

数据结构课程设计

——电网建设造价模拟系统

项目说明文档

作 者 姓 名： 韦世贸

学 号： 2351131

指 导 老 师： 软件学院

专 业： 软件工程

同济大学

Tongji University

1 介绍

1.1 背景介绍

电网建设造价模拟系统一题先假设一个城市有n个小区，为了要实现n个小区之间的电网都能够相互接通，需要构造这个城市n个小区之间的电网，使总工程造价最低。要求设计一个能够满足要求的造价方案，这是一个极具现实意义的问题，实际应用较为广泛，此项目的的目的是生成网络的最小生成树，从而达到最小造价的目的。

1.2 功能介绍

作为一个最简易的电网建设造价模拟系统，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。从基础知识可以知道：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条使总的耗费最少就是这个项目需要解决的问题。

2 设计

2.1 数据结构设计

该本项目利用图的一般存储结构，即邻接表存储，存储下不通电网点之间的路径，再通过Prim算法生成最小生成树，也利用图的邻接表存储。最终再进行输出。

2.2 类结构设计

在该项目中创建了一个电网顶点类用于进行电网顶点的存储，一个电网线路类进行电网顶点之间关系的存储，最后的电网类实现对电网最小生成树的创建。其中电网顶点类是剩余两个类的友元类，电网类以及电网线路类互为友元方便进行数据调用。

2.3 成员与操作设计

电网顶点类（Node）

// 电网的顶点类

class Node

{

friend class Linked\_list;

friend class ElectricFrence;

public:

Node();

private:

bool isVisited; // 是否访问过

string startName; // 出发顶点名称

string currentName; // 当前顶点名称

double weight; // 边的权值

Node\* nextNode; // 指向下一个节点的指针

};

电网邻接表类（Linked\_list）

// 链表类，表示顶点的邻接表

class Linked\_list

{

friend class ElectricFrence;

public:

Linked\_list(); // 构造函数

void printMinimumSpanningTree(Node\* node); // 输出最小生成树

static void insertEdge(Linked\_list\* list, string nodeName, double pathWeight); // 插入边

static bool hasEdge(Linked\_list\* list, string nodeName); // 检查是否有边

static int countAdjacentNodes(Linked\_list& list); // 统计相邻顶点数

string nodeName; // 顶点名称

Node\* adjacentNode; // 相邻顶点指针

Linked\_list\* nextLink; // 下一个顶点链表指针

};

电网类

// 电网类，表示整个图

class ElectricFrence

{

friend class Linked\_list;

public:

void createVertices(ElectricFrence& grid); // 创建顶点

void insertEdges(ElectricFrence& grid); // 插入边

bool isVertexExist(ElectricFrence& grid, string name); // 判断顶点是否存在

void changeVisitedFlag(ElectricFrence& grid, string name); // 更改访问标志

int countVertices(ElectricFrence& grid); // 统计顶点数

Node\* findMinimumPath(ElectricFrence grid, Linked\_list& edgeList, string startName); // 找最小路径

void buildMinimumSpanningTree(ElectricFrence grid, Linked\_list& edgeList, string startName); // 构建最小生成树

private:

Linked\_list\* vertexHead;

};

3 实现

3.1 创建顶点功能的实现

3.1.1 创建顶点功能流程简介

创建顶点功能是由链表链接顶点并存储下来。

3.1.2 创建顶点功能核心代码

// 创建电网顶点

void ElectricFrence::createVertices(ElectricFrence& grid)

{

cout << "请输入顶点的个数：" << endl;

int n;

cin >> n;

cout << "请依次输入各顶点的名称：" << endl;

Linked\_list\* current = nullptr;

for (int i = 0; i < n; i++)

{

Linked\_list\* newList = new Linked\_list;

cin >> newList->nodeName;

if (i == 0)

{

grid.vertexHead = newList;

current = grid.vertexHead;

}

else

{

current->nextLink = newList;

current = newList;

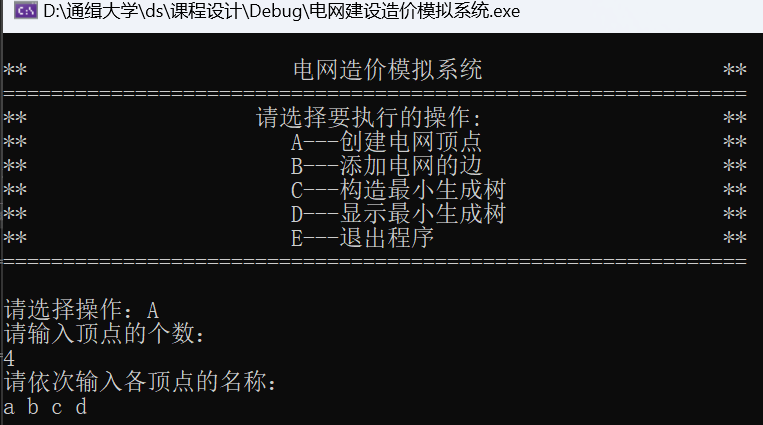
}

}

}

3.1.3创建顶点功能截屏示例

场景顶点功能展示，就是正常的输入链表内元素。



3.2 添加电网边的功能的实现

3.2.1 添加电网边的功能流程简介

添加电网边的功能先判断顶点是否存在，若存在则使用邻接表在顶点连出连接边的连接点，并且在连接结点上给出权值。

3.2.2 添加电网边的功能核心代码

// 插入边到电网中

void ElectricFrence::insertEdges(ElectricFrence& grid)

{

cout << "请输入两个顶点以及边的权值（输入? ? 0结束）：" << endl;

string name1, name2;

double pathWeight;

cin >> name1 >> name2 >> pathWeight;

if (name1 != "?" || name2 != "?" || pathWeight != 0)

{

if (isVertexExist(grid, name1) && isVertexExist(grid, name2))

{

Linked\_list\* current = grid.vertexHead;

while (current)

{

if (current->nodeName == name1)

{

Linked\_list::insertEdge(current, name2, pathWeight);

}

else if (current->nodeName == name2)

{

Linked\_list::insertEdge(current, name1, pathWeight);

}

current = current->nextLink;

}

insertEdges(grid);

}

else

{

cout << "错误的顶点，请重新输入：" << endl;

insertEdges(grid);

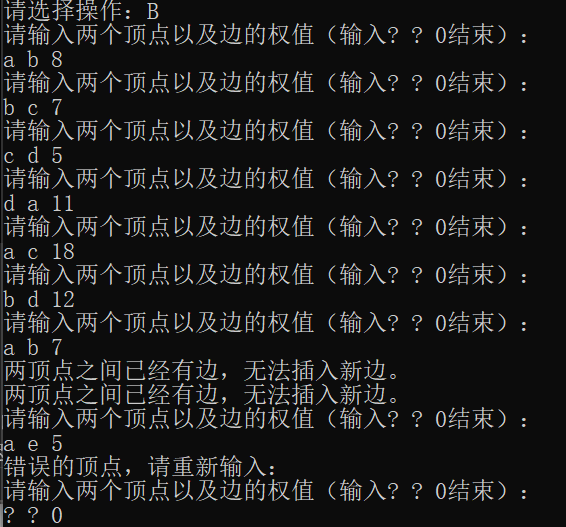
}

}

}

3.2.3 添加电网边的功能截屏示例

正常插入边展示，以及如果两顶点之间已经存在边的提示，输入顶点错误的提示，同时以？ ？ 0作为截止顶点输入。



3.3 最小生成树的功能的实现

3.3.1 最小生成树的功能流程简介

最小生成树的功能首先进行判定输入的第一个顶点名字是否存在，如果有效，建立起一个新节点为起始根节点，当最小生成树中的节点数量小于电网中的节点时进行循环，找出最小生成树中权值最小的与外相连的边并且逐步如此。

其实就是prim最小生成树算法：

将图的部分边去掉，可以将图转化为树，如果这棵树恰好是图能转化成的所有树中边权之和最小的，那么这棵树就是这个图的最小生成树。

Prim算法的思想是每次从已确定子图的所有邻接顶点中选择一个不会导致成环的、距离子图最近的顶点。

3.3.2 最小生成树的功能核心代码

// 构建最小生成树

void ElectricFrence::buildMinimumSpanningTree(ElectricFrence grid, Linked\_list& edgeList, string startName)

{

if (!isVertexExist(grid, startName))

{

cout << "错误的起始顶点，请重新输入：" << endl;

return;

}

Linked\_list\* current = grid.vertexHead;

while (current && current->nodeName != startName)

{

current = current->nextLink;

}

if (!current)

return;

Node\* startNode = new Node;

startNode->currentName = startName;

startNode->startName = startName;

startNode->weight = 0.0;

startNode->nextNode = edgeList.adjacentNode;

edgeList.adjacentNode = startNode;

changeVisitedFlag(grid, startName);

while (edgeList.countAdjacentNodes(edgeList) < grid.countVertices(grid))

{

Node\* minEdgeNode = nullptr;

Node\* tempEdge = edgeList.adjacentNode;

while (tempEdge)

{

Node\* minPathNode = findMinimumPath(grid, edgeList, tempEdge->currentName);

if (minPathNode && (!minEdgeNode || minPathNode->weight < minEdgeNode->weight))

minEdgeNode = minPathNode;

tempEdge = tempEdge->nextNode;

}

if (minEdgeNode)

{

Node\* newNode = new Node;

newNode->startName = minEdgeNode->startName;

newNode->currentName = minEdgeNode->currentName;

newNode->weight = minEdgeNode->weight;

newNode->nextNode = edgeList.adjacentNode;

edgeList.adjacentNode = newNode;

changeVisitedFlag(grid, newNode->currentName);

}

else

break;

}

}

打印最小生成树

// 输出最小生成树

void Linked\_list::printMinimumSpanningTree(Node\* node)

{

if (node->nextNode)

{

printMinimumSpanningTree(node->nextNode);

cout << node->startName << "-(" << node->weight << ")->" << node->currentName << " ";

}

else

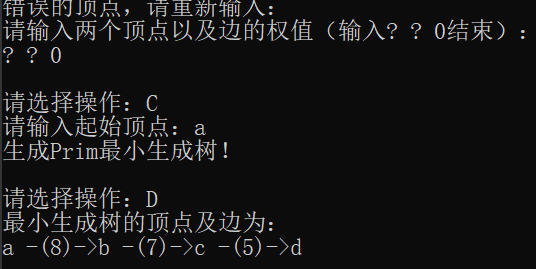
{

cout << node->currentName << " ";

}

}

3.3.3 最小生成树的功能截图示例



4 .设计小结

在这次电网造价系统的编程中我掌握了利用链表来储存图的邻接表，同时还复习了prim最小生成树算法：去掉图中的所有边，确定一个起始顶点，每次从已确定子图的所有邻接顶点中选择一个不会导致成环的、距离子图最近的顶点。也完成了要求，但其实也可以使用kruskal算法，这个算法需要并查集，堆和图的邻接矩阵，希望可以在以后的学习中使用这个算法解决问题。

但是这次的程序没有使用模板类，只能解决一种情况下的问题，希望以后编程还是多使用模板类。