中山大学计算机学院本科生实验报告

（2021 学年第 1 学期）

课程名称：Data structures and algorithms　　任课教师：张子臻

年级	20 级	专业（方向）	软件工程
学号	20337270	姓名	钟海财
电话	13397996670	Email	2940599563@qq.com
开始日期	2021/12/23	完成日期	2021/12/29

1. 实验题目

1）Highways

2.实验目的

多个输入的第一行是整数 T，然后是空行，后跟 T 个输入块。每个输入块均采用问题描述中指示的格式。输入块之间有一个空行。对于每种情况，第一行是整数 N （3 <= N <= 500），即村庄的数量。然后输入大小为 N*N 的邻接矩阵，d[i][j]表示村庄 i 到村庄 j 的距离（1<=i<=N,1<=j<=N）.

求出这些村庄的最小生成树中最长边的距离并输出。

要求分别使用 Prim 算法和 Kruskal 算法。

3.程序设计

1）Prim 算法

设计思路：

Prim 算法是基于点的最小生成树算法，开始设置最长边 d_max=0,随机选一

个节点作为起始节点（不妨选节点 1）并加入到树中，然后循环执行：找到所有与该树相邻的边中最短的且添加到树中不会成环（即该边的一个端点在当前树中，另一个端点不在当前树中）的一条边（长度为 d_min），将该边不在树中的端点加入到树中并记录，若 d_min>d_max,则令 d_max=d_min.

由于节点数为 n，所以当树中节点数为 n 时停止循环。并输出此时的 d_max 即为最小生成树中最长边的长度。

代码：

```
#include<iostream>
#include<vector>
using namespace std;
//Prim 算法
int main(){
    int times;
    cin>>times;
    while(times-->0){
        int n;
        cin>>n;
        vector<int> all_town;//用于存放已经在树中的节点
        int town[n+1]={0};//用于判断节点是否已经在树中
        int d_max=0;//要修的最长道路的长度
        int d0;//距离
        int d[n+1][n+1];//距离矩阵
        for(int i = 1 ; i<=n;++i ){//读取距离
            for(int j=1;j<=n;++j){
                cin>>d0;
                d[i][j]=d0;
            }
        }
        all_town.push_back(1);//从节点 1 开始
        town[1]=1;
        while(all_town.size()<n){ //当树中的节点树达到 n 时停止
            int key;//记录新增的节点
            int d_min=65536;//新增道路的距离
            for(int i=0;i<all_town.size();++i){//与当前树连接的最短的不成环的边
                int k = all_town[i];
                for(int j=2;j<=n;++j){
                    if(j!=k && d[k][j]<d_min && town[j]==0){
```

```
                        //更短则替换节点与边长
                        key=j;
                        d_min=d[k][j];
                    }
                }
            }
            if(d_min>d_max) d_max=d_min;//若新增边更长则替换
            town[key]=1;//记录为已在树中的节点
            all_town.push_back(key);
        }
        cout<<d_max<<endl<<endl;//输出最长边
    }
}
```

2）Kruskal 算法

设计思路：

其基本思想是将 G 的所有边按权值从小到大排序，然后依次考察每条边作为最小生成树 T 的候选树枝,如果某条边不与已经选中的边构成回路，则合乎要求，否则就被放弃而考虑下一条边，直到所有的边考虑完毕，或更简单地，只要 T 中已经有|V|－1 条边即可终止算法。

但是由于边数过多，如果要对边进行排序速度太慢，不如直接遍历所有边来找到最短的不成环的边。

由于每次得到的边都不一定是相连的，所以使用并查集来判断得到的边是否成环（如果该边得两个端点得代表元相同，说明成环，反之不成环），最后将该最短得不成环得边得两个端点记入并查集中。

代码：

```
#include<iostream>
#include<vector>
using namespace std;
//Kruskal 算法
    int p[501];//全局变量更快，并查集及其相关函数
```

```
    int find(int x) {
        return (p[x] == x) ? x : p[x] = find(p[x]);//实现了路径压缩
    }
    void union_(int x, int y) {
         if(find(y) == find(x)) return;
         p[find(y)] = find(x);
     }
     bool iscommon(int x,int y){
         if(find(y) == find(x)) return true;
         return false;
     }
int main(){
    int times;
    cin>>times;
    while(times-->0){
        int n;
        cin>>n;
        vector<int> all_side;//用于存放已经确定的通路，读入 n-1 条边则停止
        int d0;//距离
        int d[n+1][n+1];//距离矩阵
        for(int i = 1 ; i<=n;++i ){//读取距离
            p[i]=i;//并查集初始化
            for(int j=1;j<=n;++j){
                cin>>d0;
                d[i][j]=d0;
            }
        }
        while(all_side.size()<n-1){//循环读入最短不成环的边
            int d_min=65536;
            int x0,y0;//两个端点
            for(int i=1;i<=n;++i){
                for(int j=i+1;j<=n;++j){
                    if(!iscommon(i,j)&&d[i][j]<d_min){//找到不会成环的最短边
                        x0=i;
                        y0=j;
                        d_min=d[i][j];
                    }
                }
            }
            union_(x0,y0);//用并查集记录该边的两个端点
            all_side.push_back(d_min);//将该不会成环的最短边记入
        }
        cout<<all_side[n-2]<<endl<<endl;//输出最长的边
    }
}
```

4.程序运行与测试

测试输入 1

```
1
2
3
0 990 692
990 0 179
692 179 0
```

输出 1（通过）　692

测试输入 2

```
1
2
4
0 890 775  567
890 0 579  235
775 579 0  767
567 235 767 0
```

输出 2（通过）　579

测试输入 3

```
1
2
6
0 890 775  567 345 1234
890 0 579  235 456 667
775 579 0  767 567 899
567 235 767 0  789 900
345 456 567 789  0 821
1234 667 899 900 821 0
```

输出 3（通过）　667

5.实验总结与心得

Prim 算法是基于节点开始的算法，适应于边数较多，节点数相比边数明显较少的情况；而 Kruskal 算法是基于边开始的算法，首先要对边进行排序，因而适用于边数不是远多于节点数的情况。

而本次实验中，存在 n 个节点，n(n-1)/2 条边，显然边数远多于节点数，且使用 Kruskal 算法时直接从这 n(n-1)/2 条边中去找目标的 n-1 条边的时间复杂度 O([n(n-1)^2]/2)显然小于排序的时间复杂度 O([n(n-1)/2]log[n(n-1)/2]) ？,所以使用 Prim 算法明显优于 Kruskal 算法。

综上，得出结论：一般使用 Prim 算法优于 Kruskal 算法。

附录、提交文件清单