Laborator 1 - Greedy - Tehnici Avansate de programare

1) Acoperire - Pentru un interval principal $[a, b]$ și n intervale $[a_i, b_i]$, $i = \overline{1,n}$, să se determine numărul minim de intervale $[a_i, b_i]$ care îl acoperă pe $[a, b]$. Dacă nu există, se afișează -1.

Complexitate cerută: $O(n \log n)$

Idee: Sortăm în funcție de timpul de start. După sortare, traversăm intervalele care au timpul de start mai mici decât timpul de start al intervalului principal și îl alegem pe cel care se termină cel mai târziu. ($a_i <= a$ && $b_i > a$; Alegem intervalul cu b_i maxim).

Repetăm procedeul pentru intervalul $[b_i, b]$ până când acoperim intervalul complet.

Dacă nu găsim la un pas niciun interval cu $a_i \leq a$ și $b_i > a \Rightarrow$ $\Rightarrow$ nu putem acoperi zona $\Rightarrow$ returnăm -1.

Corectitudine: Presupunem prin reducere la absurd că există un algoritm optim $O \neq G$, unde G este algoritmul Greedy descris. Deoarece algoritmul optim nu este neapărat unic, o să îl alegem pe O ca fiind algoritmul optim care returnează un număr maxim de elemente în comun cu G.

Deci o să considerăm că O returnează p intervale, G returnează n intervale și $p < n$ (O este optim). O să notăm:

$O = \{[o_i, u_i] \mid i = \overline{1,p}\}$

$G = \{[a_j, b_j] \mid j = \overline{1,n}\}$

și $(\exists)\ k \geq 0$ a.î. $[o_l, u_l] = [a_l, b_l]$, $(\forall)\ l \leq k$

Deoarece $O \neq G \Rightarrow$ considerăm că la pasul $k+1$ algoritmii aleg diferit $[o_{k+1}, u_{k+1}] \neq [a_{k+1}, b_{k+1}]$

$[a_{k+1}, b_{k+1}]$ a fost ales de $G \Rightarrow$ în intervalul $[b_k, b]$:

1) $a_{k+1} \leq b_k$ (intervalul începe înainte sau odată cu cel precedent)

2) $u_{k+1} \leq b_{k+1}$ (intervalul ales de Greedy are timpul final maximal)

Avem următoarele cazuri:

I $u_{k+1} \leq b_k \Rightarrow [o_{k+1}, u_{k+1}]$ nu aduce nimic în plus intervalului total, deci e inutil

II $u_{k+1} > b_{k+1} \Rightarrow$ imposibil din (2)

III $u_{k+1} \in [b_k, b_{k+1}]$

Dacă $u_{k+1} = b_{k+1} \Rightarrow$ oricare din cele 2 poate fi aleasă de Greedy, fiind un caz irelevant pentru optimalitate.

Din III $\Rightarrow$ putem înlocui $[o_{k+1}, u_{k+1}]$ cu $[a_{k+1}, b_{k+1}]$

Deoarece $a_{k+1} \leq b_k$ și $u_{k+1} \in [b_k, b_{k+1}] \subseteq [a_{k+1}, b_{k+1}] \Rightarrow$

$\Rightarrow$ Obținem $O' = \{[o_i, u_i] / i = \overline{1,p}\}$ și $[o_i, u_i] = [a_i, b_i]$, $(\forall) i \leq k+1$.

Dar am presupus că O are număr maxim de elemente în comun cu $G \Rightarrow$

$\Rightarrow$ contradicție $\Rightarrow G$ soluție optimă.

Complexitate: Sortarea: $O(n \log n)$ } $\rightarrow O(n \log n)$

Parcurgerea: $O(n)$ ——

2) Planificarea cu întârziere minimă $\rightarrow$ Primim ca date de intrare n activități cu o durată l_i și cu timp de final t_i. Activitățile folosesc aceeași resursă. Dorim să aranjăm aceste activități astfel încât întârzierea fiecărei activități să fie minimă și să calculăm întârzierea totală

a) Detalii: Considerăm o soluție ca fiind dată de o permutare $\sigma \in S_n$.

O să considerăm că nu există intervale vide (fără activități), deoarece o activitate vidă de timp l ar adăuga l la timpul de finalizare al tuturor activităților din dreapta ei, crescând întârzierea totală.

De asemenea, vom considera funcția $f^{\sigma}_{\sigma(k)} = l_{\sigma(1)} + l_{\sigma(2)} + \dots + l_{\sigma(k)}$ care reprezintă timpul de finalizare al lui k, $(\forall) k \geq 1$. Observație: $f^{\sigma}_{\sigma(k)} = f^{\sigma}_{\sigma(k-1)} + l_{\sigma(k)}$

Notăm, de asemenea, $p_i^\sigma = \max\{0, f_i^\sigma - t_\sigma\}$, $(\forall) i = \overline{1,n}$, întârzierea activităților, și $p^\sigma = \max_{1 \le i \le n} p_i^\sigma$.

Idee: Planificăm activitățile în ordine crescătoare după termenul limită t_i.

De ce? Pentru că un termen limită „mare", care se poate termina târziu, urmat de termene limită „mici", va crește întârzierea tuturor intervalelor din dreapta sa, cauzând întârzieri raportate la lungime. Plasat în capăt, toate intervalele vor pierde l din întârziere, ceea ce este benefic chiar și atunci când cel „mare" întârzie.

Corectitudine: Sortăm activitățile în funcție de termenul limită $\Rightarrow$

$t_1 \le t_2 \le \dots \le t_n$ și permutarea asociată este e.

Intervalele sunt de forma $[l_1 + l_2 + \dots + l_{k-1},\ l_1 + l_2 + \dots + l_k] \Rightarrow$ activitatea K.

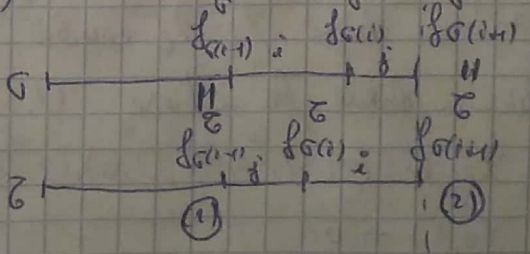
Presupunem prin reducere la absurd că e nu este optimă $\Rightarrow (\exists)\sigma$ permutare optimă cu număr minim de inversiuni. O să demonstrăm că σ nu e optimă prin găsirea unei inversiuni.

Permutarea e nu e optimă $\Rightarrow (\exists) i, j$ a.î. $t_{\sigma(i)} > t_{\sigma(j)}$ în soluția optimă. De asemenea, $(\exists) i$ a.î. $t_{\sigma(i)} > t_{\sigma(i+1)}$, pentru că orice șir care nu e crescător are două elemente consecutive inversate.

Considerăm τ a.î. 1) $\tau(k) = \sigma(k)$, $(\forall) k \ne i, i+1$

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

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



Atfel: $f^\sigma_{\sigma(i-1)} = f^\tau_{\sigma(i-1)}$ (1)

$f^\sigma_{\sigma(i+1)} = f^\tau_{\sigma(i+1)}$ (2)

$f^\sigma_{\sigma(i)} = l_{\sigma(1)} + l_{\sigma(2)} + \dots + l_{\sigma(i)}$

$f^\sigma_{\sigma(i+1)} = f^\sigma_{\sigma(i)} + l_{\sigma(i+1)}$

(3) $f^\tau_{\sigma(i)} = f^\tau_{\tau(i+1)} = f^\tau_{\sigma(i+1)}$

(4) $f^\tau_{\sigma(i+1)} = f^\tau_{\sigma(i-1)} + l_{\sigma(i+1)}$

Din ③ și ④ $\Rightarrow f^{\tau}_{G(i+1)} = f^{\tau}_{G(i)} - l_{G(i+1)} = f^{\tau}_{G(i)} - l_{G(i)} = f^{\tau}_{G(i+1)} - l_{G(i)}$

$< f^{\tau}_{G(i+1)}$

Comparăm întârzierile: $p^{\tau}_{G(i)} = f^{\tau}_{G(i)} - t_{G(i)} = f^{\tau}_{G(i+1)} - t_{G(i)}$

Dar $t_{G(i)} > t_{G(i+1)} \Rightarrow p^{\tau}_{G(i)} = f^{\tau}_{G(i+1)} - t_{G(i)} < f^{\tau}_{G(i+1)} - t_{G(i+1)} \leq p^{G}_{G(i+1)}$

$< p^{G} \Rightarrow p^{\tau} \leq p^{G} \Rightarrow G$ nu e optimal $\Rightarrow$ ↯ $\Rightarrow$ Sortarea optimă e.

Complexitate: Sortare $O(n \log n)$
Crearea intervalelor $O(n)$
Calculul întârzierilor totale $O(n)$
$\Rightarrow O(n \log n)$

b) Este corect un algoritm în care sortăm în funcție de durată? Justificați!

Exemplu: Considerăm datele de intrare: (1,10), (1,15), (3,3), (7,8)

Sortat după l_i: Varianta dată $\Rightarrow$

$\Rightarrow$ Intervalele: $[0,1)$ întârziere 0
$[1,2)$ întârziere 0
$[2,5)$ întârziere 2
$[5,12)$ întârziere 4
Întârziere totală: 6

Sortat după t_i: (3,3), (7,8), (1,10), (1,15).
$[0,3)$ întârziere 0
$[3,10)$ întârziere 2
$[10,11)$ întârziere 1
$[11,12)$ întârziere 0
Întârziere totală: 3

c) Este sortarea după $t_i - l_i$ corectă? Justificați

Considerăm datele de intrare: (1,3), (1,9), (2,10), (8,10)

Sortat $t_i - l_i$: (1,3), (8,10), (1,9), (2,10)
$[0,1] \to 0$
$[1,9] \to 0$
$[9,10] \to 1$
$[10,12] \to 2$
Total: 3

Sortat după t_i:
$[0,1] \to 0$
$[1,20] \to 0$
$[2,4] \to 0$
$[4,12] \to 2$
Total: 2

3) Numărul minim de șiruri descrescătoare dintr-un vector

Idee: Se parcurge șirul element cu element. Se extrage elementul curent, se caută în vectorul de stive elementul din top mai mare ca elementul curent și cel mai apropiat ca valoare. Dacă există, îl adăugăm în stiva respectivă, iar dacă nu, se creează o nouă stivă.

Complexitate: Căutare liniară în vectorul de stive: $O(\log n)$ $\Big|_{\Rightarrow} O(n \log n)$
Parcurgere vector: $O(n)$

Corectitudine: Observăm că orice vector primim, putem identifica în interiorul său un subșir crescător (dacă vectorul e descrescător, el va avea un singur element).
Fie $(\exists)\ c = (c_1, c_2, \ldots, c_K)$, $c_1 < c_2 < c_3 < \ldots < c_K$.

• Demonstrăm că fiecărui element din c îi corespunde un subșir, deci numărul de elemente al lui c este numărul minim de subșiruri care se pot obține.

1) Presupunem că $(\exists) c_i, c_j \in sub_i \Rightarrow$ ✗ (subșirurile sunt descrescătoare)

2) Presupunem că $(\exists) sub_i$ a.î. $c_j \notin sub_i$ $(\forall) j = \overline{1, K} \Rightarrow$ p.p. WLOG că sub_{i-1} conține un $c_p \in (c_i)_{i=\overline{1,K}}$. Dacă $c_j \notin sub_i \Rightarrow$ toate elementele lui sub_i sunt descrescătoare față de $sub_{i-1} \Rightarrow$ putem alătura (sub_{i-1}, sub_i) și să creăm un singur subșir $\Rightarrow$ algoritmul nu a returnat nr. minim de subșiruri $\Rightarrow$ ✗

(1), (2) $\Rightarrow$ În urma aplicării algoritmului, avem $c = (c_1, c_2, \ldots, c_K)$ și $(\forall) c_i$, $i = \overline{1, K}$ aparține unui subșir unic.

Vom demonstra că G construiește întotdeauna K subșiruri, de unde reiese optimalitatea.

Inducție: Considerăm ns = nr. subșiruri

$ns = 1 \rightarrow (\exists)$ un singur șir descrescător $\Rightarrow |c| = 1 \Rightarrow$ "A"

$ns = 2 \Rightarrow (\exists) c_1, c_2$ a.î. să se facă partiționarea în 2 șiruri $\Rightarrow |c| = 2 \Rightarrow$ "A"

$c_1 = (\forall) x \in [v[0], c_2)$

p.p. că pentru $ns = l$ avem l subșiruri descrescătoare și $c = (c_1, c_2, \ldots, c_l)$.

Construim $ns = l+1$

Considerăm w subșirul format din vârfurile stivelor / subșirurile construite anterior. Deci $w_l = (top[0], top[1], \ldots, top[l])$. Elementele sunt crescătoare, deci $w_l = c$

Considerăm $x = top[l+1] \Rightarrow$ elementul care nu poate fi adăugat la subșirurile din l

$\Rightarrow w_{l+1} = (top[0], top[1], \ldots, \widetilde{top[l]}, top[l+1] = x)$

$\downarrow$

e posibil să

x fi modificat

$! \widehat{top[l]} \leq top[l]$

Dacă toate elementele din w_l sunt c cu $|c| = l$ și s-a creat $l+1$ șir nou $\Rightarrow$

$\Rightarrow x > top[y], \forall y \in \overline{0,l} \Rightarrow c_{l+1} = c_l \cup \{x\} \Rightarrow$ pentru $w = l+1$ avem $|c_{l+1}| = l+1$ și $x \in v_{l+1}$.

Din inducție ne reiese că nr. de șiruri e mereu egal cu nr. de elemente crescătoare de secvență maximă $\Rightarrow$ G optim

b) Este corect următorul algoritm Greedy: Construim un subșir descrescător pornind de la $v[0]$ la care adăugăm toate elementele $m \leq v[0]$. Le afișăm, le eliminăm din vector și repetăm până când v.empty() == 1?

Da. Considerăm $c = (c_1, c_2, \ldots, c_k)$ subșir consecutiv de lungime maximă.

P.p. $v_1 \neq c_1$. Eliminăm v_1 și $\forall m \leq v_1$. Dacă c_1 a rămas $\Rightarrow c_1 > v_1 \Rightarrow$ construim $c' = (v_1, c_1, \ldots, c_k)$ șir crescător de lungime $k+1 \Rightarrow$ ↯ $\Rightarrow v_1 = c_1$

După ce am eliminat v_1 și $m \leq v_1 \Rightarrow \overline{c} = (c_2, c_3, \ldots, c_k)$ și $\overline{v}$.

Repetând procedeul, la pasul k vom fi eliminat toate elementele mai mari ca c_k și am generat $k-1$ subșiruri $\Rightarrow$ al k-lea subșir va fi vectorul final.

$\Rightarrow$ Algoritmul generează nr. minim de subșiruri $\Rightarrow$ Algoritmul e corect.