TEHNICA DE PROGRAMARE "GREEDY"

1. Prezentare generală

Tehnica de programare Greedy este utilizată, de obicei, pentru rezolvarea problemelor de optimizare, adică a acelor probleme în care se cere determinarea unei submulţimi a unei mulţimi date pentru care se minimizează sau se maximizează valoarea unei funcţii obiectiv.

Forma generală a unei probleme de optimizare:

"*Fie A o mulţime nevidă şi $f: \mathcal{P}(A) \to \mathbb{R}$ o funcţie obiectiv asociată mulţimii A, unde prin $\mathcal{P}(A)$ am notat mulţimea tuturor submulţimilor mulţimii A. Să se determine o submulţime $S \subseteq A$ astfel încât valoarea funcţiei f să fie minimă/maximă pe S (i.e., pentru orice altă submulţime $T \subseteq A, T \neq S$, valoarea funcţiei obiectiv f va fi cel puţin /cel mult egală cu valoarea funcţiei obiectiv f pe submulţimea S).*"

Exemplu:
"*Fie A o mulţime nevidă de numere întregi. Să se determine o submulţime $S \subseteq A$ cu proprietatea că suma elementelor sale este maximă.*"

"*Fie $A \subseteq \mathbb{Z}, A \neq \emptyset$ şi $f: \mathcal{P}(A) \to \mathbb{R}$,*

$$f(S) = \sum_{x \in S} x.$$

Să se determine o submulţime $S \subseteq A$ astfel încât valoarea funcţiei f să fie maximă pe S, i.e. $\forall T \subseteq A, T \neq S \Rightarrow f(T) \leq f(S)$ sau, echivalent, $\forall T \subseteq A, T \neq S \Rightarrow \sum_{x \in T} x \leq \sum_{x \in S} x$."

A = {-5, 7, -1, 10, 3} => S = {7, 10, 3}

A = {-5, -7, -1, -10, -3} => S = {-1}

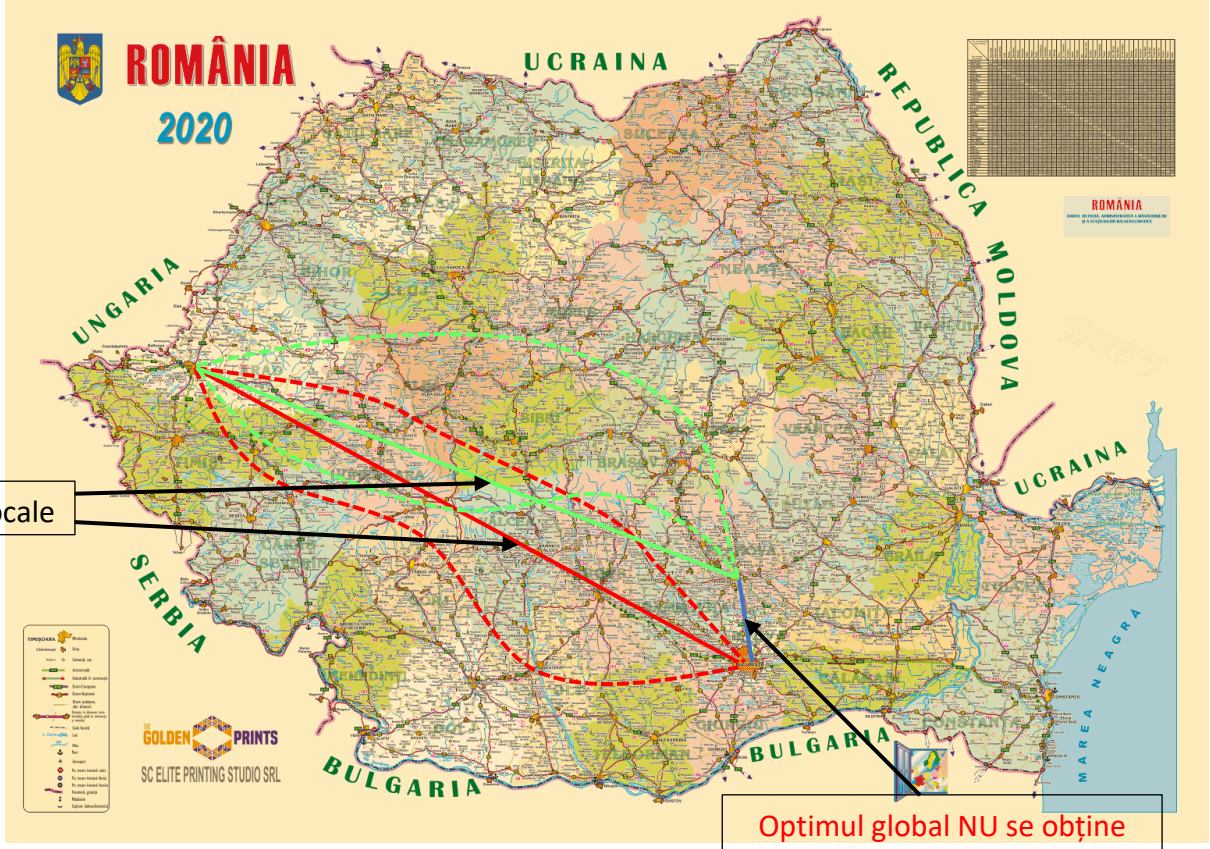
Evident, orice problemă de acest tip poate fi rezolvată prin *metoda forţei-brute*, dar soluţia va avea o complexitate exponenţială, respectiv $\mathcal{O}\left(2^{|A|}\right)$!

Tehnica de programare Greedy încearcă să rezolve problemele de optimizare adăugând în submulţimea *S*, la fiecare pas, cel mai bun element disponibil din mulţimea *A* din punct de vedere al optimizării funcţiei obiectiv.

Practic, metoda Greedy încearcă să găsească optimul global al funcţiei obiectiv combinând optimele sale locale.

Principiul de optim:

1. *Optimul global => optime locale* (forma directă) şi este întotdeauna adevărată
2. *Optime locale => optim global* (forma inversă) şi NU este întotdeauna adevărată



Revenind la problema determinării unei submulţimi S cu sumă maximă, observăm faptul că aceasta trebuie să conţină toate elementele pozitive din mulţimea A, deci criteriul de selecţie este ca elementul curent din A să fie pozitiv (demonstraţia optimalităţii este banală). Dacă mulţimea A nu conţine niciun număr pozitiv, care va fi soluţia problemei?

În anumite probleme, criteriul de selecţie poate fi aplicat mai eficient dacă se realizează o prelucrare iniţială a elementelor mulţimii A – de obicei, o sortare a lor.

Exemplu:
Fie A o mulţime nevidă formată din n numere întregi. Să se determine o submulţime $S \subseteq A$ având exact k elemente $(1 \leq k \leq n)$ *cu proprietatea că suma elementelor sale este maximă.*

A = { 1, -3, -2, ~~5~~, -7, 2, ~~3~~, -5} n = 8 şi k = 5
S = {5, 3, 2, 1, -2}

sorted(A) = {-7, -5, -3, -2, 1, 2, 3, 5}

Soluția:
- *fără sortare* (*implementare directă*): $\mathcal{O}(kn)$
- *cu sortare*: $\mathcal{O}(k + n\log_2 n) \approx \mathcal{O}(n\log_2 n)$

Forma generală a unui algoritm de tip Greedy:

```
prelucrarea inițială a elementelor mulțimii A (opțional)

S = []
for x in A
  if x verifică criteriul de selecție:
      S.append(x)

afișarea elementelor soluției S
```

Complexitățile specifice metodei Greedy:

- $\mathcal{O}(n)$ -> fără sortare și verificarea criteriului de selecție pentru un element x în $\mathcal{O}(1)$
- $\mathcal{O}(n\log_2 n)$ -> cu sortare în $\mathcal{O}(n\log_2 n)$ și verificarea criteriului de selecție pentru un element x în $\mathcal{O}(1)$
- $\mathcal{O}(n^2)$ sau $\mathcal{O}(n^2\log_2 n)$ -> cu sortare în $\mathcal{O}(n\log_2 n)$ și verificarea criteriului de selecție în $\mathcal{O}(n)$

Selecția unui element x din mulțimea A este statică, deci trebuie demonstrată corectitudinea criteriului de selecție!!!

Contraexemplu Greedy

Plata unei sume S folosind n tipuri de monede cu valorile $v_1, v_2,\ldots,v_n$

Dorim să plătim suma S folosind un număr minim de monede!

Idee de tip Greedy: folosim un număr maxim de monede cu valoare maximă la momentul respectiv

a) v = [8, 7, 5] $ și S = 23 $

S = 2*8$ + 7$ = 2*8$ + 1*7$ => Nr. monede = 3 => soluție optimă

b) v = [8, 7, 1] $ și S = 14 $

S = 1*8$ + 6$ = 1*8$ + 6*1$ => Nr. monede = 7 => soluție corectă, dar care nu este optimă (2*7$)

c) v = [8, 7, 5] $ și S = 14 $

S = 1*8$ + 6$ = 1*8$ + 1*5$ + 1$ (rest neplătibil) => nu există soluție!

2. Minimizarea timpului mediu de aşteptare

La un ghişeu, stau la coadă n persoane $p_1, p_2, \dots, p_n$ şi pentru fiecare persoană p_i se cunoaşte timpul său de servire t_i. Să se determine o modalitate de reaşezare a celor n persoane la coadă, astfel încât timpul mediu de aşteptare să fie minim.

Exemplu:

Persoana	**Timpul de servire (t_i)**	**Timp de aşteptare (a_i)**
p_1	7	7
p_2	6	7 + 6 = 13
p_3	3	13 + 3 = 16
p_4	10	16 + 10 = 26
p_5	6	26 + 6 = 32
p_6	3	32 + 3 = 35
Timpul mediu de aşteptare (TMA):		$\frac{7+13+16+26+32+35}{6} = \frac{129}{6} = 21.5$

Rearanjarea optimă (în ordinea crescătoare a timpilor de servire):

Persoana	**Timpul de servire (t_i)**	**Timp de aşteptare (a_i)**
p_3	3	3
p_6	3	3 + 3 = 6
p_2	6	6 + 6 = 12
p_5	6	12 + 6 = 18
p_1	7	18 + 7 = 25
p_4	10	25 + 10 = 35
Timpul mediu de aşteptare (TMA):		$\frac{3+6+12+18+25+35}{6} = \frac{99}{6} = 16.5$

minimizarea timpului mediu de aşteptare <=>
minimizarea timpului de aşteptare al fiecărei persoane <=>
minimizarea timpilor de aşteptare ai persoanelor aflate înaintea sa

Pentru a demonstra mai simplu corectitudinea algoritmului, mai întâi vom renumerota persoanele $p_1, p_2, \dots, p_i, \dots, p_j, \dots, p_n$ în ordinea crescătoare a timpilor de servire, astfel încât vom avea $t_1 \leq t_2 \leq \cdots \leq t_i \leq \cdots \leq t_j \leq \cdots \leq t_n$. De asemenea, vom presupune faptul că timpii individuali de servire $t_1, t_2, \dots, t_n$ nu sunt toți egali între ei (în acest caz, problema ar fi trivială), deci există $i < j$ astfel încât $t_i < t_j$. În continuare, presupunem faptul că această modalitate P_1 de aranjare a persoanelor la coadă (o permutare, de fapt) nu este optimă, deci există o altă modalitate optimă $P_2 \neq P_1$ de aranjare $p_1, p_2, \dots, p_j, \dots, p_i, \dots p_n$ diferită de cea inițială, în care $t_j > t_i$ (practic, am interschimbat persoanele p_i și p_j din varianta inițială, adică persoana p_j se află acum pe poziția i în coadă, iar persoana p_i se află acum pe poziția j, unde $i < j$).

În cazul primei modalități de aranjare P_1, timpul mediu de așteptare TMA_1 este egal cu:

$$TMA_1 = \frac{t_1 + (t_1 + t_2) + \cdots + (t_1 + \cdots + t_i) + \cdots + (t_1 + \cdots + t_j) + \cdots + (t_1 + \cdots + t_n)}{n} =$$
$$= \frac{nt_1 + (n-1)t_2 + \cdots + (n-i+1)t_i + \cdots + (n-j+1)t_j + \cdots + 2t_{n-1} + t_n}{n}$$

În cazul celei de-a doua modalități de aranjare P_2, timpul mediu de așteptare TMA_2 este egal cu:

$$TMA_2 = \frac{t_1 + (t_1 + t_2) + \cdots + (t_1 + \cdots + t_j) + \cdots + (t_1 + \cdots + t_i) + \cdots + (t_1 + \cdots + t_n)}{n} =$$
$$= \frac{nt_1 + (n-1)t_2 + \cdots + (n-i+1)t_j + \cdots + (n-j+1)t_i + \cdots + 2t_{n-1} + t_n}{n}$$

Comparăm acum TMA_1 cu TMA_2, calculând diferența dintre ele:

$$TMA_1 - TMA_2 = \frac{(n-i+1)t_i + (n-j+1)t_j - (n-i+1)t_j - (n-j+1)t_i}{n} =$$
$$= \frac{t_i(n-i+1-n+j-1) + t_j(n-j+1-n+i-1)}{n} =$$
$$= \frac{t_i(-i+j) + t_j(-j+i)}{n} = \frac{-t_i(i-j) + t_j(i-j)}{n} = \frac{(t_j - t_i)(i-j)}{n}$$

Deoarece $i < j$ și $t_j > t_i$, obținem faptul că $TMA_1 - TMA_2 = \frac{(t_j - t_i)(i-j)}{n} < 0$ (evident, $n \geq 1$), ceea ce implică $TMA_1 < TMA_2$. Contradicție!!!

Detalii de implementare + complexitate!

$\mathcal{O}(n \log_2 n)$

Forma generală a problemei:

"*Se consideră n activități cu duratele $t_1, t_2, \ldots, t_n$ care partajează o resursă comună. Știind faptul că activitățile trebuie efectuate sub excludere reciprocă* (i.e., *la un moment dat resursa comună poate fi alocată unei singure activități*), *să se determine o modalitate de planificare a activităților astfel încât timpul mediu de așteptare să fie minim.*".

3. Planificarea optimă a unor spectacole într-o singură sală

Considerăm n spectacole $S_1, S_2, \dots, S_n$ pentru care cunoaștem intervalele lor de desfășurare $[s_1, f_1), [s_2, f_2), \dots, [s_n, f_n)$, toate dintr-o singură zi. Având la dispoziție o singură sală, în care putem să planificăm un singur spectacol la un moment dat, să se determine numărul maxim de spectacole care pot fi planificate fără suprapuneri. Un spectacol S_j poate fi programat după spectacolul S_i dacă $s_j \geq f_i$.

De exemplu, să considerăm $n = 7$ spectacole având următoarele intervale de desfășurare:

$S_1: [10^{00}, 11^{20})$

$S_2: [09^{30}, 12^{10})$

$S_3: [08^{20}, 09^{50})$

$S_4: [11^{30}, 14^{00})$

$S_5: [12^{10}, 13^{10})$

$S_6: [14^{00}, 16^{00})$

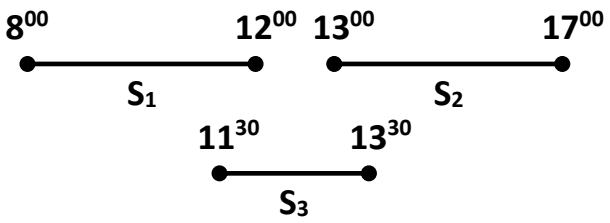
$S_7: [15^{00}, 15^{30})$

În acest caz, numărul maxim de spectacole care pot fi planificate este 4, iar o posibilă soluție este S_3, S_1, S_5 și S_7. Soluția nu este unică (S_3, S_1, S_5 și S_6)!!!

Criterii posibile de selecție:

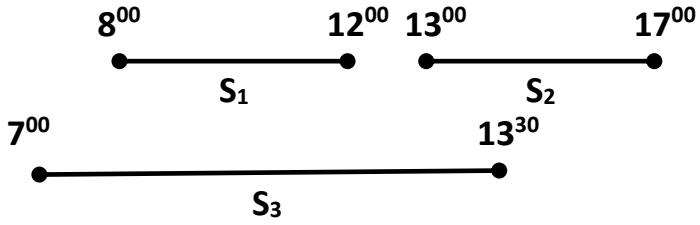
a) în ordinea crescătoare a duratelor
b) în ordinea crescătoare a orelor de început
c) în ordinea crescătoare a orelor de terminare

Criteriul a)



Programare: S3 (optim: S1, S2)

Criteriul b)



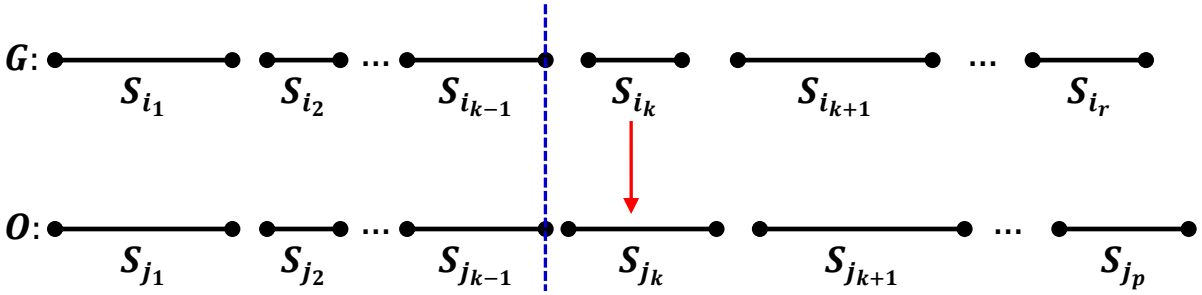
Programare: S3 (optim: S1, S2)

Criteriul c)

.....

Demonstrația optimalității - *exchange argument*:

Fie G soluția furnizată de algoritmul de tip Greedy și o soluție optimă O, diferită de G, obținută folosind orice alt algoritm:



Algoritmul Greedy:

- sortăm spectacolele în ordinea crescătoare a orelor de terminare;
- planificăm primul spectacol (problema are întotdeauna soluție!);
- pentru fiecare spectacol rămas, verificăm dacă începe după ultimul spectacol programat și, în caz afirmativ, îl planificăm și pe el.

Complexitate: $\mathcal{O}(n \log_2 n)$.