项目说明文档

数据结构课程设计

——电网建设造价模拟系统

作 者 姓 名：杨鑫

学　　号：1950787

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 分析

1.1 背景分析

电在我们的日常生活中占有非常重要的地位，如果没有电，无法想象世界会变成什么样子。城市中的电力一般都通过电网来进行运输，而电网的造价成本不菲，在保证全城都能通上电的前提下，如何将电网的造价成本降至最低，是一个非常有经济价值的问题。现在假设一个城市有 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。

1.2 功能分析

系统的功能要求是在每个小区之间都可以设置一条电网线路，但都要付出相应的经济代价。n 个小区之间最多可以有 n（n-1）/2 条线路，选择其中的 n-1 条使总的耗费最少。因此首先系统需要小区的布局构造出来，随后在小区之间增加线路。增加完毕后，就需要通过计算来得出其中的 n-1 条使总的耗费最少，并将它们显示出来。

综上所述，该电网建设造价模拟系统需要有创建电网顶点，添加电网的边，计算得出 n-1 条线路使总的耗费最少，并输出构造方案的功能。

2 设计

2.1 数据结构设计

一个连通图的每一棵生成树，都是原图的一个极大无环子图。如果每个小区可以看作一个顶点，小区与小区之间的电网线路可以看作一条边，每条线路的造价看作边上的权值，这样就可以把整个电网看作一个连通网络。若一个连通网络由 n 个顶点组成，则其生成树必含 n 个结点、n-1 条边。由于我们要求建立一个造价最低的电网系统，那就要找出该联通网络的一棵最小生成树。

构造最小生成树的方法最典型的有两种，一种为 Kruskal 算法，一种为 Prim 算法。本系统采用了 Prim 算法。

Prim 算法是在不断迭代进行的。构建两个集合 V 与 V1，V 代表当前最小生成树的顶点集合，V1 代表不属于当前生成树的顶点集合。首先选定构造最小生成树的起始顶点 u0，将它加入到集合 V 中，随后选择一条边（u，v），要求 u 属于集合 V 而 v 属于集合 V1，且该边为满足该条件的权值最小的边。将 v 从 V1

取出加入到集合 V 中，然后继续这个过程，直到网络中的所有顶点都已经加入到生成树顶点集合 V 中。此时算法过程中选取的边的集合就可以构建出最小生成树。

如上分析所述，该电网造价模拟系统会有大量创建顶点，添加边的操作。但边不会很密集，因此采用用邻接表表示的图的数据结构来保存所有小区与线路。由于 Prim 算法每次都要获得权值最小的边，因此设计了一个最小堆的数据结构，每次从堆顶取出权值最小的边。同时设计了一个最小伸展树类，用于存放生成的连通树的节点。

2.2 类结构设计

如上分析所述，首先本系统有一个邻接表表示的图类（Graphmtx），其次，为了给 Prim 算法提供最小权值的边，系统设计了一个最小堆类（MinHeap）。同时，为了保存连通树的节点，还设计了一个最小伸展树类（MinSpanTree）以及它的节点结构体（MSTEdgeNode）。最后，本系统还设计了电网系统类（ElectricNetwork），整合了图和最小伸展树类，还提供了一系列公有操作函数以实现一系列操作。为了使数据结构更具有泛用性，本系统将 MinSpanTree 类，MinHeap 类，Graphmtx 类等都设计为了模板类。

2.3 成员与操作设计

最小堆（MinHeap）：

私有成员：

```
E* heap;
int currentSize;
int maxHeapSize;
void siftDown(int start, int m); // 向下调整
void siftUp(int satrt); // 向上调整
```

公有操作：

```
MinHeap(int sz = DefaultSize); // 构造函数
~MinHeap() { delete[]heap; } // 析构函数
bool Insert(const E x); // 插入
bool RemoveMin(E& x); // 弹出最小元素
bool IsEmpty() const { return currentSize == 0; } // 是否为空
bool IsFull() const { return currentSize == maxHeapSize; } // 是否满
void MakeEmpty() { currentSize = 0; } // 置空
```

图（Graphmtx）：

私有成员：

```
// 重载输入
friend istream& operator >> (istream& in, Graphmtx<T, E>& G);
```

```
3.
4.    // 重载输出
5.    friend ostream& operator << (ostream& out, Graphmtx<T, E>& G);
6.
7.    // 邻接表的表示方法
8.    int maxVertices;
9.    int numEdges;
10.   int numVertices;
11.   T* VerticesList;
12.   E** Edge;
13.
14.   // 返回顶点元素的位置
15.   int getVertexPos(T vertex);
```

公有操作：

```
1.    Graphmtx(int sz = DefaultSize); // 构造函数
2.
3.     // 析构函数
4.    ~Graphmtx() {
5.        delete[] VerticesList;
6.        delete[] Edge;
7.    }
8.    bool GraphEmpty() const { return numEdges == 0; } // 判空
9.    bool GraphFull() const { return numEdges == maxVertices || numEdg
   es == maxVertices * (maxVertices - 1) / 2; } // 判满
10.   int NumberOfVertices() { return numVertices; } // 返回顶点个数
11.   int NumberOfEdges() { return numEdges; } // 返回边的个数
12.   T getValue(int i) { return i >= 0 && i <= numVertices ? VerticesL
   ist[i] : NULL; } // 得到顶点的值
13.   E getWeight(int v1, int v2) { return v1 != -1 && v2 != -1 ? Edge[
   v1][v2] : 0; } // 得到边的权值
14.   int getFirstNeighbor(int v); // 得到顶点的第一个邻点
15.   int getNextNeighbor(int v, int w); // 得到下一个邻点
16.   bool insertVertex(const T& vertex); // 插入顶点
17.   bool insertEdge(T& V1, T& V2, E cost); // 插入边
18.   bool removeVertex(T& V); // 删除顶点
19.   bool removeEdge(T& V1, T& V2); // 删除边
20.   bool Prim(Graphmtx<T, E>*& G, const T start, MinSpanTree<T, E>*&
   MST); // 建立最小生成树
```

最小伸展树节点结构体（MSTEdgeNode）：

```
1.    T tail, head;
2.    E key;
3.    MSTEdgeNode() : tail(-1), head(-1), key(0) {} // 构造函数
```

```
    MSTEdgeNode(const MSTEdgeNode& MN) : head(MN.head), tail(MN.tail), key(MN.key) {} // 构造函数

     // 重载运算符
    bool operator == (const MSTEdgeNode& MN);
    bool operator <= (const MSTEdgeNode& MN);
    bool operator > (const MSTEdgeNode& MN);
    void operator = (const MSTEdgeNode& MN);

    // 重载输出
    friend ostream& operator << (ostream& out, const MSTEdgeNode& MN);
```

最小伸展树类（MinSpanTree）：

私有成员：

```
    MSTEdgeNode<T, E>* edgevalue;
    int maxsize, n;
```

公有操作：

```
    MinSpanTree(int sz = DefaultSize - 1) : maxsize(sz), n(0) { edgevalue = new MSTEdgeNode<T, E>[sz]; } // 构造函数
    void Insert(MSTEdgeNode<T, E>*& item); // 插入节点
    void Show(); // 打印
```

电网系统类（ElectricNetwork）：

私有成员：

```
    Graphmtx<char, int>* graph;
    MinSpanTree<char, int>* tree;
```

公有操作：

```
    ElectricNetwork(); // 构造函数
    ~ElectricNetwork(); // 析构函数
    void CreateVertex(); // 创建顶点
    void AddEdge(); // 增加边
    void CreateMinSpanTree(); // 创建最小生成树
    void ShowTree(); // 展示
    void Loop(); // 主循环
```

2.4 系统设计

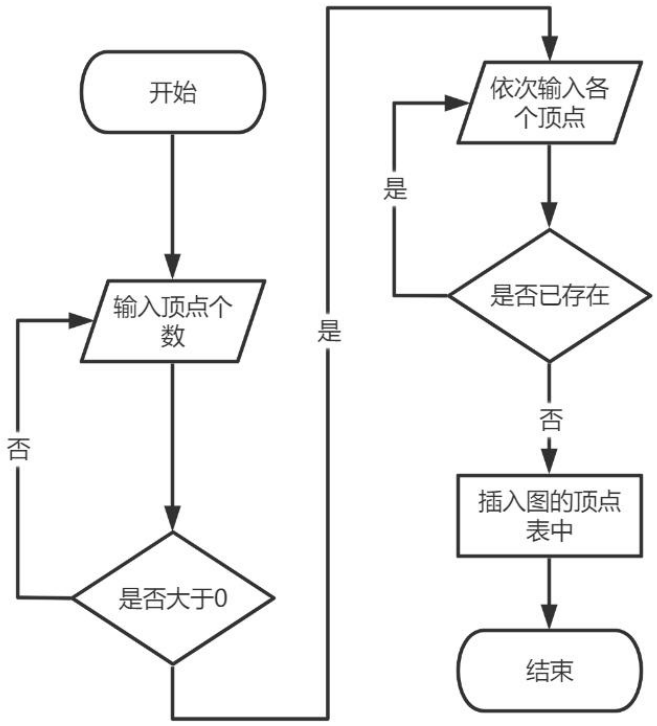
系统首先会在屏幕上显示一个用户操作菜单，根据此菜单，用户通过输入对应的操作码来执行相应的操作。功能有创建电网顶点，添加电网的边，构造最小生成树，显示最小生成树，退出程序等等。

程序兼容了 windows 和 LINUX 平台，在双平台下均可以正常运行。

3 实现

3.1 创建电网顶点功能的实现

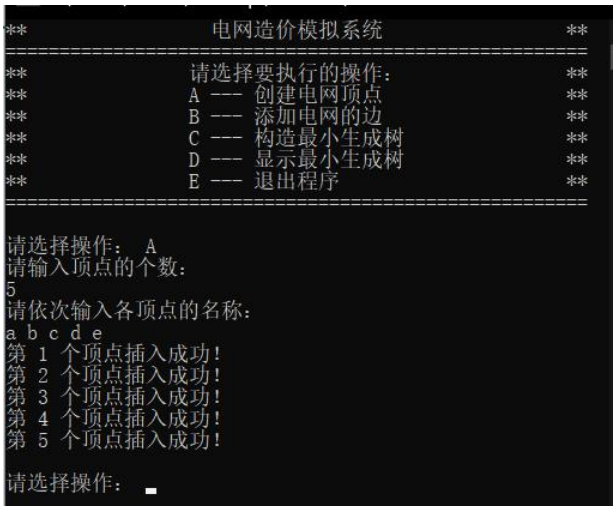
3.1.1 创建电网顶点功能流程图



3.2.2 创建电网顶点功能核心代码

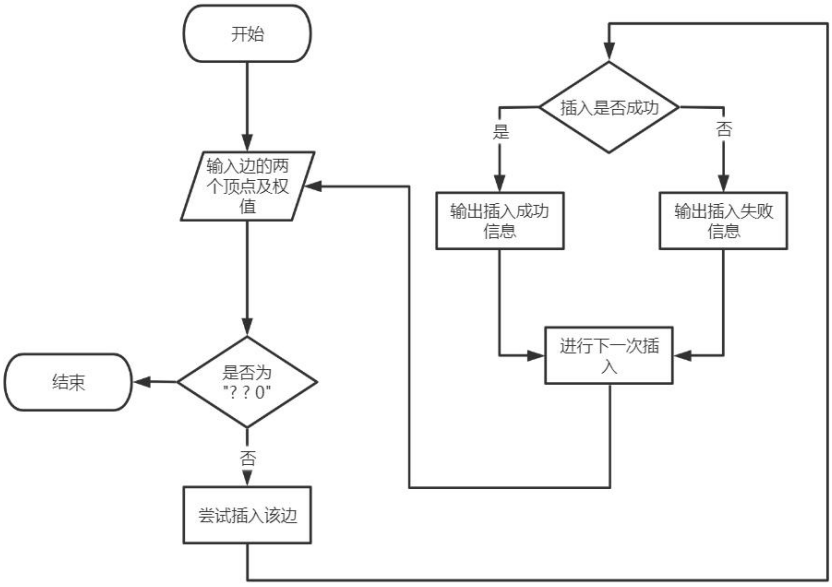
```
// 已满
if (Graphmtx<T, E>::numVertices == maxVertices) return false;

// 在最后一个位置插入
for (int i = 0; i < Graphmtx<T, E>::numVertices; i++) {
    if (VerticesList[i] == vertex) return false;
}
VerticesList[Graphmtx<T, E>::numVertices++] = vertex;
return true;
```

3.1.3 创建电网顶点功能截屏示例



3.2 添加电网的边功能的实现

3.2.1 添加电网的边功能流程图



3.2.2 添加电网的边功能核心代码

```
// 获取两个顶点的位置
int v1 = getVertexPos(V1);
int v2 = getVertexPos(V2);

```

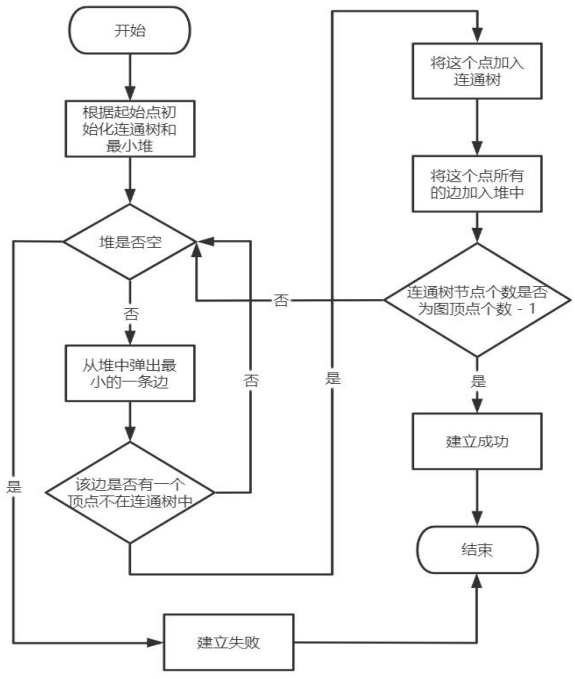
```
// 如果原来不存在这个边并且点存在， 执行插入
if (v1 > -1 && v1 < Graphmtx<T, E>::numVertices && v2 > -1 && v2 < Graphmtx<T, E>::numVertices && Edge[v1] [v2] == maxWeight) {
    Edge[v1][v2] = Edge[v2][v1] = cost;
    Graphmtx<T, E>::numEdges++;
    return true;
}
else return false;
```

3.2.3 添加电网的边功能截屏示例

```
请选择操作： A
请输入顶点的个数：
3
请依次输入各顶点的名称：
a b c
第 1 个顶点插入成功！
第 2 个顶点插入成功！
第 3 个顶点插入成功！

请选择操作： B
请输入两个顶点及边（输入 ？？ 0 以结束输入）：a b 5
该边插入成功！
请输入两个顶点及边（输入 ？？ 0 以结束输入）：b c 10
该边插入成功！
请输入两个顶点及边（输入 ？？ 0 以结束输入）：？？ 0
插入结束！

请选择操作：
```

3.3 构造最小生成树功能的实现

3.3.1 构造最小生成树功能流程图

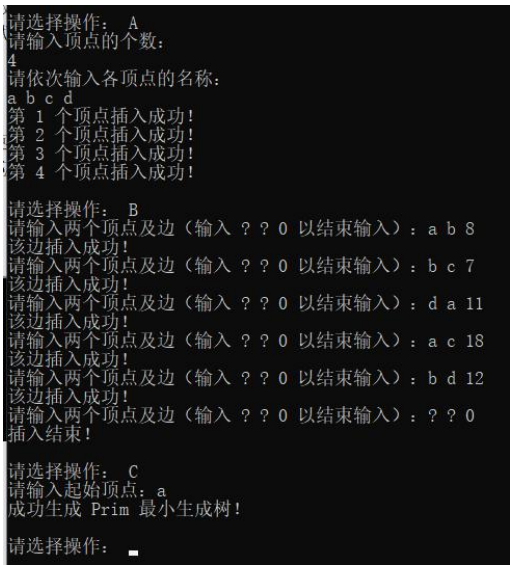


3.3.2 构造最小生成树功能核心代码

```
// 获取顶点和边的数量， 若顶点数量为 0 ，返回 false
int vertices = G->NumberOfVertices();
if (vertices == 0) return false;
int edges = G->NumberOfEdges();

// 记录上次的顶点
int last = G->getVertexPos(start);
MinHeap<E, MSTEdgeNode<T, E>>H(edges);

// 记录顶点是否已经放入
bool* MSTvertex = new bool[vertices];
for (int i = 0; i < vertices; i++) MSTvertex[i] = false;

// 一些初始化
MSTvertex[last] = true;
int count = 1;
MSTEdgeNode<T, E> MN;

// 不断扩充最小生成树直到符号条件
do {
    int next = G->getFirstNeighbor(last);

    // 往堆里面插入邻点的所有边
    while (next != -1) {
        if (MSTvertex[next] == false) {
            MN.tail = G->VerticesList[last];
            MN.head = G->VerticesList[next];
            MN.key = G->getWeight(last, next);
            H.Insert(MN);
        }
        // 下一个邻点
        next = G->getNextNeighbor(last, next);
    }

    while (count < vertices) {
        // 堆空， 返回 false
        if (H.IsEmpty()) return false;

        // 弹出权值最小的边
        H.RemoveMin(MN);

        // 若另一个顶点未被使用， 插入最小生成树中
```

```
43.             if (!MSTvertex[G->getVertexPos(MN.head)]) {
44.                 MSTEdgeNode<T, E>* ptr = &MN;
45.                 MST->Insert(ptr);
46.                 last = G->getVertexPos(MN.head);
47.                 MSTvertex[last] = true;
48.                 count++;
49.                 break;
50.             }
51.         }
52.     } while (count < vertices);
53.
54.     return true;
```

3.3.3 构造最小生成树功能截屏示例



4 测试

4.1 功能测试

4.1.1 整体功能测试

测试用例：

A

10

a b c d e f g h i j

B

d i 7

e f 7

j c 15

g a 12

c f 10

b d 10

h c 12

i j 3

g b 7

f a 5

d e 16

f g 13

c e 4

h b 6

e a 9

a b 13

i b 8

f j 5

d j 10

h g 3

c d 6

i f 14

b f 11

j e 6

e g 4

? ? 0

C

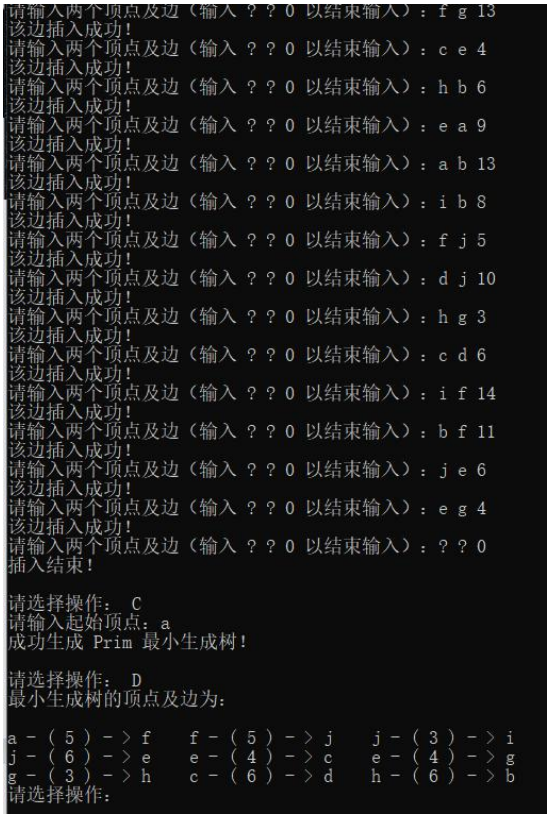
a

D

预期结果：

程序正常运行不崩溃，正确输出结果。

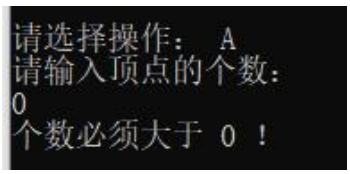
实验结果：



4.2 边界测试

4.2.1 创建电网顶点数为零

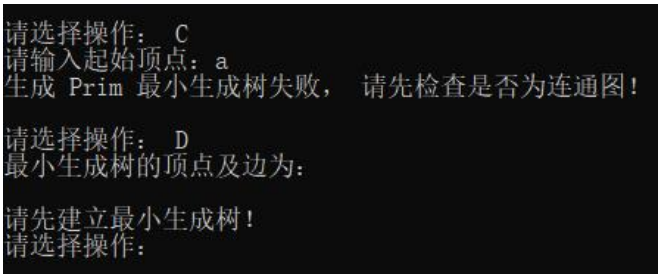
测试用例：
A
0
预期结果：程序给出提示信息，程序运行正常不崩溃。
实验结果：



4.2.2 当前电网顶点数为零或边数为零时进行 C,D 操作

测试用例：
C
a
D
预期结果：程序给出提示信息，程序运行正常不崩溃。

实验结果：



4.3 出错测试

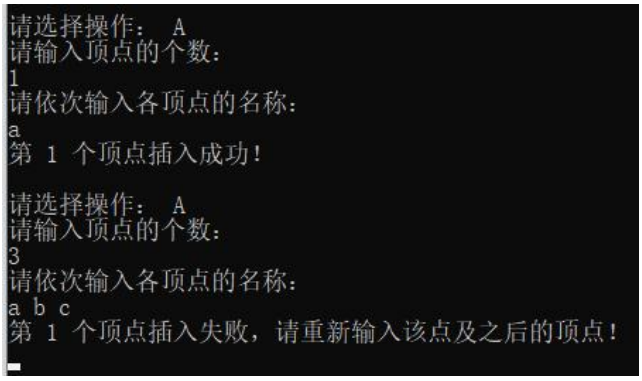
4.3.1 新添加的顶点名称在电网中已存在

测试用例：

A
1
a
A
3
a b c

预期结果：程序给出提示信息，程序正常运行不崩溃。

实验结果：

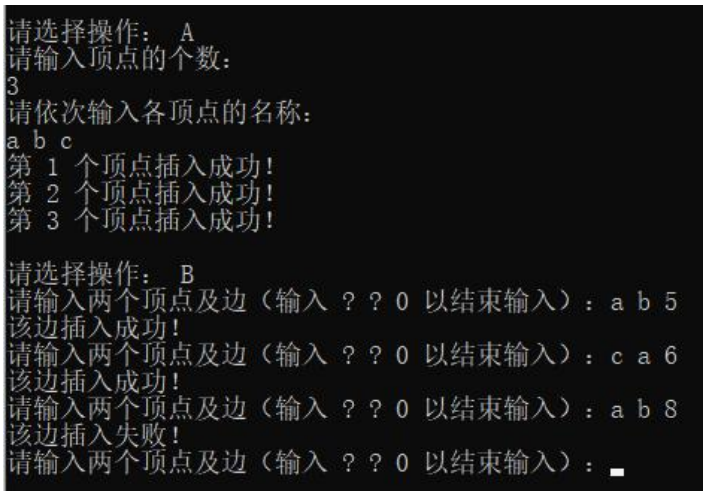


4.3.2 添加边时输入的两个顶点之间已经存在边

测试用例：

A
3
a b c
B
a b 5
c a 6
a b 8

预期结果：程序给出提示信息，程序正常运行不崩溃。
实验结果：



4.3.3 电网为非连通图时构造最小生成树

测试用例：
A
3
a b c
B
a b 5
？？0
C
a
预期结果：程序正常运行不崩溃，输出建立 Prim 树失败，并提示用户检查是否为连通图。
实验结果：