Ediția a XXIV-a

INFO-OLTENIA

https://kilonova.ro/

http://www.greceanu.ro/concursuri/InfoOltenia2024/index.html

Descriere soluții

Comisie:

- Conf. univ. dr. Doru Anastasiu Popescu (Politehnica București)
- Vlad Coneschi (Universitatea din Oxford)
- Ciprian Stanciu (TU Delft)
- Robert Vădăstreanu (TU Delft)

Proba individuală

Clasele V-VI

burger (Vlad)

Soluția (100p)

Cerința 1 (20 p)

Prima cerință se rezolvă prin adunarea valorilor pentru cele 3 preparate de burgeri.

Cerința 2 (80 p)

Pentru cea de-a doua cerință, trebuie să facem următoarea observație și anume faptul că mulțimile care se pot forma cu numărul de burgeri din fiecare tip sunt independente și astfel putem folosi o soluție care încearcă pe rând toate variantele.

Astfel, sortăm întâi cele 3 valori crescător, iar apoi verificăm, începând de la mulțimile cu un tip de burger, apoi cu două tipuri de burger, iar la final mulțimea maximă cu toate cele 3 tipuri și numărăm câte am reușit să includem.

chipsuri (Ciprian)

Dacă avem două pungi cu A și B chipsuri (A>=B), avem următoarele 3 situații (chiar cele din exemplul problemei):

- B divide A => câștigi direct jocul
- B < A < 2*B => ai o singură variantă, iei B chipsuri din punga cu A chipsuri
- A > 2 * B => mereu vei câștiga; în funcție de rezultatul situației (A%B, B):
    - dacă este situație pierzătoare, atunci poți lua chipsuri pentru a forța colegul să ajungă în această situație (A%B, B)
    - dacă este situație câștigătoare, atunci poți lua chipsuri astfel încât colegul să ajungă în situația (A%B+B, B), din care colegul de bancă este forțat să îți "ofere" situația (A%B, B) câștigătoare

Clasele VII-VIII

casa de discuri (Robert)

Tratam fiecare trimestru independent:

Ideea acestei probleme se bazeaza pe observatia ca putem rezolva orice trimestru in maxim 2 saptamani.
Practic avem 3 cazuri:
1) Cand avem toate elementele din A egale cu X afisam 0
2) Cand avem un element X in A, atunci vom afisa 1: Sa zicem ca aceste element este A[w]. In saptamana 0 noi deja detinem drepturile de autor ale acestei melodii (A[w] == X). In saptamana 1 modificam pentru toate celelalte melodii ratingul ca sa fie egal cu X si toate diferentele le adaugam la A[w].
3) Cand nu avem niciun element X in a vom afisa 2: In prima saptamana trebuie sa cream un X. (Sa zicem ca il facem pe A[1] = x si diferenta o adaugam la A[2]). In saptamana 2 aplicam aceeasi procedura ca la cazul 2).

sorti (Ciprian)

Pentru această problemă este obligatoriu să avem grijă la memoria folosită.

De exemplu, o soluție care reține pentru fiecare comandă, câte caractere au fost folosite (un vector de frecvență cu 26 de poziții) nu va lua punctajul maxim.

O alternativă ar fi să reținem caracterele folosite (ordonate într-o oarecare ordine, pentru a putea comasa ușor cuvintele ce au același multiset de caractere), deoarece un cuvânt are maxim 12 caractere.

Clasa a IX-a

scări (Robert)

Tratam fiecare query independent. Vom folosi o abordare greedy. Pentru ca trebuie sa construim cat mai multe scari de nivele diferite cu K celule, vom incerca sa construim cele mai mici scari frumoase. Incepem cu scara 1 si scadem din K 1. Continuam cu scara de nivel 3 si scadem 6 din K. Mai departe urmeaza scara de nivel 7 si scadem 28 .... si asa mai departe pana cand nu mai avem buget sa construim o noua scara frumoasa.

Pentru a afla scarile frumoase, observam un pattern:
Prima scara frumoasa costa 1
A doua scara frumoasa (lvl 3) costa 1 * 2 + 2 * 2 = 6
A treia scara frumoasa (lvl 7) costa 6 * 2 + 4 * 4 = 28
A patra scara frumoasa costa: 2 * 28 + 8 * 8
....

ramagana (Vlad)

Soluția (100p)

Cerința 1 (30 p)

Pentru a verifica dacă cele două propoziții sunt anagrame, se poate reține câte un vector de frecvență pe litere pentru ambele șiruri, iar apoi se compară valorile la fiecare literă, verificând să fie egale.

Există și alte soluții la această cerință, de exemplu sortarea crescătoare a celor două șiruri și compararea rezultatelor.

Complexitate timp: O(lungime șir).

Cerința 2 (70 p)

Pentru verificarea anagramelor pe cuvinte, trebuie întâi împărțite cele două propoziții în cuvinte și reținute acestea în câte un șir.

Odată create aceste șiruri, putem sorta crescător cuvintele pentru a avea literele din ele în ordine, iar apoi sortăm și șirurile cu totul în ordine crescătoare punând astfel cuvintele în ordine.

Dacă cele două propoziții sunt anagrame pe cuvinte, cele două șiruri sortate vor coincide.

Complexitate timp: O(lungime_șir * $\log_{\text{lungime\_șir}}$)

Clasa a X-a

antrenament (Robert)

Pentru inceput, vom rasturna vectorii A si B astfel incat sa pastram ordinea si B sa fie in ordine descrescatoare.

Gasim primul indice x astfel incat A[x] = B[1]. Daca nu exista un astfel de index sau daca min{A[1], A[2] ... A[x]} < B[1], atunci raspunsul este 0.

Mai departe vom gasi primul indice y > x pt care A[y] = B[2]. Daca nu exista un astfel de index raspunsul este 0.
Cautam urmatorul indice mid > x astfel incat A[mid] < B[1] (nu are cum sa fie acest indice dupa y). Primul subsir incepe pe pozitia 1 si se poate termina in oricare din pozitiile x, x + 1... mid - 1. Practic, avem mid - 1 variante de a sparge antrenamentele 1 si 2.

Exact la fel se procedeaza si pentru impartirea antrenmentelor 2 si 3; 3 si 4 .....
Intr-un final, raspunsul final este dat de produsul variantelor de a sparge antrenamentele.

concurs (Ciprian)

Răspunsul este dat de numărul ordonat Bell - numărul de moduri în care N competitori se pot clasa într-o competiție, cu posibilitatea de a avea egalitate.

Pentru a ajunge la formulă, pentru N competitori, ne putem gândi că primii i competitori sunt la egalitate:
- numărul de moduri în care putem alege cei i competitori din totalul de N sunt combinări de N luate câte i;
- restul de N-i competitori sunt calculați recursiv, folosit același raționament.

Cazul de bază este N=0, în care există exact o ordonare (pentru 0 competitori).

Clasele XI-XII

centrat (Ciprian)

Ignorând elementul din mijloc, diferența dintre sumele din stânga și cea din dreapta unei poziții este:
D(i) = V(1) + ... + V(i-1) - V(i+1) - V(i+2) - ... - V(n)

Atunci comparând două diferențe consecutive vom avea:
D(i+1) - D(i) = V(i) + V(i+1) clar > 0 (chiar >=2), deci diferențele pot doar să crească.

Șirul cu |D(i)| este o funcție convexă, deci am putea găsi soluția cu o căutare binară.

Dacă includem și elementul din mijloc 'e', diferența ar fi:
B(i) = |D(i)| - (e mod 2)

Deoarece D(i+1) - D(i) >= 2 (> e mod 2), șirul de |B(i)| este tot convex ca și |D(i)|, deci căutarea binară tot este o soluție valabilă.

Putem răspunde la fiecare query rapid folosind un arbore Fenwick pentru a ține și actualiza sumele parțiale, combinat cu o căutare binară pentru a găsi poziția centrală.

ucf (Vlad)

Soluția 1 (20p)

Pentru 20 de puncte, se putea efectua o soluție de complexitate $O(n^2)$ sau $O(n^2 * \log_N)$ care selecta brut culorile folosite.

Soluția 2 (100p)

Pentru a obține punctaj maxim, vom folosi o soluție bazată pe programare dinamică.

dp_i = lungimea celei mai lungi secvențe descrescătoare care se termină pe poziția *i*.

Pentru a putea recalcula eficient acest șir, vom utiliza încă un șir de dimensiune 26

$maxdp_c$ = valoarea maximă a lui dp pentru litera *c* pe prefixul deja considerat

Inițial, $dp_i = 1$, iar $maxdp_c = 0$.

Recurența: $dp_i = \max(maxdp_{s[i] + 1}, maxdp_{s[i] + 2}, \ldots, maxdp_{25}) + 1$

$maxdp_{s[i]} = dp_i$

Complexitate timp : O(n * SIGMA), SIGMA = 26.

Proba pe echipe

Clasele IX-X

despotcovit (Vlad)

Soluția 1 (15-20p)

Pentru teste în valoare de 20 de puncte, se poate realiza o soluție folosing metoda backtracking în care se generează toate metodele de a așeza caii în grajduri și pentru fiecare dintre acestea, se calculează coeficientul de tristețe.

Soluția 2 (100p)

Pentru punctajul maxim, vom utiliza o soluție bazată pe programare dinamică. Ne vom folosi de 2 șiruri pentru a ține cont de culorile cailor:

x_i = numărul de cai albaștri în primii *i* cai

y_i = numărul de cai albi în primii *i* cai

Pentru a calcula soluția, vom folosi tabloul bidimensional:

$sol_{i,j}$ = coeficientul minim de tristețe care se poate obține plasând în primele *i* grajduri primii *j* cai

Astfel, tabloul *sol* îl vom inițializa cu $sol_{1,j} = x_j * y_j$, iar recurența este:

$sol_{i,j} = \min(sol_{i,j}, sol_{i-1,t} + (x_j - x_t) * (y_j - y_t))$ pentru t = 1, 2, ..., j – 1.

Soluția se găsește apoi în $sol_{G,N}$.

Complexitate timp : $O(N^2 * G)$.

trambuline (Ciprian)

La această problemă, putem folosi un BFS, în care ținem minte la fiecare pas pentru a putea determina vecinii curenți (pozițiile la care putem ajunge) care a fost lungimea săriturii pe orizontală și pe verticală.

Pentru a nu depăși limita de memorie putem observa că, pentru a avea o săritură de lungime X pe o axă (plecând cu o săritură de pe loc și ajungând la final tot cu o săritură pe loc), ne deplasăm cel puțin 1 + 2 + ... + X-1 + X + X-1 + ... + 2 + 1 = X^2 poziții.

Deoarece limitele problemei sunt (lungime, lățime) <= 50, înseamnă că putem restrânge spațiul de căutare la maxim sărituri de 7 în fiecare direcție. Cu această restricție o soluție implementată bine intră în limite.

Clasele XI-XII

cora (Vlad)

Soluția 1 (70-90p)

Se poate folosi un algoritm asemănător cu KMP care reușește să rețină pentru fiecare cuvânt care nu se potrivește destulă informație pentru a continua căutarea eficient.

Soluția 2 (100p)

Pentru punctaj maxim la această problemă, trebuie folosit un algoritm de potrivire a șirurilor de caractere numit Aho-Corasick. Ideea din spatele algoritmului este construirea unui trie bazat pe cuvintele primite, la care pentru fiecare nod *x* se adaugă o muchie de nepotrivire care duce într-un nod cu proprietatea că este cel mai lung prefix al cuvintelor care e și sufix al lui *x*.

plimbare (Robert)

Problema poate fi simplificata: avem un graf neorientat cu n noduri si m muchii cu costuri si vrem pentru fiecare nod sa gasim cel mai ieftin ciclu.

Tratam individual fiecare nod. Folosim Dijkstra pentru a afla ruta cea mai scurta de la nodul nostrum la celelalte noduri. Dupa Dijkstra putem percepe graful nostrum ca pe un arbore. Acum vrem sa gasim muchiile nefolosite(care nu sunt incluse in arborele nostrum) care conecteaza 2 noduri care sunt pe ramuri diferite (de ex 4 si 3 sunt pe ramuri diferite fata de radacina, dar 4 si 5 sunt pe aceasi ramura fata de radacina). "Ramuri diferite" inseamna ca cel mai batran stramos (excluzand radacina din calcul) este diferit pentru cele 2 noduri. Aceste muchii formeaza cicli.

```
      1
     / \
   2.   3
  / \.  |
 4  5   6
```

Astfel, iteram prin acesti cicli si il alegem pe cel minim

Putem optimiza aceasta abordare introducand limitele unui ciclu (upper bound). Exemplu: Daca am gasit pentru nodul 1 ciclul de lungime minima 15 1-3-5-6-1, atunci cand calculam cel mai ieftin ciclu pentru nodul 5, nu are sens sa ne indepartam cu mai mult de 15 de nodul initial, adica 5. Complexitatea ramane aceeasi, dar se imbunatateste timpul de executie.

Complexitate timp O (N * M log N).