Задача А3. ОБИКОЛКА

Известният специалист по алгоритми проф. Станчев, е от най-канените лектори в ученическите школи, особено в дните преди кръг на националната олимпиада. И тази година проф. Станчев е получил покани от $N-1$ града, номерирани с числата от 2 до N. Сега пред него стои следната задача: как да направи така, че като тръгне от родния си град, номериран с 1, да мине през всеки от градовете, в които е поканен, и да се завърне в родния си град, като при това се опита да намали колкото може сумарната дължина на изминатия път. Професорът добре разбирал трудността на задачата и невъзможността да намери най-късия път за разумно време. Затова използвал един от любимите си трикове и построил маршрут с „разумна" дължина. За целта поставил родния си град в началото на правоъгълна координатна система, намерил координатите на останалите градове в тази координатна система и ги закръглил до цели числа. За дължина на пътя между два града използвал Евклидовото разстояние между точките в координатната система, съответни на тези два града. Сега е ваш ред да се проявите. Напишете програма **TRAVEL**, която да потърси колкото може по добър маршрут за обиколката на професора.

Вход

На първия ред на стандартния вход ще бъде зададено числото N ($4 \le N \le 2000$). На всеки от следващите $N-1$ реда ще бъдат зададени координатите X и Y на поредния град от обиколката, в реда по който са номерирани градовете. Координатите са цели числа в интервала [–20 000,20 000].

Изход

На стандартния изход програмата трябва да изведе една пермутация на числата от 1 до N – намерения от програмата маршрут.

Оценяване

За всеки тестов пример, за който програмата намери маршрут с дължина по-малка или равна на дължината L на маршрута, определен от професора, ще получите пълния брой точки за съответния тест. Ако дължината на намерения маршрут е по-голяма от $1.5L$ – няма да получите точки за теста. Ако дължината на намереното решение е по-голяма от L, но по-малка или равна на $1.5L$ – ще получите пропорционален на дължината на решението брой точки.

ПРИМЕР

Вход	Възможен изход
5 0 3 4 0 4 3 6 6	1 2 5 4 3

Заб. Изходът `1 3 5 2 4` също е възможен, но ще получи по-малко точки от дадения в примера.

РЕШЕНИЕ

Още една класическа задача в тази тема – „задачата на търговския пътник" или с други думи задачата за намиране на най-къс Хамилтонов път в пълен граф G с тегла на ребрата. И тази задача е *NP*- пълна и, в общия случай, да се намери точно оптимално решение е много трудно. В случая обаче, графът е геометричен, т.е. върховете му са точки в равнината, а дължините на ребрата между тях са съответните Евклидови

разстояния, значи за разстоянията между всеки три точки е в сила неравенството на триъгълника. Макар, че и в този случай намирането на оптимално решение е трудно, съществуват алгоритми („апроксимации"), които могат да гарантират решение, което е близо до оптималното. Ще разгледаме една сравнително проста апроксимация за задачата на търговския пътник. Ако разгледаме някое оптимално решение (Хамилтонов цикъл C с минимална дължина) и премахнем от него ребро, то полученият път P ще бъде покриващо дърво на графа, при това дължината му $l(P) \geq l(T)$, където T е някое минимално покриващо дърво (МПД) на G. Това ни дава следната идея. Да построим T и да превърнем всяко от ребрата му (неориентирани) в двойка противоположно ориентирани ребра. Полученият граф е Ойлеров и можем да го обходим в Ойлеров цикъл, като за да получим цикъл на G всеки път, когато се наложи да минем през вече посетен връх, пропускаме този връх и преминаваме към следващия. Тъй като „пълното" Ойлерово обхождане ще има дължина $2l(T)$, а всяко прескачане на връх v в последователността $...,u,v,w,...$ от върхове на Ойлеровия цикъл ще заменя (u,v) и (v,w) с $l(u,w)$, при това $l(u,v)+ l(v,w) \geq l(u,w)$ дължината на построения по този начин цикъл на G няма да надхвърли $2l(T) \leq 2l(P)$. Съществува апроксимация, която гарантира решение не надхвърлящо 3/2 от дължината на оптималното, но тя изисква решаването на задача много по-сложна от МПД и Ойлеров цикъл.

Нашите наблюдения при подготовката на решението на задачата показаха, че описаната апроксимация зависи от начина по който обхождаме построеното МПД, нещо което не се споменава явно в литературата по въпроса. Затова в решението сме разгледали две различни обхождания и от тях сме избрали минималното. При това тук публикуваме „версията" на професора, която извежда дължината на решението, а не исканата в условието на задачата, която трябва да изведе пермутацията. Тривиално решение на задачата, разбира се, е да се изведе пермутацията $1,2,\ldots,N$, но както ще се убедите, дори при случайното генериране на тестовете, шансът на такова решение да получи много точки са нищожни.

```
#include <stdio.h>
#include <math.h>
#define MAXN 2000
#define INF 2000000000
typedef struct
{  int N,M,directed;int T[MAXN+1][MAXN+1]; } AM_Graph;
typedef struct
{  int N; int P[MAXN+1]; } LP_Tree;
typedef struct
{  int N,M,directed;int T[MAXN+1][MAXN+1]; } LN_Graph;
typedef struct
{  int e[2*MAXN+2],top; } int_Stack;
void empty_stack(int_Stack* s) {  s->top=-1; }
int is_empty_stack(int_Stack* s)
{  if(s->top<0) return 1;
   else return 0;
}
void add_stack(int_Stack* s,int a) {  s->e[++s->top]=a;}
int look_stack(int_Stack* s) {  return s->e[s->top];}
void delete_stack(int_Stack* s) {  s->top--;}
int get_stack(int_Stack* s) {  return s->e[s->top--];}
int stack_count(int_Stack* s) { return s->top+1; }

int used[MAXN+2]={0},near[MAXN+2],int N,a[MAXN+1][2];
AM_Graph G;LP_Tree D; LN_Graph G1;int E_Loop[2*MAXN+2];
int per1[MAXN+1],per2[MAXN+1];
```

```
void Prim()
{  int i,j,k,minc,minv;
   D.P[1]=0; used[1]=1;
   for(i=1;i<=G.N;i++) near[i]=1;
   for(i=1;i<G.N;i++)
   {  minc=INF;minv=0;
      for(j=1;j<=G.N;j++)
         if(!used[j]&&G.T[j][near[j]]<minc)
         {  minc=G.T[j][near[j]];minv=j; }
      D.P[minv]=near[minv];
      G1.T[minv][++G1.T[minv][0]]=near[minv];
      G1.T[near[minv]][++G1.T[near[minv]][0]]=minv;
      used[minv]=1;
      for(j=1;j<=G.N;j++)
         if(!used[j]&&G.T[j][near[j]]>G.T[j][minv])
           near[j]=minv;
   }
   D.N=G.N;
}
int Euler_Loop_directed()
{  int i=0,v; int_Stack S;
   empty_stack(&S);add_stack(&S,1);
   while(!is_empty_stack(&S))
   {  v=look_stack(&S);
      if(G1.T[v][0]!=0)
      {  v=G1.T[v][G1.T[v][0]--]; add_stack(&S,v); }
       else E_Loop[i++]=get_stack(&S);
   }
   E_Loop[i]=0; return i;
}
int main()
{   int i,j,x,y;
    scanf("%d",&N);G.N=N;
     a[1][0]=a[1][1]=0;
    for(i=2;i<=N;i++)
    {  scanf("%d %d",&a[i][0],&a[i][1]);G.T[i][i]=0; }
    for(i=1;i<=N-1;i++)
       for(j=i+1;j<=N;j++)
       {  x=a[i][0]-a[j][0];y=a[i][1]-a[j][1];
          G.T[i][j]=G.T[j][i]=x*x+y*y;
       }
    G1.N=N;for(i=1;i<=N;i++)G1.T[i][0]=0;
    Prim(); j=Euler_Loop_directed();
    for(i=1;i<=N;i++) used[i]=0;
    j=1;double sum=0.0;double sum1=0.0;
    for(i=0;i<2*N-2;i++)
       if(!used[E_Loop[i]])
       { per1[j++]=E_Loop[i];used[E_Loop[i]]=1;}
    for(i=1;i<N;i++) sum+=sqrt((double)G.T[per1[i]][per1[i+1]]);
    sum+=sqrt((double)G.T[per1[N]][per1[1]]);
    for(i=1;i<=N;i++) used[i]=0;
    j=1;
    for(i=2*N-2;i>0;i--)
       if(!used[E_Loop[i]])
       { per2[j++]=E_Loop[i];used[E_Loop[i]]=1;}
    for(i=1;i<N;i++) sum1+=sqrt((double)G.T[per2[i]][per2[i+1]]);
    sum1+=sqrt((double)G.T[per2[N]][per2[1]]);
    if(sum<=sum1) printf("%lf\n",sum);
    else printf("%lf\n",sum1);
    return 0;
}
```

Заб. Програмното решение е направено с пакета подпрограми за работа с графи от подготвяната за печат книга „Алгоритми в графи. Част I: Основни алгоритми”.

Красимир Манев