$  reprezinta greutatea incarcata din obiectul  $i$ :

$$a = (a_1, a_2 \dots a_n) = (g_1, \dots g_{u-1}, a_u, 0 \dots 0).$$

#### Demonstrarea corectitudinii:

Profitul obtinut in urma incarcarii obiectelor propusa de algoritmul greedy este:

$$P_a = \sum_{i=1}^n (a_i * \frac{c_i}{g_i}).$$

Presupunem ca ar exista o solutie optima  $o = (o_1, o_2 \dots o_n)$  diferita de solutia Greedy. Pentru orice solutie optima suma greutatilor incarcate este  $G$ .

$$G = o_1 + \dots + o_n.$$

Fie  $1 \leq p \leq n$  prima pozitie pe care  $o$  difera de  $a$ .

Deoarece suma greutatilor incarcate de solutia  $o$  nu poate depasi  $G$  deducem ca:

$$p \leq u.$$

$$o_p < a_p.$$

Constuim solutia  $o'$  inlocuind greutatea incarcata din obiectul  $p$ , cu greutatea incarcata din obiectul  $p$  la algoritmul greedy. Primele  $p-1$  obiecte sunt incarcate cu aceleasi greutati iau greutatile obiectelor de la  $p+1$  la  $n$  sunt modificate proportional astfel incat sa se ocupe complet greutatea  $G$ .

$$o' = (o_1, \dots, o_{p-1}, a_p, \alpha o_{p+1}, \dots, \alpha o_n) \text{ cu } \alpha < 1 \text{ astfel incat:}$$

$$G = o_1 + \dots + o_{p-1} + a_p + \alpha o_{p+1} + \dots + \alpha o_n.$$

De unde deducem

$$G = o_1 + \dots + o_n = o_1 + \dots + o_{p-1} + a_p + \alpha o_{p+1} + \dots + \alpha o_n.$$

$$(*) a_p - o_p = (1 - \alpha)(o_{p+1} + \dots + o_n).$$

Comparam profitul obtinut de  $o'$  cu profitul obtinut de  $o$  si obtinem:

$$P_{o'} - P_o = \sum_{i=p}^n (o'_i - o_i) * \frac{c_i}{g_i} > 0$$

Ultima inegalitate este adevarata deoarece avem ca:

$$P_{o'} - P_o = (a_p - o_p) * \frac{c_p}{g_p} + (\alpha o_{p+1} - o_{p+1}) * \frac{c_{p+1}}{g_{p+1}} + \dots + (\alpha a_n - o_n) * \frac{c_n}{g_n}$$

iar conform (\*) si a ordonarii obiectelor in functie de profitul pe unitate:

$$\begin{aligned}
P_{O'} - P_O &= (1 - \alpha)(o_{p+1} + \dots + o_n) * \frac{c_p}{g_p} + (\alpha - 1)o_{p+1} * \frac{c_{p+1}}{g_{p+1}} + \dots + (\alpha - 1)o_n * \frac{c_n}{g_n} \\
&= \sum_{i=p+1}^n (1 - \alpha)o_i \left( \frac{c_p}{g_p} - \frac{c_i}{g_i} \right) > 0
\end{aligned}$$

Deci  $o$  nu este soluție optimă, contradicție.

2. Se dau  $n$  activități. Pentru fiecare activitate este cunoscut timpul de start  $s_i$  timpul de final  $f_i$ . Activitățile sunt împartite în grupuri. Pentru fiecare activitate este cunoscut grupul  $g_i$  din care face parte. Să se planifice o submulțime maximă de activități astfel încât intervalele de desfășurare  $[s_{a_i}, f_{a_i}]$  să fie disjuncte două câte două și să nu fie planificate două activități din același grup:  $g_{a_i} \neq g_{a_j}$  pentru orice  $i \neq j$ .

**Problema este NP-hard.**

**Algoritm Greedy:** Se parcurg în ordine activitățile după ora de sfârșit. Activitatea  $i$  se adaugă la soluție dacă nu se intersectează cu celelalte activități planificate (este suficient să nu se intersecteze cu ultima planificată) și grupul din care face parte nu este încă reprezentat de o activitate deja planificată.

**Aproximare a soluției optime:** Se demonstrează că numărul de activități planificate de soluția Greedy este, în cazul cel mai defavorabil, de două ori mai mic decât numărul de activități care se pot planifica astfel încât să fie respectate cerințele.

$$n_g \geq n_o/2$$

**Demonstratie:** Fie  $O$  o submulțime de cardinal maxim, de activități care pot fi planificate fără suprapuneri și astfel încât oricare două să fie din grupuri diferite.

Fie  $G$  submulțimea de activități planificate de algoritmul Greedy.

Pentru a demonstra că  $n_g \geq n_o/2$  alegem o funcție  $f: O \rightarrow G$  astfel încât să nu existe 3 activități  $I_1, I_2, I_3 \in O$  și  $J \in G$  a.i.  $f(I_1) = f(I_2) = f(I_3) = J$ .

Fie  $f: O \rightarrow G$  definită astfel

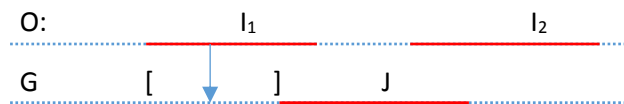
$$f(I) = \begin{cases} J, & (\alpha) J \text{ ales din grupul } g_I \text{ (} g_I = g_J \text{) dacă } g_I \text{ este reprezentat sau} \\ \beta, & (\beta) \text{ primul interval (în ordinea după } f_i \text{) care se intersectează cu } I. \end{cases}$$

$f$  este funcție (este bine definită pentru orice  $I$ ): Dacă grupul activității  $I$  nu este reprezentat în  $G$ , există o activitate  $J$  care să se intersecteze cu  $I$ , altfel putem adăuga activitatea  $I$  la soluția  $G$ . Nu pot exista două activități în  $G$  care să reprezinte același grup. Nu pot exista două activități în  $G$  care au aceeași ora de final.

Presupunem că ar exista 3 activități  $I_1, I_2, I_3 \in O$  și  $J \in G$  a.i.  $f(I_1) = f(I_2) = f(I_3) = J$ .

Conform **principiului cutiei (Dirichlet:** dacă  $n$  obiecte trebuie incluse în  $m$  cutii și  $n > m$ , cel puțin 1 cutie va conține cel puțin 2 obiecte).  $\exists i_1, i_2$  a.i.  $f(I_{i_1}), f(I_{i_2})$  sunt alese ambele în modul  $\alpha$  sau ambele în modul  $\beta$ . Presupunem  $i_1 = 1, i_2 = 2$ .

- 1) Dacă  $f(I_1) = f(I_2) = J$  și intervalul  $J$  a fost ales în modul  $\alpha$ , atunci soluția  $O$  nu este corectă,  $I_1, I_2$  aparțin aceluiași grup.
- 2) Dacă  $f(I_1) = f(I_2) = J$  (presupunem  $s_{I_1} < s_{I_2}$ ) și intervalul  $J$  a fost ales în modul  $\beta$  atunci intervalul  $[s_{I_1}, s_J]$  nu se intersectează cu nicio activitate planificată în soluția Greedy. Altfel  $J$  nu ar fi fost primul interval în soluția Greedy care se intersectează cu  $I_1$ . Dar în acest caz algoritmul Greedy ar fi ales  $I_1$  iar dacă în  $G$  în loc de  $J$  avem  $I_1$ , dacă  $f(I_2)$  este ales în modul  $\beta$  nu poate fi ales  $I_1$  deoarece  $I_1$  nu se intersectează cu  $I_2$ .



3. (tema varianta 2.1.b) **(Memorarea textelor pe bandă)**  $n$  texte cu lungimile  $L(1), \dots, L(n)$  urmează a fi așezate pe  $p$  benzi ( $p$  dat). Pentru a citi textul de pe poziția  $k$  de pe o bandă, trebuie citite textele de pe pozițiile  $1, 2, \dots, k$  de pe aceeași bandă. Fiecare text are aceeași frecvență de citire. Să se determine o modalitate de așezare a textelor pe cele  $p$  benzi astfel încât timpul total de acces să fie minimizat (se obține adunând timpul total de acces pentru fiecare din cele  $p$  benzi). Dacă notăm cu  $\sigma_{ki}$  al  $i$ -lea text așezat pe banda  $k$ , și cu  $n_k$  numărul total de texte așezate pe banda  $k$ , trebuie minimizat costul:

$$C_\sigma = \sum_{k=1}^p \sum_{i=1}^{n_k} L(\sigma_{k1}) + \dots + L(\sigma_{ki}).$$

**Algoritm Greedy:** Se sortează textele crescător după lungime. Se așază primele  $p$  texte în ordine pe prima poziție din cele  $p$  benzi. Următoarele texte, de la  $p+1$  la  $2*p$  vor fi așezate pe poziția a doua din fiecare bandă etc.

$$\sigma_{ki}^g = p * (i - 1) + (k - 1)$$

**Demonstratie corectitudine** Fie  $\sigma_{ki}^o$  o așezare optimă a textelor pe cele  $p$  benzi.

În orice soluție optimă toate cele  $p$  benzi conțin cel puțin un text.

Obs 1) Pe fiecare bandă textele sunt așezate în ordine crescătoare după lungimea textelor.

Obs 2) Benzile pot fi renumerotate astfel încât  $n_1 \geq n_2 \geq \dots \geq n_p$ . În plus avem ca:

$$(*) \forall k_1, k_2 \text{ cu } k_1 < k_2 : n_{k_1} - n_{k_2} \leq 1.$$

Presupunem că ar exista două benzi  $k_1, k_2$  cu  $k_1 < k_2$  și  $n_{k_1} - n_{k_2} > 1$ . Mutăm primul text așezat pe banda  $k_1$  astfel încât să fie primul text de pe banda  $k_2$ . Din costul pentru citirea fiecărui text de pe banda  $k_1$  se scade lungimea textului mutat  $L(\sigma_{k_1 1}^o)$ . La costul citirii oricărui text de pe banda  $k_2$  se adună lungimea textului mutat  $L(\sigma_{k_1 1}^o)$ . Se obține așezarea  $\sigma^{o'}$  care are costul:

$$\begin{aligned} C(\sigma^{o'}) &= C(\sigma^o) - L(\sigma_{k_1 1}^o) * n_{k_1} + L(\sigma_{k_1 1}^o) * (n_{k_2} + 1) \\ &= C(\sigma^o) - L(\sigma_{k_1 1}^o)(n_{k_1} - n_{k_2} - 1) < C(\sigma^o) \end{aligned}$$

Din (\*) deducem că există o soluție optimă care are pe fiecare bandă același număr de texte ca soluția Greedy.

Obs 3) Textele asezate pe prima pozitie din cele  $p$  benzi sunt ordonate crescator

$$(**) L(\sigma_{11}^o) \leq L(\sigma_{21}^o) \leq \dots \leq L(\sigma_{p1}^o)$$

Presupunem ca ar exista doua benzi  $k_1, k_2$  cu  $k_1 < k_2$   $L(\sigma_{k_1 1}^o) > L(\sigma_{k_2 1}^o)$

Stim din (\*) ca  $0 \leq n_{k_1} - n_{k_2} \leq 1$ .

Interschimbam primul text de pe benzile  $k_1$  si  $k_2$ . Se obtine diferenta intre costuri:

$$\begin{aligned} C(\sigma^{o'}) &= C(\sigma^o) + (L(\sigma_{k_2 1}^o) - L(\sigma_{k_1 1}^o)) * n_{k_1} + (L(\sigma_{k_1 1}^o) - L(\sigma_{k_2 1}^o)) * n_{k_2} \\ &= C(\sigma^o) - (L(\sigma_{k_1 1}^o) - L(\sigma_{k_2 1}^o))(n_{k_1} - n_{k_2}) < \sigma^o \end{aligned}$$

Obs 4) Textele asezate pe prima pozitie din cele  $p$  benzi sunt primele  $p$  texte daca se considera textele ordonate crescator dupa lungime:  $L(1) \leq L(2) \leq \dots \leq L(p)$ . Altfel spus, primele  $p$  texte sunt asezate ca in solutia Greedy.

$$(**) \forall k \quad L(\sigma_{k1}^o) = k.$$

Vom arata pentru  $k = 1, 2$  ca  $\sigma_{11}^o = 1$   $\sigma_{21}^o = 2$ . Pentru  $k \neq 1, 2$  se procedeaza inductiv in acelasi mod.

Este evident din observatiile 1 si 3 ca  $\sigma_{11}^o = 1$ .

Presupunem ca  $\sigma_{21}^o \neq 2$ . Tot din observatia 3 rezulta ca  $\exists j, 2 \leq j \leq n_1^o$  astfel incat:

$$L(1) \leq L(2) \leq \dots \leq L(\sigma_{1j-1}^o) \leq L(\sigma_{1j}^o) \leq L(\sigma_{21}^o) \leq L(\sigma_{1j+1}^o).$$

Mutam textul 2 de pe banda 1 astfel incat sa fie primul text pe banda 2 ( $\sigma_{21}^{o'} = 2$ ). Mutam primul text asezat pe banda 2 astfel incat sa fie al  $j$ -lea text asezat pe banda 1. Textele de la 3 la  $j$  se deplaseaza spre stanga cu o pozitie.  $\sigma_{1j}^{o'} = \sigma_{21}^o, \sigma_{1i}^{o'} = \sigma_{1i+1}^o$  pentru  $2 \leq i \leq j-1$ .

Se obtine diferenta intre costuri:

$$\begin{aligned} C(\sigma^{o'}) &= C(\sigma^o) + (L(2) - L(\sigma_{21}^o)) * n_2 + (L(3) - L(2)) + \dots + (L(\sigma_{1j}^o) - L(2)) \\ &\quad + (L(\sigma_{21}^o) - L(2))(n_1 - j + 1) \\ &\leq C(\sigma^o) + (L(2) - L(\sigma_{21}^o)) * n_2 + (L(\sigma_{21}^o) - L(2)) * (n_1 - 1) \\ &\leq C(\sigma^o) - (L(\sigma_{21}^o) - L(2)) * (1 + n_2 - n_1) < C(\sigma^o) \end{aligned}$$

4. **(tema varianta 2.1.a)** Se dau  $n$  cuburi cu laturile **diferite două câte două**. Fiecare cub are o culoare, codificată cu un număr de la 1 la  $p$  ( $p$  dat). Să se construiască un turn de înălțime maximă de cuburi în care un cub nu poate fi așezat pe un cub de aceeași culoare sau cu latură mai mică decât a sa.

**Algoritm Greedy:** La primul pas se selectează cubul cu latura cea mai mare. Ulterior, la fiecare pas este selectat cubul cu latura cea mai mare care se poate așeza peste ultimul cub selectat. Pentru aceasta vom ordona cuburile descrescător după latură.

**Demonstratie Corectitudine** Notăm cu  $l_i, i=1..n$  laturile cuburilor și cu  $c_i, i=1..n$  culorile cuburilor.

**Renumerotăm cuburile astfel încât  $l_1 > l_2 > \dots > l_n$ . Astfel, primul cub selectat de algoritmul greedy este cubul 1. Fie  $G = \{g_1=1, \dots, g_t\}$ ,  $t \leq p$  (cu  $g_1 < \dots < g_t$ ) soluția Greedy.**

**Varianta 1 (similar problemei rucsacului):**

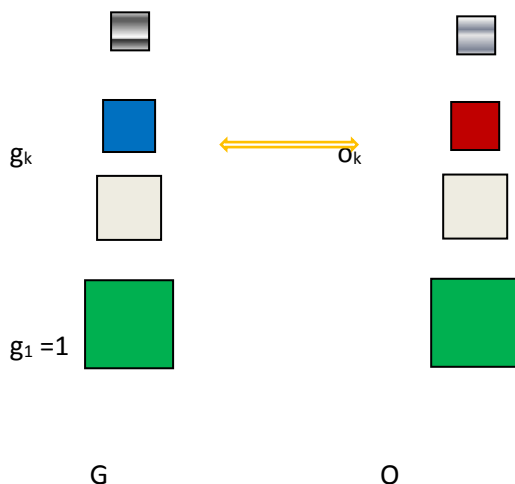
Considerăm o soluție optimă  $O = \{o_1, \dots, o_p\}$  (cu cuburile notate astfel încât  $o_1 < \dots < o_p$  care are **un număr maxim de elemente inițiale în comun cu soluția Greedy  $G$** . Presupunem  $O \neq G$ .

Atunci există un indice  $k \leq t$  astfel încât  $g_k \neq o_k$ , altfel am avea  $g_1 = o_1, \dots, g_t = o_t$ , adică  $G \subseteq O$  și  $t < p$ . Dar, deoarece cubul  $o_{t+1}$  este compatibil cu cubul  $o_t = g_t$ , algoritmul greedy ar mai fi avut cuburi compatibile cu  $g_t$  din care să selecteze, deci ar fi furnizat o soluție cu mai multe elemente (un turn mai înalt).

Fie  $k \leq t$  cel mai mic indice astfel încât  $g_k \neq o_k$  (prima poziție pe care soluția greedy  $G$  și soluția optimă  $O$  diferă). La pasul la care algoritmul a ales cubul  $g_k$  și cubul  $o_k$  era neselectat și putea fi așezat peste cubul  $g_{k-1} = o_{k-1}$ . Deoarece cubul  $g_k$  a fost cel selectat, rezultă că  $l_{g_k} > l_{o_k}$  ( $g_k$  a fost selectat deoarece avea latură mai mare). Avem atunci  $l_{g_k} > l_{o_k} > l_{o_{k-1}} > \dots > l_{o_p}$ .

Dacă  $c_{g_k} = c_{o_k}$  putem înlocui în soluția optimă  $O$  cubul  $o_k$  cu  $g_k$  și obținem tot un turn corect  $O' = O - \{o_k\} \cup \{g_k\}$ , cu înălțime mai mare decât cel optim dat de  $O$  (deoarece  $l_{g_k} > l_{o_k}$ ), contradicție.

Dacă  $c_{g_k} \neq c_{o_k}$ , atunci putem insera în turnul dat de  $O$  cubul  $g_k$  între  $o_{k-1} = g_{k-1}$  și  $o_k$  și obținem tot un turn corect  $O' = O \cup \{g_k\}$ , cu înălțime mai mare decât cel optim dat de  $O$ , contradicție.



**Varianta 2** Demonstrăm prin inducție după  $n$  că algoritmul greedy construiește o soluție optimă.

Pentru  $n = 0, 1$  afirmația este evidentă. Fie  $n \geq 2$ . Presupunem că algoritmul greedy construiește o soluție optimă pentru orice mulțime de cel mult  $n - 1$  cuburi

Fie  $S$  o mulțime de  $n$  cuburi.

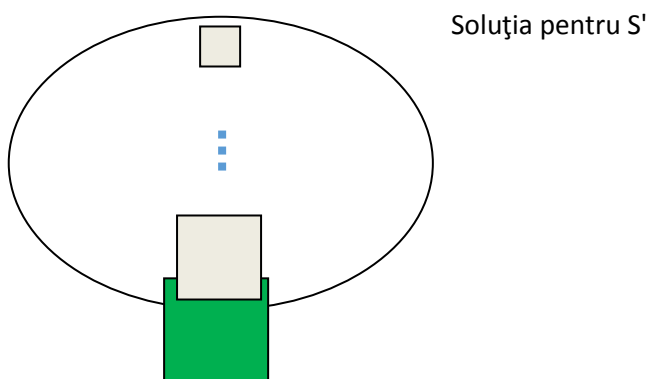
1. **Există o soluție optimă pentru  $S$  care conține cubul 1 (primul cub adăugat la soluție de algoritmul greedy).**

Într-adevăr, fie  $O = \{o_1, \dots, o_p\}$  o soluție optimă pentru  $S$  cu cuburile notate astfel încât  $o_1 < \dots < o_p$ , deci  $l_{o_1} > \dots > l_{o_p}$ . Dacă cubul 1 aparține lui  $O$ , atunci afirmația a) este adevărată. Presupunem că 1 nu aparține lui  $O$ .

Dacă  $c_1 = c_{o_1}$  putem înlocui în soluția optimă  $O$  cubul  $o_1$  cu 1 și obținem tot un turn corect  $O' = O - \{o_1\} \cup \{1\}$ , cu înălțime mai mare decât cel optim dat de  $O$  (deoarece  $l_1 > l_{o_1}$ ), contradicție.

Dacă  $c_1 \neq c_{o_1}$  putem insera în turnul dat de  $O$  cubul 1 sub cubul  $o_1$  și obținem tot un turn corect  $O' = O \cup \{1\}$ , cu înălțime mai mare decât cel optim dat de  $O$ , contradicție.

2. Fie  $r$  primul cub având culoare diferită de cubul 1 (următorul cub ales de algoritmul greedy) și  $S' = \{r, r+1, \dots, n\}$  (mulțimea cuburilor care pot aparține unui turn care are la bază cubul 1). Conform ipotezei de inducție soluția construită de algoritmul greedy pentru  $S'$ , anume  $G' = G - \{1\}$ , este soluție optimă pentru  $S'$ . Rezultă că  $G = G' \cup \{1\}$  este soluție optimă pentru  $S$  (altfel, conform punctului a) ar exista o soluție optimă  $O$  pentru  $S$  care conține cubul 1 cu înălțime mai mare decât  $G$ ; dar atunci  $O - \{1\}$  este soluție posibilă pentru  $S'$  cu înălțime mai mare decât  $G - \{1\} = G'$ , ceea ce contrazice optimalitatea lui  $G'$ )



**(tema varianta 2.1.b)** În cazul în care lungimile laturilor cuburilor nu erau diferite mai este valabilă metoda propusă?

Contraexemplu: date.in

```
4 2
10 1
10 2
9 1
8 2
```

- 5. (tema varianta 1.1) Acoperire** - Se dau un interval închis  $[a, b]$  și o mulțime de alte  $n$  intervale închise  $[a_i, b_i]$ ,  $1 \leq i \leq N$ . Scrieți un program care determină și afișează o submulțime de cardinal minim de intervale închise din mulțimea dată cu proprietatea că prin reuniunea acestora se obține un interval care include pe  $[a, b]$ . Dacă prin reuniunea tuturor intervalelor nu putem obține un interval care să includă intervalul  $[a, b]$ , se va afișa -1.

**Algoritm Greedy:** Se ordonează intervalele crescător după valorile  $a_i$ . Se caută primul interval  $[a_{g1}, b_{g1}]$  (în ordinea crescătoare după  $a_i$ ) pentru care avem:  $b_{g1} = \max_k \{b_k \mid a_k \leq a \leq b_k\}$ . Se adaugă la soluție intervalul  $[a_{g1}, b_{g1}]$ . Și se caută o acoperire optimă pentru  $[b_{g1}, b]$ . Se caută primul interval  $[a_{g2}, b_{g2}]$  pentru care avem:  $b_{g2} = \max_k \{b_k \mid a_k \leq b_{g1} \leq b_k\} \dots$

Algoritmul se oprește atunci când  $b_{gs} \geq b$  sau dacă nu a fost găsit un interval care să includă  $b_{gs}$ .

**Demonstrarea corectitudinii- Similar cu problema spectacolelor și problema 1 – cuburi.**

Fie  $o = \{[a_{o1}, b_{o1}], [a_{o2}, b_{o2}], \dots, [a_{op}, b_{op}]\}$ , o acoperire a lui  $[a, b] \subseteq \bigcup_{k=1}^p [a_{ok}, b_{ok}]$  optimă, ie  $p$  este numărul minim de intervale cu care poate fi acoperit  $[a, b]$  și intervalele sunt numerotate astfel încât  $a_{o1} \leq a \leq b_{o1}, b_{op} \leq b \leq b_{op}$  și  $a_{o1} \leq a_{o2} \leq \dots \leq a_{op}$ .

**Pasul 1:** Arătăm că  $o' = \{[a_{g1}, b_{g1}], [a_{o2}, b_{o2}], \dots, [a_{op}, b_{op}]\}$  este soluție optimă. Avem că  $b_{o1} \leq b_{g1}$ . Deci  $[a, b_{o1}] \subseteq [a, b_{g1}]$ . Rezultă că  $[a, b] \subseteq \bigcup_{k=2}^p [a_{ok}, b_{ok}] \cup [a_{g1}, b_{g1}]$ . Deci  $o'$  este o acoperire cu  $p$  intervale.

**Pasul 2:** Fie  $o = \{[a_{g1}, b_{g1}], [a_{o2}, b_{o2}], \dots, [a_{op}, b_{op}]\}$  o soluție optimă.

(\*)  $o' = \{[a_{o2}, b_{o2}], \dots, [a_{op}, b_{op}]\}$  este o acoperire optimă pentru  $[b_{g1}, b]$ .

Din (\*) va rezulta, conform pasului 1, că există  $o'' = \{[a_{g2}, b_{g2}], [a_{o2}, b_{o2}], \dots, [a_{op}, b_{op}]\}$  soluție optimă pentru  $[b_{g1}, b]$ . Inductiv, repetând pașii 1 și 2, rezultă că mulțimea intervalele alese de algoritmul greedy  $g = \{[a_{g1}, b_{g1}], [a_{g2}, b_{g2}], \dots, [a_{gp}, b_{gp}]\}$ , este soluție optimă.

Dem (\*) Arătăm că  $o' = \{[a_{o2}, b_{o2}], \dots, [a_{op}, b_{op}]\}$  este o acoperire optimă pentru  $[b_{g1}, b]$ .

Presupunem prin reducere la absurd că  $o' = \{[a_{o2}, b_{o2}], \dots, [a_{op}, b_{op}]\}$  nu este o acoperire optimă pentru  $[b_{g1}, b]$ . Fie  $u = \{[a_{u1}, b_{u1}], \dots, [a_{uq}, b_{uq}]\}$  o acoperire optimă a lui  $[b_{g1}, b]$  cu  $uq < p-1$ . Atunci  $u' = \{[a_{g1}, b_{g1}], [a_{u1}, b_{u1}], \dots, [a_{uq}, b_{uq}]\}$  este o acoperire a lui  $[a, b]$  cu  $uq+1 < p$  intervale. Contradicție cu presupunerea că  $o$  este soluție optimă.



6. **(tema varianta 1.2) Planificare cu minimizarea întârzierii maxime** – Se consideră o mulțime de  $n$  activități care trebuie planificate pentru a folosi o aceeași resursă. Această resursă poate fi folosită de o singură activitate la un moment dat. Pentru fiecare activitate  $i$  se cunosc durata  $l_i$  și termenul limită până la care se poate executa  $t_i$  (raportat la ora de început 0). Dorim să planificăm aceste activități astfel încât întârzierea fiecărei activități să fie cât mai mică. Mai exact, pentru o planificare a acestor activități astfel încât activitatea  $i$  este programată în intervalul de timp  $[s_i, f_i)$ , definim întârzierea activității  $i$  ca fiind durata cu care a depășit termenul limită:  $p_i = \max\{0, f_i - t_i\}$ .

Întârzierea planificării se definește ca fiind **maximul întârzierilor activităților**:

**Exemplu.** Pentru  $n = 3$  și  $l_1 = 1, t_1 = 3 / l_2 = 2, t_2 = 2 / l_3 = 3, t_3 = 3$  o soluție optimă se obține dacă planificăm activitățile în ordinea 2, 3, 1; astfel:

- activitatea 2 în intervalul  $[0, 2)$  – întârziere 0
- activitatea 3 în intervalul  $[2, 5)$  – întârziere  $5 - t_3 = 5 - 3 = 2$
- activitatea 1 în intervalul  $[5, 6)$  – întârziere  $6 - t_1 = 6 - 3 = 3$

Întârzierea planificării este  $\max\{0, 2, 3\} = 3$   $P = \max\{p_1, p_2, \dots, p_n\}$

**Algoritm Greedy:** Intervalele  $[s_i, f_i)$  se stabilesc parcurgând activitățile în ordinea crescătoare după  $t_i$ . Activitățile vor fi numerotate astfel încât  $t_1 \leq t_2 \leq \dots \leq t_n$ . Activitatea 1 este planificată în intervalul  $[0, l_1)$ . Fie  $[s_i, f_i)$ , intervalul în care este planificată activitatea  $i$ , atunci activitatea  $i+1$  va fi planificată în intervalul  $[f_i, f_i + l_{i+1})$ .

**Demonstrarea corectitudinii:**

Dacă activitățile au fost numerotate astfel încât  $t_1 \leq t_2 \leq \dots \leq t_n$  soluția Greedy corespunde permutării identitate  $id$  a mulțimii  $\{1, \dots, n\}$ . Fie  $f_1^g, \dots, f_n^g$  orele la care se vor termina activitățile după cum vor fi planificate de algoritmul Greedy.

Fie  $C_g = \max_{1 \leq i \leq n} p_i$  unde  $p_i = \max\{0, f_i^g - t_i\}$ , costul soluției Greedy.

Obs: O soluție optimă planifică activitățile în intervalul  $[0, \sum l_i]$ .



Astfel, o soluție optimă se obține în mod asemănător algoritmului Greedy, considerând o permutare  $\sigma^o$  a mulțimii  $\{1, \dots, n\}$  astfel:

- Activitatea  $\sigma^o(1)$  începând cu ora 0.
- Fie  $[s_{\sigma^o(i)}, f_{\sigma^o(i)})$  intervalul la care va fi planificată activitatea  $\sigma^o(i)$ ,  $2 \leq i \leq n - 1$ . Atunci activitatea  $\sigma^o(i + 1)$  va fi planificată în intervalul  $[f_{\sigma^o(i)}, f_{\sigma^o(i)} + l_{\sigma^o(i+1)})$ .

Fie  $\sigma$  o permutare optimă a activităților astfel încât  $\sigma \neq id$  cu număr minim de inversiuni (o permutare a activităților astfel încât ordinea să fie diferită de ordinea stabilită de algoritmul Greedy)

$$C_\sigma = \max_{1 \leq i \leq n} p_i \text{ unde } p_i = \max\{0, f_{\sigma(i)}^\sigma - t_{\sigma(i)}\}.$$

Deoarece  $\sigma \neq id$  rezultă că  $\exists u < v$  cu  $t_{\sigma(u)} > t_{\sigma(v)}$ .

Mai mult, dacă o astfel de inversiune există,  $\exists i$  cu  $t_{\sigma(i)} > t_{\sigma(i+1)}$

Dacă  $\forall j: u < j < v$  avem  $t_{\sigma(j)} \leq t_{\sigma(j+1)}$  luăm  $i = u$ .

În permutarea  $\sigma$ , interschimbăm două activități  $i$  și  $i+1$  pentru care  $t_{\sigma(i)} > t_{\sigma(i+1)}$ .

Obținem o permutare  $\tau$  astfel încât:

$$\tau(k) = \sigma(k) \quad \forall k \neq i, i+1$$

$$\tau(i) = \sigma(i+1)$$

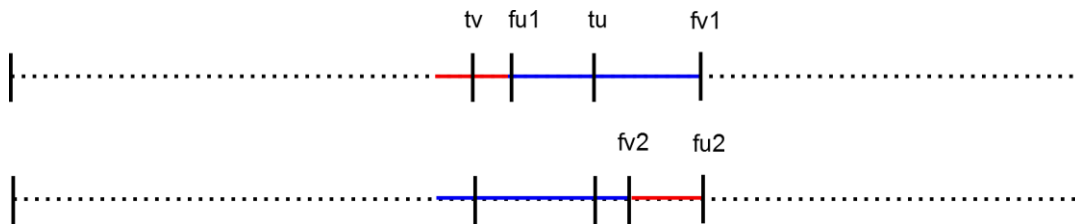
$$\tau(i+1) = \sigma(i)$$

Pentru simplificare vom nota

$$\sigma(i) = u \text{ și } \sigma(i+1) = v \text{ și } \tau(i) = v \text{ și } \tau(i+1) = u$$

$$f_u^\sigma = fu1 \quad f_v^\sigma = fv1$$

$$f_v^\tau = fv2 \quad f_u^\tau = fu2. \text{ Avem că } fu2 = fv1.$$



Singurele activități care vor avea întârzieri diferite în  $\sigma$  și  $\tau$  sunt  $u$  și  $v$ .

Calculăm întârzierea activității  $u$  în permutările  $\sigma$  respectiv  $\tau$

$$p_u^\sigma = \max\{0, f_u^\sigma - t_u\}$$

$$p_u^\tau = \max\{0, f_u^\tau - t_u\}$$

Dar  $f_u^\tau = f_v^\sigma$ . Deci  $p_u^\tau = \max\{0, f_v^\sigma - t_u\} \leq \max\{0, f_v^\sigma - t_v\} = p_v^\sigma$ , deoarece  $t_u > t_v$

Calculăm întârzierea activității  $v$  în permutările  $\sigma$  respectiv  $\tau$

$$p_v^\sigma = \max\{0, f_v^\sigma - t_v\}$$

$$p_v^\tau = \max\{0, f_v^\tau - t_v\}$$

Dar  $f_v^\tau < f_v^\sigma$ . Deci  $p_v^\tau = \max\{0, f_v^\tau - t_v\} \leq \max\{0, f_v^\sigma - t_v\} = p_v^\sigma$

Așadar

$$p_v^\tau \leq p_v^\sigma \leq C_\sigma = \max_{1 \leq i \leq n} p_i^\sigma$$

$$p_u^\tau \leq p_v^\sigma \leq C_\sigma = \max_{1 \leq i \leq n} p_i^\sigma$$

$$p_k^\tau = p_k^\sigma \leq C_\sigma \quad \forall k \neq i, i+1.$$

De unde rezultă că  $C_\tau \leq C_\sigma$ . Deci  $\tau$  este o permutare optimă cu număr mai mic de inversiuni decât  $\sigma$ , contradicție.

7. **(tema varianta 1.3.a)** Se dă un șir  $V$  de numere naturale. Să se descompună șirul de numere într-un număr minim de subșiruri descrescătoare (nestrict) și să se afișeze aceste subșiruri.

**Algoritm Greedy.**

La **pasul i** al algoritmului, sunt construite deja  $ns=k$  subșiruri descrescătoare cu elementele șirului  $v[1 \dots i-1]$ .

$u[i]$ ,  $1 \leq i \leq k$  reține ultimul element adăugat în al  $i$ -lea șir descrescător.

Avem că  $-\infty = u[0] < u[1] < \dots < u[k] < u[k+1] = \infty$ .

La **pasul i** se caută  $1 \leq j \leq k+1$  astfel încât să avem:  $u[j-1] < v[i] < u[j]$ . Se adaugă  $v[i]$  la șirul  $j$ ,  $u[j] = v[i]$ . Dacă  $j = k+1$  atunci se începe un nou subșir care va conține  $v[i]$  și  $ns = k+1$ .

**Demonstrarea corectitudinii.**

Fie  $s = (v_{i_1}, v_{i_2}, \dots, v_{i_{ls}})$   $i_1 \leq i_2 \leq \dots \leq i_{ls}$  și  $v_{i_1} < \dots < v_{i_{ls}}$  un subșir crescător al lui  $s$ . Orice descompunere a lui  $v$  în subșiruri descrescătoare conține cel puțin  $ls$  subșiruri. Dacă ar exista o descompunere cu  $ls-1$  subșiruri, două numere din  $s$  ar fi incluse în același subșir care nu ar fi ordonat descrescător. Astfel este adevărată următoarea observație:

**Obs:** Numărul optim de subșiruri  $no$  este egal cu lungimea maximă  $lmax$  a unui subșir crescător al șirului inițial.

Este suficient să demonstrăm că:

(\*) *Ultimul termen al șirului  $ns$ , aparține unui șir crescător al lui  $v$  care are lungimea  $ns$ .*

Astfel numărul de subșiruri obținut de algoritmul Greedy este egal cu lungimea unui subșir crescător al șirului  $v$ . Deci  $ns$  este optim:  $ns \leq lmax = no$ .

Demonstrăm (\*) prin inducție după  $ns$ .

Dacă  $ns = 1$  afirmația este adevărată.

Presupunem că dacă algoritmul împarte un șir  $w$  în  $ns-1$  subșiruri, ultimul termen al șirului  $ns-1$ , aparține unui șir crescător al lui  $w$  care are lungimea  $ns-1$ .

Fie  $v[ns]$  ultimul termen care se adaugă în șirul  $ns$ . Fie  $w$  șirul format cu elementele subșirurilor  $1 \dots ns-1$ . Conform presupunerii făcute:  $w$  are subșirul crescător:

$$s = (v_{i_1}, v_{i_2}, \dots, v_{i_{ns-2}}, u_{ns-1}).$$

Dar  $u[ns-1] < u[ns]$ . Altfel  $u[ns-1]$  ar fi fost adăugat la șirul  $ns-1$ . Deci există șirul crescător  $s' = (v_{i_1}, v_{i_2}, \dots, v_{i_{ns-2}}, u_{ns-1}, u_{ns})$  care are lungimea  $ns$ .

**(tema varianta 1.3.b) Demonstrarea corectitudinii:**

**Pasul 1** Primul șir ales de algoritmul b coincide cu primul șir format de algoritmul a.

**Pasul 2** Comparăm rezultatul algoritmilor a și b aplicați lui  $v'$  obținut din  $v$  dacă eliminăm primul șir descrescător.