电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2016220203031

姓 名 沈若玥

（实验） 课程名称 数据结构与算法

理论教师 郝宗波

实验教师 李美蓉

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告（3）**

**学生姓名：**沈若玥  **学 号：**2016220203031

**指导教师：**郝宗波

**实验地点：**清水河科技实验大楼 **实验时间：**2017.06.04

**一、实验室名称：**学校实验中心软件实验室

**二、实验项目名称：**编程实现最小生成树Kruskal 算法

**三、实验学时：**4

**四、实验原理：**

Kruskal 算法是一种按照图中边的权值递增的顺序构造最小生成树的方法。其基本思想是：设无向连通网为G＝（V，E），令G 的最小生成树为T，其初态为T＝（V，{}），即开始时，最小生成树T 由图G 中的n 个顶点构成，顶点之间没有一条边，这样T 中各顶点各自构成一个连通分量。然后，按照边的权值由小到大的顺序，考察G 的边集E 中的各条边。若被考察的边的两个顶点属于T 的两个不同的连通分量，则将此边作为最小生成树的边加入到T 中，同时把两个连通分量连接为一个连通分量；若被考察边的两个顶点属于同一个连通分量，则舍去此边，以免造成回路，如此下去，当T 中的连通分量个数为1 时，此连通分量便为G 的一棵最小生成树。

如教材153页的图4.21(a)所示，按照Kruskal 方法构造最小生成树的过程如图4.21 所示。在构造过程中，按照网中边的权值由小到大的顺序，不断选取当前未被选取的边集中权值最小的边。依据生成树的概念，n 个结点的生成树，有n－1 条边，故反复上述过程，直到选取了n－1 条边为止，就构成了一棵最小生成树。

**五、实验目的：**

本实验通过实现最小生成树的算法，使学生理解图的数据结构存储表示，并能理解最小生成树Kruskal 算法。通过练习，加强对算法的理解，提高编程能力。

**六、实验内容：**

（1）假定每对顶点表示图的一条边，每条边对应一个权值；

（2）输入每条边的顶点和权值；

（3）输入每条边后，计算出最小生成树；

（4）打印最小生成树边的顶点及权值。

**七、实验器材（设备、元器件）：**

PC机一台，装有C语言集成开发环境。

**八、数据结构与程序：**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VEX 10

#define MAX\_EDGE 20

typedef struct {

    int vexh;

    int vext;

    int weight;

    int flag;

}Edge;

typedef  struct {

    int vex[MAX\_VEX];

    Edge edge[MAX\_EDGE];

    int v, e;

}Graph;

void CreatGraph (Graph \*G) {

    FILE\* fp;

    fp=fopen("/Users/ryshen/Desktop/test.txt","r");

    if(!fp)

    {

        printf("The file does not exist!\n");

        exit(0);

    }

    fscanf(fp, "%d %d", &(G->v), &(G->e));

    for (int i = 0; i < G->v; i++) {

        fscanf(fp, "%d", &(G->vex[i]));

    }

    for (int i = 0; i < G->e; i++) {

        fscanf(fp, "%d %d %d", &(G->edge[i].vexh), &(G->edge[i].vext), &(G->edge[i].weight));

        G->edge[i].flag = 0;

    }

    fclose(fp);

}

void Kruskal(Graph \*G, Graph \*M) {

    Edge \*minEdge = NULL;

    int i, j = 0, k;

    int set[MAX\_VEX];

    for (i = 0; i < G->v; i++) {

        set[i] = i;

    }

    M->v = G->v;

    for (i = 0; i < G->v; i++) {

        M->vex[i] = G->vex[i];

    }

    for (i = 0; i < G->v - 1; ) {

        while (j != 0){

            j++;

        }

        minEdge = &G->edge[j];

        for (k = 0; k < G->e; k++) {

            if (G->edge[k].weight < minEdge->weight && G->edge[k].flag == 0){

                minEdge = &G->edge[k];

            }

        }

        if ((set[minEdge->vexh] != set[minEdge->vext])) {

            (\*minEdge).flag = 1;

            M->edge[M->e] = \*minEdge;

            M->e++;

            for ( j = 0; j < G->v; j++) {

                if (set[j] == set[minEdge->vext]) {

                   set[j] = set[minEdge->vexh];

                }

            }

            set[minEdge->vexh] = set[minEdge->vext];

            i++;

        }

        else {

            minEdge->flag = 2;

        }

    }

}

int main () {

    Graph G, M;

    CreatGraph(&G);

    Kruskal(&G, &M);

    for (int i = 0; i < M.e; i++){

        printf("vexh:%d  vext:%d  weight:%d  flag:%d\n", M.edge[i].vexh, M.edge[i].vext, M.edge[i].weight, M.edge[i].flag);

    }

}

**九、程序运行结果：**

****

**十、实验结论：**

测试正确，程序满足项目需求。

**十一、总结及心得体会：**

总结：这次的实验主要是考察对图的最小生成树的理解以及实现。最小生成树有十分重要的应用，也有很多方法来实现最小生成树的构造，此次实验采用的是Kruskal 算法，是一种典型的构造最小生成树的方法。实验的关键点就是对图和最小生成树的理解，还有对图的存储类型的选取，图一般选取邻接矩阵，邻接表来存储，但在此实验中稍显复杂，所以这次我没有采用这两种方式。

心得体会：这次的实验比起前两次来说不太复杂，花费的时间少一点，但是改错还是花了很多的时间，还是对知识点的掌握不熟悉导致改错改不出来，有些原理的东西理解不是很透彻。第二个致命错误就是，刚开始没有认真读题，没有按照要求来做，结果高高兴兴地做好了才发现要改，很郁闷啊。。。不过好在前两次的实验已经熟悉了C语言的操作了，所以这次的实验C语言没有成为什么问题。实验课时也看到了其他同学的新的解法，很强，还是希望自己能多学习，不断提升自我，不要拘泥于课本。