Descrierea soluțiilor

treasure

propunător, prof. Marinela Dochia

Se calculează suma Gauss pentru a afla punctajul maxim al unui set de `m` comori, apoi, pentru fiecare concurent, se calculează suma celor `k` valori ale comorilor pe care nu le-a găsit. Punctajul obținut de concurent este dat de diferența dintre cele două sume. Pentru a afla punctajul maxim, se caută dacă s-a gasit un punctaj mai mare la concurentul curent.

ifelse

propunător, prof. Sofia Vițelaru

Problema poate fi reformulată astfel: fiind dat un sir de paranteze închise și deschise să se determine care este numărul minim de modificări astfel încât să se obțină o parantezare corectă.
Cuvântul `if` corespunde unei paranteze deschise iar cuvântul `else` unei paranteze închise.
Dacă numărul de cuvinte este impar vom afișa `-1` (nu putem avea o asociere corectă).
Se parcurge șirul și dacă se intalneste cuvântul `if` creștem o variabilă `nr` cu o unitate iar dacă întâlnim cuvântul `else` scade variabila `nr` cu o unitate. Dacă `nr` devine `-1` (avem `else` care nu este ascociat niciunui `if`) înseamnă că trebuie să producem o modificare și creștem cu o unitate numărul de operații, iar variabila `nr` devine `1`.
La final la numărul de operații trebuie adăugat `nr/2` (numarul de `if` -uri ce nu au asociat `else`).

arraymode

propunători, stud. Emanuel Dicu, stud. Robert Lincă

Soluție `N * log^2(N)`

Mai întâi o să precalculam pentru fiecare element din șir (să îl numim `arr`) un vector `nxt` cu următoare semnificație:

`nxt[i]` = prima poziție `j` din dreapta poziției `i` (`j>i`) cu proprietatea că `arr[i]==arr[j]` sau `n+1` în cazul în cate nu mai există vreo valoare egală cu `arr[i]` în dreapta sa

Pentru calcularea acestui vector ne vom folosi de mai multe seturi din STL, astfel că pentru fiecare valoare va exista un set cu pozițiile aparitiilor acestei valori în șir. Astfel că atunci când vom calcula `nxt[i]` tot ce va trebui este sa găsim în setul pentru valoarea `arr[i]` primul element din set mai mare ca `i` sau vom seta `n+1` dacă nu găsim un astfel de element.

După ce am făcut această precalculare ne vom folosi de un arbore de intervale pentru a putea updata diversele modificări de apar (operațiile de tip `1`) asupra șirului și a putea răspunde rapid la operațiile de tip `2`.

Sa analizăm mai întâi cum putem răspunde la o operație de tipul `2`.

Putem observa că dacă ar fi să luăm toate cele n perechi din șir de forma (`i`, `nxt[i]`) ele formează mai multe intervale ce se pot suprapune peste pozițiile `1` - `n` din șir, iar când noi vrem sa găsim o subsecvență care pornește la poziția `pos` și să presupunem că se termină la poziția `P` având doar elemente distincte este necesar ca:

pentru orice poziție `i` din intervalul `[pos, P]`, `nxt[i] > P`

Cu alte cuvinte, pentru orice valoare din intervalul `[pos, P]`, următoarea sa apariție în șir este după poziția `P`. Astfel putem să reținem într-un arbore de intervale în fiecare nod următoarele `2` informații:

- suma pe intervalul respectiv
- valoarea minimă `nxt[i]` pentru pozițiile din intervalul respectiv (este suficient ca doar `nxt[i]` minim sa fie mai mare decât capătul din dreapta (`P`) al subsecventei soluție ca să se respecte condiția discutată anterior)

Fiind un arbore de intervale putem afla în timp logaritmic suma și `nxt`-ul minim dintr-un interval, precum și să updatăm valoarea vreunui element din arr sau `nxt`.

Acum singura problemă este modul în care selectăm capătul din dreapta (`P`) al secvenței soluție. Observăm că putem căuta binar capătul din dreapta `P` al subsecventei soluție `[pos, P]`. Odata fixat capătul, putem verifica în timp logaritmic folosind un query pe arborele de intervale dacă intervalul `[pos, P]` este valid. Daca este valid, putem creste `P`-ul, altfel trebuie sa îl scădem. Complexitatea unui query este `log^2(N)`.

Pentru o operatie de update `arr[pos] = v`, presupunem că înainte `arr[pos]` avea valoarea `w`. Trebuie să păstrăm valide structurile menținute până acum astfel:

- scoatem poziția `pos` din setul valorilor `w` și introducem pozitia `pos` în setul valorilor `v`
- `nxt[pos]` se modifică în prima poziție `pos2 > pos` cu proprietatea că `v = arr[pos2]`. Ne putem folosi de seturile cu valori ca să identificăm această pozitie
- similar, se vor modifica `pos3` = cea mai mare pozitie mai mică decât pos cu proprietatea că `arr[pos3] = v`, respectiv `pos4` = cea mai mare poziție mai mică decât pos cu proprietatea că `arr[pos4] = w`.
- `arr[pos]` devine `v`.
- pentru fiecare valoare din vectorii `nxt` sau `arr` care se modifică, trebuie să efectuăm o operație de update în arborele de intervale (în total maxim `3` poziții se pot modifica: `pos`, `pos3`, `pos4`).

inter

propunător, prof. Marius Nicoli

Descriem mai jos soluția optimă.

Considerăm toate cele `N*4` puncte laticeale de pe conturul pătratului dat. Formăm segmente cu fiecare pereche de două astfel de puncte. Dintre acestea considerăm doar segmentele nesterse

și care nu se suprapun cu laturile. Pentru un astfel de segment determinăm toate punctele laticeale din interiorul pătratului, care se află pe acest segment și incrementăm într-o matrice de frecvență elementul corespunzător coordonatele acelui punct.
Observăm că dacă reușim să calculăm această matrice, pentru fiecare element `x = a[i][j]` al său, adunăm la soluție `x*(x-1)/2`. Altfel spus, dacă `x` segmente trec prin punctul de coordonate `i,j`, avem `combinări(x,2)` perechi de segmente care se intersectează acolo.
Să vedem acum cum determinăm punctele laticeale din interiorul pătratului, care se află pe un segment dat. Considerăm `DX` = modulul diferenței coordonatelor `x` ale celor două extremități ale segmentului și `DY` valoarea calculată similar pentru coordonatele `y`. Este suficient să calculăm `D = cmmdc(DX, DY)` si apoi valorile `pasx = DX/D` și `pasy = DY/D`. Observăm că punctele laticeale dorite se obțin pornind dintr-un capăt al segmentului și deplasându-ne către celălalt pășind de fiecare deodată cu `pasx` pe axa `x` si `pasy` pe axa `y`.

O soluție mai lentă este să fixăm, de asemenea, toate segmentele posibile și pentru fiecare să parcurgem toate punctele laticeale din interiorul pătratului verificând pentru fiecare dacă se găsește pe segmentul fixat. Această soluție nu se încadrează în timp pe toate testele.

Cele două abordări de mai sus fac apel mai puțin la elemente de geometrie.

O altă abordare care obține punctaj parțial este să stabilim toate perechile de două segmente și să determinăm pentru fiecare eventualul punct de intersecție, folosind noțiuni de geometrie.

Din descrierea subtaskurilor observăm că se pot obține puncte pe diverse abordări particulare.

Echipa care a pregătit acest set este următoarea:

stud. ***Alexandra Nicola***, Universitatea Delft, Olanda
stud. ***Alexandra Udrițoiu***, Facultatea de Matematică Informatică, București
stud. ***Emanuel Dicu***, Universitatea Politehnica, București
stud. ***Radu Nicolae***, Universitatea Delft, Olanda
stud. ***Robert Lincă***, Universitatea Politehnica, București
stud. ***Ruxandra Nanu***, Universitatea Oxfors, UK
stud. ***Tudor Măgirescu***, Universitatea Delft, Olanda
prof. ***Marinela Dochia***, Colegiul Național "Frații Buzești", Craiova
prof. ***Mihaela Grindeanu***, Colegiul Național "Frații Buzești", Craiova
prof. ***Sofia Vițelaru***, Colegiul Național "Frații Buzești", Craiova
prof. ***Marius Nicoli***, Colegiul Național "Frații Buzești", Syncro Soft, Craiova