#include<stdio.h>

#include<malloc.h>

#define FINITY 5000

#define M 20

typedef char vertextype;

typedef int edgetype;

typedef struct

{

vertextype vex[M];

edgetype edges[M][M];

int n,e;

}Mgraph;

typedef struct edgedata //用于保存最小生成树的边类型定义

{

int beg,en;//beg,en是边顶点的序号

int length;//边的权值长

}edge;

//创建图

void create(Mgraph \*g,char \*s,int c)

{

int i,j,k,w;

FILE \*rf;

rf = fopen(s,"r");//从文件中读取图的信息

if(rf)

{

fscanf(rf,"%d%d",&g->n,&g->e);

for(i = 0;i<g->n;i++)

{

fscanf(rf,"%1s",&g->vex[i]);

}

for(i = 0;i < g->n;i++)

{

for(j = 0;j < g->n;j++)

{

if(i==j)

{

g->edges[i][j] = 0;

}

else

{

g->edges[i][j] = FINITY;

}

}

}

for(k = 0;k < g->e;k++)

{

fscanf(rf,"%d%d%d",&i,&j,&w);

g->edges[i][j] = w;

if(c == 0)

{

g->edges[j][i]=w;//c == 0 建立无向图邻接矩阵

}

}

fclose(rf);

}

else

{

g->n = 0;

}

}

//对边向量进行排序

void Sort(edge edges[],int count)

{

int i,j,k;

edge x;

for(i = 0;i < count-1;i++)

{

k = i;

for(j = i+1;j < count;j++)

{

if(edges[j].length < edges[k].length)

{

k = j;

}

}

if(k!=i)

{

x = edges[k];

edges[k] = edges[i];

edges[i] = x;

}

}

printf("sort finished\n");

}

//从图的邻接矩阵读取图的所有边信息

int getEdge(Mgraph g,edge edges[])

{

int i,j,k=0;

for(i = 0;i < g.n;i++)

{

for(j = 0;j < i;j++)

{

if(g.edges[i][j]!=0 && g.edges[i][j] < FINITY)

{

edges[k].beg = i;

edges[k].en = j;

edges[k++].length = g.edges[i][j];

}

}

}

return k;

}

//kruskal算法求解最小生成树

void kruskal(Mgraph g)

{

int i,j,k = 0,ltfl,count,h;

int cnvx[M];

edge edges[M\*M]; //存放图的所有边

edge tree[M]; //用于存放最小生成树的边信息

count = getEdge(g,edges);

printf("排序前\n");

for(i = 0;i < 10;i++)

{

printf("(%d->%d %d) ",edges[i].beg,edges[i].en,edges[i].length);

}

printf("\n");

Sort(edges,count);

printf("排序后\n");

for(i = 0;i < 10;i++)

{

printf("( %d->%d %d) ",edges[i].beg,edges[i].en,edges[i].length);

}

printf("\n");

for(i = 0;i <g.n;i++)

{

cnvx[i] = i; //设置每一个顶点的联通分量为其顶点编号

}

for(i = 0;i < g.n-1;i++)

{

while(cnvx[edges[k].beg] == cnvx[edges[k].en])

{

k++; //找到属于两个联通分量权最小的边

}

tree[i] = edges[k];//将边k加入生成树中

ltfl = cnvx[edges[k].en];//记录选中边的终点联通分量编号

for(j = 0;j < g.n;j++)//两个联通分量合并为一个联通分量

{

if(cnvx[j] == ltfl)

{

cnvx[j] = cnvx[edges[k].beg];

}

}

printf("\n第%d次选择后cnvx数组:\n",i);

for(h =0;h < g.n-1;h++)

{

printf("%d ",cnvx[h]);

}

k++;

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

// for(j = 0;j < g.n-1;j++)

// {

// printf("(%d--->%d) ",tree[i].beg,tree[i].en);

// }

printf("\n最小生成树是:\n");

for(j = 0;j < g.n-1;j++)

{

printf("%c---%c%6d (%d ---> %d )\n",g.vex[tree[j].beg],g.vex[tree[j].en],tree[j].length,tree[j].beg,tree[j].en);

}

}

//prim算法求最小生成树

void prim(Mgraph g,edge tree[M-1])

{

edge x;

int d,min,j,k,s,v;

for(v = 1;v < g.n-1;v++)

{

tree[v-1].beg=0;

tree[v-1].en = 0;

tree[v-1].length = g.edges[0][v];

}

for(k = 0;k < g.n-3;k++)

{

min=tree[k].length;

s = k;

for(j=k+1;j < g.n-2;j++)

{

if(tree[j].length < min)

{

min = tree[j].length;

s = j;

}

}

v = tree[s].en;//入选顶点为v

x = tree[s];

tree[s] = tree[k];//通过交换，将当前最小边加入TREE中

for(j = k+1;j <=g.n-2;j++)

{

d = g.edges[v][tree[j].en];

if(d < tree[j].length)

{

tree[j].length = d;

tree[j].beg = v;

}

}

}

printf("\n最小生成树：\n");

for(j = 0;j <= g.n-2;j++)

{

printf("\n%c---%c %d\n",g.vex[tree[j].beg],g.vex[tree[j].en],tree[j].length);

}

printf("\n\nthe root of it is %c\n",g.vex[0]);

}

int main()

{

Mgraph \*g = (Mgraph\*)malloc(sizeof(Mgraph));

edge tree[M-1];

int i,j;

char \*s = "E://graphinformation.txt";

create(g,s,0);

printf("图的节点为:\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

for(i = 0;i < g->n;i++)

{

printf("%c ",g->vex[i]);

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n");

printf("邻接矩阵如下:\n");

for(i = 0;i < g->n;i++)

{

for(j = 0;j < g->n;j++)

{

printf("%-4d ",g->edges[i][j]);

}

printf("\n");

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n");

kruskal(\*g);

prim(\*g,tree);

return 0;

}