电网建设造价模拟系统

1651573 刘客

功能介绍

- 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。
- 可以求解最短路径,得到一个耗价最少的方案。

编译说明

- 在windows平台下的.exe文件
- 在Linux平台下的out文件

一 . 设计

1. 数据结构设计

> 由题意,该问题可以抽象为求一个图的最短连通路径问题,故使用**prim最小生成树**来解题。

2. 类的设计

- ## EdgeNode 结构体
  - 成员变量

成员名称	属性	类型	描述
head	public	int	一条边的起点
tail	public	int	一条边的终点
weight	public	int	边的权值

- **GraphMatrx 带有邻接矩阵的无向图**
  - 成员变量

成员名称	属性	类型	描述
nodeVector	private	char*	存储发电站的名称(作为int到char的映射)
matrx	private	int**	邻接矩阵
nodeNum	private	int	节点数量
edgeNum	private	int	边的数量

  - 成员函数

函数名称	返回值类型	描述
GraphMatrx	无	构造函数
~GraphMatrx	无	析构函数
setNodeVector	void	设置顶点数组
getNode	int	将char转换为int的映射函数,其中int为vector数组的索引
getFirstNeighbor	int	获得当前顶点的第一个邻接点
getNextNeighbor	int	获得当前顶点在某个顶点后的邻接点

|getNextNeighbor|bool|插入边 |getWeight|int|获得一个边的权重 |getNodeNum|int|获得顶点数量 |getEdgeNum|int|获得边的数量 |isFullConnect|bool|在一定程度上判断当前输入的图是否连通|

二. 类的一些具体实现

GraphMatrx类

- 构造函数

> 为邻接矩阵matrx分配内存 将(i,j)全部设为MAX_INT(自己define的一个极大值)来代表边不存在

```
```
//初始化
    //param1 顶点个数
    GraphMatrx(int num) :nodeNum(num), edgeNum(0) {
        matrx = new int*[num + 1];
        for (int i = 0; i < num; i++) {
            *(matrx + i) = new int[num + 1];
        }
        for (int i = 0; i < num; i++) {
            for (int j = 0; j < num; j++) {
                matrx[i][j] = MAX_INT;
            }
```
```

```
        }
    }
```

- 析构函数

```
~GraphMatrx() {
        for (int i = 0; i < nodeNum; i++) {
            delete []matrx[i];
        }
        delete[] matrx;
    }
```

- 搜索第一个邻接点

> 若未找到则会返回-1

```
//直接从邻接矩阵中查找
    int getFirstNeighbor(int m) {
        for (int i = 0; i < nodeNum; i++) {
            if (matrx[m][i] != MAX_INT) {
                return i;
            }
        }
        return -1;
    }
```

- 搜索接下来的邻接点

> 同上

```
int getNextNeighbor(int m, int w) {
        bool isPass = false;
        for (int i = 0; i < nodeNum; i++) {
            if (isPass) {
                if (matrx[m][i] != MAX_INT) {
                    return i;
                }
            }
            if (i == w) {
                isPass = true;
            }

        }
        return -1;
    }
```

- 向邻接矩阵中添加边

> 包含了一些错误输入的警告,比如边已存在,或者边的信息不正确,或者不能构成连通图(部分情况下可以检测出来)等

```
    //return value
        //true -》 终止插入操作
        //false -》持续插入,错误信息在函数内部已经提示
        bool insertEdge(char fNode, char sNode, int weight) {
            if (fNode == '?' && sNode == '?' && weight == 0) {
                if (isFullConnect()) {
                    return true;
                }
                else {
                    cout << "您的各个电网之间并未全部连同,请确认您的节点是否正确\n";
                    return false;
                }
            }
            if (fNode == sNode) {
                cout << "错误的电网站点信息,请确认后在输入\n";
                return false;
            }
            int fIndex = getNode(fNode);
            int sIndex = getNode(sNode);
            //未知节点错误
            if (fIndex == -1 || sIndex == -1) {
                cerr << "错误的节点信息,请重新输入: ";
                return false;
            }
            //已有错误
            if (matrx[fIndex][sIndex] != MAX_INT || matrx[sIndex][fIndex] != MAX_INT)
{
                cerr << "这两个节点之间已经有了权重,请检查您的节点是否输入正确\n请重新输入节点:
";
                return false;
            }
            matrx[fIndex][sIndex] = weight;
            matrx[sIndex][fIndex] = weight;
            edgeNum++;
            return false;
        }
```

三. 主程序实现

- 函数

函数名称	返回值类型	描述
cout_screen	void	绘制操作台函数
primTree	void	生成prim树
main	int	主函数
cout_primTree	void	输出最小生成树
process	void	处理逻辑函数

- count_screen

```
void cout_screen() {
    cout << "**\t\t电网造价模拟系统\t\t**\n"
        << "=============================================\n"
        << "**\t\tA --- 创建电网顶点\t\t**\n"
        << "**\t\tB --- 添加电网的边\t\t**\n"
        << "**\t\tC --- 构造最小生成树\t\t**\n"
        << "**\t\tD --- 显示最小生成树\t\t**\n"
        << "**\t\tE --- 退出    程序\t\t**\n"
        << "=============================================\n\n";
}
```

- primTree

> NodeVector 存储得到的最小生成路径的信息
> graph 无向图
> ch 最小生成树的开始节点
> size 顶点的数量

```
void primTree(EdgeNode* NodeVector, GraphMatrx* graph, char ch,int size) {
    int nodeNum = graph->getNodeNum();
    int edgeNum = graph->getEdgeNum();
    int count = 0;//已经分离一个顶点(首顶点)
    int firstIndex = graph->getNode(ch);
    //初始化lowCost与nearVex数组
    int *lowCost = new int[size];
    int *nearVex = new int[size];
    for (int i = 0; i < size; i++) {
        if (firstIndex == i) {
            lowCost[i] = 0;
            nearVex[i] = -1;
        }
        else {
            lowCost[i] = MAX_INT;
            nearVex[i] = 0;
        }
    }

    while (count != size) {
        //先获取当前顶点的首个邻接点
        int tempIndex = graph->getFirstNeighbor(firstIndex);
        //如果该邻接点存在,则进入循环
        while (tempIndex != -1) {
            //获取两点间的权值
            int weight = graph->getWeight(firstIndex, tempIndex);
            //如果比lowCost中存的小,则更新lowCost和nearVex中的前驱节点
            if (weight < lowCost[tempIndex]) {
                lowCost[tempIndex] = weight;
                nearVex[tempIndex] = firstIndex;
            }
            tempIndex = graph->getNextNeighbor(firstIndex, tempIndex);
```

```
        }
        int min = MAX_INT;
        //从lowCost中选取本次循环过程中的最小值
        for (int i = 0; i < size; i++) {
            if (nearVex[i] != -1 && lowCost[i] < min) {
                firstIndex = i;
                min = lowCost[i];
            }
        }
        //将Edge添加进数组,并将该树外顶点加入树内,并且下次循环以该顶点开始
        EdgeNode tempNode;
        tempNode.head = nearVex[firstIndex];
        tempNode.tail = firstIndex;
        tempNode.weight = lowCost[firstIndex];
        nearVex[firstIndex] = -1;
        NodeVector[count++] = tempNode;
    }
    cout << "生成Prim最小生成树!\n\n";
    delete[] lowCost;
    delete[] nearVex;
}
```

- cout_primTree

```
void cout_primTree(EdgeNode* vector, int num, char* chV) {
    for (int i = 0; i < num; i++) {
        cout << chV[vector[i].head] << "-<" << vector[i].weight << ">-" <<
chV[vector[i].tail] << "\t\t\t\t";
        if ((i+1) % 3 == 0) {
            cout << "\n";
        }
    }
    cout << "\n";
    cout << "\n";
}
```

- process

> 在其中声明一些必要的变量,并使用while循环进行逻辑处理
> 在A中进行初始化
> 在B中进行输入边和权重操作
> 在C中进行prim运算
> 在D中进行输出prim树操作
> 在E中,将状态标志为false 退出循环
> 在default中给出错误输入提醒

```
 void process() {
     //以邻接矩阵的形式存储电站之间的长度
     GraphMatrx* powerGraph = NULL;
     char * nodeVector = NULL;
     int num;
     bool isA = false;
```

```
        bool isB = false;
        bool isC = false;
        bool isContinue = true;
        EdgeNode* edgeVector = NULL;
        while (isContinue) {
            cout << "请输入操作: ";
            char ch;
            cin >> ch;
            switch (ch)
            {
            case 'A':
                //如果已经进行A操作,给出提示
                if (isA) {
                    cout << "您已经建立电网,请进行其他操作\n";
                }
                else {
                    //读入操作
                    cout << "请输入顶点的个数:";

                    cin >> num;
                    cout << "请依次输入各顶点的名称:\n";
                    char tempCh;
                    powerGraph = new GraphMatrx(num);
                    nodeVector = new char[num + 1];
                    edgeVector = new EdgeNode[num + 1];
                    for (int i = 0; i < num; i++) {
                        cin >> tempCh;
                        nodeVector[i] = tempCh;
                    }
                    powerGraph->setNodeVector(nodeVector);
                    isA = true;
                }
                cout << "\n";
                break;
            case 'B': {
                if (!isA) {
                    cout << "请先进行A操作\n";
                    break;
                }
                int num = powerGraph->getNodeNum();
                char Lch;
                char Rch;
                int weight;
                do {
                    cout << "请输入两个顶点及边(ch,ch,int): ";
                    cin >> Lch >> Rch >> weight;
                } while (!powerGraph->insertEdge(Lch, Rch, weight));
                cout << "\n";
                isB = true;
                break;
                }
            case 'C':
                if (!isA) {
```

```
                cout << "请先进行A操作\n";
                break;
            }
            else if (!isB) {
                cout << "请先进行B操作\n";
                break;
            }
            //TODO prime操作
            cout << "请输入起始顶点: ";
            char ch;
            cin >> ch;
            primTree(edgeVector, powerGraph,ch,num);
            isC = true;
            break;
        case 'D':
            if (!isA) {
                cout << "请先进行A操作\n";
                break;
            }
            else if (!isB) {
                cout << "请先进行B操作\n";
                break;
            }
            else if (!isC) {
                cout << "请先进行C操作\n";
                break;
            }
            cout << "最小生成树的顶点及边为: \n\n";
            //TODO;
            cout_primTree(edgeVector, powerGraph->getNodeNum() - 1, nodeVector);
            break;
        case 'E':
            isContinue = false;
            break;
        default:
            cout << "错误的操作符,请确认后再输入\n";
            break;
        }
    }
    delete nodeVector;
    delete powerGraph;
    delete edgeVector;
}
```

四. 程序运行效果

- A操作
- B操作
- C操作
- D操作

五. 边界测试

- 操作逻辑错误时
    - 未创建电网节点就开始添加电网的边
    - 已经创建电网节点,仍继续创建
    - 其他指令也做了相应处理,此处不再展示
- 输入信息有误时
    - 在两个相同的节点之间建立一条边
    - 重复建立相同的边
    - 建立的边中有节点不存在
    - 在想要退出时所加入的边构不成连通图(部分情况可检测)