**Battle Over Cities**

**Author:**

**Chen Jiongjian**

**Date:2013-10-29**

**Chapter 1: Introduction**

Background:

This project involves application of the graph algorithm. As for graphs, they are applied widely in our days. They are used in economy, aeronautics, physics, biology(for analyzing DNA), mathematics and other areas.

Problem description:

In the given information, we got the message that all the cities are connected by highways whose building process cost disparately, and the fact that once a city is occupied, highways from/towards that cities will be closed. Under such conditions, we are required to figure out the most “repair-expensive” city in case the war breaks out.

**Chapter 2: Algorithm Specification**

Our original code deals with the following steps:

First, adopt two adjacent matrixes (both are symmetric) to label cost and statue between cities intuitively. Then call “find” function to count each city’s linking number with other cities.

Next, we set a new map and make the cities connected degenerate to be one in order that convenience of creating a minimum spanning tree can be achieved. After that process, we build a minimum spanning tree and figure out the maximum cost among these cities (the corresponding cities can be more than one, so we use an array representation). Finally the program handles with the standard output.

The improved code differs in the way we store edges and label the connections. It use set-union operation instead of adjacent matrixes, and store edge by an array of size 1000\*4. For that part, further analysis can be found in Chapter 4.

**Chapter 3: Testing Results**

**Input and Output 1 [passed]**

This case is to test a map with only two cities. Actually no matter which city is occupied, you don’t need to pay anything for keep the rest of map connected.

2 1

1 2 1 1

0

**Input and Output 2 [passed]**

This case is to test a line-shaped map. Each city is necessary for the connection of map.

5 5

1 2 1 1

2 3 1 1

3 4 1 1

4 5 1 1

5 1 1 1

0

4 3

1 2 1 1

2 3 1 1

3 4 1 1

0

**Input and Output 3 [passed]**

This case is to test a circle-shaped map. No matter which city is occupied, all of the other cities will still be combined with each other.

**Input and Output 4 [passed]**

This case is a simple of larger size.

5 7

3 5 3 0

1 3 1 1

2 3 1 1

2 3 1 1

4 5 1 1

1 2 1 0

2 4 1 0

0

**Result of large-sized data [passed][TLE]**

We designed a data generator to generate data of large range. By comparing with another program written by tester, we are sure that the answers are all corrected, but the running time can’t make us pleasant. Some of cases(which with *N*≥500) would take more than 5 seconds to get correct answer.

**Chapter 4: Analysis and Comments**

For **space complexity**, in our program, we use three arrays of size 1000\*1000 to keep the original adjacent matrix and the adjacent matrix after adjustment. The other matrix is for storing the status of edges. So the space complexity of the program is N2. But actually, there are a lot of space wasted, especially for the map that is definitely sparse. On the other hand, keeping edges by matrix can make operations easier, and can save time on some stages.

For **time complexity**, it is obviously that if we are solving the problem in the same way as our program, we must test each node for one time. Meanwhile, inside the N-sized loop, we have to check N\*N times to find which one of all edges is available for using. As a result, the time complexity of our program is O(N3).

For a better understanding of our project, we wrote another program in a brand new way. We use set-union operation to clarify the connection status of whole map, and store edge by an array of size 1000\*4. Here is the result of different reactions between two programs with the same input data.

In the program, we need to check each one of N nodes for one time, then check M times to require whether that edge can connect the two separated union. So the time complexity is O(N\*M), while space complexity is only O(M).

It is absolutely that in the case which are generated by us, the O(N\*M) method is far more faster than method takes O(N3). However, because the maximum of edges is about O(N2), so if the map is thick enough, there should not be much advantage for O(N\*M) method.

**Appendix: Source Code (in C)**

/\*

Task Name:battle over cities

Coder:Henry

IDE:Dev Cpp 5.4.2

Time:2013.10.28

\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int cost[1001][1001];//store the distance by adjacent matrix

int state[1001][1001];//store the state of the edge

int map[1001][1001];//a new adjacent matrix

int s[1001];//disjoint sets

int outmax[1001]; //output queue of city numbers

int maxn,max,part;

int mincost,sum,l,r;

int n,m,i,j,k,first;

int city1,city2,cost1,state1;

int lowcost[1001];

int v[1001];

void find(int left,int sign,int del)

//mark cities connected with the same sign as floodfill algorithm

{

int right;

s[left]=sign;//mark the city

for(right=1;right<=n;right++)

if((cost[left][right]!=0)&&(state[left][right]==1)&&

(s[right]==0)&&(right!=del))

find(right,sign,del); //mark other cities connected with it in a recursion process

}

int main()

{

scanf("%d",&n);

while(n!=0)

{

scanf("%d",&m);

memset(cost,0,sizeof(cost));

memset(state,0,sizeof(state)); //initializing

for(i=1;i<=m;i++)

{

scanf("%d %d %d %d",&city1,&city2,&cost1,&state1);

cost[city1][city2]=cost1;

cost[city2][city1]=cost1; //the ways are undirected

state[city1][city2]=state1;

state[city2][city1]=state1;

} //initialize adjacetn matrix

max=0;

maxn=0;

memset(outmax,0,sizeof(outmax));

for(i=1;i<=n;i++)

{

memset(s,0,sizeof(s));

part=0;

for(first=1;first<=n;first++)

{

if((first!=i)&&(s[first]==0))

find(first,++part,i);

} //find how many parts were separated

for(l=1;l<=part;l++)

for(r=1;r<=part;r++)

if(l!=r)

map[l][r]=10000;//initialize new map

for(l=1;l<=n;l++)

for(r=1;r<=n;r++)

if((s[l]!=s[r])&&

(state[l][r]==0)&&(cost[l][r]!=0)

&&(cost[l][r]<map[s[l]][s[r]]))

map[s[l]][s[r]]=cost[l][r];

//here we make the cities connected degenerate to be one city

for(j=1;j<=part;j++)

{

lowcost[j]=map[1][j];

v[j]=0;

}

sum=0;//recording minimum cost to connect all the cities

v[1]=1;

for(j=1;j<=part;j++)

{

if(v[j]==0)

{

mincost=10000;

for(k=1;k<=part;k++)

if((v[k]!=0)&&(map[k][j]<mincost))

mincost=map[k][j];

lowcost[j]=mincost;

sum+=mincost;

}

} //process of minimun spanning tree

if(sum>max)

{

maxn=1;

outmax[maxn]=i;

max=sum;

}

else

if((sum==max)&&(sum!=0))

{

maxn++;

outmax[maxn]=i;

} //record the maximum

}

if(maxn==0)

printf("0");

else

{

for(i=1;i<=maxn-1;i++)

printf("%d ",outmax[i]);

printf("%d",outmax[maxn]);

} //output process

printf("\n");

scanf("%d",&n);

}

return 0;

}

**Declaration**

*We hereby declare that all the work done in this project titled "Battle Over Cities" is of our independent effort as a group.*

**Duty Assignments:**

Programmer: 秦昇

Tester: 安磊

Report Writer: 陈炯坚