Focșani, 18-24 aprilie 2003

Olimpiada NAȚIONALĂ de INFORMATICĂ

Mihai Stroe

Vă prezentăm în continuare soluțiile problemelor propuse la ONI 2003, la clasele a XI-a și a XII-a și la cele două probe de baraj. Aceste soluții au fost realizate de redacția GInfo pe baza soluțiilor oficiale prezentate de autorii problemelor.

Clasele a XI-a și a XII-a

P050325: A007

Problema se reduce la găsirea unui centru de simetrie într-o mulțime de n puncte având coordonate întregi.

Punctul de simetrie pentru mulțimea de puncte are proprietatea că simetricul fiecărui punct din mulțime față de centrul de simetrie este tot un punct din mulțime. În concluzie, pentru a exista centru de simetrie trebuie ca n să fie par.

Dacă n este par, se observă că punctul căutat (centrul de simetrie) trebuie să fie mijlocul a $n/2$ segmente cu capetele în punctele date. Fiecare punct din mulțime ar fi capăt pentru exact un segment. Dacă am putea identifica aceste segmente, problema s-ar reduce la determinarea mijlocului unuia dintre ele și la testarea dacă acesta este și mijlocul celorlalte.

Notăm cu x și y vectorii care reține coordonatele celor n puncte.

Ordonăm punctele crescător în funcție de abscise. În cazul în care două abscise sunt egale vom ordona crescător în funcție de ordonată.

Pentru un segment cu capetele de coordonate (x_1, y_1) și (x_2, y_2) mijlocul segmentului are coordonatele $x = (x_1 + x_2) / 2$ și $y = (y_1 + y_2) / 2$.

Dacă există centru de simetrie, atunci acesta este plasat în mijlocul segmentului având capetele în primul punct respectiv în ultimul punct (adică cu un capăt în cel mai din stânga punct și celălalt în cel mai din dreapta). Notăm acest punct cu M.

Pentru ca să existe centru de simetrie trebuie ca segmentele având coordonatele (x_i, y_i) și (x_{n-i+1}, y_{n-i+1}) să aibă mijlocul în M, pentru orice i =1, 2, ..., n **div** 2. Se verifică aceste condiții pentru punctul M determinat.

Fără a observa legătura dintre sortarea punctelor în ordinea precizată și relația de simetrie, putea fi încercată o abordare bazată pe determinarea repetată a înfășurătorii convexe. Această abordare are complexitatea $O(N^2)$ pe cazul cel mai defavorabil, deci probabil că nu s-ar fi încadrat în timp.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $O(N)$.

Sortarea punctelor se va realiza cu unul din algoritmii clasici având ordinul de complexitate $O(N \cdot \log N)$.

Operațiile de determinare a mijlocului unuia din segmente și verificarea faptului că acest punct este și mijlocul celorlalte segmente au, în total, ordinul de complexitate $O(N)$.

Operația de scriere a rezultatelor are ordinul de complexitate $O(1)$.

În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N) + O(N \cdot \log N) + O(N) = \boldsymbol{O(N \cdot \log N)}$.

P050326: Asmin

Pentru această problemă există o soluție a cărei ordin de complexitate este $O(N^2)$. În continuare prezentăm această soluție.

Se consideră fiecare nod (pe rând) ca fiind rădăcină. Pentru rădăcina astfel fixată, valorile asociate nodurilor se pot calcula în timp liniar. Dacă rădăcină este nodul i, atunci valoarea rădăcinii este R_i. Pentru fiecare alt nod, valoarea asociată lui este cea mai mică valoare, astfel încât suma valorilor până la tatăl său în arborele cu rădăcina fixată plus această valoare să dea restul cerut la împărțirea cu numărul K. Valorile sunt atribuite nodurilor de la rădăcină către frunze. Astfel, o simplă parcurgere ***DF*** (*Depth First* - parcurgere în adâncime) este suficientă, odată ce rădăcina arborelui a fost fixată. Evident, această soluție nu se va încadra în timp pe testele mari.

Soluția optimă are ordinul de complexitate $O(N)$. Se fixează mai întâi ca rădăcină nodul 1 și se calculează valorile asociate nodurilor arborelui având rădăcina în nodul 1, conform algoritmului descris mai sus. Astfel se obține o valoare C_1 egală cu suma valorilor nodurilor arborelui când nodul 1 este rădăcina. Vom numi această valoare *costul asociat nodului* 1.

Pentru a calcula C_i ($i \neq 1$) nu este nevoie să repetăm parcurgerea *DF* pentru nodul i considerat rădăcină. În schimb, încă o parcurgere *DF* din nodul 1 este suficientă pentru a calcula costurile asociate fiecărui nod. Să presupunem că vrem să calculăm C_i și am calculat deja C_j, unde j este tatăl lui i, în arborele având rădăcina fixată în nodul 1.

Atunci, $C_i = C_j - R_j - \text{minim}(R_j, R_i) + R_i + \text{minim}(R_i, R_j)$, unde $\text{minim}(a, b)$ (cu $0 \leq a, b \leq K\text{-}1$) reprezintă valoarea x minimă ($0 \leq x$), astfel încât $a + x = b \pmod K$. Se alege minimul din vectorul C și se tipăresc nodurile i având valoarea C_i egală cu minimul.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $O(N)$, iar cea de scriere a rezultatului are ordinul de complexitate $O(1)$.

Prima rezolvare constă într-o parcurgere *DF* pentru fiecare nod. Deoarece ordinul de complexitate al unei parcurgeri este $O(N)$, ordinul de complexitate al acestei soluții este $O(N^2)$.

Rezolvarea optimă constă în două parcurgeri *DF*, în care se execută calcule elementare. Ordinul de complexitate al unei astfel de parcurgeri este $O(N)$.

În concluzie, ordinul de complexitate al soluției optime pentru această problemă este $O(N) + O(1) + O(N) = \mathbf{O(N)}$.

P050327: Căutare

În prima fază a rezolvării, se citesc și se sortează valorile care vor fi plasate în nodurile arborelui.

Se citesc interogările și, folosind căutări binare, se determină de câte ori se caută fiecare din numerele date și de câte ori se caută numere din intervalul dintre două numere date, consecutive în șirul sortat (sau unul din intervalele exterioare arborelui). Se obține astfel un șir de ponderi.

În continuare, se folosește metoda *programării dinamice*. Se calculează o matrice A, unde $a_{i,j}$ = costul minim pentru a forma un arbore de căutare cu nodurile de la i la j (unde i și j sunt poziții, deci indici pe șirul sortat de valori).

Pentru calculul unei astfel de valori se încearcă fixarea rădăcinii în fiecare poziție k (între i și j inclusiv). Costul arborelui se determină pe baza lui $a_{i,k}$ și $a_{k+1,j}$; determinarea relației exacte este lăsată ca exercițiu pentru cititor.

Valorile $a_{i,j}$ sunt calculate în ordinea crescătoare a diferenței i - j; matricea este deci completată pe diagonale. Rezultatul cerut se va afla în $a_{1,N}$.

Ordinul de complexitate al acestei rezolvări este $O(N^3)$ și este dat de cele 3 cicluri `for` imbricate (pentru diferența între i și j, pentru i și pentru k). Timpul de rezolvare pentru un set de date de intrare este rezonabil, dar fișierul de intrare poate conține un număr mare de astfel de seturi de date. În acest caz, rezolvarea nu se va încadra în timp.

Pentru a reduce complexitatea la $O(N^2)$ putem face următoarea observație. Dacă notăm cu $R_{i,j}$ rădăcina arborelui optim care ar conține pozițiile de la i la j din șirul sortat de valori, atunci:

$$R_{i,j-1} \leq R_{i,j} \leq R_{i+1,j}.$$

Folosind această observație, valorile de pe o anumită diagonală a matricei se pot calcula în timp liniar.

Analiza complexității

În continuare vom analiza complexitatea rezolvării pentru un singur set de date.

Operația de citire a datelor de intrare are ordinul de complexitate $O(N + M)$, iar scrierea rezultatului se realizează în timp constant.

Ordinul de complexitate al primei variante de construire a matricei este $O(N^3)$. Această variantă nu s-ar fi încadrat în timp pentru testele mari.

Folosind relația existentă între valorile din R și calculând elementele de pe fiecare diagonală a matricei în $O(N)$, ordinul de complexitate al construirii matricei este $O(N^2)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru un singur set de date este $O(N + M) + O(1) + O(N^2) = \mathbf{O(N^2 + M)}$.

Dacă luăm în considerare toate cele T teste, ordinul de complexitate devine $\mathbf{O(T \cdot (N^2 + M))}$.

P050328: Inter

Sunt $N \cdot (N - 1)/2$ puncte de intersecție. Din aceste puncte ne interesează punctele situate în extremități pe fiecare dreaptă. Aceste puncte, cel mult N, sunt vârfurile înfășurătorii convexe a celor $N \cdot (N - 1)/2$ puncte.

Se sortează dreptele după pantă. Se observă că punctele de interes sunt intersecțiile a câte două drepte consecutive în vectorul sortat, și în plus intersecția primei drepte cu ultima.

Se determină înfășurătoarea convexă a acestor N puncte, folosind un algoritm clasic, și se calculează aria zonei rezultate.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $O(N)$, iar cea de scriere a rezultatului are ordinul de complexitate $O(1)$.

Sortarea dreptelor în funție de pantă are ordinul de complexitate $O(N \cdot \log N)$.

Determinarea punctelor de interes are ordinul de complexitate $O(N)$.

Determinarea înfășurătorii convexe a acestor puncte are ordinul de complexitate $O(N \cdot \log N)$.

Calcularea ariei poligonului obținut are ordinul de complexitate $O(N)$.

În final, ordinul de complexitate al rezolvării este $O(N) + O(N \cdot \log N) + O(N) + O(N) + O(N) = \mathbf{O(N \cdot \log N)}$.

P050329: Număr

Se determină numărul de apariții pentru fiecare cifră având valori în intervalul [0, 9] în numărul dat x. Acest număr trebuie să fie par (cu excepția eventuală a unei singure cifre; în caz că o astfel de cifră există, trebuie să fie plasată la mijlocul palindromului), altfel nu există soluție.

În continuare se va construi prima jumătate a palindromului; cealaltă jumătate va fi obținută prin oglindirea primei jumătăți.

Se determină numărul de apariții ale fiecărei cifre (0..9) în prima jumătate a numărului x, precum și numărul de apariții necesar (pentru numărul cerut). Dacă pentru o cifră numărul de apariții existent în prima jumătate a numărului dat este prea mare, se elimină aparițiile suplimentare, începând de la dreapta spre stânga.

Se încearcă să se distribuie cifrele care mai sunt necesare în prima jumătate a numărului astfel încât să se obțină cel mai mic număr posibil mai mare sau egal cu x (cu o eventuală rearanjare a cifrelor existente). Distribuirea cifrelor se face căutând de la stânga la dreapta pozițiile cu cifre egale și completarea numărului cu cele din numărul dat.

Când acest lucru nu mai este posibil, ne aflăm în unul din următoarele două cazuri:

- am obținut o jumătate mai mică sau egală cu cea din numărul dat (cea mai mare cu această proprietate). Determinăm anagrama imediat următoare din punct de vedere lexicografic a primei jumătăți a lui x; dacă o astfel de anagramă există, ea (împreună cu cea de-a doua jumătate a palindromului) dă soluția problemei.
- am găsit-o pe cea mai mică cifră mai mare decât cea de pe poziția curentă. Plasăm cifra respectivă, ordonăm crescător cifrele rămase, construim cea de-a doua jumătate și obținem soluția problemei.

Analiza complexității

Ordinul de complexitate al operațiilor de citire a datelor de intrare și furnizare a rezultatelor este $O(N)$.

Determinarea numărului de apariții al fiecărei cifre în numărul x și în numărul dorit au ordinul de complexitate $O(N)$.

Construirea pas cu pas a primei jumătăți a palindromului prin adăugarea unei cifre cât timp este posibil are ordinul de complexitate $O(N)$.

Fiecare din cele două cazuri care încheie rezolvarea se tratează cu algoritmi de complexitate $O(N)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(N) + O(N) + O(N) + O(N) + O(N) = \boldsymbol{O(N)}$.

P050330: Proc

Se observă că timpul minim după care se va termina aplicația este între 0 și $2 \cdot N$ (în cazul când toate fragmentele de cod au timpul 2 și sunt executate pe primul procesor). De asemenea, se observă că, fixând o limită de timp T, se poate determina dacă putem executa toate fragmentele de cod în timpul limită acordat, folosind procesoarele existente. Ne interesează limita minimă de timp pentru care această condiție este respectată.

Se va folosi o căutare binară pentru determinarea limitei de timp. La un moment dat, în cadrul căutării, se va testa dacă putem executa toate fragmentele de cod. Dacă da, atunci se încearcă un T mai mic. Altfel, se mărește T-ul.

Funcția care decide dacă putem executa toate fragmentele de cod în timpul T are complexitatea $O(P)$, unde P este numărul de procesoare disponibile. Se observă că fiecare procesor K asigură $[T/K]$ unități de timp elementare.

Se calculează $N_2 = \sum_{k=1}^{P} [[T/K]/2]$, având semnificația: numărul maxim de fragmente de cod cu timpul 2 care pot fi executate dacă timpul minim este cel mult T. Dacă N_2 este mai mare sau egal cu numărul de fragmente de cod cu timpul 2 existente (să notăm acest număr cu DOI), se verifică dacă se pot executa destule fragmente de cod cu timpul 1. Pentru aceasta se calculează $N_1 = \sum_{k=1}^{P} [[T/K] \bmod 2]$. Dacă $N_1 + 2 \cdot (N_2 - DOI)$ este mai mare sau egal cu numărul de fragmente de cod cu timpul 1, atunci putem executa toate fragmentele de cod în timpul T.

Analiza complexității

Operațiile de citire a datelor și afișarea rezultatelor au ordinul de complexitate $O(1)$.

Sunt $\log N$ pași de căutare binară, deoarece valoarea cautată este limitată superior de $2 \cdot N$.

În cadrul fiecărui pas se execută o funcție care are ordinul de complexitate $O(P)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(1)+O(\log N) \cdot O(P) = \boldsymbol{O(P \cdot \log N)}$.

Baraj

P050331: Excursie

Această problemă admite o rezolvare matematică. Se evidențiază următoarele cazuri pe care le prezentăm în continuare.

În cazul în care $K > P$, atunci rezultatul este 0.

Dacă $P = K$, vom distribui cei K ghizi în primele K excursii. Rămân $N - K$ persoane, care pot fi repartizate în cele K excursii disponibile. În concluzie, există K^{N-K} posibilități.

Ultimul caz corespunde relației $P > K$.

Să notăm $r = P - K$, $r \geq 1$.

Repartizăm mai întâi cei K ghizi în K excursii distincte (acestea vor fi excursiile 1, 2,... , K).

Rămân $N - K$ persoane. Dintre acestea, unele vor fi repartizate în cele $P - K = r$ excursii care nu au ghizi și celelalte vor fi repartizate în excursii care au ghizi.

Notăm cu x numărul de persoane care vor fi repartizate în excursii care nu au ghizi. Acestea pot fi partiționate în r excursii în $S(x, r)$ moduri (cu $S(x, r)$ notăm *numerele lui Stirling de speța a II-a*).

Deci există $N - K - x$ persoane care trebuie repartizate în cele K excursii care au ghizi. Acest lucru se poate realiza în K^{N-K-x} moduri.

Deoarece putem selecta x persoane din cele $N - K$ rămase în C_{N-K}^{x} moduri, rezultă că se obțin:

$$NR = \sum_{x=r}^{N-K} \left(C_{N-K}^{x} \cdot S(x,r) \cdot K^{N-K-x} \right) \text{ posibilități.}$$

Numerele lui Stirling de speța a II-a, $S(n, m)$ reprezintă numărul de partiții ale unei mulțimi cu n elemente în m clase ($n \le m$). Evident, clasele partiției trebuie să fie nevide.

Pentru a determina numerele lui *Stirling de speța a II-a* se poate utiliza următoarea relație de recurență:

$S(n, 1) = S(n, n) = 1$;

$S(n, m) = S(n - 1, m - 1) + m \cdot S(n - 1, m)$.

Remarcăm că pentru rezolvarea problemei nu este necesară cunoașterea în prealabil a numerelor lui *Stirling* .

Analiza complexității

Operațiile de citire și scriere a datelor au ordinul de complexitate $O(1)$.

Cazul cel mai defavorabil, din punct de vedere al complexității se obține pentru $P > K$.

Pentru calculul rezultatului se folosesc numere mari și vom considera că folosim numere mari de M cifre.

Calculul numerelor *Stirling* de speța a II-a are ordinul de complexitate $O(N \cdot P \cdot M)$. Se folosesc operațiile de adunare a numerelor mari și înmulțire a unui întreg mic cu un număr mare.

Combinările din cadrul formulei se pot calcula într-un timp $O(N^2 \cdot M)$, folosind formula $C(n, k) = C(n - 1, k) + C(n - 1, k - 1)$ și operația de adunare a numerelor mari.

Puterile lui K se pot calcula cu un algoritm de complexitate $O(N \cdot M)$.

Rezultatul final se poate calcula cu un algoritm de complexitate $O(N \cdot M^2)$, folosind operația de înmulțire a numerelor mari.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $\boldsymbol{O(N \cdot M^2)}$. Nu am ținut cont de modul în care crește numărul M de cifre pe parcursul rezolvării, dar îl putem considera ca fiind tot timpul egal cu cea mai mare valoare pe care o poate atinge.

P050332: Robot

Pentru a rezolva această problemă pornim de la următoarele observații.

Regiunea accesibilă unui braț format dintr-un singur segment de lungime L_1 este, în mod evident un cerc cu raza L_1 și centrul în origine (umărul robotului).

Regiunea accesibilă unui braț format din două segmente de lungimi L_1, respectiv L_2 este un inel, cu raza exterioară $L_1 + L_2$ și raza interioară $|L_1 - L_2|$.

Regiunea de accesibilitate a unui braț format din n segmente este un disc centrat în origine care are raza exterioară $ro = L_1 + L_2 + \ldots + L_n$ (corespunzător alinierii tuturor segmentelor). Raza interioară este 0 dacă lungimea celui mai lung segment L_{max} este mai mică decât jumătate din lungimea totală a brațului. În caz contrar, raza interioară este egală cu $2 \cdot L_{max} - ro$. Se poate demonstra că orice punct din acest disc este accesibil.

Cu aceste observații, în continuare trebuie construită o configurație a brațului care să permită atingerea punctului dorit.

Dacă brațul are două segmente, de lungimi L_1 și L_2, problema se reduce la o simplă intersecție de cercuri. Fie p punctul care trebuie să fie atins. Intersectăm cercul C_1 de rază L_1, centrat în origine, cu cercul C_2, de rază L_2, centrat în p. În general pot exista 0, 1, 2 sau o infinitate de puncte de intersecție în cazul în care cele două cercuri coincid. Fie R un punct de intersecție. Pentru a determina configurația brațului, trebuie să calculăm unghiul dintre axele OX și OR.

Dacă brațul are trei sau mai multe segmente, se poate demonstra că orice punct din mulțimea punctelor accesibile poate fi atins dacă îndoim brațul în cel mult două locuri, și anume în extremitățile segmentului care conține mijlocul brațului, dacă brațul ar fi întins.

Prima impresie ar fi că o singură îndoire este suficientă, dar există contraexemple clare care demonstrează incorectitudinea acestui raționament.

După ce am stabilit unde îndoim brațul, având la dispoziție centrul și punctul care trebuie atins, configurația brațelor se determină cu ajutorul unor calcule simple.

Analiza complexității

Operațiile de citire și scriere a datelor au ordinul de complexitate $O(N)$.

Complexitatea algoritmului este $O(N)$, datorită calculării sumei lungimilor segmentelor și a poziției mijlocului brațului. Calculele matematice care urmează se realizează în timp constant.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(N) + O(N) + O(N) = \boldsymbol{O(N)}$.

P050333: Telegraf

Personal, problema mi s-a părut foarte dificilă și, cel puțin în concurs, nu aș fi știut să o rezolv. În aceste condiții, ea putea fi tratată ca un exercițiu de "*furat puncte*", cu rezultate mai mult sau mai puțin satisfăcătoare. Oricum, nivelul de dificultate al problemelor crește de la an la an, fapt ce se evidențiază și în rezultatele excelente obținute de elevii români participanți la concursurile internaționale.

Soluția problemei, descrisă mai jos, este construită numai pe baza indicațiilor autorului problemei.

Pentru cunoscători, problema seamănă cu codificarea *Huffman*, diferența (esențială) constând în faptul că unul din simbolurile elementare din textul codificat necesită două unități de timp pentru a fi transmis.

Soluția problemei constă în determinarea unui arbore binar, în care:

- frunzele sunt asociate simbolurilor din șir;
- deplasarea dintr-un nod spre stânga corespunde unui punct, iar spre dreapta, unei linii;
- durata transmisiei unui simbol este dată de lungimea drumului de la rădăcină la frunza asociată. Punctul contribuie cu 1 la lungime, iar linia, cu 2.

Simbolurile fiind asociate frunzelor, textul transmis codificat nu este ambiguu (reprezentarea unui simbol din șirul inițial nu poate fi prefixul reprezentării unui alt simbol).

În plus, se observă că arborele este binar strict. Asftel, dacă ar exista în arbore un nod cu un singur descendent, putem șterge acest nod și îl înlocuim cu arborele care are ca rădăcină descendentul, obținând o codificare cu timp de transmisie mai mic.

Dacă vom considera rădăcina pe nivelul 0, fiul stâng al unui nod de pe nivelul i fiind situat pe nivelul $i + 1$, iar fiul drept pe nivelul $i + 2$ (datorită lungimii diferite a muchiilor), se observă că o parcurgere pe niveluri a arborelui va lista simbolurile în ordinea descrescătoare a frecvenței. Altfel spus, simbolurile care apar mai des vor fi situate mai aproape de rădăcină.

Folosind aceste observații putem construi un algoritm bazat pe *metoda programării dinamice*. Vom construi (conceptual) arborele pe niveluri de sus în jos. Următoarele informații reprezintă starea:

- A - numărul de noduri interne care sunt pe nivelul $L - 1$;
- B - numărul de noduri interne care sunt pe nivelul $L - 2$;
- F - numărul de frunze care au apărut deja pe nivelurile mai mici decât L.

Este posibil să găsim optimul pentru o anumită configurație (A, B, F), la orice nivel s-ar afla. Bineînțeles, dacă nu știm valoarea lui L nu putem calcula efectiv costul unei frunze când o construim. De aceea costurile trebuie adunate "pe parcurs": când coborâm cu un nivel în arbore, vom aduna costul unui nivel în contul tuturor frunzelor aflate mai jos decât nivelul respectiv. Evident, frunzele situate mai jos de nivelul L au indicii $F + 1$, $F + 2$, … (dacă am ordonat simbolurile descrescător în funcție de frecvența lor de apariție).

Deci, costul unui nivel L va fi dat de suma frecvențelor simbolurilor începând cu $F + 1$; aceste sume le vom calcula la începutul algoritmului.

Așa cum am menționat, algoritmul construiește un astfel de tablou tridimensional folosind *metoda programării dinamice*. Memoria folosită este $O(N^3)$, unde am folosit N pentru numărul de simboluri (36 în cazul de față). Cel mai simplu mod de a calcula elementele tabloului este o soluție recursivă cu memoizare. Pentru aceasta, trebuie să vedem din ce poate proveni o soluție (A, B, F). Singura variabilă este numărul de frunze de pe nivelul L; pentru X frunze pe nivelul L, starea precedentă este (B - A, A, F - X). Aceasta ne dă un algoritm în timp $O(N^4)$ pentru construirea tabloului.

Analiza complexității

Notăm cu N numărul de simboluri cu frecvențe strict pozitive. În enunțul acestei probleme, N este limitat superior de 36.

Datorită limitei superioare pentru numărul N, citirea datelor are ordinul de complexitate $O(36)$.

Operația de construire a matricei, prin metoda programării dinamice, are ordinul de complexitate $O(N^4)$.

Operațiile de găsire a optimului se pot realiza în timpul construirii matricei.

Operația de afișare a rezultatelor are ordinul de complexitate $O(1)$.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(36) + O(N^4) + O(1) = \boldsymbol{O(N^4)}$.

P050334: Ajutor

Problema se poate rezolva în mai multe moduri. În continuare vom prezenta două soluții ale acesteia.

Se consideră mai întâi cele N puncte de prim ajutor. Se iau coordonatele x ale acestor puncte într-un vector, care este sortat. Se face același lucru și cu coordonatele y. Fiecare punct având abscisa în vectorul x și ordonata în vectorul y va reprezenta un nod într-o rețea ortogonală (caroiaj). Pentru aceste noduri se calculează, folosind *metoda programării dinamice*, distanța până la cel mai apropiat post de prim ajutor.

Se folosește metoda programarii dinamice de patru ori, începând din stânga-sus, apoi începând din stânga-jos, apoi începând din dreapta-jos, apoi începând din dreapta-sus. Calculele se fac (conceptual) cu ajutorul unor matrice $a[i, j]$, reprezentând distanța până la cel mai apropiat post de prim ajutor, dacă am merge, de exemplu, numai în stânga și în sus (pentru prima matrice), din pozitia (i, j) din caroiaj, adică a i-a coordonată y, după sortare și a j-a coordonată x după sortare. Calculul se face $O(N^2)$, pentru că, de exemplu, valoarea lui $a[i, j]$ nu depinde decât de vecinii din stânga-sus.

Dintre cele patru valori se reține minimul și se memorează valoarea sa într-o matrice.

Pentru a calcula valorile respective nu este nevoie de matrice, ci de doi vectori: nivelul curent și nivelul anterior.

Fiecare dintre cele M locații ale accidentului se rezolva (se află minimul) în $O(\log N)$, pentru că nu trebuie decât să aflăm în care celulă din caroiaj se află punctul în care ne situăm (două căutări binare în vectorii x și y), iar acolo să vedem în care dintre cele 4 colțuri este mai bine să mergem.

Complexitatea acestei variante de rezolvare este, în consecință, $O(N^2 + M \cdot \log N)$

În cele ce urmează vom prezenta o altă soluție care are ordinul de complexitate mai mic.

Datorită limitărilor coordonatelor pentru puncte, apare și posibilitatea de a răspunde la o cerere în $O(1)$. Acest lucru se face prin preprocesarea vectorilor x și y. Se mai rețin două tablouri în care, pentru fiecare coordonată x, se află coloana i și, pentru fiecare coordonată y, se află linia j a caroiajului obținut prin preprocesare.

Este nevoie deci de două tablouri a câte 32000 de elemente de tip `word` (respectiv, `unsigned int`). În acest caz, răspunsul pentru fiecare locație posibilă a accidentului poate fi dat în timp constant (după rezolvarea subproblemei de programare dinamică).

Complexitatea acestei variante de rezolvare a problemei este $O(N^2 + M)$.

Analiza complexității

Operația de citirea a datelor are ordinul de complexitate $O(M + N)$, iar cea de scriere a rezultatelor are ordinul de complexitate $O(M)$.

Preprocesarea, constând în aplicarea *metodei programării dinamice* pentru a calcula distanțele minime în cazul caroiajului, are ordinul de complexitate $O(N^2)$.

Preprocesarea pe axele x și y, din cadrul celei de-a doua variante de rezolvare, are ordinul de complexitate constant (datorită limitării la 32000).

Pentru fiecare locație posibilă a accidentului, răspunsul se dă în timp constant în varianta 2, în timp ce în cazul primei variante de rezolvare ordinul de complexitate este $O(\log N)$ datorită căutării binare.

În concluzie, ordinul de complexitate al rezolvării este $O(M + N) + O(N^2) + O(M) \cdot O(\log N) + O(M) = \boldsymbol{O(N^2 + M \cdot \log N)}$ în cazul primei variante, respectiv $O(M + N) + O(N^2) + O(M) + O(M) = \boldsymbol{O(N^2 + M)}$ pentru cea de-a doua variantă.

P050335: Ghizi

Se observă că problema devine o problemă de flux într-o rețea de transport dacă facem următoarele transformări:

- momentele întregi de timp se transformă în nodurile unui graf;
- un interval $[T_1, T_2)$ asociat unui ghid introduce un arc de la T_1 la T_2, cu capacitatea 1. Două noduri pot fi conectate prin mai multe arce.

Se observă că un drum de la sursă (nodul 0) la destinație (nodul 100) acoperă fiecare moment de timp din [0, 100) exact o dată. Astfel, problema poate fi reformulată:

"*Dându-se graful construit conform transformării de mai sus, să se determine K drumuri, disjuncte din punct de vedere al muchiilor, de la sursă la destinație.*"

Problema este în mod evident o problemă de flux; se introduce un arc de capacitate K de la o sursă fictivă la nodul 0 și se execută un algoritm de flux maxim.

Trebuie aleasă o variantă de implementare care să se încadreze în timpul de execuție acordat. Nu descriem algoritmii de flux deoarece, pentru scopul acestui articol, îi considerăm cunoscuți.

Analiza complexității

Notăm cu T numărul de momente distincte de timp care pot fi capete ale unui interval asociat unui ghid (maxim 101).

Operațiile de citire a datelor de intrare și scriere a rezultatelor au ordinul de complexitate $O(N)$.

Construirea grafului are, de asemenea, ordinul de complexitate $O(N)$. Se obține un graf cu T noduri și N muchii.

Aplicarea algoritmului de flux maxim *Ford-Fulkerson* presupune găsirea a K drumuri de creștere de la sursă la destinație. Presupunând că fiecare drum se determină printr-o singură parcurgere a grafului cu complexitatea $O(N)$, găsirea celor K drumuri de creștere are ordinul de complexitate $O(N \cdot K)$.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(N) + O(N \cdot K) + O(N) = \boldsymbol{O(N \cdot K)}$.

P050336: Sala

Se poate demonstra, prin inducție matematică (cu pasul 3), că numărul maxim de băieți în sală este egal cu:

$$x_n = [(n \cdot (n+1)+1)/3].$$

Pentru $n \le 3$ verificările sunt imediate.

Să considerăm afirmația adevărată pentru n și să o demonstrăm pentru $n + 3$.

Fie $a_1, a_2, \ldots, a_n, a_{n+1}, a_{n+2}, a_{n+3}$ o așezare pentru $n + 3$ persoane. Următoarele două rânduri sunt date prin vectorii:

$b_1, b_2, \ldots, b_n, b_{n+1}, b_{n+2}$

$c_1, c_2, \ldots, c_n, c_{n+1}$

Pentru aceste trei rânduri de persoane numărul minim de fete este $n + 2$, deci numărul maxim de băieți este $2 \cdot n + 4$.

Astfel obținem $x_{n+3} \le 2 \cdot n + 4 + x_n$.

Conform ipotezei de inducție se obține că:

$$x_{n+3} \le 2 \cdot n + 4 + [(n \cdot (n+1)+1)/3] = [((n+3) \cdot (n+4)+1)/3].$$

Relația de mai sus reprezintă exact ceea ce trebuia demonstrat.

Valoarea maximă $x_n = [(n \cdot (n+1)+1)/3]$ se poate obține pentru următoarea așezare în sală:

```
bbfbbfbbf…
fbbfbbfb…
bfbbfbb…
…
b
```

Calculul valorii necesită implementarea operațiilor cu numere mari, însă programul poate fi redactat ușor.

Analiza complexității

Notăm cu K numărul de cifre al numărului dat la intrare.

Operația de citire a datelor de intrare are ordinul de complexitate $O(K)$.

Pentru calculul formulei folosim înmulțirea pe numere mari, de complexitate $O(K^2)$.

Operația de afișare a rezultatului are ordinul de complexitatea $O(K)$, deoarece rezultatul nu poate avea mai mult de $2 \cdot K$ cifre.

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(K) + O(K^2) + O(K) = \boldsymbol{O(K^2)}$.

Observații

Toate variabilele, care au fost folosite în rezolvarea acestor probleme și nu au fost explicate, au semnificația dată în enunțurile problemelor, care au fost publicate în *GInfo* 13/5 (mai 2003).