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：陈垲昕

学　　号：[redacted]

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

可通过按住 Ctrl 并单击访问说明文档各个模块：

1. 分析

1.1 背景分析

图是一种非线性的数据结构，可用于解决许多现实问题。 图可以用于表示网络，包括城市或电话网络或电路网络中的路径。 图还广泛使用于于诸如 linkedIn，Facebook 之类的社交网络。 例如，在 Facebook 中，每个人都用 vertex（或 node）表示。 每个节点都是一个结构，并包含诸如人的身分，姓名，性别，语言环境等信息。

本题目是一个经典的图论问题：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

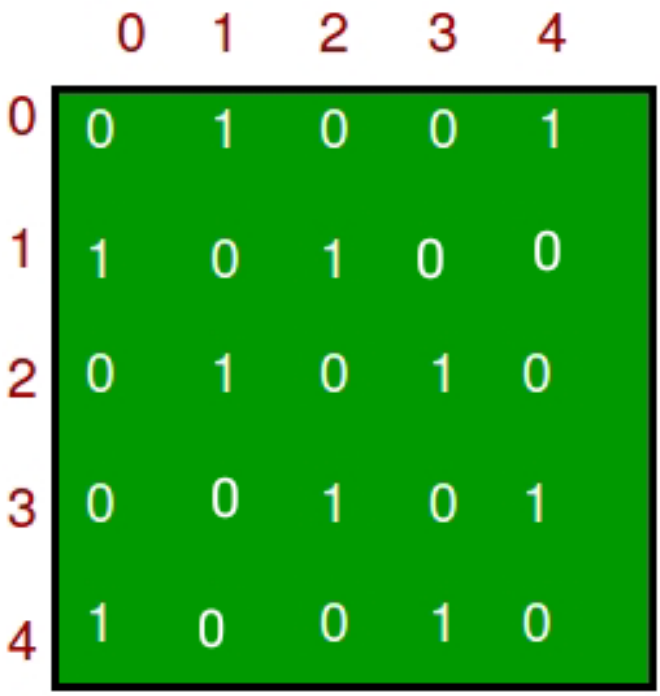
构建 n 个小区间连通的造价最小的电网线路，是典型的图论中对无向图生成最小生成树的问题。故需要设计的数据结构有：1.一个图的存储结构，使用邻接矩阵或邻接表；2.一个最小生成树数据结构，存储该图的最小生成树的信息；3.一个系统类，用于程序的运行以及最小生成树的生成算法。

要表示图这种数据结构，我们需要存储以下信息：1、 有限的一组顶点；2、 数量有限的一组有序对，就是图的边。

存储这样的信息，可以用图的两种基本数据结构：邻接矩阵与邻接表：

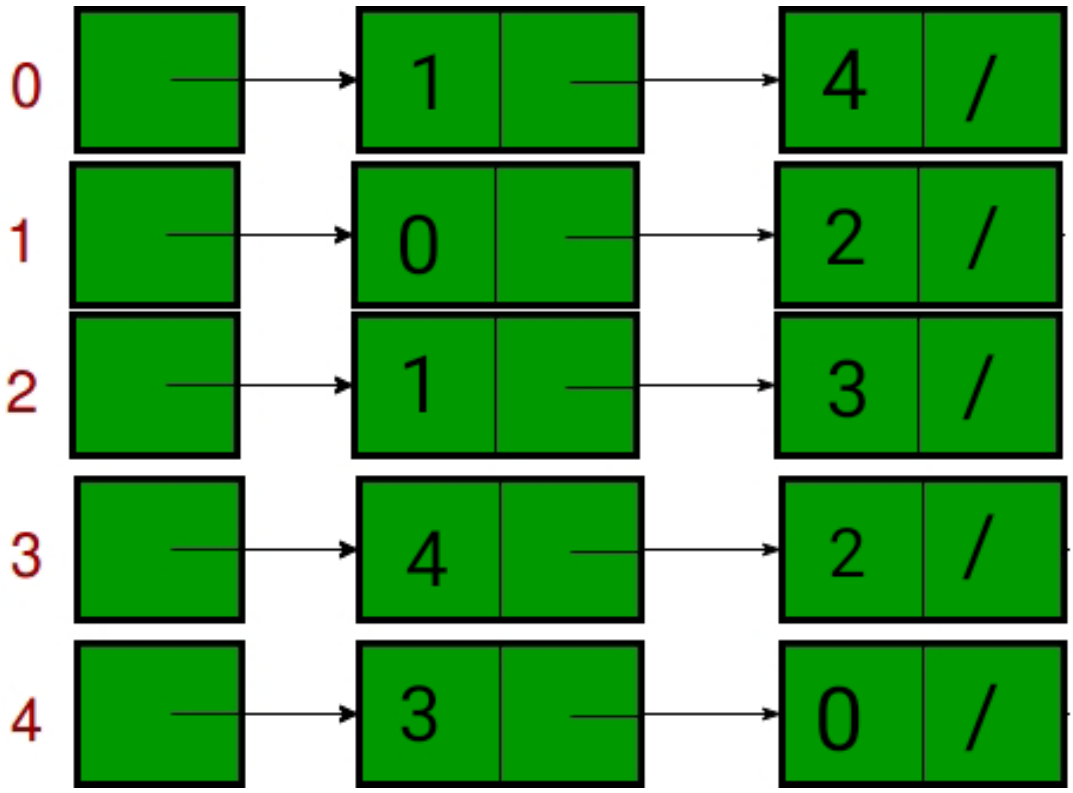
邻接矩阵

邻接矩阵的设计利用了图的点边关系的交互。对所有的顶点，显然可以使用一个一维数组存储，而边的构建需要的信息有：两个邻接顶点，以及有权情况下边的权值。故可考虑用二维数组表示边的关系。故邻接矩阵可设计为行列表示顶点编号的矩阵（行列表示的点的含义有所不同，一个是入点，一个是出点），而连接两个顶点的边即可用两个分别代表行与列的索引表示，在无权的情况下可用 bool 类型或 0,1 表示两点之间有无边，在有权的情况下，可用当前边的权值为单元赋值。



邻接表

邻接表的基本原理是以一维数组与若干一维链表的形式存储图的信息，对顶点而言，由于实际运用中直接访问顶点的操作被调用的次数较为频繁，故将顶点采用一维数组的方式存储，有多少个顶点便开辟多少空间的一维数组。对于边而言，为了避免使用一维数组分配时造成的空间浪费，故使用了链表的形式存储。对顶点数组的每一个顶点，都添加一个指向边结点的指针域作为边的链表的起始，链表中的每一个边即是与对应顶点相接的边，边结点中存储的数据主要有：1、边上另外一个顶点的信息（通常存储该点在数组中的索引值），2、下一条边的边结点的指针域，3、（在有权图中）边所带有的权值。



通过 **EletricityNetworkSimulator** 类实现程序的主体，存储一个邻接表图成员与对应的最小生成树成员，并通过 prim 算法实现最小生成树的生成

2.1 图存储数据结构设计

1.表示边的数据结构

本程序定义了以**<Name,Weight>**为模板参数的边存储数据结构，包括终点的索引，边的权值，下一条边结点的指针域。

```
template<class Name, class Weight>
struct Edge {
	int destination;                    //终点的索引
	Weight cost;                        //权值
	Edge<Name, Weight>* next;           //下一条边


	Edge() :next(nullptr) {}
	Edge(int n, Weight weight);
	bool operator !=(const Edge<Name, Weight>& rhs)const;

	bool operator ==(const Edge<Name, Weight>& rhs)const;
```

2.表示点的数据结构

本程序定义了以**<Name,Weight>**为模板参数的点存储数据结构，包括点的名字，以及一个边结点的链表域。

```
template<class Name, class Weight>
struct Point {
	Name data;                          //名字域
	Edge<Name, Weight>* edgeList;       //边的头指针域
};
```

3.图的存储结构

本程序定义了以**<Name,Weight>**为模板参数的图存储数据结构，以邻接表为主要形式，存储了一个表示点域的动态数组指针，当前开辟的点域最大空间，当前存储的点的个数，边的个数；成员函数方面，实现了构造与析构函数，获取点与边的权重，插入点与边的操作。

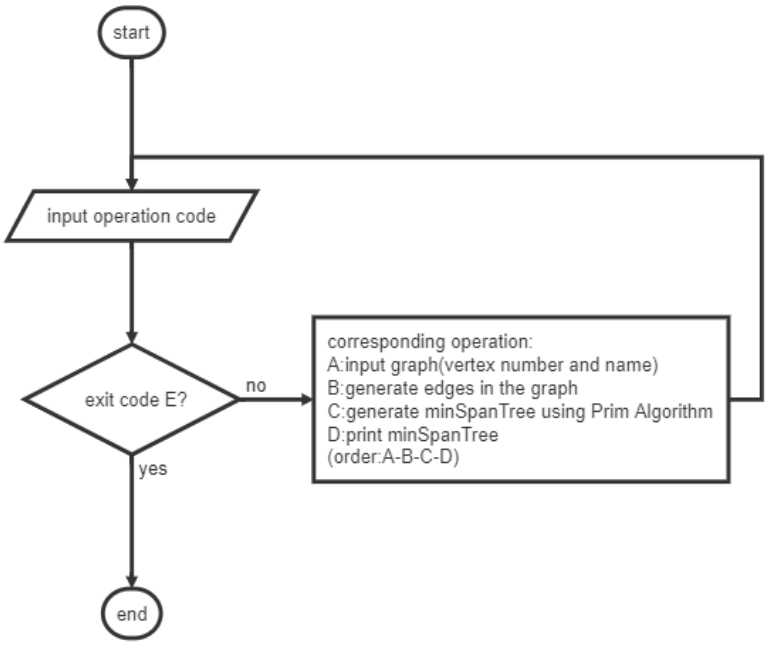
```
template<class Name, class Weight>
class Graph {
public:
	//构造与析构函数
	Graph(int sz);
	~Graph();

	//信息获取:点，边权重
     Name getPointData(int i) { return (i >= 0 && i <= _currPointNum) ?
 _pointArr[i].data : "NONE"; }
	Weight getEdgeWeight(int v1, int v2);
	int getPointPos(const Name& pointData);
	int getCurrPointNum()const { return _currPointNum; }

	//插入删除点边操作
	bool insertPoint(const Name& pointData);
	bool insertEdge(int v1, int v2, Weight const);


private:
	Point<Name, Weight>* _pointArr;		//点的指针域（动态数组）
	int _maxPointNum;
	int _currEdgeNum;
	int _currPointNum;

};
```

2.2 系统类设计

通过 **EletricityNetworkSimulator** 类实现程序的主体，存储一个邻接表图成员与对应的最小生成树成员，并通过 prim 算法实现最小生成树的生成

```
class EletricityNetworkSimulator {
public:
	EletricityNetworkSimulator();
	void prim(const string& startPoint);
	void run();

private:
	Graph<string, double>* adjList;	//领接表图类
	MinSpanTree<string,double>* mst; //最小生成树
};
```

流程图



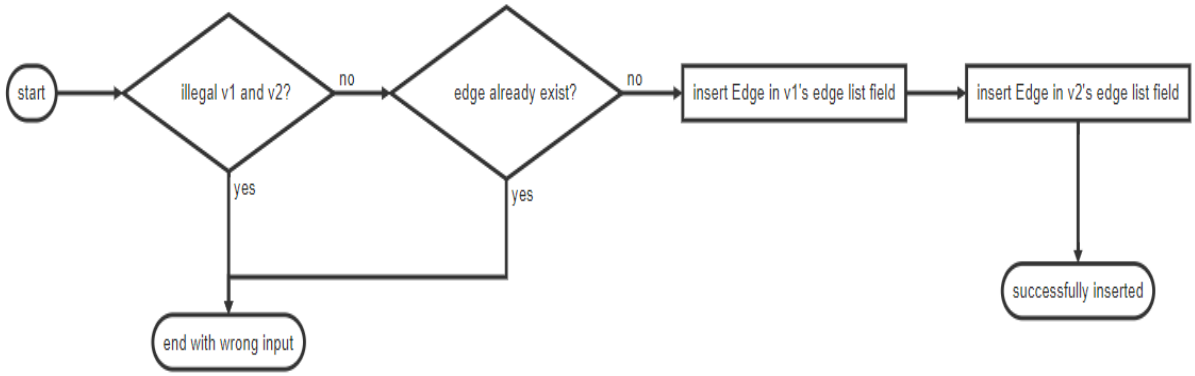
3.核心功能实现

3.1 邻接表的插入边操作

当电网模拟系统输入指令 B 时，执行为图（经过 A 指令后生成了若干顶点，顶点之间尚未通过带权边连接）插入边结点的操作。输入连接边的两个结点（记为 **v1** 与 **v2**）以及权值后，首先以其中一个顶点 **v1** 为遍历起始点，搜素该顶点的链表域中是否已经含有 **destination** 为 **v2** 的边（我们规定两点之间只能有一条电网线路连接），若搜索到了，提示该边已经存在于邻接表图中；若搜索不到，则代表可以插入，在该顶点的链表域尾部插入该边结点，同时由于该图为无向图，一个边结点所表示的边需要存储在对应两个点的指针域中，故继续定位到 **v2**，在 **v2** 的指针域头部插入该边结点（**destination** 为 **v1**）。此时就完成了邻接表边的插入操作

复杂度分析：对于邻接表而言，单纯插入边的操作即将边结点插入对应顶点链表域的第一个空间中，在有向图的时间复杂度为$O(1)$，但若需要预先检查是否该边已经插入，则需要遍历一遍该顶点的所有边，确认不存在后再插入边，故时间复杂度为$O(e)$

流程图



核心代码

```
Edge<Name, Weight>* pInsert;
```

```
		Edge<Name, Weight>* pFind = _pointArr[index1].edgeList;
		//找到(index1,index2)找得到即返回假
		while (pFind)
		{
			if (pFind->destination == index2)
			{
				cout << "当前需要插入的边已存在，无需重复插入" <<
endl;
				return false;
			}
			else
			{
				pFind = pFind->next;
			}
		}
		pFind = new Edge<Name, Weight>;
		pInsert = new Edge<Name, Weight>;//准备后续插入
		//index1 插入新边
		pFind->destination = index2;
		pFind->cost = weight;
		//插入头部
		pFind->next = _pointArr[index1].edgeList;
		_pointArr[index1].edgeList = pFind;

		//index2 插入新边
		pInsert->destination = index1;
		pInsert->cost = weight;
		//插入头部
		pInsert->next = _pointArr[index2].edgeList;
		_pointArr[index2].edgeList = pInsert;
```

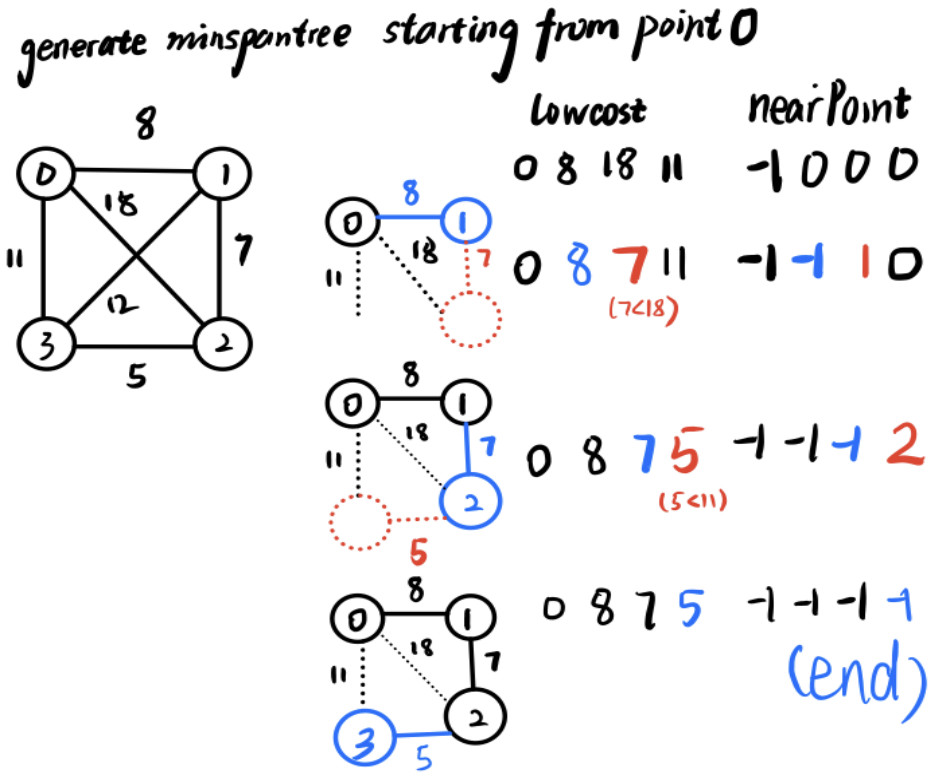
3.2 Prim 最小生成树算法

本程序设计涉及的核心算法是构造最小生成树的 Prim 算法。它是一种贪心算法，背后的思想很简单——生成树意味着所有的顶点都必须是连通的。因此，两个不相交的顶点子集(如上所述)必须互相连通，才能构成生成树，而它们必须与最小权边相连才能使它成为最小生成树。

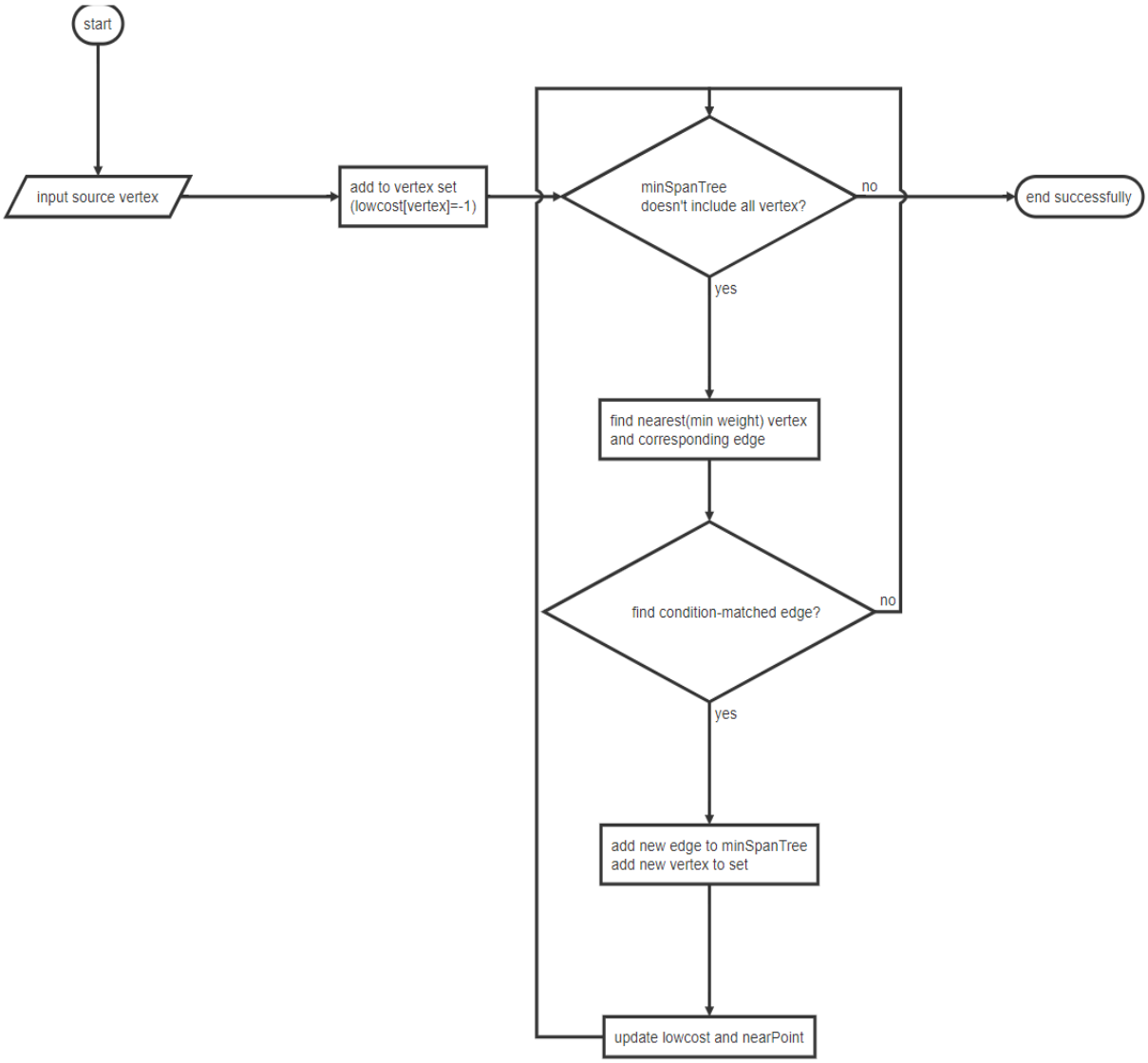
于是，我们从一个空的生成树开始，维护两个信息数组 **lowcost** 与 **nearPoint**，并且维护一个最小生成树 **minSpanTree**。第一个数组 **lowcost** 表示**生成树顶点集合内顶点 生成树顶点集合内顶点到生成树外各顶点的各边上的当前最小权值**，另一个数组 **nearPoint** 表示了**记录生成树顶点集合外各顶点 生成树顶点集合外各顶点距离集合内最近 (即权值最小)的顶点**。初始化条件中，置起始顶点入生成树点集合中（**lowcost** 置为-1）在每一次循环中，对不在顶点集合中的点，查找距离集合中的点权值最小的点并获取该边的权值,获取到该边后，放入最小生成树中。我们知道，该边一定与原来集合中的点相连，即一个顶点在原顶点集合内，另一个顶点原来不在顶点集合内，故此时将这个不在顶点集合中的顶点加入集合（同样的有对应的 **lowcost** 置为-1）。此后根据当前的图信息，更新 lowcost 的最小权值与 nearPoint 邻近点。

算法示例图

如图。显示了 lowcost 与 nearPoint 数组在最小生成树生成过程中的变化过程



流程图


start
input source vertex
add to vertex set
(lowcost[vertex]=-1)
minSpanTree
doesn't include all vertex?
no
end successfully
yes
find nearest(min weight) vertex
and corresponding edge
find condition-matched edge?
no
yes
add new edge to minSpanTree
add new vertex to set
update lowcost and nearPoint

核心代码

初始化 lowcost 与 nearPoint 数组

```
for (int i = 0; i < pointSum; i++)
{
	//lowcost[i]为索引 i 的点到startIndex 点的边的权值（无边时为DBL_MAX）
	lowcost[i] = (i == startIndex) ? 0 : adjList->getEdgeWeight(start
Index, i);
	//nearPoint[i]为生成树集合中距离索引 i 的点最近的点
	nearPoint[i] = (i == startIndex) ? -1 : startIndex;
}
```

对不在集合中的点，查找距离集合中的点权值最小的点并获取该边的权值

```
for (int j = 0; j < pointSum; j++)
{
	if (nearPoint[j] != -1 && lowcost[j] < minCost)
	{
		newPointIndex = j; minCost = lowcost[j];
	}
}
```

添加入最小生成树，并置新的顶点 newPointIndex 于顶点集合中

```
e.head = name1;
e.tail = name2;
e.cost = minCost;
//边加入最小生成树
mst->insert(e);
//置目前的点为-1，加入集合
nearPoint[newPointIndex] = -1;
```

更新 lowcost 的最小权值与 nearPoint 邻近点

```
for (int j = 0; j < pointSum; j++)
{
     if (nearPoint[j] != -1 && adjList->getEdgeWeight(newPointIndex, j)
 < lowcost[j])
	{
		//更新 lowcost 的最小权值与 nearPoint 邻近点
		lowcost[j] = adjList->getEdgeWeight(newPointIndex, j);
		nearPoint[j] = newPointIndex;
	}
}
```

4.测试

4.1 常规结果测试

Task 1 示例测试

测试样例

见 `3.2 Prim` 算法代码示意图

预期结果

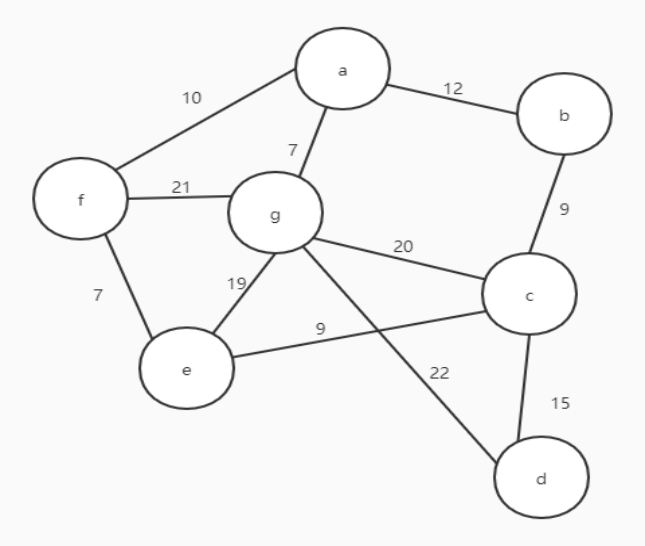
最小生成树的顶点及边：
`a-(8)->b  b-(7)->c  c-(5)->d`

实际结果：

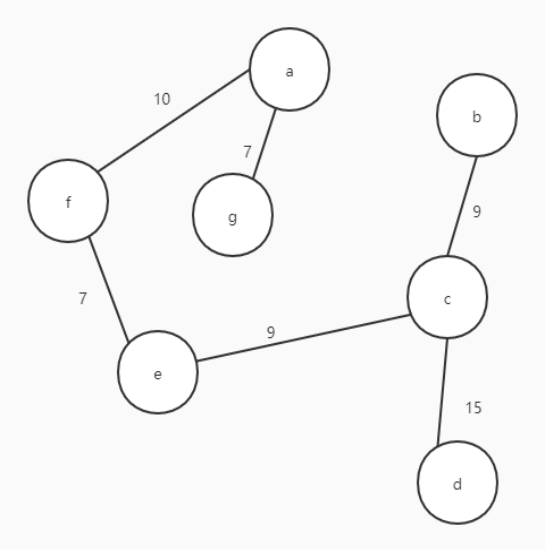
```
ckx@VM-0-10-ubuntu:[redacted]
**          电网造价模拟系统          **
==========================================
**          A --- 创建电网顶点        **
**          B --- 添加电网的边        **
**          C --- 构造最小生成树      **
**          D --- 显示最小生成树      **
**          E --- 退出程序            **
==========================================

请选择操作：A
请输入顶点的个数：4
请依次输入各顶点的名称：a b c d
已经建立含4个顶点的邻接表图

请选择操作：B
请输入两个顶点及边(边权值为负结束操作)：a b 8
请输入两个顶点及边(边权值为负结束操作)：b c 7
请输入两个顶点及边(边权值为负结束操作)：c d 5
请输入两个顶点及边(边权值为负结束操作)：d a 11
请输入两个顶点及边(边权值为负结束操作)：a c 18
请输入两个顶点及边(边权值为负结束操作)：b d 12
请输入两个顶点及边(边权值为负结束操作)：? ? -1

请选择操作：C
请输入起始起点
a
生成最小生成树
请选择操作：D
最小生成树的顶点及边：
a-(8)->b        b-(7)->c        c-(5)->d
请选择操作：E
程序已成功退出
```

Task 2 其他稍微复杂的图的测试

测试样例 1:输入如下图所示的图（以 g 为起点）



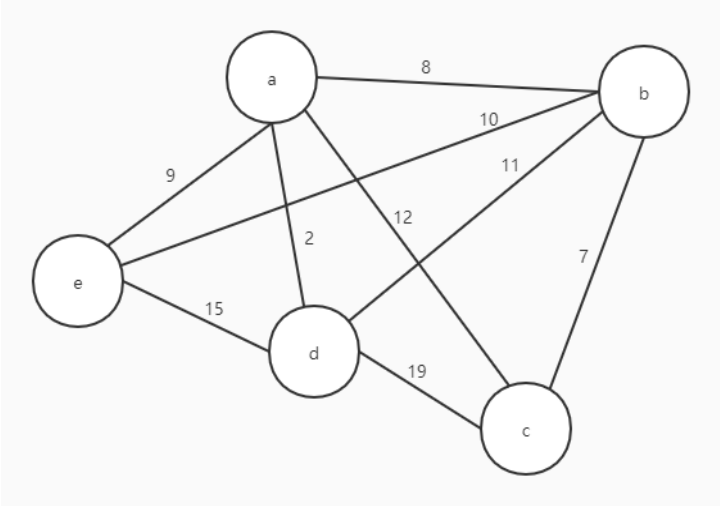
预期结果 1：输出如下图所示的最小生成树



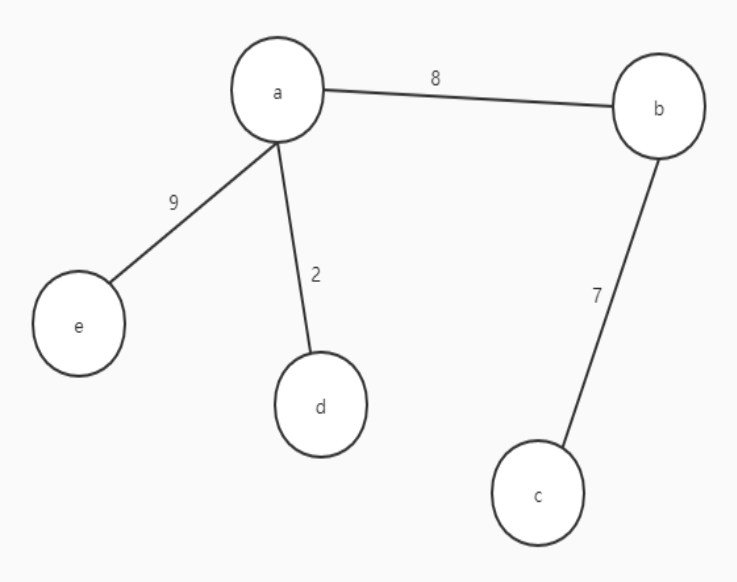
实际结果 1：

```
**            电网造价模拟系统            **
========================================
**            A --- 创建电网顶点          **
**            B --- 添加电网的边          **
**            C --- 构造最小生成树        **
**            D --- 显示最小生成树        **
**            E --- 退出程序              **
========================================

请选择操作：A
请输入顶点的个数：7
请依次输入各顶点的名称：a b c d e f g
已经建立含7个顶点的邻接表图

请选择操作：B
请输入两个顶点及边(边权值为负结束操作)：a b 12
请输入两个顶点及边(边权值为负结束操作)：a g 7
请输入两个顶点及边(边权值为负结束操作)：a f 10
请输入两个顶点及边(边权值为负结束操作)：b c 9
请输入两个顶点及边(边权值为负结束操作)：c g 20
请输入两个顶点及边(边权值为负结束操作)：c d 15
请输入两个顶点及边(边权值为负结束操作)：c e 9
请输入两个顶点及边(边权值为负结束操作)：e g 19
请输入两个顶点及边(边权值为负结束操作)：e f 7
请输入两个顶点及边(边权值为负结束操作)：d g 22
请输入两个顶点及边(边权值为负结束操作)：f g 21
请输入两个顶点及边(边权值为负结束操作)：? ? -1

请选择操作：C
请输入起始起点
g
生成最小生成树
请选择操作：D
最小生成树的顶点及边:
g-(7)->a        a-(10)->f       f-(7)->e        e-(9)->c        c-(9)->b        c-(15)->d
请选择操作：E
程序已成功退出
```

测试样例 2:输入如下图所示的图（以 b 为起点）



预期结果 2：输出如下图所示的最小生成树



实际结果 2：

```
ckx@VM-0-10-ubuntu:[redacted]
**          电网造价模拟系统          **
========================================
**          A --- 创建电网顶点         **
**          B --- 添加电网的边         **
**          C --- 构造最小生成树       **
**          D --- 显示最小生成树       **
**          E --- 退出程序             **
========================================

请选择操作：A
请输入顶点的个数：5
请依次输入各顶点的名称：a b c d e
已经建立含5个顶点的邻接表图

请选择操作：B
请输入两个顶点及边(边权值为负结束操作)：a b 8
请输入两个顶点及边(边权值为负结束操作)：a e 9
请输入两个顶点及边(边权值为负结束操作)：a d 2
请输入两个顶点及边(边权值为负结束操作)：a c 12
请输入两个顶点及边(边权值为负结束操作)：b e 10
请输入两个顶点及边(边权值为负结束操作)：b d 11
请输入两个顶点及边(边权值为负结束操作)：b c 7
请输入两个顶点及边(边权值为负结束操作)：c d 19
请输入两个顶点及边(边权值为负结束操作)：d e 15
请输入两个顶点及边(边权值为负结束操作)：? ? -1

请选择操作：C
请输入起始起点
b
生成最小生成树
请选择操作：D
最小生成树的顶点及边:
b-(7)->c        b-(8)->a        a-(2)->d        a-(9)->e
请选择操作：E
程序已成功退出
ckx@VM-0-10-ubuntu:[redacted]
```

4.2 边界测试

Task 1 图中只有两个顶点的图

测试样例：

插入顶点：a b
插入边：a b 1
选取生成起点： b

预期结果：

b-(1)->a

实际结果：

4.3 错误测试

Task 1 未定义的操作码

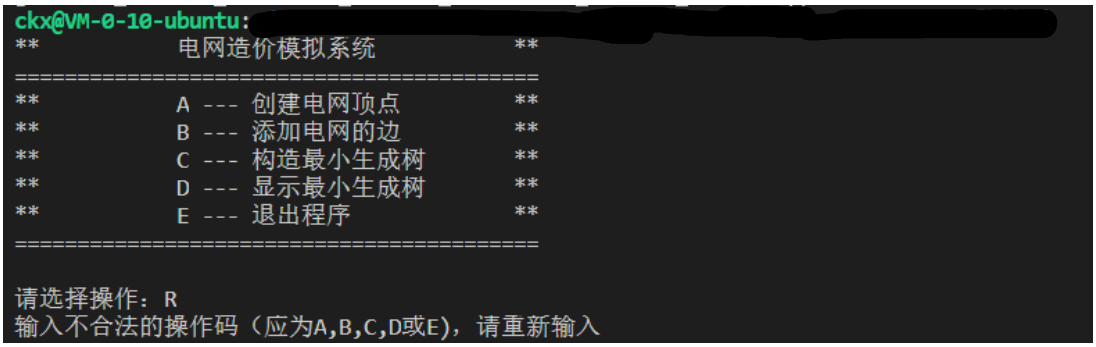
测试样例：

输入了 A,B,C,D,E 之外的操作码

预期结果：

提示操作码有误，要求用户重新输入

实际结果：



Task 2 输入的顶点数不足以建立图

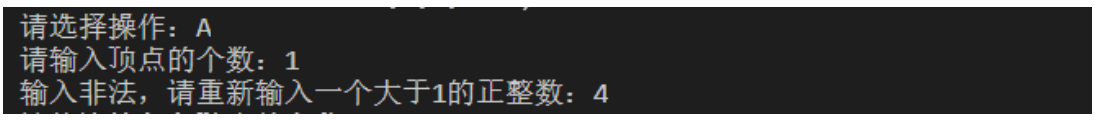
测试样例：

输入顶点数时 输入了小于 2 的数字

预期结果：

报错，要求重新输入

实际结果：



Task 3 输入了重名的顶点

测试样例：

插入顶点：a a b c

预期结果：

报错，要求重新输入各顶点的名称

实际结果：

```
请依次输入各顶点的名称：  a a b c
错误：输入了两个以上一样的顶点名字，请重新输入各顶点的名称
```

Task 4 两个顶点之间重复插入多条边

测试样例：

插入边:a c 7
插入边:a c 7

预期结果：

报错，提示当前两个顶点之间已经存在边，无需重复插入

实际结果：

```
请选择操作：B
请输入两个顶点及边(边权值为负结束操作)：a c 7
请输入两个顶点及边(边权值为负结束操作)：a c 7
当前需要插入的边已存在，无需重复插入
```

Task 5 插入的边对应的点不存在

测试样例：

原有顶点：a b c d
尝试插入：b f 3（f 不存在于原顶点序列中）

预期结果：

报错，提示顶点位置越界，要求用户重新输入

实际结果：

```
请选择操作：A
请输入顶点的个数：4
请依次输入各顶点的名称：a b c d
已经建立含4个顶点的邻接表图

请选择操作：B
请输入两个顶点及边(边权值为负结束操作)：b d 4
请输入两个顶点及边(边权值为负结束操作)：b f 3
错误：顶点位置越界，请重新输入
```