天津大学

数据结构上机实验报告

1. 题目:两种算法实现最小生成树

学生姓名 王雨朦

学生学号 **2016229082**

学院名称 国际工程师学院

专业 计算机

时间 **2016/11/29**

目　　录

第一章　需求分析

1.1 原题表述

某市为实现交通畅行，计划使全市中的任何两个村庄之间都实现公路互通，虽然不需要直接的公路相连，只要能够间接可达即可。现在给出了任意两个城镇之间修建公路的费用列表，以及此两个城镇之间的道路是否修通的状态，要求编写程序求出任意两村庄都实现公路互通的最小成本。

输入参数

测试输入包含若干测试用例。每个测试用例的第 1 行给出村庄数目 N (1< N < 100)；随后的 N(N-1)/2 行对应村庄间道路的成本及修建状态，每行给 4 个正整数，分别是两个村庄的编号（从 1 编号到 N），此两村庄间道路的成本，以及修建状态：1 表示已建，0 表示未建。

当 N 为 0 时输入结束。

输出参数

每个测试用例占输出的一行，输出实现后所需的最小成本值

1.2 解决方案

prim 算法：

1) 确定一个生成树的开始结点，找出与该点有关联的最小权值的点，将该点加入；
2) 把已经加入结点集合当作一个整体，查找外部与这个整体最权值最低的点加入集合；
3) 以此迭代下去，直到所有的点都被加入，问题就得到了解决。

kruskal 算法：

1) 对各边的权值进行排序，遍历找到最小的边，判断是否和已经生成的部分最小二叉树形成环；
2) 若不为环，则继续加入，若形成环，则继续遍历；
3) 重复直到找到 n-1 条边，此时算法结束，所得到的就是一个最小生成树。

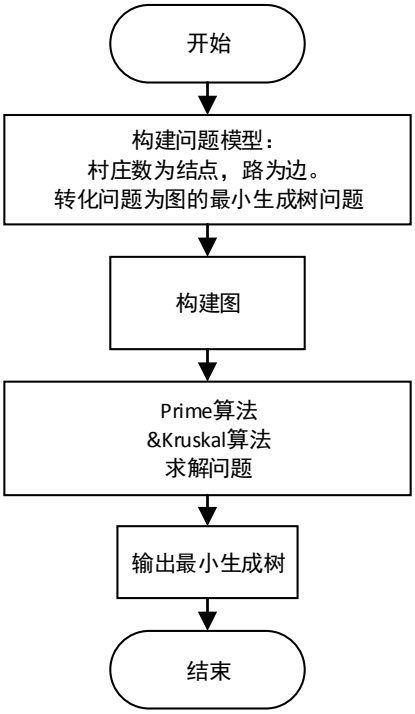
第二章　概要设计

2.1 抽象数据类型

图的矩阵&图的邻接点存储存储:

```
#define MAX 1000000;
struct AdjG
{
    int edges[200][200];
    int n;
    int e;
}
:
struct AdjG
{
    int edges[200][200]; int n; int e;
};
struct Edge
{
    int head;int tail;int weight;
};
```

2.2 算法



第三章　详细设计

3.1 程序代码

Prim:

```
#include <iostream>
using namespace std;

#define MAX 1000000;
typedef struct AdjG
{
    int edges[200][200];
    int n;
    int e;
}AdjG;
void CreateMatrix(AdjG & g, int v)
{
    g.n = v;
    g.e = v*(v-1)/2;

    for(int i=1; i<=g.n; i++)
        for(int j=1; j<=g.n; j++)
            g.edges[i][j]=MAX;

    for(int i=1; i<=g.e; i++)
    {
        int vs, ve, cost, judge;
        cin >> vs;
        cin >> ve;
        cin >> cost;
        cin >> judge;
        if(judge == 1) cost=0;

        g.edges[vs][ve]=cost;
        g.edges[ve][vs]=cost;
    }
}
int Prim(AdjG g){
    int result=0;
    int low[200], vex[200];
    for(int i=1; i<=g.n; i++)
    {
```

```
        low[i]=g.edges[1][i];
        vex[i]=1;
    }
    vex[1]=-1;
    for(int i=2; i<=g.n; i++)
    {
        int minL = MAX;
        int k;
        for(int j=1; j<=g.n; j++)
        {
            if(vex[j]!=-1&&low[j]<minL)
            {
                minL = low[j];
                k=j;
            }
        }
        result+=low[k];
        vex[k]=-1;
        for(int j=1; j<=g.n; j++)
        {
            if(vex[j]!=-1&&g.edges[k][j]<low[j])
            {
                low[j] = g.edges[k][j];
                vex[j]=k;
            }
        }
    }
    return result;
}
int main()
{
    while(true)
    {
        int v;
        cin >> v;
        if(v==0)
            break;
        AdjG g;
        CreateMatrix(g, v);
        int cost = Prim(g);
        cout << cost << endl;
    }
    return 0;
}
```

Kruskal：

```
#include <iostream>
using namespace std;

#define MAX 1000000;
struct AdjG
{
    int edges[200][200];
    int n;
    int e;
};
struct Edge
{
    int head;
    int tail;
    int weight;
};
void CreateMatrix(AdjG & g, int v)
{
    g.n = v;
    g.e = v*(v-1)/2;

    for(int i=1; i<=g.n; i++)
        for(int j=1; j<=g.n; j++)
            g.edges[i][j]=MAX;
    for(int i=1; i<=g.e; i++)
    {
        int vs, ve, cost, judge;
        cin >> vs;
        cin >> ve;
        cin >> cost;
        cin >> judge;
        if(judge == 1) cost=0;
        g.edges[vs][ve]=cost;
        g.edges[ve][vs]=cost;
    }
}
int findStart(int vexEnd[], int m)
{
    while(vexEnd[m]>0)
        m=vexEnd[m];
    return m;
}
```

```
int Kruskal(AdjG g)
{
    int vexEnd[200];
    Edge edge[200];
    int k=1;
    for(int i=1; i<=g.n; i++)
    {
        for(int j=i+1; j<=g.n; j++)
        {
            edge[k].head=i;
            edge[k].tail=j;
            edge[k].weight=g.edges[i][j];;
            k++;
        }
    }
    for(int i=1; i<k; i++)
    {
        for(int j=i+1; j<k; j++)
        {
            if(edge[i].weight > edge[j].weight)
            {
                int temp = edge[i].head;
                edge[i].head = edge[j].head;
                edge[j].head = temp;
                temp = edge[i].tail;
                edge[i].tail = edge[j].tail;
                edge[j].tail = temp;
                temp = edge[i].weight;
                edge[i].weight = edge[j].weight;
                edge[j].weight = temp;
            }
        }
    }
    for(int i=1; i<=g.n; i++)
    {
        vexEnd[i]=0;
    }
    int s,e;
    int result=0;
    for(int i=1; i<=g.e; i++)
    {
        s=findStart(vexEnd, edge[i].head);
        e=findStart(vexEnd, edge[i].tail);
        if(s!=e)
```

```
        {
            vexEnd[e]=s;
            result+=edge[i].weight;
        }
    }
    return result;
}
int main()
{
    while(true)
    {
        int v;
        cin >> v;
        if(v==0)
            break;
        AdjG g;
        CreateMatrix(g, v);
        int cost = Kruskal(g);
        cout << cost << endl;
    }
    return 0;
}
```

第四章　调试分析

4.1 调试过程

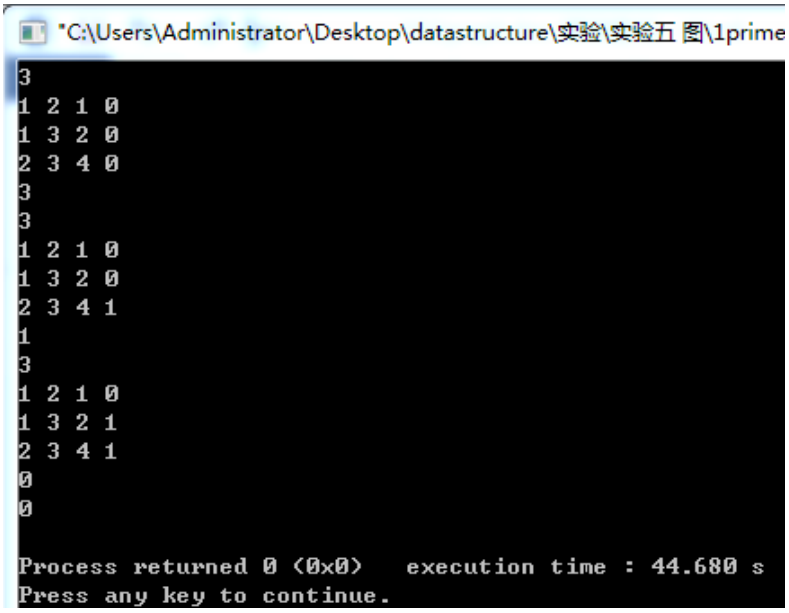
Bug 名称	数据结构选择错误
Bug 描述	图有矩阵存储和邻接表存储两种数据结构，不同的算法有更加适用的结构，因此在存储图的时候要合理选择
Bug 原因	选择了矩阵存储图再用 kruskal 算法，出现问题
Bug 解决方案	用 Kruskal 算法时改为邻接表存图
Bug 总结	数据结构的选择和运用不够熟练

Bug 名称	知识错误
Bug 描述	在 Krustal 算法时，对环的判断不清楚
Bug 原因	不能理解判断环的方法
Bug 解决方案	搜索网上一些方法和解释，分析理解最终用辅助数组解决
Bug 总结	通过学习，学会了新的解决问题的方法

第五章　测试结果

5.1 测试过程

测试编号	1
测试对象	Prim 算法功能输出最小成本值
测试输入参数	3 1 2 1 0 1 3 2 0 2 3 4 0
测试步骤	先输入邻接矩阵存储的图，再观察输出的最小成本是否正确
测试预期结果	3
测试输出结果	3
测试分析	程序正确



测试编号	2
测试对象	Kruskal 算法功能输出最小成本值
测试输入参数	3 1 2 1 0 1 3 2 1 2 3 4 1
测试步骤	先输入邻接矩阵存储的图，再观察输出的最小成本是否正确
测试预期结果	0
测试输出结果	0
测试分析	程序正确