Yong-In, Coreea de Sud

IOI 2002

În continuare vă vom prezenta soluțiile oficiale ale celor șase probleme propuse spre rezolvare la ediția din acest an a Olimpiadei Internaționale de Informatică, desfășurată în perioada 18-25 august 2002.

P060213: Broscuțele buclucașe

Un algoritm naiv de rezolvare a acestei probleme are ordinul de complexitate $O(N^3)$ și implică parcurgerea tuturor celor $O(N^2)$ segmente care au ca extremități puncte din mulțimea *S*. Un segment de lungime *L* poate fi extins în oricare din cele două direcții ale sale cu un alt segment de lungime *L*, dacă extremitățile noului segment fac parte din mulțimea *S*. Se obține astfel un segment cu lungimea $2 \cdot L$ care poate fi extins cu un alt segment de lungime *L*, dacă se respectă aceeași condiție. Procedeul continuă până în momentul în care segmentul nu mai poate fi extins.

Numărul plantelor culcate la pământ de o broscuță care parcurge un astfel de segment extins va fi dat de numărul extinderilor la care se adaugă valoarea 2 (cele două plante corespunzătoare segmentului inițial). Dacă un segment nu poate fi extins până în afara lanului, atunci el nu este luat în considerare. Operația de extindere are ordinul de complexitate $O(N)$. Soluția va fi dată de numărul maxim de plante culcate la pământ corespunzătoare unui anumit segment.

Un algoritm eficient, cu ordinul de complexitate $O(N^2)$ se bazează pe algoritmul de determinare a unei ***submulțimi coliniare egal spațiate***. Algoritmul funcționează prin suprapunerea tuturor tripletelor egal spațiate cu scopul de a determina toate mulțimile coliniare egal spațiate maximale. Suprapunerea este realizată prin construirea unui graf neorientat unde, pentru fiecare triplet egal spațiat (p_A, p_B, p_C) sunt introduse nodurile <*A*, *B*> și <*B*, *C*> și muchia (<*A*, *B*>, <*B*, *C*>); componentele conexe ale acestui graf corespund submulțimilor coliniare egal spațiate maximale din mulțimea *S*. Se observă că drumul unei broscuțe este un lanț liniar format din noduri conectate (cu cel puțin o muchie și două noduri, adică cel puțin trei plante culcate la pământ) ale acestui graf. Fiecare nod al grafului are gradul cel mult 2, deci mulțimea muchiilor și cea a vârfurilor grafului au ambele dimensiunea $O(N^2)$, așadar putem determina toate submulțimile coliniare egal spațiate maximale într-un timp de ordinul $O(N^2)$.

Singurul detaliu rămas în discuție este modul în care pot fi determinate eficient tripletele egal spațiate folosite în construirea grafului. O metodă evidentă este parcurgerea tuturor tripletelor, dar aceasta ar face ca ordinul de complexitate să devină $O(N^3)$. Dacă păstrăm în memorie lanul de orez sub forma unei matrice (fiecare plantă este un element al acesteia) care conține numărul de ordine al aterizării pe acea plantă (de exemplu, dacă cea de-a 100-a plantă culcată la pământ are coordonatele (10, 12), atunci valoarea elementului corespunzător poziției (10, 12) va fi 100), putem lua în considerare toate perechile de plante (p_A, p_B) și apoi determinăm în timp constant poziția plantei p_C, deoarece locația acesteia este determinată în mod unic de p_A, și p_B. Ordinul de complexitate al acestei operații este $O(N^2)$, dar avem nevoie de un spațiu de memorie de ordinul O(*dimensiunea lanului*). În cazul cel mai defavorabil, matricea va avea $5000 \cdot 5000$ elemente, fiecare fiind un întreg reprezentat pe 2 bytes. Așadar, această matrice va ocupa aproximativ 50 MB memorie. Datorită faptului că stocarea grafului folosit pentru rezolvare necesită, de asemenea, un spațiu de memorie de ordinul $O(N^2)$, limita de 64 MB, impusă în enunțul problemei, este depășită. Totuși, această matrice este foarte rarefiată (majoritatea elementelor au valoarea 0), deci poate fi stocată sub forma unei tabele de dispersie. Din nefericire, folosirea unei tabele de dispersie afectează ordinul de complexitate al algoritmului. Cea de-a treia și cea mai bună opțiune este construirea grafului în timp liniar și folosirea unei memorii cu dimensiune constantă prin sortarea locațiilor (în funcție de linie și în cadrul aceleiași linii în funcție de coloană) și păstrarea a trei referințe în lista obținută (referințele *A*, *B*, *C* vor indica punctele p_A, p_B, p_C - $A < B < C$). Construirea grafului se realizează astfel: se iau în considerare toate valorile posibile pentru *A* și apoi, la fiecare pas se incrementează fie valoarea *B*, fie valoarea *C*, astfel încât să fie menținută o spațiere cât mai apropiată între p_A, p_B și p_C; când spațierea este egală, nodurile și muchia corespunzătoare sunt introduse în graf.

Există de asemenea un algoritm cu ordinul de complexitate $O(N^2)$ care folosește *metoda programării dinamice*. Acest algoritm se confruntă cu aceleași probleme de memorie ca și cele descrise anterior. Pe lângă matricea folosită

în algoritmul anterior se stochează o altă matrice de dimensiune $O(N^2)$ ale cărei linii sunt indexate de p_A; fiecare linie conține N elemente, unul pentru fiecare plantă p_B, ale căror valori reprezintă numărul de aterizări dintr-un traseu al unei broscuțe care trece prin p_A și p_B, iar celelalte puncte se află înaintea lui p_A în lista ordonată. Cu alte cuvinte, presupunem că lanul se termină în poziția p_A și căutăm trasee ale broscuțelor (care pot avea orice lungime) în acest lan de dimensiune mai mică. Ideea este de a determina trasee parțiale ale broscuțelor care nu încalcă nici una dintre condițiile referitoare la trasee în regiunea deja examinată a lanului. Presupunând că primele A linii ale matricei sunt completate, a $(A + 1)$-a linie este completată luând în considerare toate cele $O(N)$ plante culcate la pământ B, care se află în listă înainte de p_A și, dacă există o plantă culcată la pământ C, astfel încât A, B și C sunt egal spațiate, se caută în șir numărul aterizărilor din traseul de la C la B, se incrementează cu 1 și se stochează ca al B-lea element de pe linia A. Dacă C este în afara lanului, atunci valoarea corespunzătoare va fi 2. În același timp, se verifică dacă următoarea plantă culcată la pământ (B) se află în afara grafului; în acest caz a fost determinat un traseu complet. Pentru determinarea eficientă a valorii C avem nevoie de aceeași matrice de 50 MB, descrisă pentru algoritmul anterior. Din nou, poate fi folosită o tabelă de dispersie, dar în cazul cel mai defavorabil, ordinul de complexitate al algoritmului crește.

P060214: Utopia divizată

Aceasta este o problemă în spațiul bidimensional (plan), dar este mai ușor de rezolvat dacă o considerăm ca două probleme într-un spațiu unidimensional. Pentru început vom arăta cum poate fi rezolvată problema mai simplă, într-un spațiu unidimensional. Enunțul acestei probleme ar fi:

Dându-se N numere de cod naturale și o secvență de N semne corespunzătoare celor două regiuni ('+' sau '-'), determinați o secvență de valori de cod cu semn x, astfel încât $\sum_{i=1}^{k} x_i$ *are semnul identic cu cel corespunzător celei de a k-a regiuni.*

Soluția este intuitivă, dar demonstrarea corectitudinii acesteia este puțin mai dificilă. La început, vom sorta în ordine crescătoare cele N numere de cod și apoi le vom atribui semne alternante. Cu alte cuvinte, pentru $|x_i| > |x_i - 1|$ avem $x_i > 0$ dacă și numai dacă $x_{i-1} < 0$. Semnul atribuit celui mai mic cod determină semnele atribuite tuturor celorlalte coduri. Vom începe dintr-o poziție aflată în "mijlocul" secvenței ordonate și ne vom deplasa înspre exterior. Vom folosi numere mai mici pentru a nu modifica semnul sumei și numere mai mari pentru a-l modifica. Pentru a clarifica faptul că acest algoritm este corect, vom prezenta câteva rezultate matematice.

Definiție

Vom numi ***secvență alternantă*** o secvență $X = (x_a, x_{a+1}, \ldots, x_b)$, $a \le b$ pentru care:

- $|x_a| < |x_{a+1}| < |x_{a+2}| < \ldots < |x_b|$ și
- pentru fiecare valoare i ($a < i \le b$), semnele valorilor x_i și x_{i-1} sunt diferite.

Lemă

Fie $X = (x_a, x_{a+1}, \ldots, x_b)$, $a \le b$ o secvență alternantă. Semnul valorii x_b este același cu semnul sumei tuturor elementelor din X.

Demonstrație

Vom presupune fără a reduce generalitatea că x_b este un număr întreg pozitiv. Dacă numărul elementelor din X ($b - a + 1$) este par, atunci sumele parțiale $x_a + x_{a+1}$, $x_{a+2} + x_{a+3}$, ..., $x_{b-1} + x_b$ sunt pozitive, deci valoarea sumei tuturor elementelor din X este pozitivă. Dacă numărul elementelor din X este impar, atunci sumele parțiale x_a, $x_{a+1} + x_{a+2}$, ..., $x_{b-1} + x_b$ sunt pozitive, deci valoarea sumei tuturor elementelor din X este din nou pozitivă.

Cazul în care valoarea x_b este negativă se tratează într-o manieră asemănătoare (toate sumele parțiale vor fi negative, deci suma tuturor elementelor va fi negativă).

Exemple

Suma elementelor din secvența alternantă $X = (-1, +2, -5, +6)$ este egală cu $(-1 + 2) + (-5 + 6) = +2$ și este pozitivă la fel ca ultimul element al secvenței.

Suma elementelor din secvența alternantă $X = (+3, -4, +5, -6, +7)$ este egală cu $(+3) + (-4 + 5) + (-6 + 7) = +5$ și este pozitivă la fel ca ultimul element al secvenței.

Fie $S = (s_a, s_{a+1}, \ldots, s_b)$, $a \le b$ o secvență de semne. O secvență $X' = (x'_a, x'_{a+1}, \ldots, x'_b)$, $a \le b$ este validă pentru S, dacă semnul sumei $\sum_{i=a}^{k} x'_i$ este s_k pentru fiecare k cuprins între a și b.

Teoremă

Fie $(x_a, x_{a+1}, \ldots, x_b)$, $a \le b$ o secvență alternantă și $S = (s_a, s_{a+1}, \ldots, s_b)$, $a \le b$ o secvență de semne. Dacă semnul valorii x_b este s_b, atunci există o secvență $X' = (x'_a, x'_{a+1}, \ldots, x'_b)$, $a \le b$ astfel încât:

- $\{x'_a, x'_{a+1}, \ldots, x'_b\} = \{x_a, x_{a+1}, \ldots, x_b\}$ și
- X' este validă pentru S.

Demonstrație

Demonstrația se realizează pe baza unei inducții matematice după numărul k al elementelor din X. Pentru $k = 1$ se observă foarte ușor că $X' = X$ este o secvență validă pentru S. Presupunem că $k \ge 2$ și notăm $S_1 = S - s_b$. Cu alte cuvinte, avem $S_1 = (s_a, s_{a+1}, \ldots, s_{b-1})$.

Distingem două cazuri. Dacă semnul valorii x_b este s_b, atunci notăm $X_1 = X - x_a$ și $t = x_a$. Dacă semnele sunt diferite atunci avem $X_1 = X - x_b$ și $t = x_b$.

Se observă că X_1 este o secvență alternantă și S_1 este o secvență de semne, astfel încât semnul ultimului element din X_1 este s_{b-1} (ultimul element din S_1). Așadar, folosind ipoteza de inducție, deducem că există o secvență X'_1

validă pentru S_1. Prin urmare, $X' = (X'_1, t)$ este o secvență validă pentru S.

Exemple

Pentru secvența alternantă $X = (-4, +5, -7, +8)$ și secvența de semne $S = (+, -, -, +)$ avem:
$S_1 = (+, -, -)$, $X_1 = (-4, +5, -7)$
$S_2 = (+, -)$, $X_2 = (+5, -7)$
$S_3 = (+)$, $X_3 = (+5)$

Ca urmare, obținem:
$X'_3 = (+5)$
$X'_2 = (+5, -7)$
$X'_1 = (+5, -7, -4)$
$X' = (+5, -7, -4, +8)$

Pentru secvența alternantă $X = (-1, +2, -3)$ și secvența de semne $S = (-, -, -)$ avem:
$S_1 = (-, -)$, $X_1 = (+2, -3)$
$S_2 = (-)$, $X_2 = (-3)$

Ca urmare, obținem:
$X'_2 = (-3)$
$X'_1 = (-3, +2)$
$X' = (-3, +2, -1)$

Acum putem prezenta pașii algoritmului de rezolvare a acestei probleme:

Pasul 1: //*citirea datelor de intrare*
- 1.1. citirea valorii N.
- 1.2. citirea celor $2 \cdot N$ numere de cod și partiționarea lor în două mulțimi A și B cu același cardinal.
- 1.3. citirea secvenței de regiuni $R = (r_1, r_2, ..., r_N)$.

Pasul 2: //*determinarea coordonatelor orizontale ale perechilor de coduri*
- 2.1. determinarea unei secvențe de semne $S = (s_1, s_2, \ldots, s_N)$ astfel încât pentru toate valorile j cuprinse între 1 și N avem s_j = '+' dacă $r_j = 1$ sau $r_j = 4$ și s_j = '-' dacă $r_j = 2$ sau $r_j = 3$.
- 2.2. determinarea unei secvențe alternante $X = (x_1, x_2, \ldots, x_N)$ construită pe baza mulțimii A, astfel încât semnul valorii x_N este s_N.
- 2.3. determinarea, pe baza teoremei anterioare, a unei secvențe $X' = (x'_1, x'_2, \ldots, x'_N)$ care este validă pentru S.

Pasul 3: //*determinarea coordonatelor verticale ale perechilor de coduri*
- 3.1. determinarea unei secvențe de semne $S = (s_1, s_2, \ldots, s_N)$ astfel încât pentru toate valorile j cuprinse între 1 și N avem s_j = '+' dacă $r_j = 1$ sau $r_j = 2$ și s_j = '-' dacă $r_j = 3$ sau $r_j = 4$
- 3.2. determinarea unei secvențe alternante $Y = (y_1, y_2, \ldots, y_N)$ construită pe baza mulțimii B, astfel încât semnul valorii y_N este s_N.
- 3.3. determinarea unei secvențe $Y' = (y'_1, y'_2, \ldots, y'_N)$ care este validă pentru S.

Pasul 4: //*scrierea datelor de ieșire*

Afișarea perechilor $(x'_1, y'_1), (x'_2, y'_2), ..., (x'_N, y'_N)$.

Teoremă

Algoritmul prezentat anterior este corect și are ordinul de complexitate $O(N \cdot \log N)$.

Demonstrație

Corectitudinea algoritmului rezultă direct din teorema anterioară. Ordinul de complexitate al tuturor pașilor (excepție 2.2 și 3.2) este $O(N)$. Pașii 2.2 și 3.2 au ordinul de complexitate $O(N \cdot \log N)$ datorită faptului că necesită efectuarea unei sortări.

Observație

Nu se cunoaște o soluție cu un ordin de complexitate mai mic, dar nici nu s-a reușit demonstrarea faptului că sortarea este neapărat necesară pentru rezolvarea acestei probleme.

P060215: Compresie XOR

Vom începe prezentarea soluției acestei probleme definind anumiți termeni pe care îi vom folosi ulterior.

Definiții

Grila corespunzătoare ecranului este grila obținută prin trasarea unor linii imaginare care separă pixelii.

Un ***punct al grilei*** este punctul de intersecție a două linii ale grilei corespunzătoare ecranului (o linie verticală și una orizontală).

Un ***colț*** este un punct al grilei care este adiacent cu un număr impar de pixeli negri. Dacă un punct p este colț, spunem că ***ic-valoarea*** sa este 1 și notăm acest fapt prin $ic(p) = 1$, iar dacă p nu este colț, spunem că *ic-valoarea* sa este 0 și $ic(p) = 0$.

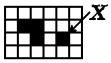

Figura 1

De exemplu, în figura 1, punctul x este adiacent cu patru pixeli dintre care 1 este negru și 3 sunt albi. Așadar, punctul x este colț. În figura 1 avem în total 10 colțuri.

Este evident că următoarea afirmație este adevărată: ***O imagine este complet albă dacă și numai dacă ea nu conține nici un colț.***

Putem acum să încercăm să găsim o soluție folosind imaginea furnizată ca intrare și încercând să obținem imaginea complet albă cu ajutorul operațiilor *XOR*. Această soluție se bazează pe următoarea lemă.

Lemă

Presupunem că efectuăm operația $XOR(L, R, T, B)$; fie Q dreptunghiul afectat de această operație. Vom nota prin a punctul din colțul stânga-sus al dreptunghiului Q, prin b punctul din colțul stânga-jos, prin c punctul din colțul dreapta-jos și prin d punctul din colțul dreapta-sus. Prin efectuarea operației $XOR(L, R, T, B)$ *ic-valorile* punctelor a, b, c și d se modifică (dacă au fost 0 devin 1 și invers), iar

ic-valorile tuturor celorlalte puncte ale grilei rămân neschimbate.

Demonstrație
În urma operației, exact un pixel adiacent fiecăruia dintre punctele *a*, *b*, *c* și *d* își schimbă culoarea. Ca urmare, *ic-valorile* acestor puncte se modifică.

Luăm acum în considerare punctele de pe laturile dreptunghiului *Q*. Pentru fiecare dintre acestea, exact doi pixeli adiacenți își schimbă culoarea (cei din interiorul dreptunghiului), deci paritatea numărului de pixeli negri din jurul acestor puncte nu se modifică.

În final considerăm punctele din interiorul dreptunghiului *Q*. Toți cei patru pixeli adiacenți cu aceste puncte își schimbă culoarea, deci paritatea numărului de pixeli negri adiacenți rămâne aceeași.

Vom încerca să reducem numărul de colțuri la fiecare pas; algoritmul se va opri când nu va mai exista nici un colț. Evident, vom încerca să reducem cât mai mult acest număr, la fiecare pas. Totuși, nu vom putea întotdeauna să efectuăm o operație *XOR* în urma căreia să se modifice *ic-valorile* tuturor colțurilor din regiunea afectată de această operație.

Lemă
Dacă imaginea nu este complet albă, este întotdeauna posibil să efectuăm o operație *XOR*, astfel încât numărul colțurilor să se reducă fie cu 2, fie cu 4.

Demonstrație
Pentru început, trebuie să observăm că există o linie pe care se află cel puțin două colțuri. Este evident că linia cea mai de sus care conține pixeli negri conține un colț aflat în partea din dreapta-sus a celui mai din dreapta pixel negru și un colț aflat în partea din stânga-sus a celui mai din stânga pixel negru. Aceste colțuri sunt distincte și se află pe aceeași linie.

În continuare, vom observa că nu este posibil ca toate colțurile să se afle pe aceeași linie. Este evident că cel mai de sus pixel negru și cel mai de jos pixel negru au în jurul lor (deasupra, respectiv dedesubt) cel puțin câte un colț care nu se află pe aceeași linie orizontală. De asemenea, cel mai din dreapta pixel negru și cel mai din stânga pixel negru au în jurul lor (la dreapta, respectiv la stânga) cel puțin câte un colț care nu se află pe aceeași linie verticală.

Vom putea alege acum un dreptunghi determinat de trei colțuri. Mai întâi alegem cel mai din dreapta colț de pe cea mai de sus linie și apoi un alt colț care se află pe aceeași linie verticală. Cel de-al treilea colț poate fi determinat astfel: începem de la primul colț ales; acesta are un singur pixel adiacent negru (cel din stânga-jos), ceilalți fiind albi; parcurgem punctele aflate la stânga pe aceeași linie până în momentul când pixelul aflat la stânga și sub punctul curent devine alb. Aceste trei colțuri determină dreptunghiul asupra căruia vom efectua operația *XOR*.

Așadar, cel puțin trei dintre punctele care delimitează dreptunghiul sunt colțuri. Dacă toate cele patru puncte sunt colțuri, atunci în urma operației numărul total al colțurilor se reduce cu 4. Dacă cel de-al patrulea punct nu este colț, numărul total al colțurilor se reduce cu 2 (se elimină trei colțuri și se creează unu).

Așadar, un algoritm de rezolvare al acestei probleme constă în alegerea succesivă a trei colțuri care determină un dreptunghi și efectuarea de operații *XOR* asupra acestor dreptunghiuri până în momentul în care nu mai rămâne nici un pixel negru în imagine.

Teoremă
Algoritmul prezentat anterior este corect și numărul operațiilor *XOR* care trebuie efectuate este de cel mult două ori mai mare decât numărul minim de operații *XOR* care trebuie efectuate pentru eliminarea tuturor pixelilor negri.

Demonstrație
Lema anterioară ne asigură că la fiecare pas vom găsi 3 colțuri care determină un dreptunghi. Datorită faptului că numărul inițial al colțurilor este finit și la fiecare pas acesta se reduce, după un număr finit de pași imaginea nu va mai conține nici un colț, deci va fi complet albă.

În cazul ideal, pentru fiecare operație *XOR*, numărul colțurilor se reduce cu 4. În cazul algoritmului nostru, numărul colțurilor se reduce cu cel puțin 2. Așadar, în cel mai defavorabil caz, vom elimina câte două colțuri la fiecare pas. Ca urmare, numărul pașilor va fi de cel mult două ori mai mare față de cazul ideal.

Vom prezenta în continuare o versiune a acestui algoritm:

```
pentru fiecare linie i execută
  pentru fiecare coloană j execută
    dacă colț(i, j) atunci
      fie k cel mai apropiat colț aflat pe linia i la dreapta față de punctul (i, j)
      fie l cel mai apropiat colț aflat pe coloana j sub punctul (i, j)
      xor(j, k-1, i, l+1)
    sfârșit dacă
  sfârșit pentru
sfârșit pentru
```

Folosind acest algoritm se puteau obține aproximativ 85 de puncte la această problemă.

Problema poate fi abordată și folosind tehnici euristice. Una dintre acestea constă în determinarea, la fiecare pas, a unui dreptunghi care conține doar pixeli negri și efectuarea operației *XOR* asupra dreptunghiului respectiv. Punctajul obținut în acest caz este de aproximativ 35 de puncte.

O a treia abordare folosește un algoritm de determinare a unui cuplaj maxim într-un graf. Se iau în considerare

toate colțurile de pe o linie, fiecare devenind un nod al grafului. Costul unei muchii dintre două vârfuri este 1 dacă și numai dacă există un dreptunghi determinat de cele două colțuri corespunzătoare celor două vârfuri și alte două colțuri din imagine. În caz contrar, costul muchiei este 0. Se determină un cuplaj maxim în graful construit și se aplică operații *XOR* asupra tuturor dreptunghiurilor determinate de muchiile care apar în acest cuplaj. Operația continuă până în momentul în care au fost eliminați toți pixelii. Această abordare ar fi dus la obținerea tuturor celor 100 de puncte (mai exact 99.7, care prin rotunjire devine 100).

Pentru unul singur (ultimul) dintre cele zece teste prima metodă duce la obținerea unui rezultat mai bun decât a treia.

P060216: Programarea pachetelor

O posibilitate de a aborda această problemă este folosirea *programării dinamice*. Vom nota prin C_i costul minim de partiționare a operațiilor J_i, J_{i+1}, ..., J_n în pachete și prin $C_i(k)$ costul minim de partiționare ale acestor operații, dacă primul pachet este format din operațiile $J_i, J_{i+1}, ..., J_k$. Vom obține următoarea relație:

$C_i(k) = C_k + (S + T_i + T_{i+1} + \ldots + T_{k-1}) \cdot (F_i + F_{i+1} + \ldots + F_n)$.

Pe baza acestei relații deducem următoarea formulă de calcul pentru determinarea valorilor C_i:

$C_i = \min\{C_i(k) \mid k = i + 1, \ldots, n + 1\}$ pentru $1 \le i \le n$ și $C_{n+1} = 0$.

Este evident că ordinul de complexitate al acestui algoritm este $O(n^2)$, deoarece trebuie calculate valorile $C_i(k)$ pentru toate valorile i și k astfel încât $1 \le i \le k \le n$).

Studiind valorile $C_i(k)$ se poate deduce că această problemă se poate rezolva în timp liniar după cum se va arăta în continuare.

Din relația $C_i(k) = C_k + (S + T_i + T_{i+1} + \ldots + T_{k-1}) \cdot (F_i + F_{i+1} + \ldots + F_n)$ deducem că, pentru $i < k < l$ avem:

$C_i(k) < C_i(l) \Leftrightarrow$

$\Leftrightarrow C_l - C_k + (T_i + T_{i+1} + \ldots + T_{k-1}) \cdot (F_i + F_{i+1} + \ldots + F_n) \ge 0$

$\Leftrightarrow (C_k - C_l) / (T_i + T_{i+1} + \ldots + T_{k-1}) \le (F_i + F_{i+1} + \ldots + F_n)$.

Vom nota prin $g(k, l)$ valoarea $(C_k - C_l) / (T_i + T_{i+1} + \ldots + T_{k-1})$ și prin $f(i)$ valoarea $(F_i + F_{i+1} + \ldots + F_n)$.

Proprietăți

- Dacă $g(k, l) \le f(i)$ pentru $1 \le i < k < l$, atunci $C_i(k) \le C_i(l)$.
- Dacă $g(j, k) \le g(k, l)$ pentru $1 \le i < k < l$, atunci pentru toate valorile i cuprinse între 1 și j - 1 avem $C_i(j) \le C_i(k)$ sau $C_i(l) \le C_i(k)$.

Din cea de-a doua proprietate rezultă că dacă avem $g(j, k) \le g(k, l)$ pentru $1 \le i < k < l \le n$, atunci valoarea $C_i(k)$ nu este necesară pentru a calcula valoarea C_i.

Vom calcula valorile C_i în ordinea C_n, C_{n-1}, ..., C_1. Vom folosi o structură $Q = (i_r, i_{r-1}, \ldots, i_2, i_1)$ asemănătoare unei cozi, ultimul element introdus (coada) fiind i_r, iar primul (capul) fiind i_1. Singura diferență față de o coadă "clasică" este faptul că pot fi eliminate elemente din ambele capete ale cozii. Această structură respectă următoarele proprietăți:

- $i_r < i_{r-1} < \ldots < i_2 < i_1$ și
- $g(i_r, i_{r-1}) > g(i_{r-1}, i_{r-2}) > \ldots > g(i_2, i_1)$.

În momentul calculării unei valori C_i va trebui, mai întâi, să eliminăm elementele care nu sunt necesare din capul cozii Q. Pentru aceasta vom folosi valoarea $f(i)$ și prima proprietate enunțată anterior. Dacă $f(i) \ge g(i_2, i_1)$ vom elimina din coadă elementul i_1, deoarece din proprietatea amintită rezultă că pentru toate valorile $h \le i$, avem $f(h) \ge f(i) \ge g(i_2, i_1)$, deci $C_h(i_2) \le C_h(i_1)$. Vom continua să eliminăm elemente până ajungem în situația în care, pentru o anumită valoare $t \ge 1$, avem $g(i_r, i_{r-1}) > g(i_{r-1}, i_{r-2}) > \ldots > g(i_{t+1}, i_t) > f(i)$. Acum, pe baza aceleiași proprietăți, deducem că $C_i(i_{v+1}) > C_i(i_v)$ pentru toate valorile v cuprinse între t și r - 1, sau $r = t$ (caz în care în coadă a rămas un singur element: i_r). Rezultă imediat că:

$C_i(i_t) = \min\{C_i(k) \mid k = i + 1, \ldots, n + 1\}$.

În momentul inserării elementului i în coadă, trebuie efectuate operațiile necesare pentru ca cele două proprietăți ale acesteia să fie respectate. Dacă avem $g(i, i_r) \le g(i_r, i_{r-1})$, atunci din cea de-a doua proprietate enunțată anterior, rezultă că elementul i_r trebuie eliminat. Vom continua să eliminăm elemente până ajungem în situația în care $g(i, i_r) > g(i_r, i_{r-1})$. În acest moment i este adăugat în coadă (devine ultimul element al acesteia).

Fiecare valoare i este inserată și eliminată din coadă cel mult o dată. Pentru fiecare inserare și pentru fiecare eliminare operațiile necesare sunt realizate în timp constant. Așadar, ordinul de complexitate al algoritmului prezentat este $O(n)$.

P060217: Stații de autobuz

Soluția acestei probleme se bazează pe un algoritm al cărui ordine de complexitate este $O(n^3)$. ***Diametrul*** unei rețele de autobuze va fi dat de cel mai lung traseu dintre două stații ale rețelei. Va trebui să găsim cel mai mic diametru posibil, considerarând toate posibilitățile de a alege stațiile centrale. Vom lua în considerare două cazuri și, pentru început, vom introduce câteva notații:

- Fie D_1 valoarea minimă a celui mai lung traseu dintre două stații care sunt legate prin intermediul unei singure stații centrale, ținând cont de toate posibilitățile de a alege stația centrală respectivă.
- Fie D_2 valoarea minimă a celui mai lung traseu dintre două stații care sunt legate prin intermediul ambelor stații centrale, luând în considerare toate posibilitățile de a alege cele două stații centrale.

Vom determina valorile D_1 și D_2 și vom furniza la ieșire minimul dintre aceste două valori.

Vom descrie acum modul în care poate fi calculată valoarea D_1. Dacă stația centrală se află într-un punct p, atunci cel mai lung traseu care trece prin p este determinat de cele mai depărtate două stații față de p. Dacă aceste două stații se află în punctele q și r, atunci vom avea D_1 =

$d(p, q) + d(p, r)$, unde $d(x, y)$ reprezintă distanța dintre punctele x și y. Vom lua în considerare toate cele n posibilități de a alege punctul p și vom alege acel punct pentru care suma distanțelor de la el la cele mai depărtate două stații față de punctul respectiv este minimă. Cele mai depărtate două stații față de un punct pot fi determinate în timp liniar, deci, datorită faptului că vom efectua n astfel de determinări, ordinul de complexitate al acestei operații este $O(n^2)$.

Pentru a ilustra modul în care se poate calcula valoarea D_2 vom folosi un exemplu simplu. Trebuie observat faptul că, în acest caz, cel mai lung traseu dintre două stații va trece prin ambele stații centrale.

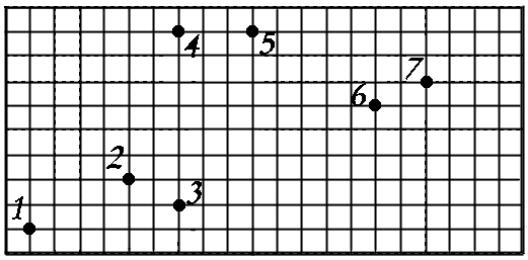

Figura 1

În figura 1 este prezentată o rețea cu șapte stații. Vom lua în considerare toate posibilitățile de a alege stațiile centrale și vom reține perechea pentru care diametrul rețelei este minim. Pentru început vom considera că avem $D_2 = \infty$. Pentru fiecare pereche de stații centrale (H_1 și H_2) vom considera că acestea sunt fixate. Celelalte n - 2 puncte (stații), vor fi inițial legate la una dintre cele două stații centrale (de exemplu la H_1).

Vom sorta aceste n - 2 puncte în ordine crescătoare în funcție de distanța față de H_1, rezultatul sortării fiind păstrat într-un vector P ($P[1]$ va fi cea mai apropiată stație față de H_1, iar $P[n - 2]$ va fi cea mai depărtată). Rezultatul sortării este ilustrat în figura 2.

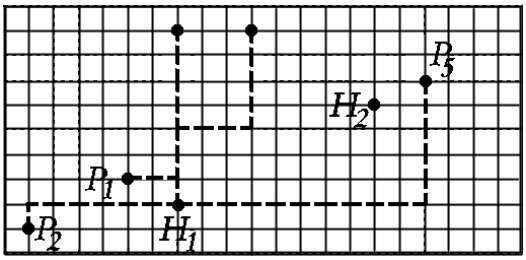

Figura 2

Vom nota prin r_1 valoarea $d(H_1, P[n - 3])$, prin r_2 valoarea $d(H_2, P[n - 2])$ și prin d_{12} valoarea $d(H_1, H_2)$. Dacă avem $r_1 + r_2 + d_{12} < D_2$, atunci punctul $P[n - 2]$ va fi legat de stația H_2 și valoarea D_2 va deveni $D_2 = r_1 + r_2 + d_{12}$. Acest pas este ilustrat în figura 3.

Avem $r_1 = d(H_1, P[n - 3]) = d(H_1, P[4]) = 10$, $r_2 = d(H_2, P[n - 2]) = d(H_2, P[5]) = 3$, $d_{12} = d(H_1, H_2) = 12$, deci $D_2 = r_1 + d_{12} + r_2 = 10 + 12 + 3 = 25$.

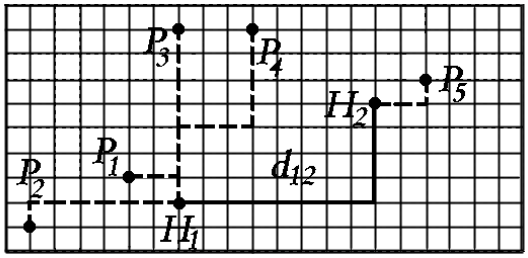

Figura 3

În continuare, aplicăm aceeași metodă pentru $r_1 = d(H_1, P[n-4])$, $r_2 = d(H_2, P[n - 3])$ și aceeași valoare $d_{12} = d(H_1, H_2)$ și obținem $r_1 + r_2 + d_{12} = d(H_1, P[3]) + d(H_1, H_2) + d(H_2, P[4]) = 7 + 12 + 8 = 27$. Deoarece noua distanță este mai mare decât diametrul anterior, valoarea D_2 rămâne nemodificată. Dacă, la final, valoarea D_2 va rămâne 25, atunci punctul $P[4]$ va fi legat de H_1 cu toate că distanța sa față de H_2 este mai mică decât distanța față de H_1. Situația este ilustrată în figura 4, în care punctul $P[4]$ este legat printr-o linie îngroșată de H_2 deoarece distanță față de H_1 este mai mare.

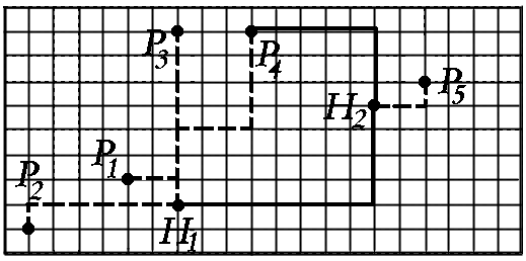

Figura 4

Algoritmul continuă luându-se în considerare celelalte elemente din P (în ordine descrescătoare a indicilor) până în momentul în care indicele devine 1. În figura 5 este ilustrată situația rețelei la final; valoarea D_2 a rămas 25.

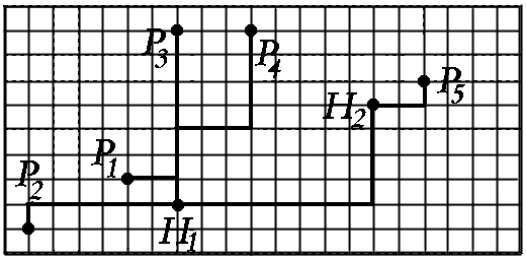

Figura 5

Pentru a obține ordinul de complexitate $O(n^3)$, trebuie respectată următoarea ordine de efectuare a operațiilor.

- Se iau în considerare toate punctele p ca fiind prima stație centrală.
- Se sortează celelalte n-1 puncte în funcție de distanța față de p.
- Se iau în considerare toate punctele q ca fiind a doua stație centrală.
- Se aplică algoritmul de determinare a diametrului pentru stațiile centrale p și q.

Dacă se efectuează sortarea numai după alegerea ambelor stații centrale, atunci ordinul de complexitate devine $O(n^3 \cdot \log n)$.

P060218: Două bețe

Cea mai rapidă metodă de rezolvare a acestei probleme implică efectuarea a cinci sau șase căutări binare. Cu ajutorul primelor patru căutări binare vom determina cel mai mic dreptunghi care conține în interiorul său ambele bețe.

Pentru prima căutare binară vom apela funcția `rect` pentru dreptunghiuri delimitate în stânga, dreapta și dedesubt de marginile grilei. Latura de sus a dreptunghiului va fi "variabilă" și, în urma căutării binare, va fi linia pe care se află cea mai de jos celulă care face parte dintr-un băț. Inițial, latura de sus va fi la jumătatea grilei. Dacă în urma apelului se returnează valoarea 0, atunci înseamnă că nu există în jumătatea de jos nici o celulă care face parte dintr-un băț și căutarea continuă în jumătatea de sus. Dacă se returnează valoarea 1, înseamnă că în jumătatea de jos există cel puțin o celulă care face parte dintr-un băț și căutarea continuă în această jumătate. Regiunea determinată se împarte din nou în două subregiuni și algoritmul continuă până în momentul în care ajungem la regiuni formate dintr-o singură linie.

Cu ajutorul celorlalte trei căutări binare vom determina celelalte trei laturi ale dreptunghiului căutat.

Acum, prin patru apeluri ale funcției `rect` (asupra celor patru colțuri ale dreptunghiului; dreptunghiurile pentru care se realizează apelul vor avea lungimea și lățimea egală cu unitatea, deci vor fi pătrate de latură 1) putem determina forma generală a structurii. În funcție de numărul și poziția colțurilor care conțin celule ocupate de bețe, deosebim patru cazuri (există și alte variante care pot fi obținute prin rotații):

- bețele ocupă trei dintre colțurile dreptunghiului;
- bețele ocupă două colțuri opuse (în diagonală) ale dreptunghiului;
- bețele ocupă două colțuri alăturate (pe aceeași latură) ale dreptunghiului;
- bețele nu ocupă nici unul dintre colțurile dreptunghiului.

Cele patru posibilități sunt ilustrate în figura 1 .

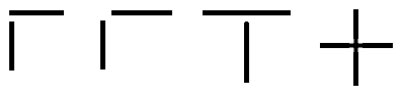

Figura 1

Pentru a determina poziția exactă a bețelor mai avem nevoie de una sau două căutări binare în funcție de forma generală a structurii.

Datorită faptului că se realizează un număr finit de căutări binare, ordinul de complexitate al algoritmului este $O(\log N)$, unde N este latura grilei; numărul de apeluri ale funcției `rect` are același ordin. Mai exact, vom efectua cel mult $6 \cdot \lceil log_2 N \rceil + 4$ apeluri.

Pentru valoarea maximă $N = 10000$ vom avea cel mult $6 \cdot \lceil \log_2 10000 \rceil + 4 = 6 \cdot 14 + 4 = 88$ de apeluri, deci întotdeauna va fi determinată soluția folosindu-se mai puțin de 100 de apeluri, ceea ce permite obținerea punctajului maxim.

Bibliografie

1. IOI 2002 Scientific Committee, *IOI 2002 Competition*,
2. A.B. Kahng, G. Robbins, *Optimal Algorithms Extracting Spatial Regularity in Images*, *Pattern Recognition Letters*, 12, p. 757-764, 1991
3. P. Brucker, *Efficient Algorithm for some Path Problems*, *Discrete Applied Mathematics*, 62, p. 77-85, 1995
4. S. Albers, P. Brucker, *The Complexity of One-machine Batching Problems*, *Discrete Applied Mathematics*, 47, p. 87-107, 1993
5. J. M. Ho, D. T. Lee, C. H. Chang, C. K. Wong, *Minimum Diameter Spanning Trees and Related Problems* , *SIAM J. on Computing*, 20(5), p. 987-997, 1991
6. T. Chan, *Semi-online Maintenance of Geometric Optima and Measures*, 13th *ACM-SODA*, p. 474-483, 2002

Despre IOI 2002...

La IOI 2002 concurenții au avut la dispoziție calculatoare dotate cu procesoare Intel Pentium 4 cu frecvența de 1,7 GHz și 256 MB de memorie RAM. În timpul concursului putea fi folosit fie sistemul de operare Microsoft Windows XP, fie Debian GNU/Linux 3.0.

Compilatoarele oficiale la IOI 2002 au fost GCC 2.95.3 și FPC 1.0.6. Mediile de programare puse la dispoziția concurenților au fost FreePascal IDE și RHIDE.

Au fost două zile de concurs în care participanții au avut la dispoziție cinci ore pentru a rezolva cele trei probleme propuse.

În ziua premergătoare începerii concursului propriu-zis a avut loc o sesiune de antrenament care avea ca scop familiarizarea concurenților cu modul în care se va desfășura concursul.

Mai multe detalii referitoare la IOI 2002 puteți găsi la adresa www.ioi2002.or.kr.