《离散数学》课程实验报告4 最小生成树

**1.项目要求**

用户从键盘输入所求图的顶点数目和边的数目，然后依次输入各边两端顶点的序号和边的对应的权值，要求程序输出该图对应的最小生成树的路径和总耗费。

**2.实验原理**

**2.1.最小生成树**

图G的每一条边e附加一个实数w(e), 称作边e的权。图G连同附加在边上的权称作带权图。设H是G的子图, H所有边的权的和称作H的权。而最小生成树就是带权图权最小的生成树。

**2.2.Kruskal算法**

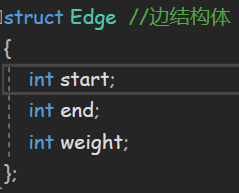
设有一个有n个顶点的连通网络N = { V, E } , 最初先构造一个只有n个顶点，没有边的非连通图T = { V, ∅ }, 图中每个顶点自成一个连通分量。当在E中选到一条具有最小权值的边时, 若该边的两个顶点落在不同的连通分量上，则将此边加入到T中；则将此边舍去，重新选择一条权值最小的边。如此重复下去，直到所有顶点在同一个连通分量上为止。

**3.实验所用的数据结构**

**3.1.并查集**

在本项目中，主要运用并查集的并和查操作，通过合并操作将边添加进最小生成树节点集合中，通过查操作判断想要加入的边的两个顶点是否落在不同的联通分量上，若两顶点同根，则说明两顶点在相同的联通分量中，若将该边添加进最小生成树集合中会形成回路，故不能将其添加到最小生成树集合中。

**3.2.最小堆**

最小堆要求出队本项目中，通过最小堆来选出当前未被合并进最小生成树中权值最小的边，这里我直接通过把边按照边权排序之后每次取当前队列中权值最小的元素来模拟最小堆的操作。

**3.3.图**

本题中图以边集的方式存储，每个边需要存储的信息为边的起点、终点和权重（如右图）。

**4.实现思路及重要函数**

**4.1.输入处理流程**

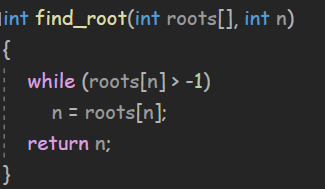
输入操作通过函数ini\_graph实现。该函数有3个参数：vex存储点的数目，edge存储边的数目，graph是一个指向边数组的指针，存储图的相关信息。该函数有一个返回值，若返回0则说明用户将要退出程序，若返回1则说明用户想要继续运行程序。首先要求用户输入所求图的顶点数目(vex)和边的数目(edge)，同时进行相应错误处理，若出现不合理的输入（输入小于0或输入字符），则给出相应提示并要求用户重新输入。

然后开始输入各边的信息，要求用户从键盘中输入各边的起点、终点和权重。若用户输入其他字符或输入的点的编号不合理（点的编号大于点的数目或小于1）则会给出相应错误提示并要求用户重新输入。

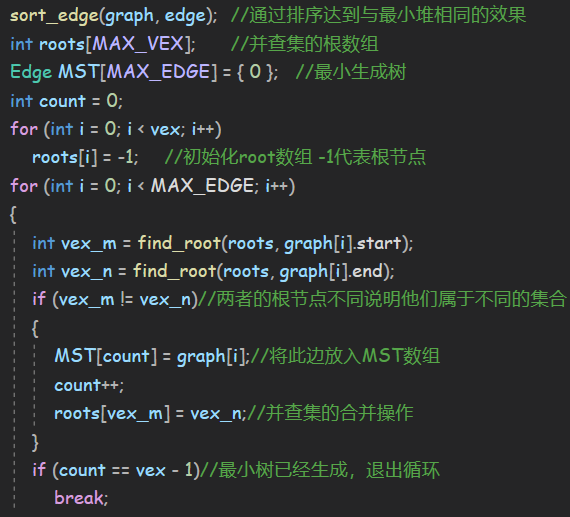
**4.2.排序函数**

对边按权值由小到大排列通过函数sort\_edge实现。这个函数有2个参数：edges\_arr是待排序的边数组，edge是边数组中边的数量。该函数通过冒泡排序的方式将边集数组按权值大小进行排序，以便模拟最小堆的操作。

**4.3.寻根操作**

并查集的寻根操作通过函数find\_root实现。该函数有2个参数：roots[]代表存储并查集的数组，n代表待寻根集合元素的编号。函数通过回溯的方式寻找元素n的根，因为根节点的root值一定为-1，所以当回溯到roots[n]==-1时，说明已经回溯到根节点，返回该根节点元素的索引即可，代码如右图所示。

**4.4.主函数及Kruskal算法的实现**

Kruskal算法的主体部分被放在一个大循环之中，循环的条件为ini\_graph函数的返回值，若返回1则循环继续进行，若返回0则退出循环，程序结束。

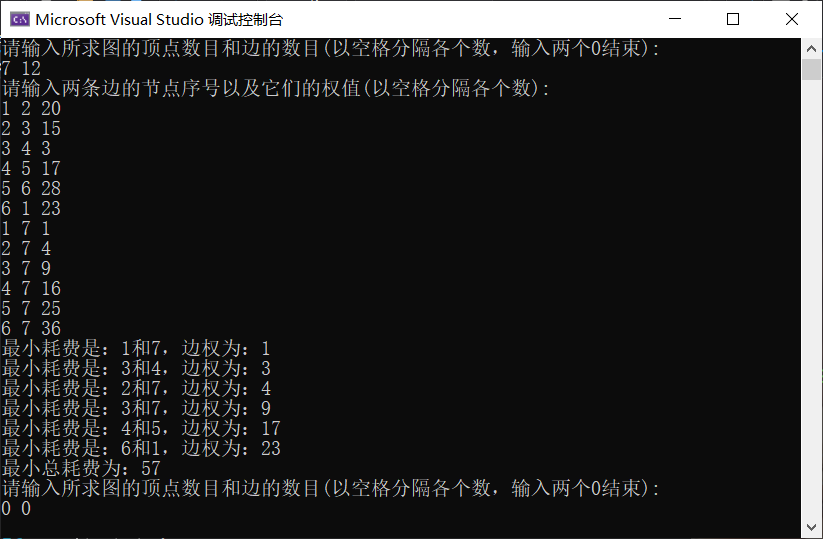
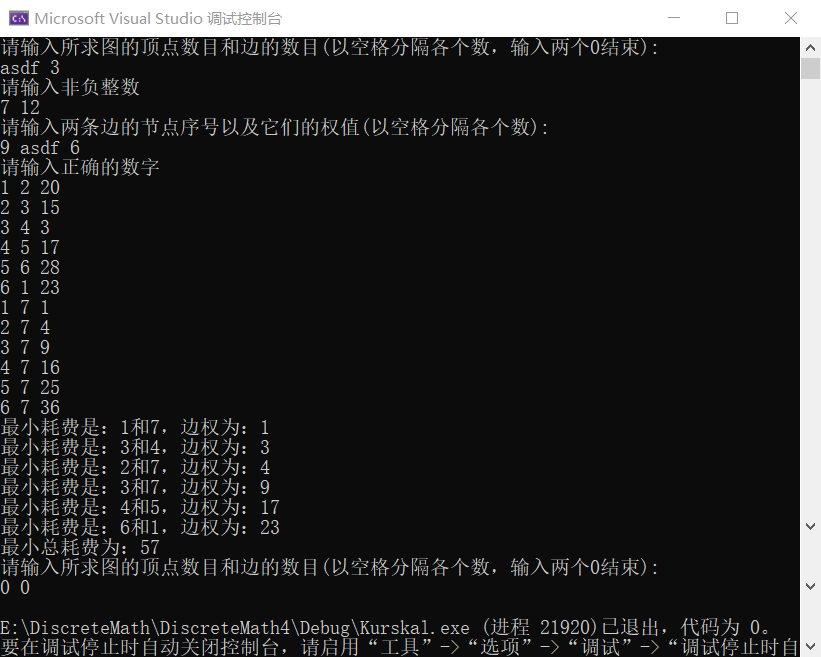
算法的主要代码如右图所示。首先调用sort\_edge函数将图中所有的边按权值从小到大排序，然后定义并查集的根数组和最小生成树集合数组。初始化根数组，将所有节点的值赋值为-1，说明当前所有节点均是根节点，还未发生合并操作。定义计数变量count,当count和vex-1相等时，说明最小生成树已经生成。然后进入一个循环，通过循环按权值由小到大的顺序遍历所有边，并判断其是否应该加入最小生成树集合中，通过调用find\_root函数返回当前边起点和终点的的根，若该边的起点和终点的根相同则不应将改变添加进最小生成树中；若该边的起点和终点的根不同则应该将该边放入最小生成树MST数组中，并将该边的起点和终点合并进并查集数组中，同时计数变量加1。每次循环完成后都需要判断计数变量count是否等于vex-1，若相等，则说明最小生成树已经生成，直接退出循环即可。

循环退出后，格式化输出MST数组中各边的起点、终点和边权，同时通过累加计算总耗费，此时算出的总耗费即为最小总耗费。

**5.心得体会**

这次作业要求我们使用Kruskal算法实现最小生成树。Kruskal算法虽然看起来不难理解，但要用到的数据结构比较多，实现起来也不简单，不过我还是见识到了很多未曾见过的技巧。在写代码的过程中，最小堆的实现需要重新封装数据结构和一系列操作，比较麻烦，但在本题中可以通过排序的方式代替最小堆，起到同样的效果；在判断将新的边加入后是否会形成回路的过程中可以借助并查集，通过判断边的起点和终点是否有相同的根来确定该边加入后是否会形成回路。在完成算法本身的同时，我还给程序添加了一些错误处理，使之更加健壮。本次作业加深了我对Kruskal算法和并查集的理解，收获很多。

程序效果如下：

1、正常输入：

2、错误处理：