《离散数学》课程实验报告

4-最小生成树

一、题目背景与简介

最小生成树的概念：在给定连通无向图 G=(V，E)中，(u，v)代表连接顶点 u 与顶点 v 的边，而 w(u，v)代表此的边权重，若存在 T 为 E 的子集且为无循环图，使得的 w（T）最小（树中的边权和最小），则该 T 为 G 的最小生成树，也称最小权重生成树。

最小生成树有 3 个性质：

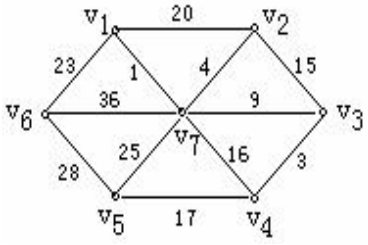
①最小生成树是树，因此其边数等于顶点数减 1，且树内一定不会有环。

②对给定的图 G(V,E)，其最小生成树可以不唯一，但其边权之和一定是唯一的。

③由于最小生成树是在无向图上生成的，因此其根结点可以是这棵树上的任意一个结点。

最小生成树可以通过 Kruskal 算法或 Prim 算法求出。这两个算法都是采用了贪心法的思想，只是贪心的策略不一样。

如下图所示的赋权图表示某七个城市{v_1，v_2，……v_7}，预先计算出它们之间的一些直接通信道路造价（单位：万元），如（v_1，v_2）=20。试给出一个设计方案，使得各城市之间既能够保持通信，又使得总造价最小，并计算其最小值。



（七个城市赋权图）

二、原理与核心算法

Prim 算法又称为“加点法”。该算法基于贪心法，每次找出距离最小生成树最近的边对所应的点，从某一个顶点 v_0 开始，逐渐将 n 个点纳入最小生成树中。

（1）设图中所有顶点的集合为 V，u 代表已经加入最小生成树的顶点的集合，v 代表未加入最小生成树的顶点的集合。由于从某点 v_0 开始，所以此时 u={v_0}，v=V-u。

（2）在两个集合 u,v 中选择一条权值最小的边，并将该边处于 v 中的端点 v_1 加入到集合 u 中，并且更新与 v_1 邻接的所有顶点

（3）重复上述步骤，直到最小生成树顶点集合 u 中包含所有顶点为止。

三、解题思路

3.1 涉及的主要变量

由题目可知，整个无向图以及边的权重存放可以用二维数组 Graph[N][N]表示，其中 N 为宏定义的顶点最大个数，并且由于是无向图，所以该二维数组对应的矩阵是对称阵（Graph[i][j]=Graph[j][i]）。而对于每一次找出距离最小生成树最近（造价最小）的顶点时，需要用到 cost[N]数组来存放剩余的点与已选中的点之间的最小造价，同时当其值变味 0 时，也意味着对应的点已被选中。上述变量以全局变量的形式出现。

由于需要进行造价的大小对比，所以将不可到达的城市之间用无穷大表示，在代码中表示为新增宏定义 INF 为 2147483647。

而在主函数体中，需要两个变量：vertex_num——存储顶点数（城市总数），edge_num——存储边数（道路总数），在程序一开始由用户进行输入。

3.2 输入函数以及错误处理

由于需要输入的变量较多，所以我打算分为两个部分进行输入以及错误的判断处理。

第一部分是顶点数和边数的输入，由 input_num()函数进行包装。两者都需要进行 cin.fail()的判定以及正负零的判定。但边数还要多一个检查，就是由于两个顶点确定一条边，边数最大只能为点数*（点数-1）÷2，所以超过就会产生重边，从而与题意不符合。

第二部分是每条边（道路）的端点（城市）和边权值（造价）的输入，我打算采取一次性输入，再用一个 bool 型变量 input_good 来判断是否正确输入。但凡有一个变量的一种情况出现了错误，则 input_good 置为 false，并统一重新该边的输入。

3.3 Prim 算法的实现部分

根据算法的原理不难进行理解。我们需要一个 int 型变量 total_cost 来计算最小耗费的总价值，当最小生成树出来之后，只需要依次进行叠加就行。另外还需要 int 型变量 min_cost 以筛选并存储已被处理的点和未被处理的点之间的最小造价，和 int 型变量 vertex 保存选中点的下标。

同样由于两点确定一条直线，所以我们覆盖到所有点要经历 vertex_num-1 轮选择，于是由 i 控制的外层循环从 1 变化到 vertex_nun-1。而内部，首先每一轮循环都要对最低造价 min_cost 进行初始化，并将当前的 vertex 作为下标传输到 disposed 中进行选中。随后由一个 j 控制的循环执行“新加入的点更新 cost”这一步骤，注意当 Graph[vertex][j]<cost[j]时才执行覆盖，随后不要忘了把自身造价的值置为 0。

在内部第一个循环结束后开始第二个循环，主要功能为在未被标记过的点中找出最低造价，并将该最低造价的下标覆盖 vertex，同时更新最新造价 min_cost 的内容，最终得到新的 vertex 和 min_cost。

进行第三个内部循环，是用来进行判断选中的最低造价道路的两端是那两个城市，其中的一个我们已经知道是 vertex，另一个就要在 Graph[vertex][j]中寻找与 min_cost 相等的下标 j，以便进行输出 j->vertex，并表明选中道路的造价。

最后的最后，将 total_cost 进行总结输出就完成该算法。

四、代码与运行结果

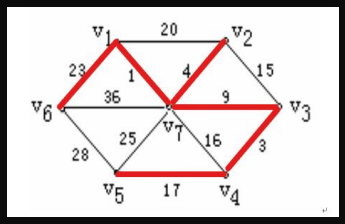
```
/*2151133 孙韩雅*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <iostream>
using namespace std;

#define N 100//宏定义点的最大个数为 100
#define INF 2147483647//宏定义一个 INF 表示无穷大 infinity
int Graph[N][N], cost[N];//用 Graph 存放题目图，cost 存放已选中点与未选中点之间的最小造价，同时也可显示已被处理的点

//工具函数
void init() {//进行初始化操作
	for (int i = 0; i < N; i++) {
		cost[i] = INF;//各邻接城市间道路造价置无穷大
		for (int j = 0; j < N; j++)
			Graph[i][j] = INF;//原图各道路造价置无穷大
	}
}
bool input_num(int &vertex_num,int &edge_num) {//用以输入顶点数和边数并进行错误判定和处理
PART1:
	cout << "请输入城市总数（输入 0 结束程序）：" << endl;//输入所求的顶点数
	cin >> vertex_num;
	if (cin.fail() || vertex_num < 0) {
		cout << "城市总数错误，请重新输入！" << endl << endl;

		cin.clear();
		cin.ignore(100, '\n');
		goto PART1;
	}
	else if (vertex_num == 0)
		return false;//程序结束
PART2:
	cout <<"请输入城市间通信道路总数（输入 0 结束程序）：" << endl;//输入所求的边数
	cin >> edge_num;
```

```
        if (cin.fail() || edge_num < 0) {
                cout << "通信道路总数错误，请重新输入！" << endl
<< endl;
                cin.clear();
                cin.ignore(100, '\n');
                goto PART2;
        }
        else if (edge_num > vertex_num * (vertex_num - 1) / 2) {//
进行重边的特判
                //注意：两个顶点确定一条边，所以边数最大为点数*
（点数-1）÷2
                cout << "出现重边，请重新输入！" << endl;
                cin.clear();
                cin.ignore(100, '\n');
                goto PART2;
        }
        else if (vertex_num == 0)
                return false;//程序结束
        return true;
}
void input_weight(int& vertex_num, int& edge_num) {
        int u = 1, v = 1, weight = INF;//边的两个端点u，v以及
该边权值weight
        bool input_good = true;//定义输入正误的判断变量，以便更
高效地完成输入错误判断与提示。

        for (int i = 1; i < edge_num + 1; i++) { //输入所有边的
权值，所以用edge_num控制循环
                cout << "请输入第" << i  << "条道路的两个端点城
市序号[1，" << vertex_num << "]以及该道路造价：";
                input_good = true;//每轮循环更新依次判断变量，以
保证输入的正确
                cin >> u;
                if (cin.fail() || u<1 || u>vertex_num)
                        input_good = false;

                cin >> v;
                if (cin.fail() || v<1 || v>vertex_num)
                        input_good = false;

                cin >> weight;
                if (cin.fail() || weight < 0)
                        input_good = false;
                if (!input_good) {//如果输入有误的情况出现
                        cout << "输入错误，请重新输入！" << endl <<
endl;
                        cin.clear();
                        cin.ignore(100, '\n');
                        i--;//重新开始本轮输入，相当于continue
                }
                Graph[u][v] = Graph[v][u] = weight;
        }
}
//Prim算法
void Prim(int& vertex_num){
        cout << "最小耗费所需的城市通信道路为：" << endl;
        int i = 0, j = 0;//循环控制变量
        int total_cost = 0;
        int min_cost = INF, vertex = 1;//最小造价与选中点的下标

        for (i = 1; i < vertex_num; i++) {//循环直到所有城市都
覆盖到
                min_cost = INF;//每轮循环都要将最低造价初始化
                for (j = 1; j < vertex_num + 1; j++) {
                        if (Graph[vertex][j] < cost[j])
                                cost[j] = Graph[vertex][j];//用新加
入的城市更新cost
                        cost[vertex] = 0;//因为选中该点，所以到自
身的造价为0
                }
                for (j = 1; j < vertex_num+1; j++) {
                        if (cost[j] && (cost[j] < min_cost)) {//
找出最低造价，注意寻找的顶点必须是未被标记过的
                                vertex = j;//覆盖当前顶点，找到最终
下一轮增加到树中的顶点
                                min_cost = cost[j];//更新最低造价，
最终得到最小值
                        }
                }
                for (j = 1; j < vertex_num; j++) {
                        if (Graph[vertex][j] == min_cost
&& !cost[j])//道路另一端是被选中的城市
                                break;
                }
                cout << j << "->" << vertex << ": " <<
Graph[j][vertex] << "万元" << endl;
                total_cost += Graph[j][vertex];
        }
```

```
        cout << "最小耗费共计" << total_cost << "万元" << endl;
}

int main(){
        int vertex_num = 0, edge_num = 0;//顶点数和边数
        while (input_num(vertex_num, edge_num)) {
                init();//首先进行初始化
                input_weight(vertex_num, edge_num);//输入每条道路的端点及造价
                Prim(vertex_num);//用 Prim 算法构建最小生成树
        }
        cout << "程序已退出！" << endl;
        return 0;
}
```

运行结果：

```
请输入城市间通信道路总数（输入0结束程序）：
12
请输入第1条道路的两个端点城市序号[1, 7]以及该道路造价：1 2 20
请输入第2条道路的两个端点城市序号[1, 7]以及该道路造价：2 3 15
请输入第3条道路的两个端点城市序号[1, 7]以及该道路造价：3 4 3
请输入第4条道路的两个端点城市序号[1, 7]以及该道路造价：4 5 17
请输入第5条道路的两个端点城市序号[1, 7]以及该道路造价：5 6 28
请输入第6条道路的两个端点城市序号[1, 7]以及该道路造价：6 1 23
请输入第7条道路的两个端点城市序号[1, 7]以及该道路造价：1 7 1
请输入第8条道路的两个端点城市序号[1, 7]以及该道路造价：2 7 4
请输入第9条道路的两个端点城市序号[1, 7]以及该道路造价：3 7 9
请输入第10条道路的两个端点城市序号[1, 7]以及该道路造价：4 7 16
请输入第11条道路的两个端点城市序号[1, 7]以及该道路造价：5 7 25
请输入第12条道路的两个端点城市序号[1, 7]以及该道路造价：6 7 36
最小耗费所需的城市通信道路为：
1->7：1万元
7->2：4万元
7->3：9万元
3->4：3万元
4->5：17万元
1->6：23万元
最小耗费共计57万元
请输入城市总数（输入0结束程序）：
0
程序已退出！
```



五、体会与心得

本题是我离散数学大作业里最后一个完成的题目，因为需要完成这个项目就得进行自学。老师课上将了 Kruskal 算法，我也算大致明白了，但是给出的示例代码用的是 Prim。对比之下，我还是选择阅读并自学 Prim 算法，因为从实现和操作层面来看，Prim 算法要比较简洁一些。也是因为该项目，让我熟悉并掌握了两种求解最小生成树的方法。

虽然在完成之后对算法有了了解，但是在编写的过程中，还是有些云里雾里的，这就导致开辟了一些重复功能的变量。比如我一开始还增加了 bool 型一位数组 disposed[N]来记录那些顶点时处理选中的，哪些是未选中的。但是在后来的检查中发现，该功能由 cost 数组在不断的更新中就可以实现，数组中 0 值对应的下标就是已被选中的顶点。所以在出血的过程中我们往往会有空间过度使用的情况，也是要在之后的检查中不断完善、不断简洁的。

另外就是要学会将主函数进行清洁，将一些输入操作放入工具函数中，这样会使得主函数更加清晰。因为主函数是用来进行思维的清理与执行的，具体的执行内容交给具体的功能函数做，所以不应太过繁杂。