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 ∪ { a_i })
                S = S ∪ { a_i }
```

EXEMPLE DE PROBLEME

1. (**Problema continua a rucsacului**) Fiind dat un rucsac care are capacitatea G (greutatea maxima ce poate fi incarcata in rucsac) si *n* obiecte, pentru fiecare obiect fiind cunoscute greutatea g_i si profitul total obtinut prin incarcarea obiectului in intregime c_i , sa se obtina profitul maxim obtinut prin incarcarea rucsacului. Un obiect poate fi incarcat partial.

   Solutie: Se ordoneaza obiectele in functie de profitul obtinut pe unitate c_i/g_i. Se vor incarca obiectele $1, 2, \dots, u-1$ in intregime iar din ultimul obiect *u<=n,* se va incarca greutatea

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

   u este cel mai mare indice pentru care este adevarata inegalitatea

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

   Obiectele *u+1, … n* nu vor fi incarcate.

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 cu numar maxim de elemente initiale $o_1, \dots o_{p-1}$ identice cu cele alese in solutia Greedy. Pentru orice solutie optima suma greutatilor incarcate este G.

$G = o_1 + \cdots + o_n$.

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*.

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

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

De unde deducem

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

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

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 deoarce avem ca:

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

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

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

$$= \sum_{i=p+1}^{n} (1-\alpha)o_i(\frac{c_p}{g_p} - \frac{c_i}{g_i}) \geq 0$$

Deci *o'* este o solutie optima care are mai multe elemente initiale identice cu elementele alese de solutia Greedy, contradictie cu alegerea lui o, optima cu numar maxim de elemente initiale comune cu solutia Greedy.

2. Se dau *n* activitati. Pentru fiecare activitate este cunoscut timpul de start s_i timpul de final f_i. Activitatile sunt impartite in grupuri. Pentru fiecare activitate este cunoscut grupul g_i din care face parte. Sa se planifice o submultime maxima de activitati astfel incat intervalele de desfasurare $[s_{a_i}, f_{a_i}]$ sa fie disjuncte doua cate doua si sa nu fie planificate doua activitati din acelasi grup: $g_{a_i} \neq g_{a_j}$ pentru orice $i \neq j$.

Problema este NP-hard.

Algoritm Greedy: Se parcurg in ordine activitatile dupa ora de sfarsit. Activitatea i se adauga la solutie daca nu se intersecteza cu celelalte activitati planificate (este suficient sa nu se intersecteze cu ultima planificata) si grupul din care face parte nu este inca reprezentat de o activitate deja planificata.

Aproximare a solutiei optime: Se demonstreaza ca numarul de activitati planificate de solutia Greedy este, in cel mai rau caz, de doua ori mai mic decat numarul de activitati care se pot planifica astfel incat sa fie respectate cerintele.

$$n_g \geq n_o/2$$

Demonstratie: Fie *O* o submultime de cardinal maxim, de activitati care pot fi planificate fara suprapuneri si astfel incat oricare doua sa fie din grupuri diferite.

Fie *G* submultimea de activitati planificate de algoritmul Greedy.

Pentru a demonstra ca $n_g \geq n_o/2$ alegem o functie $f: O \to G$ astfel incat sa nu existe 3 activitati $I_1, I_2, I_3 \in O$ si $J \in G$ a.i $f(I_1) = f(I_2) = f(I_3) = J$.

Fie $f: O \to G$ definita astfel

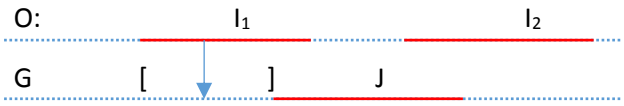
$$f(I) = \begin{cases} J, & (\alpha)\ J\ ales\ din\ grupul\ g_I\ (g_I = g_J)\ daca\ g_I\ este\ reprezentat\ sau \\ J, & (\beta)\ primul\ interval\ (in\ ordinea\ dupa\ f_i)\ care\ se\ intersecteaza\ cu\ I. \end{cases}$$

f este functie (este bine definita pentru orice I): Daca grupul activitatii *I* nu este reprezentat in *G*, exista o activitate *J* care sa se intersecteze cu *I*, altfel putem adauga activitatea *I* la solutia *G*. Nu pot exista doua activitati in G care sa reprezinte acelasi grup. Nu pot exista doua activitati in *G* care au aceasi ora de final.

Presupunem ca ar exista 3 activitati $I_1, I_2, I_3 \in O$ si $J \in G$ a.i $f(I_1) = f(I_2) = f(I_3) = J$.

Conform **principiului cutiei** (**Dirichlet**: daca *n* obiecte trebuie incluse in *m* cutii si *n>m*, cel putin 1 cutie va contine cel putin 2 obiecte). $\exists\ i_1, i_2\ a.i\ f(I_{i1}), f(I_{i2})$ sunt alese ambele in modul α sau ambele in modul β. Presupunem $i_1 = 1, i_2 = 2$.

1) Daca $f(I_1) = f(I_2) = J$ si intervalul *J* a fost ales in modul α, atunci solutia *O* nu este corecta, I_1, I_2 apartin aceluiasi grup.
2) Daca $f(I_1) = f(I_2) = J$ (presupunem $s_{I_1} < s_{I_2}$) si intervalul *J* a fost ales in modul β atunci intervalul $[s_{I_1}, s_J]$ nu se intersecteaza cu nicio activitate planificata in solutia Greedy.
Altfel J nu ar fi fost primul interval in solutia Greedy care se intersecteaza cu I_1. Dar in acest caz algoritmul Greedy ar fi ales I_1 iar daca in G in loc de *J* avem I_1, daca $f(I_2)$ este ales in modul β nu poate fi ales I_1 deoarece I_1 nu se intersecteaza cu I_2.



3. (tema varianta 2.1.b) **(Memorarea textelor pe banda)** n texte cu lungimile *L(1),...,L(n)* urmează a fi aşezate pe *p* benzi (*p* dat). Pentru a citi textul de pe poziţia *k* de pe o banda, trebuie citite textele de pe poziţiile *1,2,...,k* de pe aceea bandă. Fiecare text are aceeaşi frecvenţă de citire. Să se determine o modalitate de aşezare a textelor pe cele *p* benzi astfel încât timpul total de acces să fie minimizat (se obţine adunând timpul total de acces pentru fiecare din cele *p* benzi). Daca notam cu σ_{ki} al i-lea text asezat pe banda *k*, si cu n_k numarul total de texte asezate pe banda *k*, trebuie minimizat costul:

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

Algoritm Greedy: Se sorteaza textele crescator dupa lungime. Se aseaza primele *p* texte in ordine pe prima pozitie din cele *p* benzi. Urmatoarele texte, de la *p+1* la *2*p* vor fi asezate pe pozitia a doua din fiecare banda etc.

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

Demonstratie corectitudine Fie σ_{ki}^{o} o asezare optima a textelor pe cele p benzi.

In orice solutie optima toate cele *p* benzi contin cel putin un text.
Obs 1) Pe fiecare banda textele sunt asezate in ordine crescatoare dupa lungimea textelor.
Obs 2) Benzile pot fi renumerotate astfel incat $n_1 \geq n_2 \geq \cdots \geq n_p$. In plus avem ca:

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

Presupunem ca ar exista doua benzi $k_1, k_2\ cu\ k_1 < k_2$ si $n_{k_1} - n_{k_2} > 1$. Mutam primul text asezat pe banda k_1 astfel incat sa fie primul text de pe banda k_2. Din constul pentru citirea fiecarul text de pe banda k_1 se scade lungimea textului mutat $L(\sigma_{k_1 1}^{o})$. La costul citirii oricarui text de pe banda k_2 se aduna lungimea textului mutat $L(\sigma_{k_1 1}^{o})$. Se obtine asezarea $\sigma^{o\prime}$ care are costul:

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

Din (*) deducem ca exista o sulutie optima care are pe fiecare banda acelasi numar de texte ca solutia Greedy.

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

$$(**)\ L(\sigma_{11}^o) \le L(\sigma_{21}^o) \le \cdots \le L(\sigma_{p1}^o)$$

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 $0 \le n_{k_1} - n_{k_2} \le 1$.

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

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

Obs 4) Textele asezate pe prima pozitie din cele *p* benzi sunt primele p texte daca se considera textele ordonate crescator dupa lungime: $L(1) \le L(2) \le \cdots \le L(p)$. 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 $k \ne 1{,}2$ se procedeaza inductiv in acelasi mod.
Este evident din observatiile 1 si 3 ca $\sigma_{11}^o = 1$.
Presupunem ca $\sigma_{21}^o \ne 2$. Tot din observatia 3 rezulta ca $\exists\ j, 2 \le j \le n_1^o$ astel incat:

$$L(1) \le L(2) \le \ldots L(\sigma_{1j-1}^o) \le L(\sigma_{1j}^o) \le L(\sigma_{21}^o) \le L(\sigma_{1j+1}^o)\,.$$

Mutam textul 2 de pe banda 1 astfel incat sa fie primul text pe banda 2 ($\sigma_{21}^{o\prime} = 2$). 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. $\sigma_{1j}^{o\prime} = \sigma_{21}^o, \sigma_{1i}^{o\prime} = \sigma_{1i+1}^o$ pentru $2 \le i \le j-1$.

Se obtine diferenta intre costuri:

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

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 înălţime maximă de cuburi în care un cub nu poate fi aşezat pe un cub de aceeaşi culoare sau cu latură mai mică decât a sa.

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

Demonstratie Corectitudine Notăm cu l_i, i=1..n laturile cuburilor şi cu c_i, i=1..n culorile cuburilor.

Renumerotăm cuburile astfel încât $l_1 > 1_2 > \ldots > l_n$. Astfel, primul cub selectat de algoritmul greedy este cubul 1.

Fie $G = \{g_1=1, \ldots, g_t\}$, $t \leq p$ (cu $g_1 < \ldots < g_t$) soluţia Greedy.

Varianta 1 (similar problemei rucsacului):

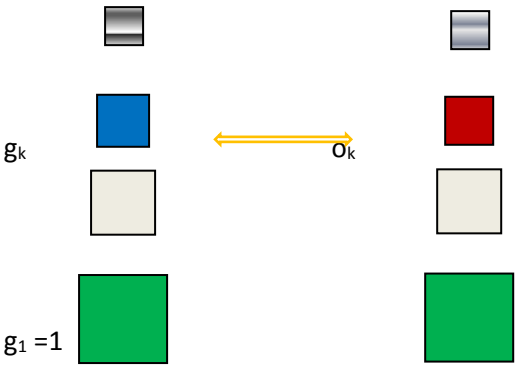
Considerăm o soluţie optimă $O = \{o_1, \ldots, o_p\}$ (cu cuburile notate astfel încât $o_1 < \ldots < o_p$ care are **un număr maxim de elemente iniţiale în comun cu soluţia Greedy** G. Presupunem $O \neq G$.

Atunci există un indice $k \leq t$ astfel încât $g_k \neq o_k$, altfel am avea $g_1 = o_1, \ldots, g_t = o_t$, adică $G \subseteq O$ şi $t < p$. 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 soluţie cu mai multe elemente (un turn mai înalt).

Fie $k \leq t$ cel mai mic indice astfel încât $g_k \neq o_k$ (prima poziţie pe care soluţia greedy G şi soluţia optimă O diferă). La pasul la care algoritmul a ales cubul g_k şi cubul o_k era neselectat şi putea fi aşezat peste cubul $g_{k-1} = o_{k-1}$. Deoarece cubul g_k a fost cel selectat, rezultă că $lg_k > lo_k$ (g_k a fost selectat deoarece avea latură mai mare). Avem atunci $lg_k > lo_k > lo_{k-1} > \ldots > lo_p$.

Dacă $c_{gk} = c_{ok}$ putem înlocui în soluţia optimă O cubul o_k cu g_k şi obţinem tot un turn corect $O' = O - \{o_k\} \cup \{g_k\}$, cu înălţime mai mare decât cel optim dat de O (deoarece $lg_k > lo_k$) , contradicţie.

Dacă $c_{gk} \neq c_{ok}$, atunci putem insera în turnul dat de O cubul g_k între $o_{k-1} = g_{k-1}$ şi o_k şi obţinem tot un turn corect $O' = O \cup \{g_k\}$, cu înălţime mai mare decât cel optim dat de O, contradicţie.



G O

Varianta 2 Demonstrăm prin inducţie după n că algoritmul greedy construieşte o soluţie optimă.

Pentru n = 0, 1 afirmaţia este evidentă. Fie $n \geq 2$. Presupunem că algoritmul greedy construieşte o soluţie optimă pentru orice mulţime de cel mult n - 1 cuburi

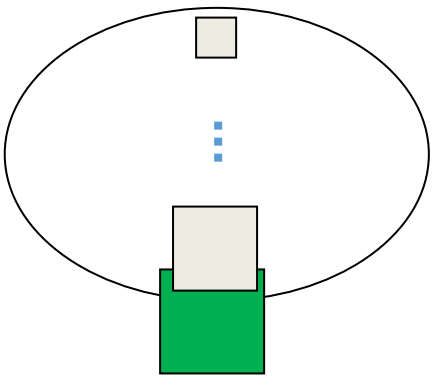
Fie S o mulţime de n cuburi.

1. **Există o soluţie optimă pentru S care conţine cubul 1 (primul cub adăugat la soluţie de algoritmul greedy).**
Într-adevăr, fie $O = \{o_1,\dots,o_p\}$ o soluţie optimă pentru S cu cuburile notate astfel încât $o_1 < \dots < o_p$, deci $lo_1 > \dots > lo_p$. Dacă cubul 1 aparţine lui O, atunci afirmaţia a) este adevărată. Presupunem că 1 nu aparţine lui O.

Dacă $c_1 = c_{o1}$ putem înlocui în soluţia optimă O cubul o_1 cu 1 şi obţinem tot un turn corect $O' = O - \{o_1\} \cup \{1\}$, cu înălţime mai mare decât cel optim dat de O (deoarece $l_1 > l_{o1}$) , contradicţie.

Dacă $c_1 \neq c_{o1}$ putem insera în turnul dat de O cubul 1 sub cubul o_1 şi obţinem tot un turn corect $O' = O \cup \{1\}$, cu înălţime mai mare decât cel optim dat de O, contradicţie.

2. Fie r primul cub având culoare diferită de cubul 1 (următorul cub ales de algoritmul greedy) şi $S' = \{r, r+1, \dots, n\}$ (mulţimea cuburilor care pot aparţine unui turnul care are la bază cubul 1). Conform ipotezei de inducţie soluţia construită de algoritmul greedy pentru S', anume $G' = G - \{1\}$, este soluţie optimă pentru S'. Rezultă că $G = G' \cup \{1\}$ este soluţie optimă pentru S (altfel, conform punctului a) ar exista o soluţie optimă O pentru S care conţine cubul 1 cu înălţime mai mare decât G; dar atunci $O - \{1\}$ este soluţie posibilă pentru S' cu înălţime mai mare decât $G - \{1\} = G'$, ceea ce contrazice optimalitatea lui G')



Soluţia pentru S'

(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] si o mulţime de alte n intervale închise $[a_i,b_i]$, $1 \le i \le N$. Scrieţi un program care determină şi afişează o submulţime de cardinal minim de intervale închise din mulţimea dată cu proprietatea că prin reuniunea acestora se obţine un interval care include pe [a,b]. Dacă prin reuniunea tuturor intervalelor nu putem obţine un interval care sa includă intervalul [a,b], se va afişa -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: $b_{g1} = \max_k\{b_k |\ a_k \le a \le b_k\}$. 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: $b_{g2} = \max_k\{b_k |\ a_k \le b_{g1} \le\ b_k\}$ ...

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

Demonstrarea corectitudinii- Similar cu problema spectacolelor şi problema 1 – cuburi.

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

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

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

Întârzierea planificării se defineşte ca fiind **maximul întârzierilor activităţilor**:

Exemplu. Pentru n = 3 şi $l_1 = 1$, $t_1 = 3$ / $l_2 = 2$, $t_2 = 2$ / $l_3 = 3$, $t_3 = 3$
o soluţie optimă se obţine dacă planificăm activităţile 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 $\max\{0, 2, 3\}=3$ $\boldsymbol{P = \max\{p_1, p_2, ..., p_n\}}$

Algoritm Greedy: Intervalele $[s_i, f_i)$ se stabilesc parcurgând activităţile în ordinea crescătoare după t_i . Activităţile vor fi numerotate astfel încât $t_1 \le t_2 \le \cdots \le t_n$. 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 $[f_i, f_i + f_{i+1})$.

Demonstrarea corectitudinii:

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

Fie $C_g = \max\limits_{1 \le i \le n} p_i$ unde $p_i = \max\{0, f_i^g - t_i\}$, costul soluţiei Greedy.

Obs: O soluţie optimă planifică activităţile în intervalul $[0, \sum_i l_i]$.

Astfel, o soluţie optima se obţine în mod asemănător algoritmului Greedy, considerând o permutare σ^o a mulţimii $\{1, \dots, n\}$ 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 $\sigma^o(i)$, $2 \le i \le n-1$. 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 σ 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)

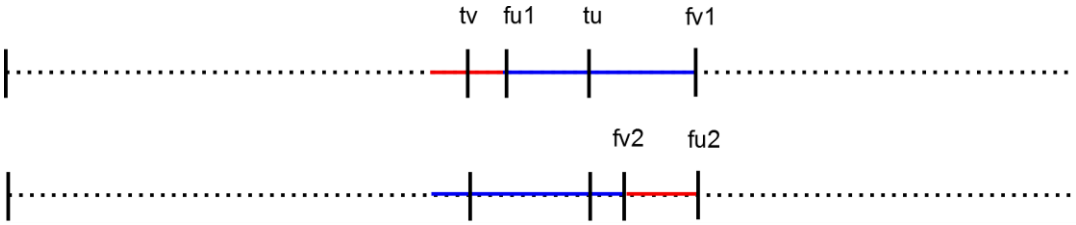
$C_\sigma = \max\limits_{1\le i\le n} p_i$ unde $p_i = \max\{0, f^\sigma_{\sigma(i)} - t_{\sigma(i)}\}$.

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 $t_{\sigma(j)} \le t_{\sigma(j+1)}$ luăm *i* = *u*.

În permutarea σ, interschimbăm două activități i și i+1 pentru care $t_{\sigma(i)} > t_{\sigma(i+1)}$.
Obținem o permutare τ astfel încât:

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

Pentru simplificare vom nota
$\sigma(i) = u$ și $\sigma(i+1) = v$ și $\tau(i) = v$ și $\tau(i+1) = u$
$f^\sigma_u = fu1 \ f^\sigma_v = fv1$
$f^\tau_v = fv2 \ f^\tau_u = fu2$. Avem că fu2 = fv1.



Singurele activități care vor avea întârzieri diferite în σ și τ sunt u și v.
Calculăm întârzierea activității u în permutările σ respectiv τ
$p^\sigma_u = \max\{0, f^\sigma_u - t_u\}$
$p^\tau_u = \max\{0, f^\tau_u - t_u\}$

Dar $f^\tau_u = f^\sigma_v$. Deci $p^\tau_u = \max\{0, f^\sigma_v - t_u\} \le \max\{0, f^\sigma_v - t_v\} = p^\sigma_v$, deoarece $t_u > t_v$

Calculăm întârzierea activității v în permutările σ respectiv τ
$p^\sigma_v = \max\{0, f^\sigma_v - t_v\}$
$p^\tau_v = \max\{0, f^\tau_v - t_v\}$

Dar $f^\tau_v < f^\sigma_v$. Deci $p^\tau_v = \max\{0, f^\sigma_v - t_v\} \le \max\{0, f^\sigma_v - t_v\} = p^\sigma_v$
Așadar
$p^\tau_v \le p^\sigma_v \le C_\sigma = \max\limits_{1\le i\le n} p^\sigma_i$

$p^\tau_u \le p^\sigma_v \le C_\sigma = \max\limits_{1\le i\le n} p^\sigma_i$

$p^\tau_k = p^\sigma_k \le C_\sigma \ \forall k \neq i, i+1$.

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

7. (tema varianta 1.3.a) Se dă un şir V de numere naturale. Să se descompună şirul de numere într-un număr minim de subşiruri descrescătoare (nestrict) şi să se afişeze aceste subşiruri.

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 $-\infty = u[0] < u[1] < \cdots < u[k] < u[k+1] = \infty$.
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 $s = (v_{i_1}, v_{i_2}, \dots, v_{i_{ls}})$ $i_1 \leq i_2 \dots \leq i_{ls}$ și $v_{i_1} < \cdots < v_{i_{ls}}$ 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 subşiruri no este egal cu lungimea maximă $lmax$ a unui subşir crescător al şirului iniţial.

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