《离散数学》课程实验报告 4 最小生成树

2253893 苗君文 软件工程

1. 题目简介

1.1. 背景与目的

图是由节点（或顶点）和边组成的数据结构，节点表示图中的对象，边表示对象之间的关系。最小生成树是一个连通图的生成树（Spanning Tree）中，边的权重之和最小的生成树。生成树是一个树，它包含图中的所有顶点，并且通过边将这些顶点相连而形成，同时不形成回路。这个概念在许多应用中都有实际意义，比如在通信网络、电路设计、城市规划等领域。找到一个图的最小生成树有助于优化资源的利用，减小总体成本。

常见的最小生成树算法包括 Prim 算法和 Kruskal 算法。这两种算法都属于贪心算法的范畴，通过每一步选择局部最优解来达到全局最优解。它们的目标都是找到一个包含所有顶点的树，并且边的权重之和最小。

1.2. 问题描述

为了是抽象的问题更具体，选择用以下问题来实例化。

如下图所示的赋权图表示某七个城市，预先计算出它们之间的一些直接通信道路造价（单位：万元），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。

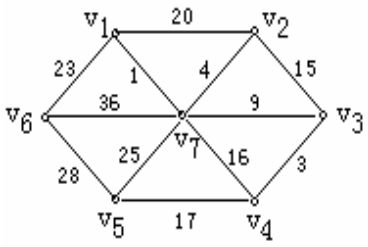


图 1 七个城市赋权图

1.3. 程序输入输出

1.3.1. 输入

用户通过程序输入所求图的顶点数目和边的数目，并输入每条边的信息（包括头尾结点序号以及该边的权值）。输入完图的结构后，可以选择使用 Prim 算法

或 Kruskal 算法。如果选择 Prim 算法，用户需要输入一个起始节点的序号。每次运行结束后，程序会询问用户是否继续运行。用户可以输入以决定。并且所有输入都有相应的错误处理。

1.3.2. 输出

程序会输出所选择算法求解得到的最小生成树的各边，它们的权值，以及最小生成树的总权值。

1.4. 功能选择

用户在输入完图的结构后，可以选择两种不同的算法进行最小生成树的求解：

（1）Prim 算法： 该算法需要用户输入一个起始节点，从该节点开始构建最小生成树。

（2）Kruskal 算法： 该算法不需要用户额外的输入，直接对图中的边进行排序并判断是否形成回路，逐步构建最小生成树。

同时，用户可以选择是否继续运行程序。

2. 解题思路

2.1. 明确最小生成树定义

使用不同的遍历图的方法，可以得到不同的生成树；从不同的顶点出发，也可能得到不同的生成树。按照生成树的定义，n 个顶点的连通网络的生成树有 n 个顶点，n-1 条边。构造最小生成树的准则如下：必须只使用该网络中的边来构造最小生成树；必须使用且仅使用 n-1 条边来联结网络中的 n 个顶点；不能使用产生回路的边

2.2. Kruskal 算法

2.2.1. 基本思想

设有 n 个顶点的连通网络 N = { V, E } ，最初先构造一个只有 n 个顶点，没有边的非连通图 T = { V, ∅ }，图中每个顶点自成一个连通分量。当 E 中选到一条具有最小权值的边时，若该边的两个顶点落在不同的连通分量上，则此边加入到 T 中；否则将此边舍去，重新选择一条权值最小的边。如此重复下去，直到所有顶点在同一个连通分量上为止。

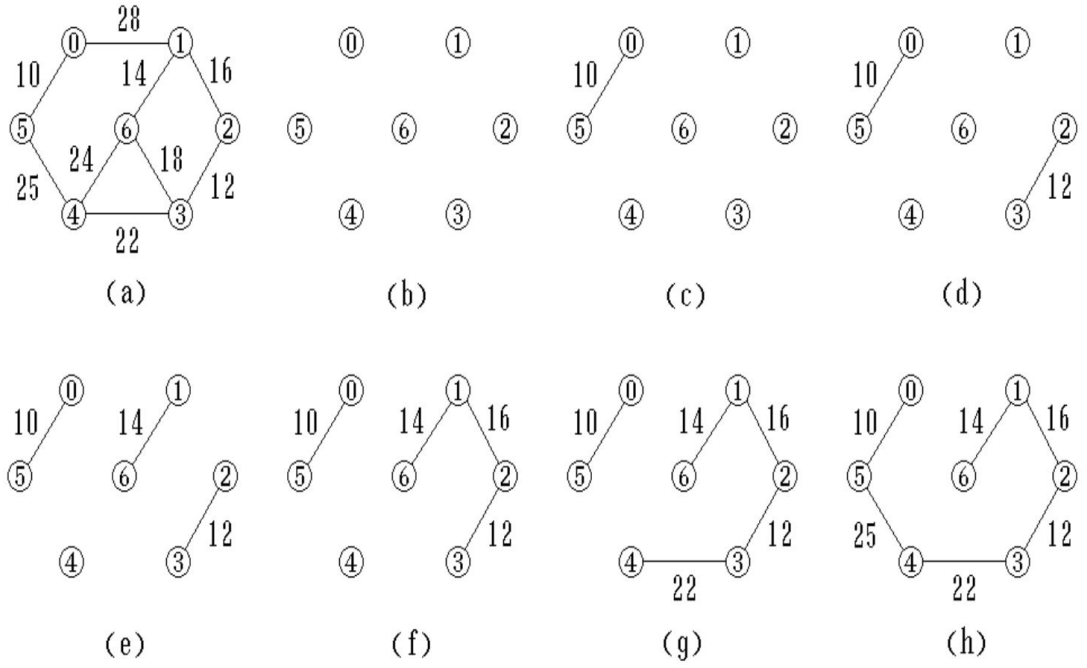
2.2.2. 算法框架

可以利用最小堆（MinHeap）和并查集（DisjoinSets）来实现克鲁斯卡尔算法。

首先，利用最小堆来存放 E 中的所有的边，堆中每个结点包含边的两个顶点位置和边的权值。

在构造最小生成树的过程中，最小堆中存放剩余的边，并且利用并查集的运算检查依附于一条边的两个顶点是否在同一个连通分量（即并查集的同一个子集合）上，是则舍去这条边，否则将此边加入 T，同时将这两个顶点放在同一个连通分量上。随着 各边逐步加入到最小生成树的边集合中，各连通分量也在逐步合并，直到形成一个连通分量位置。

2.2.3. 应用 Kruskal 算法构造最小生成树的过程



2.3. Prim 算法

2.3.1. 基本思想

从连通网络 N = { V, E } 中的某一顶点 u_0出发，选择与它关联的具有最小权值的边 (u_0, v)，将其顶点加入到生成树的顶点集合 U 中。

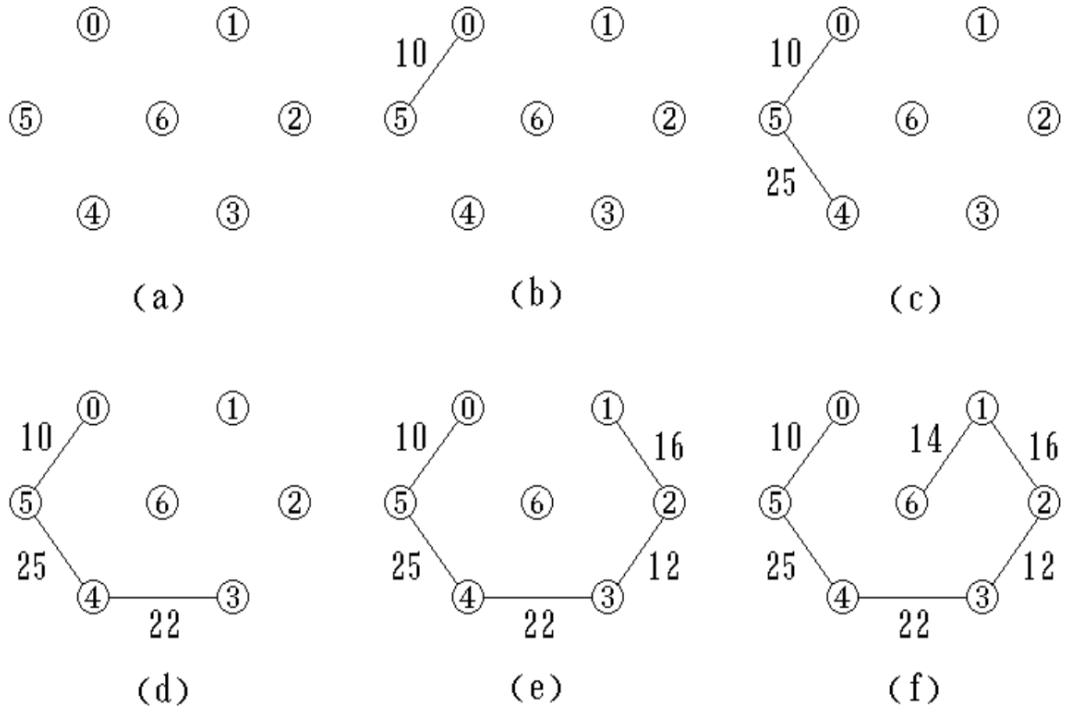
以后每一步从一个顶点在 U 中，而另一个顶点不在 U 中的各条边中选择权值最小的边(u, v)，把它的顶点加入到集合 U 中。如此继续下去，直到网络中的所有顶点都加入到生成树顶点集合 U 中位置。

2.3.2. 算法框架

采用邻接矩阵作为图的存储表示。

在构造的过程中，使用了 3 个辅助数组，visited 数组用来记录结点是否已经加入最小生成树中，minDist 数组用来记录当前结点到最小生成树的最小距离，parent 数组用来存储最小权值边两端顶点信息。

2.3.3. 用 Prim 算法构造最小生成树的过程



2.4. 总体思路

（1）用户输入图的结构和选择的算法。

（2）根据算法选择，执行相应的最小生成树算法。

（3）输出最小生成树的边集合和总权值。

（4）询问用户是否继续运行程序，若是则返回第 1 步，否则结束程序。

2.5. 界面设计

本程序为方便使用者调试与使用，增设了让用户判断是否需要继续运行程序的环节。使用循环与判断语句，并做了详尽的输入错误处理。

3. 所用数据结构

本程序为表示图的数据结构，定义了 Graph 类，其中使用了邻接矩阵和边集合来表示图的结构，主要操作如下图：

```
class Graph {
private:
    int V;//节点数
    vector<vector<int>>adjMatrix; //邻接矩阵,用于prim
    vector<pair<int, pair<int, int>>>edges;//存储边的信息（权值，（顶点1，顶点2）），用于kruskal
public:
    Graph(int vertices) :V(vertices), adjMatrix(vertices, vector<int>(vertices, INT_MAX)) {}//构造函数
    void addEdge(int u, int v, int weight);  //添加有权边
    int Prim(int startNode, vector<pair<int, int>>& resultEdges); //prim算法
    int Kruskal(vector<pair<int, int>>& resultEdges); //kruskal算法
    void printMST(const vector<pair<int, int>>& resultEdges);//打印最小生成树的各边
};
```

3.1. 邻接矩阵

邻接矩阵（Adjacency Matrix）是一种图数据结构，用于表示图中顶点之间的关系。对于无向图，邻接矩阵是对称的。在邻接矩阵中，图的顶点用矩阵的行和列表示，矩阵的元素表示顶点之间的关系。如果图中的顶点 i 和顶点 j 之间存在边，则邻接矩阵中的第 i 行第 j 列和第 j 行第 i 列的元素为 1。本程序中，adjMatrix 就是一个邻接矩阵，通过 adjMatrix[u][v]来表示顶点 u 和 v 之间的关系，其中 u 和 v 是顶点的索引。

邻接矩阵的数据结构运用在 Prim 算法中，使用 vector<vector<int>>表示，其中 adjMatrix[u][v]表示顶点 u 和 v 之间的权值。如果两个顶点之间没有边相连，则权值初始化为 INT_MAX。

3.2. 边集合

在本程序的 Kruskal 算法中，图的边集合用于按照权值升序排序，并进行后续的边的选择。在该程序中，边的信息以 pair<int, pair<int, int>>的形式存储，其中 pair<int, int>表示边连接的两个顶点，而外层的 int 表示边的权值。这种表示方式方便后续对边集合进行排序，并且在 Kruskal 算法中选择边。在 Kruskal 算法的实现中，边集合会使用 sort 函数按照权值升序排序，以便选择权值最小的边。

3.3. 并查集

并查集是一种数据结构，主要支持两种操作：查找和合并。在程序中，定义了一个 UnionFind 类用于实现并查集的相关操作。

在并查集中，parent 数组用来表示每个元素所属的集合（或者说是树的根

节点)。初始时，每个元素都是一个独立的集合，它的父节点就是它自己。findSet 函数用于查找某个节点所属的集合，实现了路径压缩，将节点直接连接到根节点，以提高查找效率。unionSets 函数用于合并两个集合，将一个集合的根节点设为另一个集合的根节点。

```
class UnionFind {
private:
    vector<int>parent;//表示每个元素所属的集合，即父节点
public:
    UnionFind(int size) :parent(size) {
        for (int i = 0;i < size;i++)
            parent[i] = i; //初始化每个节点为独立的集合
    }//构造函数
    int findSet(int node);//用于查找某个节点所属的集合
    void unionSets(int u, int v);//合并集合
};
```

其中，findSet` 操作用于查找某个节点所属的集合，它的实现涉及路径压缩，其目标是找到该节点所在集合的根节点。路径压缩一种优化技术，它在查找的同时将节点直接连接到根节点，减少后续查找的时间。具体来说，当 `findSet` 操作在节点 `node` 上执行时，它会追溯整个路径直到找到根节点，然后将该路径上的所有节点直接连接到根节点。路径压缩的目标是尽可能地减小树的高度，从而提高后续的查找效率。

unionSets 操作是并查集中的合并操作，它将两个不同集合的根节点连接在一起，形成一个更大的集合。在 Kruskal 算法中，unionSets 的目的是合并两个不同集合，确保加入的边不会形成环。对应函数中，setU 和 setV 分别表示节点 u 和 v 所在集合的根节点。通过将 setU 所在集合的根节点设置为 setV，实现了两个集合的合并。此操作保证了在并查集中，任意节点所在的集合都是一个由根节点引导的树形结构。

在 Kruskal 算法中，通过并查集来判断是否形成回路，以保证最小生成树的生成过程中不会有环产生。

4. 核心算法

4.1. Prim 算法

```cpp
int Graph::Prim(int startNode, vector<pair<int, int>>& resultEdges)
{
    vector<bool>visited(V, false);//记录结点是否已经加入最小生成树
    vector<int>minDist(V, INT_MAX);//记录当前节点到最小生成树的最小距离
    vector<int>parent(V, -1);//存储最小权值边两段顶点信息
    int totalWeight = 0;// 记录最小生成树的总权重
    int selectedNodes = 0;//记录加入最小生成树的节点树

    minDist[startNode] = 0;// 从指定的起始结点开始
    int i, j;
    for (i = 0;i < V;i++) {
        int u = -1;
        // 找到未加入最小生成树且具有最小距离的节点
        for (j = 0;j < V;j++) {
            if (!visited[j] && (u == -1 || minDist[j] < minDist[u])) {
                u = j;
            }
        }

        if (minDist[u] == INT_MAX) {
            break;//如果找不到合适的节点，则结束
        }

        visited[u] = true;// 将节点标记为已访问
        totalWeight += minDist[u];//将权重累加到最小生成树的总权重
        selectedNodes++;//增加加入最小生成树的节点树

        //将边加入结果集
        if (u != startNode)
            resultEdges.push_back({ parent[u],u });

        //更新与当前结点和相邻的未访问的节点的最小距离和父亲列表
        for (int v = 0;v < V;v++) {
            if (!visited[v] && adjMatrix[u][v] < minDist[v]) {
                minDist[v] = adjMatrix[u][v];
                parent[v] = u;
            }
        }
    }

    //检查是否存在最小生成树
    if (selectedNodes != V)
        return -1;//不存在返回值为-1
    return totalWeight;
}
```

4.1.1. 初始化

首先该函数会初始化一些数据结构，包括：visited，用于标记每个节点是否已经加入最小生成树；初始时，所有节点都未被访问。minDist：用于记录每个节点到最小生成树的最小距离；初始时，所有节点的距离被设置为无穷大（INT_MAX），表示它们与最小生成树的距离尚未确定。Parent，用于存储与最小权值边两端相连的顶点信息；初始时，所有节点的父节点被设置为-1。

totalWeight，用于记录最小生成树的总权重。selectedNodes，记录已经加入最小生成树的节点数。

4.1.2. 指定起始结点

将指定的起始结点的距离设为 0。

4.1.3. 迭代加入结点

该函数内进行循环，每次选择一个未加入最小生成树且具有最小距离的节点（u），将该节点标记为已访问，更新总权重，记录该边，并更新与该节点相邻的未访问节点的最小距离和父节点信息。

4.1.4. 终止条件

当已经加入最小生成树的节点数等于图的节点数时，表示最小生成树构建完成，算法终止。

4.1.5. 返回结果

没有最小生成树的情况也是可能存在的，因此在返回时要注意这一点。在无向图中，如果图是非连通的，即存在不可达的节点，或者存在孤立的节点（度为 0 的节点），则可能没有最小生成树。因为最小生成树是要包含所有的节点，如果有节点不可达，那么最小生成树就无法包含所有的节点。

在算法实现中，可以通过检查最终加入最小生成树的节点数来确定是否存在最小生成树。如果最小生成树的节点数等于图的总节点数，说明图是连通的，存在最小生成树。如果节点数小于总节点数，说明存在不可达的节点，最小生成树不存在。

不连通，返回-1 表示不存在最小生成树。否则，返回最小生成树的总权重。

4.2. Kruskal 算法

通过不断选择权值最小的边，并使用并查集判断是否形成回路，逐步构建最小生成树。Kruskal 算法同样采用了贪心策略，每次选择当前权值最小的边，确保每一步都是局部最优的，最终得到全局最优解。

```cpp
int Graph::Kruskal(vector<pair<int, int>>& resultEdges)
{
    sort(edges.begin(), edges.end());//将边按权值升序排序
    UnionFind uf(V);
    int totalWeight = 0;
    for (const auto& edge : edges) {
        int u = edge.second.first;
        int v = edge.second.second;
        int weight = edge.first;

        //判断加入该边是否会形成回路
        if (uf.findSet(u) != uf.findSet(v)) {
            uf.unionSets(u, v);//合并集合
            totalWeight += weight;//累加权值
            resultEdges.push_back({ u,v });//// 将边加入最小生成树的边集合
        }
    }

    //检查是否存在最小生成树
    if (resultEdges.size() != V - 1)
        return -1;//不存在返回值为-1

    return totalWeight;
}
```

4.2.1. 排序边集合

首先，对图的所有边按照权值升序进行排序。这样，算法每次选择权值最小的边。

4.2.2. 初始化并查集

使用并查集来维护各个节点所属的集合。初始化并查集，每个节点单独成为一个集合。

4.2.3. 迭代选择边

遍历排好序的边集合，对于每条边，判断它的两个端点是否属于同一个集合。如果不属于同一个集合，说明加入该边不会形成回路，可以将它加入最小生成树。通过并查集的 findSet 和 unionSets 操作，将这两个节点合并到同一个集合中。

4.2.4. 返回结果

最终，检查结果集合中边的数量是否为图的节点数减一。如果是，说明最小生成树构建成功；否则，表示图不连通，返回-1 表示不存在最小生成树。如果存在最小生成树，返回最小生成树的总权重。

4.3. 输入错误处理相关函数

4.3.1. 单个参数的输入错误处理

使用 while 循环不断尝试获取用户输入，直到输入满足要求。使用 cin.fail() 及输入内容的限制来判断输入是否出错，检查输入是否在有效范围内。如果输入无效，则输出错误信息，清除输入缓冲区并忽略之后的字符。通过一个字符来获取输入一个数之后的字符以检查输入字符的个数，若不正确，也输出错误信息，执行相应清除操作。如果输入有效，则跳出循环。

此方法可以用在判断输入的算法选择序号的正误，输入是否继续运行程序的正误。

4.3.2. 顶点数和边数的输入错误处理

与上方类似，使用 while 循环不断尝试获取用户输入，直到输入满足要求。而由于 n 个顶点的最小生成树中必有 n-1 条边，如果用户输入边的数目时，边数比 n - 1 小，则必定没有最小生成树，因此这种情况也归为输入内容错误。与前面类似，数据类型，输入的个数错误都需要重新输入，直到输入符合要求为止。

4.3.3. 边的头尾节点序号及权值的输入

用循环嵌套，让用户逐边输入各边的节点序号以及该边的权值，节点序号需要在已知的节点范围之内，而权值程序设计了宏定义的最大最小值，同样输入内容及输入错误时都需要重新输入，直到输入正确，跳出循环，继续输入下一条边的信息。

5. 心得体会

5.1. 关于程序结构

这次实验，我使用了不同的类封装不同的数据结构，例如：UnionFind 类：用于实现并查集的功能封装了集合的查找和合并操作。Graph 类：表示图的结构，包含邻接矩阵和边集合。也对不同的函数功能进行了清晰的划分，分别实现了图的构建、Prim 算法、Kruskal 算法以及结果的输出，addEdge 函数用于添加边，Prim 和 Kruskal 分别实现了两个算法等。同时，在输入输出部分采用了用户友好的交互方式，清晰地提示用户输入信息。

5.2. 关于算法实现

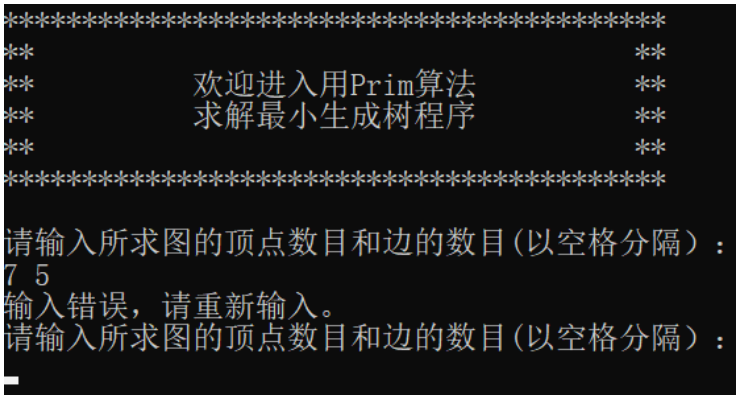
一开始使用的是 Prim 算法，使用了邻接矩阵表示图，让用户选择起始的结点，提高了程序的可维护性，通过标记已访问节点和维护最小距离数组，逐步构建最小生成树。后来又增加了 Kruskal 算法获得最小生成树，使用了边集合表示图，通过排序边的权值，使用并查集来判断是否形成回路。整体用到的结构都清晰易懂。在写代码的时候，让我逐步掌握两种算法，对最小生成树有了更为深刻的理解。

6. 测试结果

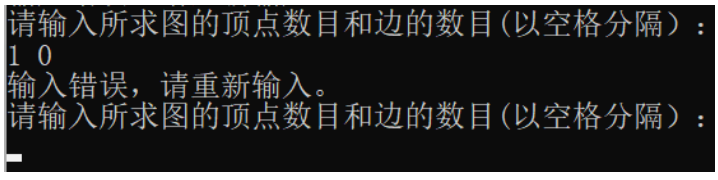
6.1. 输入顶点数和边数

本程序设定了顶点数目至少是 2 个，至多是 100 个，并且用宏定义 MIN_VERTICES，MAX_VERTICES 来表示。此外，如果要有最小生成树，边数至少是顶点数减一。边数和顶点数若不符合上述要求，则属于输入错误，则要求重新输入。

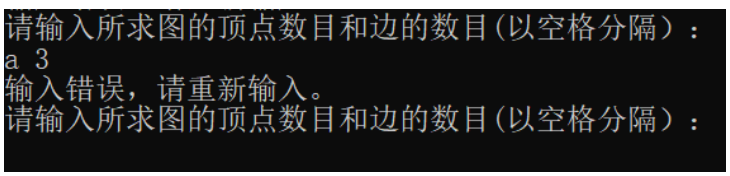
（1）顶点数值符合而边的数值错误



（2）顶点数错误



（3）输入类型不合法



（4）输入的个数不合法

```
请输入所求图的顶点数目和边的数目(以空格分隔)：
7 12 5
输入错误，请重新输入。
请输入所求图的顶点数目和边的数目(以空格分隔)：
```

（5）输入正确则可进入下一环节

```
请输入所求图的顶点数目和边的数目(以空格分隔)：
7 12
请输入各边的头尾节点序号及该边的权值(以空格分隔)：
请输入第1条边的信息：
```

6.2. 输入各边的头尾节点序号及其权值

任何节点序号一定都小于顶点数且都大于等于 1，同时本程序设定权值最大值为 100000000，并用宏定义 MAX_WEIGHT 来表示

（1）输入的值不在规定范围内（下图例子中顶点数为 7，但输入的其中一个节点序号为 8）

```
请输入所求图的顶点数目和边的数目(以空格分隔)：
7 12
请输入各边的头尾节点序号及该边的权值(以空格分隔)：
请输入第1条边的信息：8 6 3
输入错误，请重新输入。
请输入第1条边的信息：
```

（2）输入的类型不合法

```
请输入第1条边的信息：a 3 n
输入错误，请重新输入。
请输入第1条边的信息：
```

（3）输入的个数不合法，超过要求的 3 个

```
请输入第1条边的信息：1 2 3 4
输入错误，请重新输入。
请输入第1条边的信息：
```

（4）输入正确则可进入下一环节

```
请输入第1条边的信息：1 2 20
请输入第2条边的信息：2 3 15
请输入第3条边的信息：3 4 3
请输入第4条边的信息：4 5 17
请输入第5条边的信息：5 6 28
请输入第6条边的信息：6 1 23
请输入第7条边的信息：1 7 1
请输入第8条边的信息：2 7 4
请输入第9条边的信息：3 7 9
请输入第10条边的信息：4 7 16
请输入第11条边的信息：5 7 25
请输入第12条边的信息：6 7 36

1.用Prim算法求解最小生成树
2.用Kruskal算法求解最小生成树
请选择使用的算法（1/2）：
```

6.3. 输入选择的算法编号

（1）输入内容错误

```
1.用Prim算法求解最小生成树
2.用Kruskal算法求解最小生成树
请选择使用的算法（1/2）：a
输入无效，请重新输入。
请选择使用的算法（1/2）：
```

（2）输入个数错误（超过一个）

```
请选择使用的算法（1/2）：1 2
输入无效，请重新输入。
请选择使用的算法（1/2）：
```

（3）输入正确则可进入下一环节

```
请选择使用的算法（1/2）：2

使用Kruskal算法:
```

6.4. 选择算法 1，即 kruskal 算法的结果展示

```
使用Kruskal算法:

最小生成树中的各边:
1---7
3---4
2---7
3---7
4---5
6---1

最小生成树的总权值为：57

是否继续运行该程序？（y/n）：
```

6.5. 若选择算法 2，即 prim 算法

需要输入最小生成树起始的节点序号，节点序号一定都小于顶点数且都大于等于 1，否则视为输入错误。

（1）输入的值不在规定范围内（下图例子中图的顶点数为 7）

```
请输入起始的节点序号：0
输入错误，请重新输入。
请输入起始的节点序号:
```

（2）输入的类型不合法

```
请输入起始的节点序号：a
输入错误，请重新输入。
请输入起始的节点序号:
```

（3）输入的个数不合法（此处仅要求输入一个数据）

```
请输入起始的节点序号：1 2 6
输入错误，请重新输入。
请输入起始的节点序号：
```

（4）输入正确即可进入下一环节

```
请输入起始的节点序号：1

最小生成树中的各边：
```

6.6. 输出最小生成树中的各边及最小生成树的总权值（此处展示不同起点的结果）

```
1.用Prim算法求解最小生成树
2.用Kruskal算法求解最小生成树
请选择使用的算法（1/2）：1

使用Prim算法：

请输入起始的节点序号：1

最小生成树中的各边：
1---7
7---2
7---3
3---4
4---5
1---6

最小生成树的总权值为：57

是否继续运行该程序？（y/n）：
```

```
1.用Prim算法求解最小生成树
2.用Kruskal算法求解最小生成树
请选择使用的算法（1/2）：1

使用Prim算法：

请输入起始的节点序号：4

最小生成树中的各边：
4---3
3---7
7---1
7---2
4---5
1---6

最小生成树的总权值为：57

是否继续运行该程序？（y/n）：
```

6.7. 选择是否继续运行该程序

（1）输入错误

```
是否继续运行该程序？（y/n）：3
输入无效，请重新输入。

是否继续运行该程序？（y/n）：36
输入无效，请重新输入。

是否继续运行该程序？（y/n）：j6
输入无效，请重新输入。

是否继续运行该程序？（y/n）：a
输入无效，请重新输入。

是否继续运行该程序？（y/n）：
```

（2）输入 y/Y/n/N 均输入正确（下图以 y 为例）

```
是否继续运行该程序？（y/n）：y

请输入所求图的顶点数目和边的数目(以空格分隔）：
```

6.8. 能生成最小生成树的结果总览（此处输入均正确）

（1）使用 Kruskal 算法

```
*****************************************
**                                     **
**                欢迎进入             **
**          用Prim和Kruskal算法        **
**          求解最小生成树程序         **
**                                     **
*****************************************

请输入所求图的顶点数目和边的数目(以空格分隔）：
7 12
请输入各边的头尾节点序号及该边的权值(以空格分隔)：
请输入第1条边的信息：1 2 20
请输入第2条边的信息：2 3 15
请输入第3条边的信息：3 4 3
请输入第4条边的信息：4 5 17
请输入第5条边的信息：5 6 28
请输入第6条边的信息：6 1 23
请输入第7条边的信息：1 7 1
请输入第8条边的信息：2 7 4
请输入第9条边的信息：3 7 9
请输入第10条边的信息：4 7 16
请输入第11条边的信息：5 7 25
请输入第12条边的信息：6 7 36

1.用Prim算法求解最小生成树
2.用Kruskal算法求解最小生成树
请选择使用的算法（1/2）：2

使用Kruskal算法：

最小生成树中的各边：
1---7
3---4
2---7
3---7
4---5
6---1

最小生成树的总权值为：57

是否继续运行该程序？（y/n）：
```

（2）使用 Prim 算法

```
1.用Prim算法求解最小生成树
2.用Kruskal算法求解最小生成树
请选择使用的算法（1/2）：1

使用Prim算法：

请输入起始的节点序号：4

最小生成树中的各边：
4---3
3---7
7---1
7---2
4---5
1---6

最小生成树的总权值为：57

是否继续运行该程序？（y/n）：
```

6.9. 无向图为非连通图，即不存在最小生成树的情况的总览

（1）使用 Kruskal 算法

```
请输入所求图的顶点数目和边的数目(以空格分隔）：
4 3
请输入各边的头尾节点序号及该边的权值(以空格分隔)：
请输入第1条边的信息：2 3 9
请输入第2条边的信息：3 4 6
请输入第3条边的信息：2 4 8

1.用Prim算法求解最小生成树
2.用Kruskal算法求解最小生成树
请选择使用的算法（1/2）：2

使用Kruskal算法：

此图不存在最小生成树！

是否继续运行该程序？（y/n）：
```

（2）使用 Prim 算法

```
请输入所求图的顶点数目和边的数目(以空格分隔）：
4 3
请输入各边的头尾节点序号及该边的权值(以空格分隔)：
请输入第1条边的信息：2 3 9
请输入第2条边的信息：3 4 6
请输入第3条边的信息：2 4 8

1.用Prim算法求解最小生成树
2.用Kruskal算法求解最小生成树
请选择使用的算法（1/2）：1

使用Prim算法：

请输入起始的节点序号：2

此图不存在最小生成树！

是否继续运行该程序？（y/n）：
```