数据结构课程设计

项目说明文档

电网建造造价模拟系统

|  |  |
| --- | --- |
| 作者姓名： | 罗斌江 |
| 学 号： | 2053285 |
| 指导教师： | 张 颖 |
| 学院专业： | 软件学院 软件工程 |



同济大学

Tongji University

目录

[1项目分析 3](#_Toc91014038)

[1.1项目背景 3](#_Toc91014039)

[1.2 项目需求分析 4](#_Toc91014040)

[1.3 项目要求 4](#_Toc91014041)

[1.3.1 功能要求 4](#_Toc91014042)

[1.3.2 输入格式 5](#_Toc91014043)

[1.3.3 输出格式 5](#_Toc91014044)

[1.3.4 项目示例 5](#_Toc91014045)

[2 项目设计 5](#_Toc91014046)

[2.1 数据结构设计 5](#_Toc91014047)

[2.2 类设计 6](#_Toc91014048)

[2.2.1 图类（Graph） 6](#_Toc91014049)

[2.2.2 边类（Edges） 8](#_Toc91014050)

[3 项目实现 8](#_Toc91014051)

[3.1 项目主体功能 8](#_Toc91014052)

[3.1.1 项目主体功能流程图 8](#_Toc91014053)

[3.1.2 项目主体功能代码 8](#_Toc91014054)

[3.2 Prim最小生成树 10](#_Toc91014055)

[3.2.1 Prim最小生成树流程图 10](#_Toc91014056)

[3.2.2 Prim最小生成树算法 10](#_Toc91014057)

[3.3 添加边 12](#_Toc91014058)

[4 项目测试 12](#_Toc91014059)

[4.1 创建顶点测试 12](#_Toc91014060)

[4.2 添加边测试 13](#_Toc91014061)

[4.3 生成最小生成树测试 13](#_Toc91014062)

[4.4 二次生成最小生成树测试 14](#_Toc91014063)

[4.5 边界测试 14](#_Toc91014064)

[4.5.1 只有两个顶点的最小生成树测试 14](#_Toc91014065)

[4.7.2 只有一个顶点的最小生成树测试 15](#_Toc91014066)

# 1项目分析

## 1.1项目背景

图（Graph）结构是一种非线性的数据结构，图在实际生活中有很多例子，比如交通运输网，地铁网络，社交网络，计算机中的状态执行（自动机）等等都可以抽象成图结构。图结构是一种比树结构更加复杂的非线性结构。

电网是电力系统中各种电压的变电及输配电线路组成的整体，在一个国家中电网的构建极为重要。但与此同时，如何合理的构造一个电网系统也是需要被解决的一个问题。为了使得构建电网的总工程造价最低，要设计一个能够满足要求的造价方案通常需要借助最小生成树来完成。

## 1.2 项目需求分析

针对于电网建造造价模拟系统，本项目在实现的过程中，考虑并且满足了以下的需求：

* 功能完善

系统所构造的用来解决电网构建问题的最小生成树应当正确无误。

* 执行效率高

针对数据量比较大的情况，本系统也应该具有在较短时间内求解出正确答案的能力。

* 代码可读性强

本项目在实现过程中，将代码根据功能的不同划分为了不同的代码块，同时进行了合理封装。

* 健壮性

当用户输入的数据不合理时，系统应当给予相应的提示而非直接报错。

* 可视化

该系统通过打印相应操作，使得各种操作可以直观的在使用时被用户感知。

## 1.3 项目要求

### 1.3.1 功能要求

假设一个城市有n个小区，要实现n个小区之间的电网都能够相互接通，构造这个城市n个小区之间的电网，使总工程造价最低。请设计一个能够满足要求的造价方案。

在每个小区之间都可以设置一条电网线路，都要付出相应的经济代价。n个小区之间最多可以有条线路，选择其中的n-1条使总的耗费最少。

### 1.3.2 输入格式

输入相应的操作。

### 1.3.3 输出格式

执行相应的操作，绘制最小生成树。

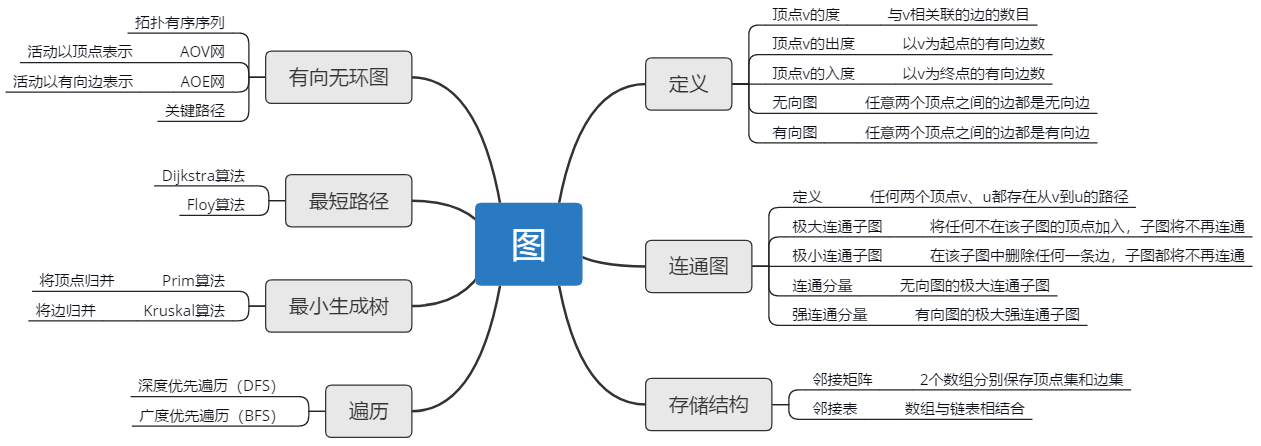
### 1.3.4 项目示例



# 2 项目设计

## 2.1 数据结构设计

图是由若干给定的点及连接两点的线所构成的图形，这种图形通常用来描述某些事物之间的某种特定关系，用点代表事物，用连接两点的线表示相应两个事物间具有这种关系。



如上图所示，本项目中设计了一个**图**（Graph）类。由于本项目中所述电网之间任意两个顶点不存在方向关系，因此属于无向图，即任意两个顶点之间的边都是无向边。

此外，图的存储结构有两种：一种是邻接矩阵法，另一种则是邻接表法。这两种存储结构在执行下列操作时的时间复杂度如下所示：

|  |  |  |
| --- | --- | --- |
|  | 邻接矩阵 | 邻接表 |
| 遍历 | O (n²) | O（n+e) |
| 判断某一边 | O（1） | O (n) |
| Prim算法 | O（n²） | O（n+e) |

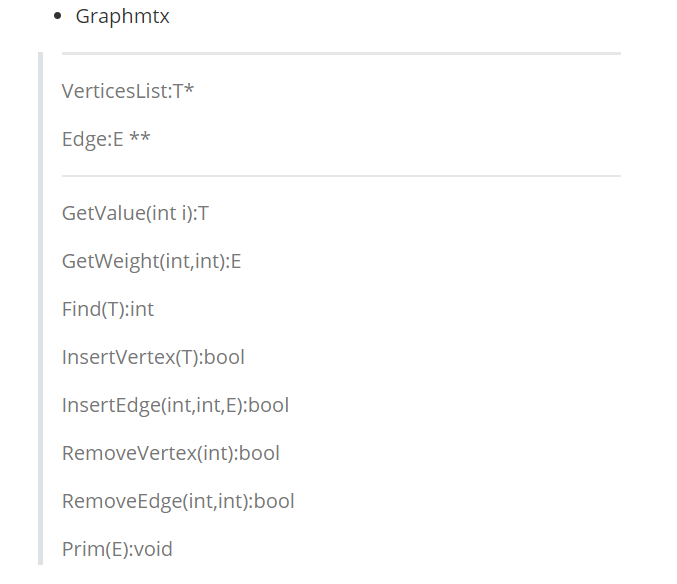
为了降低时间复杂度，本项目中采用了邻接表法存储图。为了实现邻接表，本项目中使用了邻接矩阵。

## 2.2 类设计

### 2.2.1 图类（Graph）

一个图的内部主要需要存储结点和边。而邻接矩阵表示的图只需要邻接矩阵和一个用来存储顶点的定点表就可以记录所有的结点和边；

图的UML图如下所示：



类中主要函数如下所示：

* T GetValue(int i)

返回顶点表中处于i位置顶点的值。

* E GetWeight(int v1,int v2)

返回顶点表中v1和v2两顶点之间边的权值。

* void **Prim**(T start)

该函数用于生成Prim最小生成树。

* bool **InsertVertex**(T vertex)

向顶点表中插入顶点vertex。

* bool InsertEdge(int v1,int v2,E cost)

向顶点v1和v2建立权值为cost的边。

* bool **RemoveVertex**(int i)

该函数用于清除顶点表中i位置的顶点。

* bool **RemoveEdge**(int v1,int v2)

移除v1和v2之间的边。

通过上述函数操作，即完成了定义了一个图类，各函数的详细实现方法见第3部分。

### 2.2.2 边类（Edges）

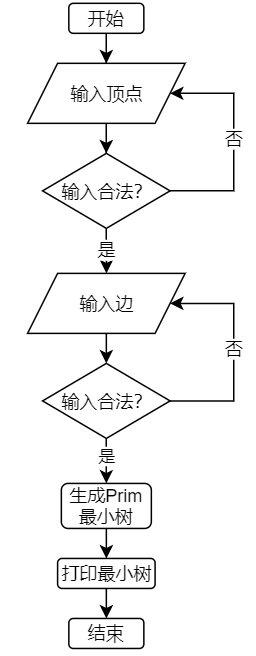
Edges类用于存储最小生成树的结果序列



# 3 项目实现

## 3.1 项目主体功能

### 3.1.1 项目主体功能流程图



### 3.1.2 项目主体功能代码

menu();

    Graphmtx<std::string, *int*> p;

    std::cout<<std::endl;

*//std::cout<<"初始化电网系统:"<< std::endl;*

*//std::cin >> p;*

    std::string start;

*bool* con=true;

    while(con){

        std::cout<<"请选择操作:";

*char* option;

        std::cin >> option;

        switch(option){

            case 'A':{

*int* num;

                std::cout<<"请输入顶点的个数:";

                std::cin>>num;

                std::string vertex;

                std::cout<<"请依次输入各顶点的名称:"<<std::endl;

                for(*int* i=0;i<num;i++){

                    std::cin>>vertex;

                    p.InsertVertex(vertex);

                }

                break;

            }

            case 'B':{

*int* num;

                std::cout<<"请输入边的个数:";

                std::cin>>num;

                std::string vertex1,vertex2;

*int* cost = 0;

                std::cout<<"请依次输入各顶点的名称:"<<std::endl;

                for(*int* i=0;i<num;i++){

                    std::cin>>vertex1>>vertex2>>cost;

*int* v1=p.Find(vertex1);

*int* v2=p.Find(vertex2);

                    if(v1!=-1&&v2!=-1){

                        p.InsertEdge(v1,v2,cost);

                    }

                }

                break;

            }

            case 'C':{

                std::cout<<"请输入起始顶点:";

                std::cin>>start;

                std::cout<<"最小生成树的顶点及边为:";

                p.Prim(start);

                break;

            }

            case 'D':{

                con=false;

                break;

            }

            default: {

                std::cout << "Error input" << std::endl;

                break;

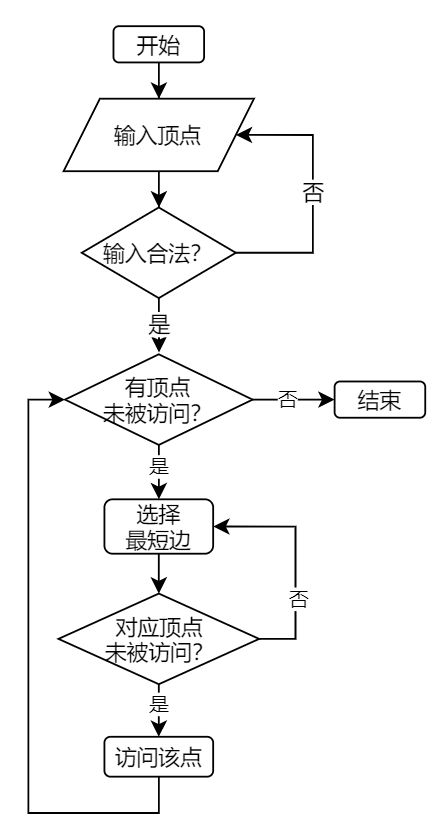
            }

        }

    }

## 3.2 Prim最小生成树

### 3.2.1 Prim最小生成树流程图



### 3.2.2 Prim最小生成树算法

*template*<*class* T, *class* E>

*void* Graphmtx<T, E>::Prim(T *start*) {

*int* pos = Find(*start*);

    if (pos == -1) {

        std::cout << "could not find " << *start* << std::endl;

        return;

    }

    std::stack<*int*> opStack;

*int* count = numOfVertices;

*int* min = -1;

*int* vertex2 = -1, vertex1 = -1;

    result = new Edges<T, E>[count];

*bool*\* visited = new *bool*[count];

    for (*int* i = 0; i < count; i++) {

        visited[i] = false;

    }

    opStack.push(pos);

    visited[pos] = true;

    for (*int* i = 1; i < count; i++) {

        min = INF;

        vertex2 = -1;

        while (!opStack.empty()) {

*int* vertex = opStack.top();

            opStack.pop();

            for (*int* j = 0; j < numOfVertices; j++) {

*int* cost = GetWeight(vertex, j);

                if (visited[j] == false && cost < INF) {

                    if (min > cost) {

                        min = cost;

                        vertex2 = j;

                        vertex1 = vertex;

                    }

                }

            }

        }

        if (vertex2 != -1) {

            visited[vertex2] = true;

            for (*int* j = 0; j < count; j++) {

                if (visited[j] == true) {

                    opStack.push(j);

                }

            }

            result[i].data1 = VerticesList[vertex1];

            result[i].data2 = VerticesList[vertex2];

            result[i].weight = min;

        }

    }

    for (*int* i = 1; i < count ; i++) {

        std::cout << result[i].data1 << "-<" << result[i].weight << ">->" << result[i].data2 << std::endl;

    }

    std::cout<<"生成Prim最小生成树!"<<std::endl;

    delete[] result;

    return;

}

## 3.3 添加边

采用邻接矩阵存取的图，在添加边的时候可以直接更改邻接矩阵对应的值，。代码如下：

*template*<*class* T, *class* E>

*bool* Graphmtx<T, E>::InsertEdge(*int* *v1*, *int* *v2*, E *cost*) {

    if (*v1* > -1 && *v1*<numOfVertices &&

*v2*>-1 && *v2* < numOfVertices &&

        Edge[*v1*][*v2*] == INF) {

        Edge[*v1*][*v2*] = Edge[*v2*][*v1*] = *cost*;

        numOfEdges++;

        return true;

    }

    return false;

}

# 4 项目测试

## 4.1 创建顶点测试

测试用例：

A

4

a b c d

预期结果：

添加顶点

实验结果：



## 4.2 添加边测试

测试用例：

B

6

a b 8

b c 7

c d 5

d a 11

a c 18

b d 12

预期结果：

添加边

实验结果：



## 4.3 生成最小生成树测试

测试用例：

C a

预期结果：生成以a为起点的Prim最小生成树

实验结果：

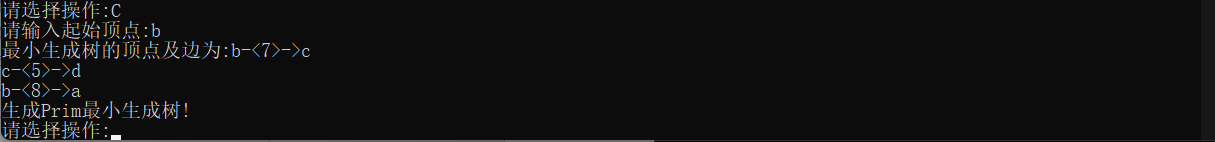


## 4.4 二次生成最小生成树测试

测试用例：C b

预期结果：生成以b为起点的Prim最小生成树

实验结果：



## 4.5 边界测试

### 4.5.1 只有两个顶点的最小生成树测试

测试用例：

A

2 a b

B

a b 10

? ? 0

C

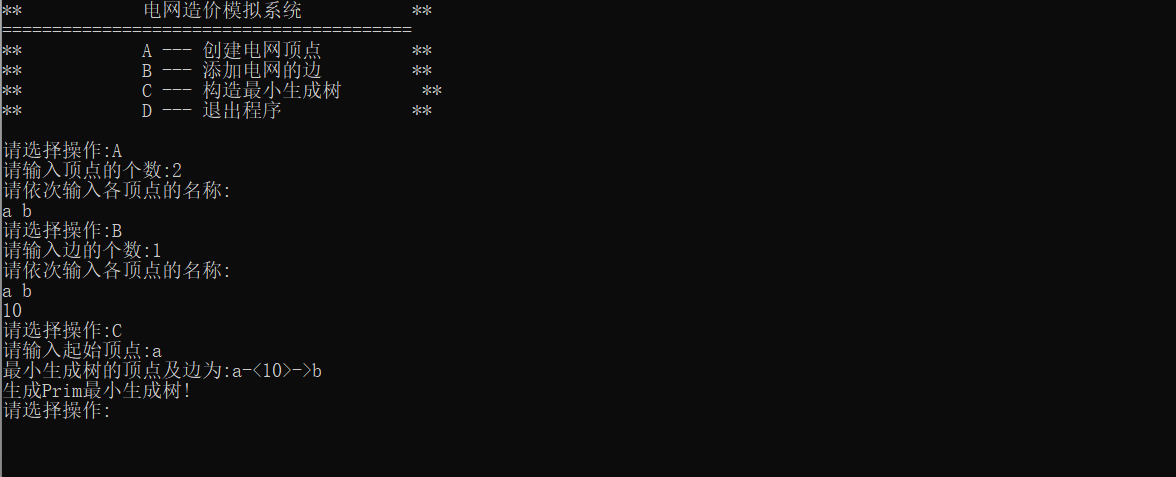
a

D

预期结果：

打印只含两个顶点的图中以a为起点生成的Prim最小生成树

实验结果：



### 4.7.2 只有一个顶点的最小生成树测试

测试用例：

A

1 a

C

预期结果：

打印只含一个顶点的图中以a为起点生成的Prim最小生成树

实验结果：

