empowersoft

Concurs online de informatică
Categoria PROGRAMARE

Secţiunea 11-12

DESCRIERE SOLUŢII

PROBLEMA 1 PIEPTBICEPS

Autor: Victor Popa
University of Oxford, anul I

Soluţie de 10 puncte

Se generează toate posibilitățile unui antrenament de lungime N cu ajutorul backtrackingului sau cu ajutorul generărilor tuturor submulțimilor cu operații pe biți, apoi se calculează eficient sau ineficient cât durează corectarea acestui antrenament, numărându-se cele care sunt corectate în M secunde.

Soluţie de 100 de puncte

Este necesară rezolvarea unei subprobleme: având un antrenament, calculează numărul de secunde necesare pentru că Mihai să corecteze antrenamentul în O(N). În primul rând putem să ignorăm momentan cel mai lung prefix format doar din P-uri.
Fie F(i) timpul necesar pentru al i-lea exercițiu de piept să ajungă în punctul în care va fi în configurația finala (adică pentru BPBP F(1) = 1 și F(2) = 2). Fie NRB(i) numărul de B-uri dinaintea celui de-al i-lea P. Sunt necesare următoarele observații:
1. Ordinea exercițiilor de piept nu se schimbă unul fața de celălalt – deci F(i) >= F(i-1)
2. Doua exerciții de piept nu se pot termina în același timp – deci conform 1. F(i) >= F(i-1) + 1
3. Pentru a ajunge în poziția finală al i-lea P trebuie să treacă de toate B-urile din stânga sa - deci F(i)>=NRB(i)
4. Dacă fix în fata celui al i-lea P se va afla un P la un moment dat în drumul sau spre poziția finală, atunci F(i) = F(i-1) + 1.
5. Dacă în drumul său spre poziția finală al i-lea P nu se va afla niciodată direct în spatele celui de-al (i-1)-lea P, atunci nu se va afla niciodată în spatele direct al vreunui alt P și deci în acest caz F(i) = NRB(i)
6. Având în vedere cele 2 posibilități și observațiile 2 și 3 => F(i) = max(F(i-1)+1 , NRB(i))
7. Răspunsul va fi cel mai mare F(i), dar conform observației 1, răspunsul este F(ultimul i)
Acum rezolvarea subproblemei se poate face prin parcurgerea și completarea a F(i)

Problema mare se rezolvă cu ajutorul programării dinamice.
Fie dp[lastF][nr_P][nr_B] = numărul de antrenamente ce au nr_P exerciții de piept și nr_B exerciții de biceps și durează lastF secunde pentru a fi corectate. Daca la un astfel de antrenament se mai adaugă un exercițiu de biceps se formează un antrenament care este corectat în lastF timp, are nr_P exerciții de piept și nr_B+1 exerciții de biceps. Daca se adaugă în schimb un exercițiu de piept, antrenamentul va dura max(lastF + 1 , nr_B) , va avea nr_P+1 exerciții de piept și nr_B exerciții de biceps. Așadar recurenta este:
dp[lastF][nr_P][nr_B+1] = (dp[lastF][nr_P][nr_B+1] + dp[lastF][nr_P][nr_B])%MOD
dp[max(lastF + 1,nr_B)][nr_P+1][nr_B] =
(dp[max(lastF + 1,nr_B)][nr_P+1][nr_B] + dp[lastF][nr_P][nr_B])%MOD

Atenție însă pentru că daca lastF = 0 și nr_B = 0 și se adaugă un P, lastF vă rămâne 0 întrucât în demonstrarea soluției subproblemei am ignorat cel mai mare prefix de P-uri. Ordinea efectuării recurentei vă fi după dimensiunea antrenamentului care este egal cu nr_P + nr_B. Răspunsul va fi suma tuturor elementelor de forma dp[M][i][N-i].
Complexitate timp: O(N^3)
Complexitate spatiu: O(N^3)

Ploieşti, 25 martie 2019

PROBLEMA 2 LUCKY7

Autor: Tamio Nakajima Vesa
University of Oxford, anul II
Teste + Soluţie : Victor Popa, Dinu Cristian, University of Oxford, anul I

Soluţie 100 Puncte

Pentru a putea răspunde în O(logN), se precalculează următoarea dinamică:
dp[i][j] = numărul de numere (fie x acest număr) de i cifre pentru care f(x) = i
Aceasta se poate calcula în următorul mod:
Daca la începutul unui număr de i-1 cifre ale căror suma mod 7 este j se va adauga o cifra k (evident 0<=k<10) atunci se va obține un număr de i cifre a căror suma mod 7 este (j+k) mod 7
Deci: dp[i][(j+k)%7] += dp[i-1][j]
Fie N[i] a i-a cifra a numărului N și L numărul de cifre al lui N.
Pentru un prefix N[1..i] se calculează toate numerele de L cifre (cu excepția cazului primei cifre) care sunt mai mici **sau egale** ca numărul de L cifre având prefixul N[1..i] și urmat de 0-uri, dar sunt **strict** mai mari decât numărul format din L cifre având prefixul N[1..i-1] și 0-uri în rest. Acest lucru poate fi făcut în următorul mod:
După ce am parcurs prefixul N[1..i-1] putem calcula suma acestor cifre mod 7, fie sum aceasta suma. Apoi, pentru a obține o secvență de cifre căutată utilizam o cifră num (0 <= num < N[i]) într-o secvență al carei prefix este N[1..i-1], urmat de num , urmat de orice secvență de lungime L-i.
Pentru a afla suma sumelor cifrelor mod 7 a acestor tip de numere, se utilizează următoarea formula: dp[L-i][j]*((sum + num + j)%7) pentru 0 <= j < 7.
Rezultatul pentru o cifră este așadar suma tuturor acestor sume de cifre mod 7, totul fiind luat mod 1000000007.
Rezultatul final este suma rezultatelor pentru o cifra mod 1000000007.
Complexitate timp : O(7*10*logN) = O(logN)
Complexitate spatiu : O(7*logN) = O(logN)

PROBLEMA 3 SOS

Autor: Dinu Cristian
University of Oxford, anul I

Soluţie 100 Puncte

Observam în primul rând că datele de intrare pot fi împărțite la 2 (primele 2 dimensiuni), respectiv la 3, înălțimea. Se va folosi aceasta optimizare pentru că altfel programul va ieși din memorie. (1 1 2 3 este echivalent cu 1 1 1 3 sau cu 2 2 3 3).
Vom folosi principiul folosit la sume parțiale într-un vector. Doar că acum, avem 4 dimensiuni (lungime, lățime, înălțime și timp) (50*50*33*100).
Daca se folosește o matrice de 50*50*33*100 în care d[i][j][k][t] este cantitatea de sos folosita în paralelipipedul cu colturile opuse (1,1,1) și (i,j,k) în perioada [1,t], scorul obținut va fi de aproximativ 30 puncte, în funcție de implementare. Dar în acest fel, operațiile vor avea timp liniar pentru fiecare dimensiune => O(n^4)
Pentru a putea face update și query în timp logaritmic, vom folosi arbori indexați binar (arbori fenwick). (Se poate face și cu arbori de intervale dar este mai complicat de implementat).
Observația este că, vom tine un arbore indexat binar pentru fiecare dimensiune, apoi fiecare poziție din acesta reprezintă un alt abore indexat binar, etc.

Ploiești, 25 martie 2019

De exemplu, pentru 2 dimensiuni vom avea:

```
inline int zeros(int x)
                                {
                                   return ((x^(x-1))&x);
                                }

void update(int x, int y, int val)
{       for(int i=x; i<=N; i+=zeros(i))
// vom itera pe fiecare pe care trebuie actualizata in primul arbore
for(int j=y; j<=M; j+=zeros(j))
// iteram acum pe fiecare pozitie care trebuie actualizata in al doilea arbore
                                v[i][j]+=val;
}
int compute(int x, int y)
{
        Int ans = 0;
        For(int i=x; i>0; i-=zeros(i))
                For(int j=y; j>0; j-=zeros(j))
                        Ans+=v[i][j];
}
```

Astel, vom avea 4 aib uri (4 for uri pentru fiecare operatie).

Pentru operaţia de calculare a răspunsului, folosim principiul includerii excluderii.
De exemplu, pentru o secventa în 3 dimensiuni între x,y,z si X,Y,Z vom avea
Ans = compute(X,Y,Z) - compute(X,Y,z) - compute(X,y,Z) - compute(x,Y,Z)+ compute(x,y,Z) + compute(x,Y,z) + compute(X,y,z) - compute(x,y,z)
Folosim acelaşi principiu în 4 dimensiuni.
Complexitate finală O (q * log(N*m*k*t/2/2/3)
Link util: (https://infoarena.ro/aib)

PROBLEMA 4 TANAKAGAME

Autor: Tamio Nakajima Vesa
University of Oxford, anul II
Descrierea solutiei : Dinu Cristian
University of Oxford, anul I

Soluţie 100 de puncte

O observaţie iniţială: jocul trebuie să se termine. Pentru a arăta aceasta, vom demonstra inductiv următoarea ipoteză: "pentru oricare K, dacă toate poziţiile mai sus de cea de a K-a diagonală secundară conţin doar 0, atunci jocul trebuie să se termine" (folosim inducţie "inversă", pentru K de la N+M până la 0):

- Cazul de bază: pentru K = N+M, vrem ca dacă toate poziţiile mai sus de cea de a N+M-a diagonală secundară sunt 0 (adica totul mai putin colţul jos-dreapta a matricii e 0), atunci jocul se termina. Cum fiecare mutare neapărat scade numărul acela, până ajunge la 0, în care caz jocul se termină, e clar ca jocul se termină în situaţia asta.

- Cazul inductiv: dacă ştim ipoteza pentru un K fix, o voi demonstra şi pentru K-1: Observăm că orice mutare e fie o mutare pe diagonala K-1, sau mai jos de diagonala K-1 (căci toate poziţiile mai

sus de diagonala K-1 conţin 0). Presupunem prin absurd că s-ar putea juca la infinit. Observăm că pot exista doar un număr finit de mutari pe diagonala K-1 (căci oricare mutare va reduce suma valorilor pe K-1 până ajunge la 0, în care caz nu se mai poate juca pe diagonala acesta). Astfel trebuie să fie un număr infinit de mutări doar pe sau sub diagonala K. Dar asta contrazice ipoteza inductivă - jocul ar fi echivalent cu unul care are doar 0 pe diagonala K-1 (căci oricum nu se joacă acolo), şi deci ar fi un exemplu de un joc care are doar 0 peste tot peste diagonala K care nu se termină. Contradicţia implică rezultatul dorit.

Astfel, jocul se termina. Considerăm, pentru orice stare M a matricii, următorul vector v de mărime K, definit astfel:

v[i] = suma XOR a valorilor (M[x][y]%4) unde (N - x + M - y) % K = I

Voi demonstra trei lucruri:

1. Dacă v conţine doar 0, atunci după oricare mutare, v va conţine măcar o valoare nenulă.
2. Dacă v conţine o valoare nenulă, există o mutare care îl face pe v să conţină doar 0.
3. Orice stare pierzatoare corespunde unui v plin de 0.

Dem. 1: Observăm că orice mutare nu poate schimba două celule care contribuie la aceeasi valoare din v. Mai mult, cum din ea vom scădea un număr prim sau 1, adica un număr ne-divizibil cu 4, este clar ca valoarea iniţială selectată într-o mutare va fi schimbată, modulo 4. Asta înseamnă că valoarea din v la care contribuie acea poziţie va fi schimbată (cum doar acea pozitie, din cele afectate de o mutare, va fi schimbată). Asta inseamnă ca acea valoare va deveni nenulă.

Dem. 2: Să luăm indicele maximal din v care corespunde unei valori nenule. Acea poziţie corespunde sumei XOR a mai multor valori. Se poate demonstra uşor că măcar una dintre acele valori poate fi redusă astfel încât acea suma XOR să dea 0. Să selectăm ca poziţie iniţială, astfel, poziţia care corespunde acelei valori, şi, să scădem din acea pozitie 1, 2 sau 3, încât să facem ca suma XOR a valorii corespunzatoare din v să dea 0. Acum, oricare drum maximal de lungime cel mult K ce începe la acea poziţie va trece neapărat cel puţin prin poziţii ce corespund valorilor în v cu indici mai mici decat indicele maximal care contine o valoare nenula. Pentru acele pozitii, se poate adaugă şi respectiv scădea 1, 2, 3, 5 sau 7 încât valoarea, modulo 4, a acelei poziţii să fie făcută orice vrem noi. Putem, deci, să le setam la fix valoarea necesară pentru a face suma XOR a valorilor din v corespunzătoare să fie 0.

Dem. 3: Singura stare pierzatoare este o matrice ce contine doar valori nule.

Astfel, observăm că dacă un câştigător are o stare în care v conţine o valoare nenulă, el poate câştiga cu siguranţa (căci el poate muta astfel încât să dea oponentului un v plin de 0, la care oponentul este fortat să îi dea înapoi o matrice cu un v care conţine măcar o valoare nenulă), având în vedere ca jocul e şi clar finit.