Fantome

Această problemă a fost rezolvată corect de către 43 dintre participanții la cea de-a treia rundă a ***Concursului de Programare Agora***. O rezolvare pentru o problemă echivalentă a fost publicată în numărul 2/1994 al *Gazetei de Informatică*.

Enunțul problemei echivalente era următorul: *O placă dreptunghiulară de dimensiuni* $L1 \times L2$ ($L1 \leq 600$, $L2 \leq 300$) *este împărțită într-o rețea ortogonală de pătrate de latură* 1, *în care sunt marcate* N *puncte albe și* N *puncte negre, toate având coordonate întregi. Oricare trei dintre cele* $2 \cdot N$ *puncte nu sunt coliniare. În scopul obținerii unui circuit imprimat prin corodarea plăcii, fiecare punct alb urmează a fi unit printr-un segment de dreaptă cu un punct negru oarecare, dar astfel încât oricare două dintre cele* N *segmente să nu se intersecteze*.

Singurele diferențe, care au apărut în cazul problemei propuse la runda a treia, sunt reprezentate de stabilirea dimensiunilor tablei la 1.000×1.000 și posibilitatea ca punctele să aibă coordonate reale cu trei zecimale. Putem înmulți fiecare valoare cu 1.000 și obținem o placă de dimensiuni $1.000.000 \times 1.000.000$.

În *Gazeta de Informatică* 2/1994 erau prezentate două soluții pentru rezolvarea problemei. Prima se baza pe metoda *backtracking*: se încearcă o conectare a punctului alb (vânătorului) curent cu un punct negru (fantomă) neconectat încă, astfel încât segmentul (raza) format(ă) să nu se intersecteze cu nici unul dintre segmentele (razele) deja formate. Este evident că o astfel de soluție nu se va încadra în limita de timp admisă, așadar nu vom mai insista asupra ei.

Cea de-a doua soluție folosește un algoritm care evită metoda *backtracking*. Pentru început vom conecta punctele într-un anumit mod. De exemplu, am putea conecta punctele (vânătorii și fantomele) în ordinea dată la intrare: primul punct alb (vânător) cu primul punct negru (fantomă) etc.

Vom indentifica prin V_i punctele albe (vânătorii) și cu F_i punctele negre (fantomele). Vom verifica apoi dacă există două segmente V_1F_1 și V_2F_2 care se intersectează. În acest caz conexiunile se vor modifica și noile segmente vor fi V_1F_2 și V_2F_1. Este evident că aceste segmente nu se vor mai intersecta.

Vom demonstra în continuare că, prin schimbarea conexiunilor, suma lungimilor celor două segmente se reduce. Vom nota cu O intersecția segmentelor V_1F_1 și V_2F_2. Știm că într-un triunghi lungimea unei laturi este întotdeauna mai mică decât suma lungimilor celorlalte două. Așadar, în triunghiul V_1OF_2 avem $V_1F_2 < V_1O + OF_2$, iar în triunghiul V_2OF_1 avem $V_2F_1 < V_2O + OF_1$. Adunând cele două inegalități obținem $V_1F_2 + V_2F_1 < V_1O + OF_1 + V_2O + OF_2$, adică $V_1F_2 + V_2F_1 < V_1F_1 + V_2F_2$.

Așadar, după fiecare modificare a conexiunilor suma totală a lungimilor segmentelor se reduce. Deoarece această sumă nu va fi niciodată negativă, înseamnă că după un număr finit de pași nu vom mai avea intersecții.

Se conturează astfel următorul algoritm de rezolvare a problemei: *atâta timp cât există două segmente care se intersectează se modifică conexiunile pentru punctele corespunzătoare*.

Aceasta a fost soluția prezentată de majoritatea celor care au obținut punctaj maxim la această problemă. Acest lucru s-a datorat faptului că redacția a considerat că o rezolvare corectă care nu folosește metoda *backtracking* merită 100 de puncte. Totuși, puteau fi găsite destul de ușor date de test pentru care programele să nu se încadreze în limita de timp admisă. Ordinul de complexitate al algoritmului prezentat anterior este liniar în suma lungimilor segmentelor, dar nu este polinomial în numărul de puncte, deoarece după modificarea conexiunilor pentru două segmente, segmentele nou formate se pot intersecta cu alte segmente, deci numărul de intersecții nu se reduce neapărat, ci numai suma totală a lungimilor.

Enunțul problemei, așa cum a fost el publicat în numărul 10/8 al *GInfo* a fost preluat din cartea ***Introducere în algoritmi***. Problema apare la sfârșitul capitolului 35 (*Geometrie computațională*) - pagina 783. În ***CLR*** (aceasta este denumirea folosită pe plan internațional pentru cartea ***Introducere în algoritmi***; denumirea provine de la inițialele celor trei autori: ***C****ormen*, ***L****eiserson*, ***R****ivest*) apar următoarele indicații pentru rezolvarea problemei:

- *Demonstrați că există o linie care trece printr-un vânător de fantome și printr-o fantomă, astfel încât numărul vânătorilor de fantome aflați de o parte a liniei este egal cu numărul fantomelor aflate de cealaltă parte. Descrieți cum se poate determina o astfel de linie într-un timp* $O(n \cdot \log n)$.
- *Se cere un algoritm de timp* $O(n^2 \cdot \log n)$ *care să echivaleze numărul vânătorilor de fantome cu numărul fantomelor în așa fel încât să nu se intersecteze nici o rază.*

Să prespunem că am reușit să determinăm linia cerută la prima indicație. Am putea acorda un premiu celui care

va găsi un algoritm de rezolvare folosind astfel de linii. Noi nu am reușit.

Din păcate, varianta în limba română a ***CLR*** conține o gravă eroare de traducere care probabil că a creat confuzii pentru mulți dintre cei care au încercat să rezolve această problemă bazându-se pe aceste indicații. Traducerea corectă a primei indicații este următoarea: *demonstrați că există o linie care trece printr-un vânător de fantome și printr-o fantomă, astfel încât numărul vânătorilor de fantome aflați de o parte a liniei este egal cu numărul fantomelor aflate de* ***aceeași*** *parte*.

Acum este destul de simplu să găsim un algoritm, dacă am determinat o astfel de dreaptă. Pentru rezolvare vom folosi metoda *divide et impera*. Dreapta va împărți vânătorii și fantomele în două mulțimi fiecare dintre ele conținând un număr egal de vânători și fantome. Așadar, același algoritm poate fi aplicat pentru cele două mulțimi fără a exista pericolul ca vreun segment trasat în una dintre mulțimi să se intersecteze cu un segment trasat în cealaltă mulțime sau cu segmentul care unește cele două puncte care determină dreapta care separă mulțimile.

Problema mai dificilă este determinare dreptei care are proprietatea cerută. Ordinul de complexitate $O(n \cdot \log n)$ indicat arată că este foarte probabil să folosim o anumită sortare pentru a determina acea dreaptă. Va trebui să folosim un algoritm performant cum ar fi *heapsort* sau *mergesort* pentru a ne încadra în ordinul de complexitate cerut.

Vom considera punctul `P` având coordonata `y` cea mai mică, indiferent dacă reprezintă poziția unui vânător sau a unei fantome.

Vom sorta toate celelalte puncte în funcție de unghiul format de dreapta care unește `P` de acest punct și dreapta orizontală care trece prin `P`.

Vom avea doi indicatori `V` și `F`, primul va indica numărul vânătorilor luați în considerare, iar al doilea numărul fantomelor.

Dacă punctul `P` reprezintă o fantomă atunci inițializăm `V` cu `0` și `F` cu `1`, iar dacă reprezintă un vânător inițializăm `V` cu `1` și `F` cu `0`.

Vom considera pe rând punctele sortate; dacă punctul curent reprezintă un vânător incrementăm indicatorul `V`, iar dacă reprezintă o fantomă incrementăm indicatorul `F`.

În momentul în care vom avea `V = F` vom ști că sub dreapta care unește punctul `P` cu punctul curent se află un număr egal de vânători și fantome.

Vom demonstra în continuare că, la un moment dat, vom ajunge în situația în care `V = F`. Presupunem, fără a restrânge generalitatea, că în punctul `P` se află un vânător.

Dacă în primul punct se află o fantomă atunci ajungem în situația `V = F = 1`, deci am determinat deja o dreaptă cu proprietatea cerută.

Dacă în ultimul punct se află o fantomă atunci, fie am găsit o dreaptă anterior, fie acest ultim punct și punctul `P` determină dreapta pe care o căutăm.

Așadar, în cazul în care primul sau ultimul punct reprezintă o fantomă este rezolvat.

Să presupunem acum că atât primul, cât și ultimul punct reprezintă un vânător. În acest caz inițial vom avea `V = 1` și `F = 0` (`V - F = 1`), iar înaintea considerării ultimului punct `V = n - 1` și `F = n` (`V - F = -1`).

Deoarece valoarea `V - F` nu se poate modifica decât cu `1` la fiecare pas, pentru a ajunge de la `1` la `-1` ea trebuie neapărat să treacă și prin `0`. Așadar, la un moment dat ne vom afla în situația pentru care `V = F`, deci vom avea dreapta cerută.

Datorită faptului că după sortare nu se face decât o singură parcurgere (într-un timp $O(n)$) a punctelor, ordinul de complexitate rămâne $O(n \cdot \log n)$.

În total trebuie să determinăm n astfel de drepte, deci ordinul de complexitate al algoritmului devine $O(n^2 \cdot \log n)$.

La implementare am folosit numere de la `0` la `n - 1` pentru a identifica vânătorii și numere de la `n` la `2·n - 1` pentru a identifica fantomele.

Să presupunem că în sistemul de coordonate cu originea în `P`, avem două puncte `A` și `B` de coordonate (x_1, y_1) și (x_2, y_2). Pentru a sorta punctele în funcție de unghi avem trei cazuri:

- Dacă punctele `A` și `B` se află în primul cadran al sistemului de coordonate cu originea în `P`, atunci punctul `A` se află înaintea punctului `B` în șirul sortat dacă și numai dacă $y_1/x_1 < y_2/x_2$, adică $y_1 \cdot x_2 < y_2 \cdot x_1$, deoarece coordonatele orizontale sunt pozitive. (Am înlocuit împărțirile cu înmulțiri pentru a mări viteza de execuție și a evita împărțirea cu zero.)
- Dacă punctele `A` și `B` se află în cadrane diferite, atunci punctul din primul cadran îl va precede pe cel din al doilea cadran.
- Dacă punctele `A` și `B` se află în al doilea cadran al sistemului de coordonate cu originea în `P`, atunci punctul `A` se află înaintea punctului `B` în șirul sortat dacă și numai dacă $y_1/x_1 < y_2/x_2$, adică $y_1 \cdot x_2 > y_2 \cdot x_1$, deoarece coordonatele orizontale sunt negative.

Datorită faptului că am ales punctul `P` având coordonata `y` minimă, punctele `A` și `B` nu se pot afla în al treilea sau al patrulera cadran.

Vă prezentăm în continuare versiunea oficială pentru rezolvarea acestei probleme.

Listing GHOSTS.CPP

```
#include <stdio.h>

float x[1000],y[1000];
   // coordonatele vanatorilor si fantomelor
int n, // numarul de vanatori si fantome
    pair[500], // pair[i] va indica fantoma
               // ucisa de vanatorul i
    p[1000]; // sir de indici folosit pentru
             // sortare si prelucrare

void ReadData(void){
  FILE *f=fopen("GHOSTS.IN","rt");
```

```
  fscanf(f,"%d",&n);
  for (int i=0;i<n<<1;i++)
    fscanf(f,"%f%f",&x[i],&y[i]);
  fclose(f);
}

int FindMinimum(int left,int right){
// determina punctul din multumimea curenta
// care are coordonata y minima
  int ind=left;
  float min=y[p[left]];
  for (int i=left+1;i<=right;i++)
    if (min>y[p[i]])
      min=y[p[ind=i]];
  return ind;
}

int HigherAngle(int left,int p1,int p2){
// functia folosita pentru a stabili relatia
// de ordine dintre doua puncte
  if (x[p[p1]]-x[p[left]]>=0 &&
      x[p[p2]]-x[p[left]]>=0 &&
      (x[p[p1]]-x[p[left]])*
      (y[p[p2]]-y[p[left]])<
      (x[p[p2]]-x[p[left]])*
      (y[p[p1]]-y[p[left]]))
    return 1; // puncte in primul cadran
  if (x[p[p1]]-x[p[left]]>=0 &&
      x[p[p2]]-x[p[left]]>=0 &&
      (x[p[p1]]-x[p[left]])*
      (y[p[p2]]-y[p[left]])<
      (x[p[p2]]-x[p[left]])*
      (y[p[p1]]-y[p[left]]))
    return 1; // puncte in al doilea cadran
  if (x[p[p1]]-x[p[left]]<=0 &&
      x[p[p2]]-x[p[left]]>=0)
    return 1; // puncte din cadrane diferite
  return 0;
}

void Sift(int left,int right,int k){
// functia pentru scufundarea unui element
// intr-un heap
  int son;
  if ((k<<1)+left<=right){
    son=k<<1;
    if (((k<<1)+left)<right &&
        HigherAngle(left,left+1+(k<<1),
                    left+(k<<1)))
      son++;
    if (HigherAngle(left,k+left,son+left))
      son=0;
  }
  else
    son=0;
```

```
  while (son){
    int aux=p[k+left];
    p[k+left]=p[son+left];
    p[son+left]=aux;
    k=son;
    if ((k<<1)+left<=right){
      son=k<<1;
      if (((k<<1)+left)<right &&
        HigherAngle(left,left+1+(k<<1),
                    left+(k<<1)))
        son++;
      if (HigherAngle(left,k+left,son+left))
        son=0;
    }
    else
      son=0;
  }
}

void HeapSort(int left,int right){
// crearea unui heap pe baza unui vector
  for (int i=(right-left)>>1;i;i--)
    Sift(left,right,i);
// sortarea heap-ului
  for (i=right-left;i>=2;i--){
    int aux=p[left+1];
    p[left+1]=p[left+i];
    p[left+i]=aux;
    Sift(left,left+i-1,1);
  }
}

int FindLine(int left,int right){
// determina punctul cu care trebuie uni
// punctul de coordonata y minima pentru a
// determina o dreapta astfel incat numarul
// de vanatori care se afla de o parte a
// dreptei sa fie egal cu numarul de fantome
// care se afla de aceeasi parte a dreptei
  int v=0,f=0;
  for (int i=left;;i++){
    (p[i]<n)?v++:f++;
    if (v==f)
      return i;
  }
}

void DivideEtImpera(int left,int right){
// functia care implementeaza metoda divide
// et impera folosita pentru rezolvarea
// problemei
  if (left>=right)
    return; // multimea este vida
  // se alege punctul de coordonata y minima
  int ind=FindMinimum(left,right);
```

```
  // punctul determinat este mutat pe prima
  // pozitie, iar punctul care se afla pe
  // aceasta pozitie este mutat in locul
  // ramas liber
  float aux=p[ind];
  p[ind]=p[left];
  p[left]=aux;
  // se sorteaza celelalte puncte in functie
  // de unghiul format de dreapta care le
  // uneste de punctul ales si dreapta
  // orizontala acre trece prin punctul ales
  HeapSort(left,right);
  // se determina linia care va imparti
  // multimea de puncte in doua multimi
  ind=FindLine(left,right);
  // se testeaza daca punctul ales reprezinta
  // pozitia unui vanator sau a unei fantome
  // si se realizeaza corespondenta, adica
  // este aleasa fantoma care va fi ucisa de
  // vanator
  if (p[ind]<n)
    pair[p[ind]]=p[left];
  else pair[p[left]]=p[ind];
  // se va folosi acelasi algoritm pentru
  // cele doua multimi de puncte in care a
  // fost impartita multimea initiala;
  // pentru aceasta se va autoapela functia
  // care implementeaza metoda divide et
  // impera
  DivideEtImpera(left+1,ind-1);
  DivideEtImpera(ind+1,right);
}

void KillGhosts(void){
  // se initializeaza sirul de indici
  for (int i=0;i<n<<1;i++)
    p[i]=i;
  // se apeleaza functia care implementeaza
  // metoda divide et impera pentru multimea
  // initiala de puncte
  DivideEtImpera(0,(n<<1)-1);
}

void WriteSolution(void){
  FILE *f=fopen("GHOSTS.OUT","wt");
  for (int i=0;i<n;i++)
    fprintf(f,"%d\n",pair[i]-n+1);
  fclose(f);
}

void main(void){
  ReadData();
  KillGhosts();
  WriteSolution();
}
```

Au rezolvat corect:

Marius Andrei, Bucureşti
Mugurel Ionuţ Andreica, Bucureşti
Liviu-Cosmin Andreicuţ, Bucureşti
Cristian Baicoianu, Ploieşti
Ciprian Baicu, Drobeta Turnu Severin
Livia Bana, Drobeta Turnu Severin
Valentin Bişa, Lugoj
Denis Bogdănaş, Iaşi
Vencel Bors, Oradea
Ioana-Violeta Brutaru, Hunedoara
Ciprian Cană, Vaslui
Adrian Cârcu, Bistriţa
Horea Coroiu, Cluj-Napoca
Daniel Crişan, Râmnicu Vâlcea
Vlad Dascălu, Bacău
Andrei David, Râmnicu Vâlcea
Laurent Demonet, Franţa
Radu Dondera, Bucureşti
George Drumea, Bucureşti
Vincent Groenhuis, Olanda
Ioana-Maria Ileană, Alba Iulia
Liviu Lalescu, Craiova
Maximilian Machedon, Bucureşti
Paul Marinescu, Buzău
Cosmin-Silvestru Negruseri, Bistriţa
Mihai-Vlad Pantiş, Cluj-Napoca
Gabriel Pârvan, Râmnicu Sărat
Mihai Pătraşcu, Craiova
Bogdan-Milovan Piloga, Timişoara
Mohammad Ali Safari Ghahsareh, Iran
Bogdan Stan, Câmpina
Mihai Stroe, Bucureşti
Radu Ştefan, Braşov
Radu Vătavu, Suceava
Victor Vernescu, Constanţa