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： 吴英豪

学 号： 1953608

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

1.分析

1.1背景分析

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

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n（n-1）/2条线路，选择其中的n-1条使总的耗费最少。

2.设计

2.1数据结构设计

本题可以简化为求图的最小生成树的问题，故采用Kruskal算法来构造图的最小生成树。需要的数据结构有:图、并查集、和队列。

图(Graph)由顶点和边构成，顶点集合和边的集合采用链表的形式，涉及到顶点结点(VertexNode)和边结点(EdgeNode).图主要涉及到的操作有插入节点，插入边，利用Kruskal算法构造最小生成树。

并查集(UFSet)主要用于Kruskal算法构造最小生成树,以数组的形式呈现，在同一个并查集中的元素具有相同的根结点，涉及的操作有并查集的查询，初始化，插入和合并。

队列类(Queue)主要用于保存最小生成树的相关结点，便于最终最小生成树边的输出，能有提高用户和系统交互的便利性

2.2类结构设计

2.2.1顶点类Vertex

顶点集合是图中不可或缺的元素，考虑到图使用链表来表示顶点集合，所以Vertex中的成员有顶点的名字\_name和指向下一个顶点的指针\_next.

2.2.2 边结点EdgeNode

边集合是图中不可或缺的元素，考虑到图使用链表来表示边集合，所以EdgeNode中的成员有第一个顶点的名字\_vertex\_name\_1,第二个顶点的名字\_vertex\_name\_2,边的权值\_value,指向下一个边结点的指针\_next;

2.2.3 图类 Graph

图的成员有链表形式的顶点集合,其头指针为\_vertex\_first;链表形式的边集合,其头指针为\_edge\_first;顶点的个数vertex\_num

2.2.4 队列类 Queue

链式队列,主要用于保存最小生成树的各个边，所以采用边界点EdgeNode作为队列的元素。

2.2.5 并查集UFSet

数组形式的并查集,其成员有最大长度max\_length,存放数据的数组data\_array，存放父结点的数组parent\_array.

2.2.6 相关说明

考虑到代码的复用性，将所有的类都设计为模板。

此系统各个类之间设计频繁的相互访问数据的操作，所以需要设置友元。其中,EdgeNode,Graph,UFSet为顶点Vertex的友元;Graph,Queue,UFSet为边界点EdgeNode的友元;UFSet为Graph的友元。

2.3成员与操作设计

2.3.1顶点类Vertex

//顶点类

template<class T,class E>

class VertexNode

{

private:

T \_name; //此顶点的名字

VertexNode<T,E>\* \_next; //下一个顶点

public:

VertexNode(T name=0);

~VertexNode();

friend class EdgeNode<T,E>;//友元类，边

friend class Graph<T,E>; //友元类，图

friend class UFSet<T,E>; //友元类，并查集

};

2.3.2边结点EdgeNode

//边类

template<class T,class E>

class EdgeNode

{

private:

T \_vertex\_name\_1; //第一个顶点的名字

T \_vertex\_name\_2; //第二个顶点的名字

E \_value; //当前边的权值

EdgeNode<T, E>\* \_next; //下一个图结点

public:

EdgeNode(T name1=0,T name2=0,E value=0);

EdgeNode(EdgeNode<T, E>\* edge);

~EdgeNode();

void printEdge();

friend class Graph<T,E>; //友元类，图

friend class Queue<T,E>; //友元类，队列

friend class UFSet<T, E>; //友元类，并查集

};

2.3.3图类 Graph

//图类

template<class T,class E>

class Graph

{

private:

VertexNode<T,E>\* \_vertex\_first; //顶点链表

EdgeNode<T, E>\* \_edge\_first; //边列表

int vertex\_num; //顶点的个数

protected:

bool findVertex(T vertex) const; //查找名为vertex的结点是否存在

bool findMinEdge(EdgeNode<T, E>\*& edge); //寻找最短边

public:

Graph(int length=0);

~Graph();

bool insertVertex(T vertex1); //插入一个名为vertex的顶点

bool insertEdge(T vertex1, T vertex2, E value); //用现有的两个顶点vertex1和vertex2来构造一个权值为value的边

bool createMinTree(Queue<T,E>& queue); //生成最小生成树

bool isSet()const { return(!(NULL == this->\_vertex\_first->\_next)); }; //是否已经创建图

friend class UFSet<T,E>;

};

2.3.4队列类 Queue

//以边结点为内容的队列

template<class T, class E>

class Queue

{

private:

EdgeNode<T, E>\* \_first;

EdgeNode<T, E>\* \_last;

public:

Queue();

~Queue();

void push(EdgeNode<T, E>\* edge);

bool pop(EdgeNode<T,E>\* & edge);

void printNode();

bool isSet()const { return(!(NULL == \_first->\_next)); };

};

2.3.5并查集UFSet

//并查集

template<class T,class E>

class UFSet

{

private:

int max\_length;

int\* parent\_array;

T\* data\_array;

protected:

bool getNodePos(T name,int &pos);

public:

UFSet();

UFSet(int length,Graph<T,E>\*graph);

~UFSet();

bool merge(EdgeNode<T,E>\*& node);

bool isConnected()const; //所有的结点是否均被连接起来0

};

2.4文件设计

本系统支持windows系统和Linux系统。

(1)Windows系统

windows系统的相关文件在文件夹08\_1953608\_WuYinghao\_Windows中，

其中类的定义以及其成员函数的定义在08\_1953608\_WuYinghao.h中;main函数以及相关的函数定义在08\_1953608\_WuYinghao.cpp中.08\_1953608\_WuYinghao.exe为Windows平台下的可执行文件。

**08\_1953608\_WuYinghao.cpp和08\_1953608\_WuYinghao.h的编码格式均为**

**简体中文(GB2312)-代码页 936**

(2)Linux 系统

Linux系统的相关文件在文件夹08\_1953608\_WuYinghao\_Linux中,

其中类的定义以及其成员函数的定义在08\_1953608\_WuYinghao\_Linux.h中;main函数以及相关的函数定义在08\_1953608\_WuYinghao\_Linux.cpp中.08\_1953608\_WuYinghao.out为Linux平台下的可执行文件。

**08\_1953608\_WuYinghao\_Linux.cpp和08\_1953608\_WuYinghao\_Linux.h的编码格式均为Unicode(UTF-8 无签名)-代码页65001**

3.实现

3.1算法设计

3.1.1 算法的文字表述

采用Kruskal算法借助并查集来构造最小生成树。首先选择权值最小的边作为最小生成树的一个元素，将这一条边的两个顶点存入并查集中(具有相同的根结点)。随后按照边权值从小到大的顺序分析每一条边，若这一条边的顶点已经在并查集中，则不能选择这一条边(否则会生成回路)。否则将这一条边进入最小生成树，这条边的顶点存入同一个并查集中。

重复上述操作知道对于有n条边的图，其最小生成树的边为n-1.最终得到的结果即为最小生成树。

3.1.2 核心代码展示

template<class T,class E>

bool Graph<T,E>::createMinTree(Queue<T,E>& queue)

{

EdgeNode<T, E>\* minEdge=new EdgeNode<T,E>;//最小边

EdgeNode<T, E>\* edge = this->\_edge\_first;

UFSet<T,E> ufset( vertex\_num,this);

while(!ufset.isConnected())

{

if (!findMinEdge(minEdge)) //找不到最小边

{

delete minEdge;

return false;

}

else

{

if (ufset.merge(minEdge)) //合并成功，入队

{

queue.push(minEdge);

}

}

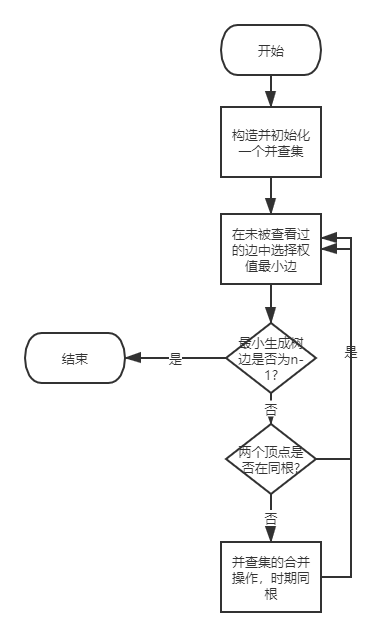
}

delete minEdge;

return true;

}

3.1.3 流程图展示



3.2 构造图

3.2.1 功能介绍

构造图的操作需要用户输入一些数据，首先输入顶点个数和每一个顶点的名称，然后输入边。在输入边的时候需要用户输入边的两个顶点名称和边的权值。如果边的顶点未出现在图的顶点集中，系统会给出相应的错误提示。

图中的顶点集合以及边集合均用链表的数据结构表示。

**特别强调:**

由于输入的边的个数完全是由用户自定义的，边的个数可以为任意值，这就需要一个结束标志，标志着边输入的结束。

此系统的结束表达式格式为:? ? n

前两个字符中需要输入至少一个英文问号?,第三个数为大于0的正整数.例如 ? ? 1即可退出边的输入

3.2.2 核心代码展示

(1)创建并插入顶点

template<class T,class E>

bool Graph<T, E>::insertVertex(T vertex)

{

if (!findVertex(vertex)) //如果没有查到这个结点

{

try

{

VertexNode<T,E>\* newNode=newVertexNode<T,E>(vertex);

newNode->\_next = \_vertex\_first->\_next;

\_vertex\_first->\_next = newNode;

vertex\_num++;

return true;

}

catch (bad\_alloc& ba)

{

cerr<<"无法创建顶点"<<endl<<ba.what()<<endl;

system("pause");

exit(1);

}

}

else

{

return false;

}

}

(2)创建并插入边

template<class T,class E>

bool Graph<T, E>::findVertex(T vertex) const

{

VertexNode<T,E>\* current = \_vertex\_first->\_next;

while (current)

{

if (current->\_name == vertex)

{

return true;

}

else

{

current = current->\_next;

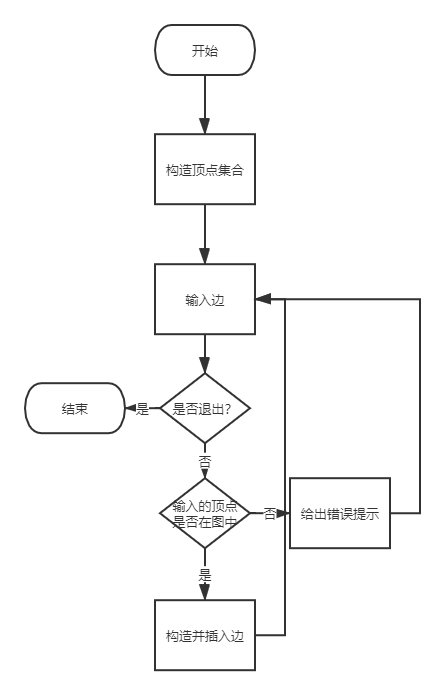
}

}

return false;

}

3.2.3 流程图展示



3.3 并查集的合并

3.3.1功能介绍

采用Kruskal算法构造最小生成树时,涉及并查集的合并操作，本系统采用将子孙结点较少的树合并到子孙结点较多的结点的方法。

并查集采用数组parent\_array来存储对应位置结点的父节点，若某一个结点位置i已经为根结点，则parent\_array[i]为一个负值这个负值的绝对值与1的差值表示以这个结点为根的并查集的元素个数，例如-3表示这个集合一共有3个元素，这个位置上的结点有两个子孙结点。

在向最小生成树中插入某条边的时候，首先判断这条边的两个顶点是否在同一个并查集内(是否同根),如果在同一个并查集中，则不能插入这条边。若不在同一个并查集中，则可以进行合并。首先通过数组parent\_array找到其根结点的子节点(parent\_array[i]为正),在判断parent\_array[parent\_array[i]]的绝对值的大小，绝对值大说明这个并查集的元素多。

合并操作:个数多的并查集根结点的值+=个数少的并查集的根节点值，个数少的并查集的根结点作为个数多的并查集根结点的子节点(指向个数多的并查集的根结点即可)。

3.3.2核心代码展示

template<class T,class E>

bool UFSet<T,E>::merge(EdgeNode<T,E>\*& node)

{

int current, first, second;

if ((getNodePos(node->\_vertex\_name\_1, first)) && (getNodePos(node->\_vertex\_name\_2, second))) //两个位置都找到了

{

if (first >= 0) //寻找根结点

{

while (parent\_array[first] >= 0)

{

first = parent\_array[first];

}

}

if (second >= 0) //寻找根结点

{

while (parent\_array[second] >= 0)

{

second = parent\_array[second];

}

}

if (first != second) //根结点不相同

{

if (parent\_array[first] <= parent\_array[second]) //first上的元素比较多,或相等,把second合并到first上

{

parent\_array[first] += parent\_array[second];

parent\_array[second] = first;

}

else

{

parent\_array[second] += parent\_array[first];

parent\_array[first] = second;

}

return true;

}

else //两个结点是同根的

{

return false;

}

}

else

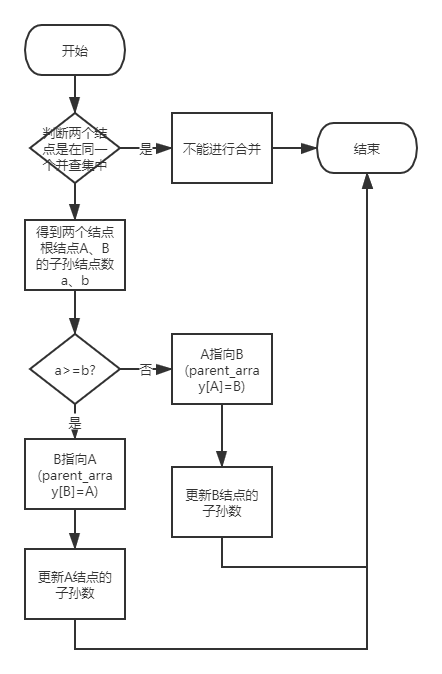
{

return false;

}

}

3.2.3 流程图展示

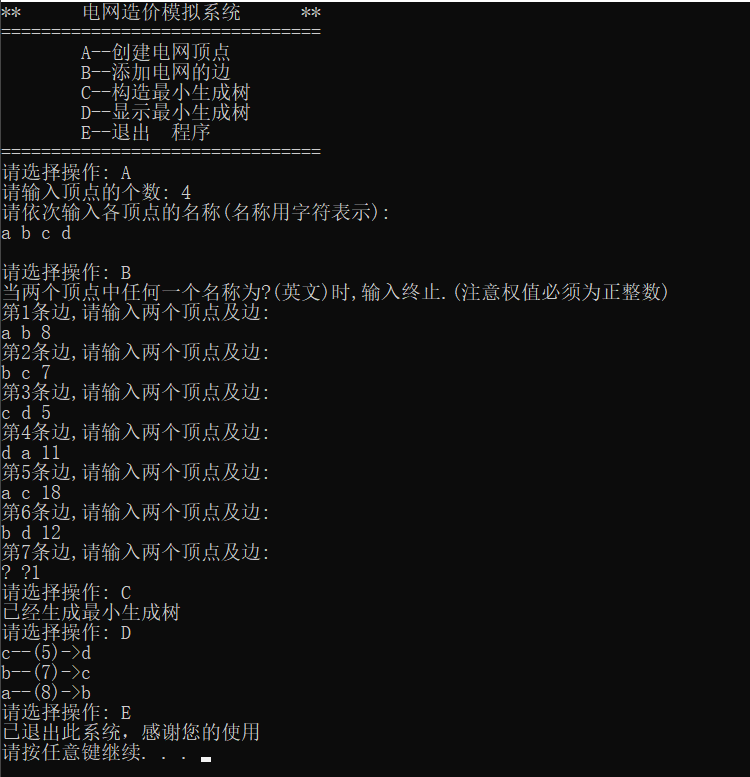


4.测试

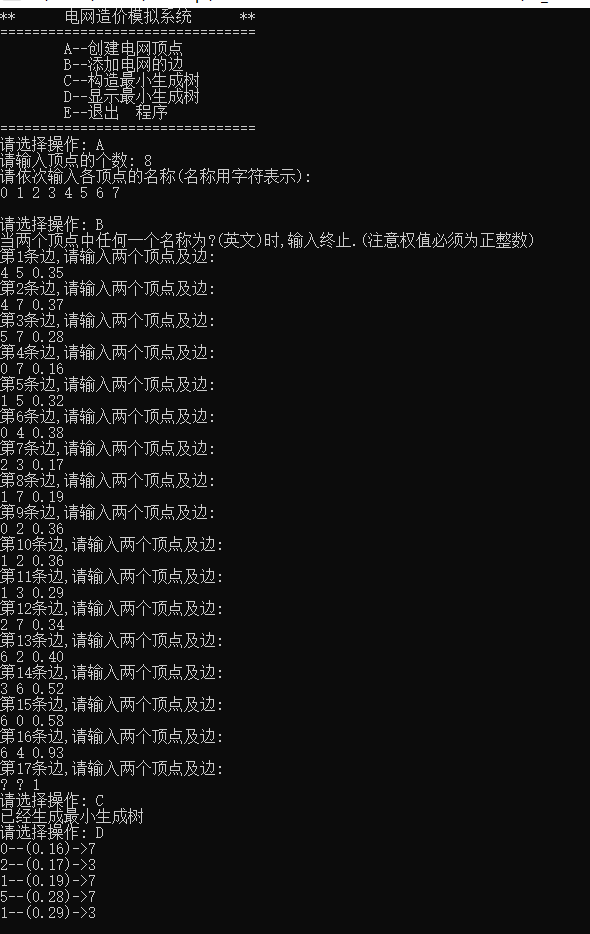
**4.1功能测试**

4.1.1 样例测试

测试题目中给出的样例



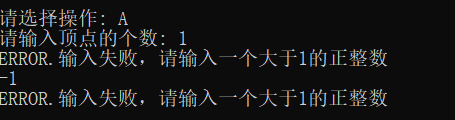
4.1.2 其他例子测试



4.2 边界测试

4.2.1 检测顶点个数输入是否合法

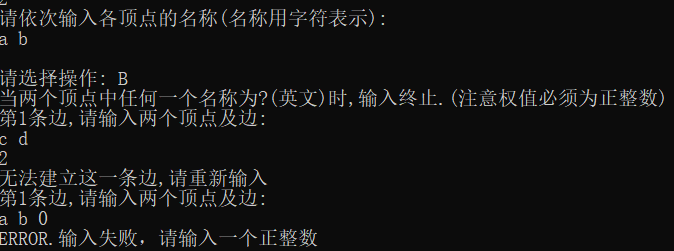
由于题目中涉及顶点个数的输入，考虑到图中顶点的个数既不能为负数，也不能为1（因为要构成一个边）。所以当用户输入不合法的数据时系统会给出相应的提示。



4.2.2 检测能否成功构造一条边

本系统中图构造步骤为:先构造顶点，再根据顶点构造边，当输入图中不存在的顶点时，或输入的边构成回路时，系统会给出相应的提示。

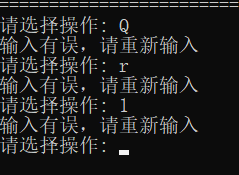
与此同时，此题拥有一定的实际情景，当输入边的权值为正数时，系统会给出相应的提示。



4.3 错误测试

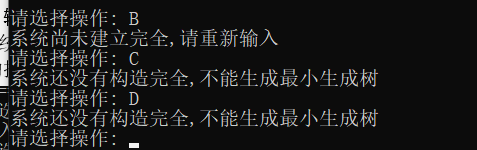
4.3.1 输入非法的指令

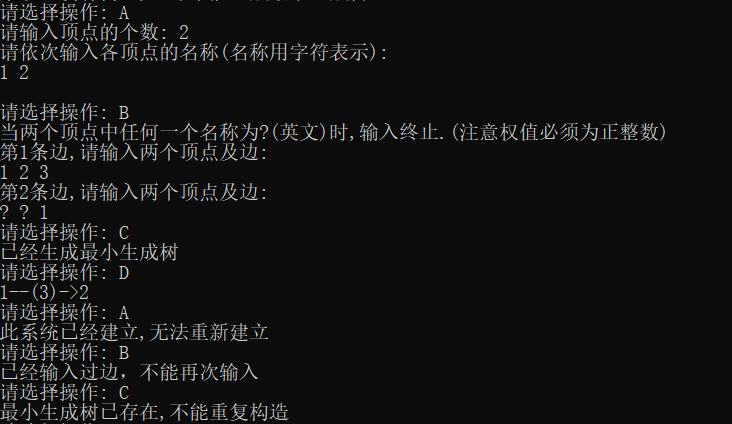
系统可供用户的指令有:A、B、C、D、E，若输入除此以外的命令，应该给出相应的提示。



4.3.2 未按照步骤输入指令

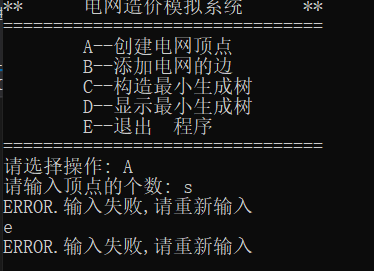
需要先输入顶点、边等才能构造最小生成树，不按照顺序进行操作，系统会给出相应的提示；当构造最小生成树完毕之后，无法再向图中插入顶点和边，否则系统会做相应的提示。





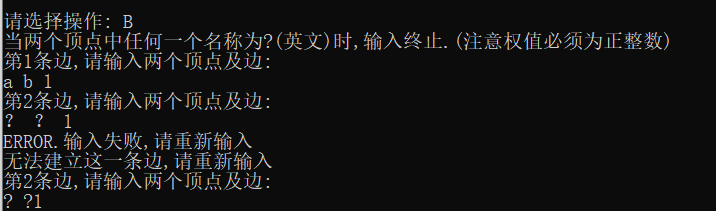
4.3.3 应该输入正整数时输入非法数据

由于题目中涉及输入顶点的个数，考虑到顶点的个数应该为大于1的正整数，所以输入非法的数据(例如字符)会产生错误，系统会给出相应的提示。



4.3.4 未按照正确的格式输入

边集合输入完毕时，需要输入标志字符? ? \*(\*为任意正整数).系统设置的输入结束符为英文的问号? 若输入中文问号？会给出一定的提示。



4.3.5 无法构造最小生成树

对于含有n的结点的图，其最小生成树应该有n-1条边，若用户输入的边个数小于n-1，则无法构造最小生成树，这时会给出相应的提示

