#include <iostream>  
#include <string>  
#include <vector>  
#include <utility>  
#include <queue>  
#include <functional>  
  
#define Max 10000  
using namespace std;  
  
/\*---最小生成树类---\*/  
class Tree {  
private:  
 int\*\* map; //地图  
 int num; //节点数目  
 vector <pair <string, int> > name; //节点名和对应的序号  
 vector <pair<int, int> > ans; //最小生成树序列  
public:  
 Tree() :map(NULL), num(0) {} //默认构造函数  
 ~Tree(); //析构函数  
  
 void A\_InputName(); //输入节点名称  
 void B\_InputMap(); //输入地图  
 void C\_buildTree(); //生成最小生成树  
 void D\_print(); //打印最小生成树  
};  
/\*---析构函数---\*/  
Tree::~Tree() {  
 if (map != NULL) {  
 for (int i = 0; i < num; i++) {  
 delete[] map[i];  
 }  
 delete[] map;  
 }  
}  
  
/\*---输入节点名称---\*/  
void Tree::A\_InputName() {  
 if (!ans.empty()) {  
 cout << "已生成最小生成树，非法操作！" << endl << endl;  
 return;  
 }  
 if (map != NULL) {  
 cout << "已生成地图， 非法操作！" << endl << endl;  
 return;  
 }  
 string temp;  
 cout << "请输入顶点的个数:";  
 cin >> num;  
 cout << "请依次输入顶点的名称:" << endl;  
  
 for (int i = 0; i < num; i++) {  
 cin >> temp;  
 name.push\_back(make\_pair(temp, i));  
 }  
 cout << endl;  
}  
/\*---输入地图---\*/  
void Tree::B\_InputMap() {  
 if (!ans.empty()) {  
 cout << "已生成最小生成树，非法操作！" << endl << endl;  
 return;  
 }  
 if (num == 1) {  
 cout << "非法城市个数！" << endl << endl;  
 return;  
 }  
  
 map = new int\* [num];  
 for (int i = 0; i < num; i++)  
 map[i] = new int[num];  
  
 for (int i = 0; i < num - 1; i++) {  
 cout << "请输入给出的两个顶点间的距离，不存在距离则输入-1: ";  
 map[i][i] = 0;  
 for (int k = i + 1; k < num; k++) {  
 cout << name[i].first << " ";  
 cout << name[k].first << " ";  
 cin >> map[i][k];  
 if (map[i][k] == -1)  
 map[i][k] = Max;  
 if (k != num - 1)  
 cout << "请输入给出的两个顶点间的距离，不存在距离则输入-1: ";  
 }  
 }  
 for (int i = 0; i < num; i++) {  
 map[i][i] = 0;  
 }  
 for (int i = 0; i < num; i++) {  
 for (int k = 0; k < i; k++) {  
 map[i][k] = map[k][i];  
 }  
 }  
 cout << endl;  
}  
/\*---生成最小生成树---\*/  
void Tree::C\_buildTree() {  
 if (num == 0 || map == nullptr) {  
 cout << "请先输入节点名称和地图信息。" << endl;  
 return;  
 }  
 ans.clear();  
 vector<bool> visited(num, false); // 添加一个用于跟踪访问过的顶点的向量  
 priority\_queue<pair<int, pair<int, int>>,  
 vector<pair<int, pair<int, int>>>,  
 greater<pair<int, pair<int, int>>>> que;  
  
 // 添加第一个节点到优先队列  
 string tempname;  
 cout << "请输入起始顶点：";  
 cin >> tempname;  
 int startIndex = -1;  
 for (int i = 0; i < num; i++) {  
 if (tempname == name[i].first) {  
 startIndex = i;  
 visited[i] = true;  
 break;  
 }  
 }  
 if (startIndex == -1) {  
 cout << "未找到起始顶点。" << endl;  
 return;  
 }  
  
 // 从起始顶点添加边到优先队列  
 for (int i = 0; i < num; i++) {  
 if (i != startIndex) {  
 que.push({ map[startIndex][i], {startIndex, i} });  
 }  
 }  
  
 // Prim算法的主循环  
 while (!que.empty()) {  
 auto edge = que.top();  
 que.pop();  
  
 int to = edge.second.second;  
 if (visited[to]) continue; // 跳过已经访问的节点  
  
 visited[to] = true; // 标记节点已访问  
 ans.push\_back(edge.second); // 添加到结果中  
  
 // 将所有连接的未访问节点的边加入到优先队列  
 for (int i = 0; i < num; i++) {  
 if (!visited[i]) {  
 que.push({ map[to][i], {to, i} });  
 }  
 }  
 }  
  
 cout << "生成Prim最小生成树！" << endl << endl;  
}  
  
/\*---打印最小生成树---\*/  
void Tree::D\_print() {  
 string name1, name2;  
 cout << "最小生成树的顶点以及边为：" << endl;  
 for (int i = 0; i < ans.size(); i++) {  
  
 for (int k = 0; k < name.size(); k++) {  
 if (ans[i].first == name[k].second)  
 name1 = name[k].first;  
 if (ans[i].second == name[k].second)  
 name2 = name[k].first;  
 }  
  
 cout << name1 << "-(" << map[ans[i].first][ans[i].second]  
 << ")-" << name2 << " ";  
 }  
 cout << endl << endl;  
}  
  
int main() {  
 cout << "\*\* 电网造价模拟系统 \*\*" << endl;  
 cout << "==============================================================" << endl;  
 cout << "\*\* A---创建电网定点 \*\*" << endl;  
 cout << "\*\* B---添加电网的边 \*\*" << endl;  
 cout << "\*\* C---构造最小生成树 \*\*" << endl;  
 cout << "\*\* D---显示最小生成树 \*\*" << endl;  
 cout << "\*\* E---退出程序 \*\*" << endl;  
 cout << "==============================================================" << endl;  
  
 char order;  
 Tree tree;  
  
 //Node \* p = tree.create();  
  
 cout << "请选择操作:";  
 cin >> order;  
  
 while (order != 'E') {  
 switch (order) {  
 case 'A':  
 tree.A\_InputName();  
 break;  
 case 'B':  
 tree.B\_InputMap();  
 break;  
 case 'C':  
 tree.C\_buildTree();  
 break;  
 case 'D':  
 tree.D\_print();  
 break;  
 default:  
 break;  
 }  
 cout << "请选择操作:";  
 cin >> order;  
 }  
 return 0;  
}