Maxq (Daniel Păsăilă)

Problema se poate rezolva în mai multe moduri. Prima posibilitate ar fi folosirea unui arbore de intervale. Astfel, în fiecare nod al arborelui se ţin următoarele valori:

- A[nod] – valoarea subsecvenţei de sumă maximă situată la începutul intervalului curent
- B[nod] – valoarea subsecvenţei de sumă maximă situată la sfârşitul intervalului curent
- C[nod] – valoarea subsecvenţei de sumă maximă situată oriunde în intervalul curent
- D[nod] – suma elementelor din intervalul current

Astfel, fiecare update se poate efectua în O(log N) (se apelează procedura pentru fiii nodului curent, apoi se calculează valorile pentru intervalul curent). Query-ul se efectuează parcurgând recursiv intervalele consecutive din arbore ce sunt incluse in intervalul curent (complexitate O(log N)). Soluţia aceasta obtine 100p.

O altă modalitate de rezolvare a problemei ar fi împărţirea şirului de la intrare în sqrt(N) secvenţe. Pentru fiecare secvenţă trebuie ţinute aceleaşi valori ca şi în cazul arborelui de intervale, iar query-ul se efectuează asemănator. Soluţia poate lua de la 60p la 100p, în funcţie de implementare.

Tric (Ştefan Ciobâcă)

Problema cere găsirea tuturor ciclurilor de grad 3 din graful dat la intrare. Graful acesta este cordal (fiecare ciclu de lungime 4 trebuie să aibă cel puţin o coardă), fapt care ne permite să numărăm triunghiurile în timp O(n + m).

Se realizează o ordonare a nodurile v_1, v_2, … v_n, cu proprietatea că v_i, împreună cu toţii vecinii săi care se găsesc la dreapta lui în ordonare, formează o clică. Deoarece graful este cordal, această ordonare se poate realiza întotdeauna. O dată stabilită această ordine, pentru orice i, putem număra triunghiurile formate de nodul v_i împreună cu nodurile v_{i+1}, v_{i+2}, …, v_n în timp O(1) (dacă t e numărul de vecini ai lui v_i aflaţi la dreapta acestuia, sunt t * (t-1) / 2 astfel de triunghiuri).

Pentru a găsi ordonarea, există doi algoritmi cunoscuţi: LexBFS şi MCS. Cel mai simplu este MCS şi funcţionează astfel:

Alegem nodurile din ordonare pe rând, de la v_n la v_1.

- asociem fiecărui nod numărul de vecini aflaţi la stânga sa, iniţializând aceste valori cu 0
- presupunând că am ales v_{i+1}, v_{i+2}, …, v_n, alegem v_i ca fiind nodul cu valoarea asociată cea mai mare
  - incrementăm valoarea asociată tuturor vecinilor lui v_i

Algoritmul se poate implementa în O(n + m), dacă se ţin toate nodurile care au aceeaşi valoare asociată într-un bucket.

Se poate demonstra că acest algorithm găseşte o ordonare cu proprietatea de mai sus (vezi Tarjan şi Yannakakis, "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs").

Log (Stelian Ciurea)

Soluţia 1 (Stelian Ciurea)

Pentru 70 puncte, problema se poate rezolva în complexitate $O(n^3)$ folosind un algoritm asemănător cu cel pentru înmulţirea optimală a unui şir de matrici. Astfel se calculează o matrice în care fiecare element o[i,j,0] reţine timpul minim necesar pentru a calcula expresia formata din logaritmii şi a o aduce la forma unui logaritm, iar o[i,j,1] reţine timpul minim necesar pentru a calcula expresia formată din logaritmii si a o aduce la forma unui număr real.

Soluţia se găseşte în o[1][n][1].

Soluţia 2 (Tiberiu Florea, Daniel Păsăilă)

Pentru 100 puncte era necesară o soluţie in $O(n^2)$. Pentru asta trebuie să folosim un tablou V, unde V[i] reprezintă costul minim pentru a calcula primii i logaritmi. V[i] se calculează usor în timp liniar, considerând toate secvenţele posibile din care ar putea face parte logaritmul i (înainte de a fi transformat într-un real). Relaţiile de recurenţă se deduc uşor.