/\* 求最小生成树的prim算法

lowcost数组，顾名思义，最小代价。也就是 lowcost[k] 保存着V-U中编号为k的顶点到U中所有顶点的最小权值。

closest数组，顾名思义，距离最近。 也就是 closest[k] 保存着U中到V-U中编号为K的顶点权值最小的顶点的编号。

这两个数组的元素是随着顶点不断加入U集合而动态变化的 \*/

#include <stdio.h>

#include <string.h>

#define MaxSize 20

#define MAX 10000

typedef char VertexType; //用VertexType 表示char

//定义图 的邻接矩阵表示法结构

typedef struct Graph {

VertexType ver[MaxSize+1]; //char ver[MaxSize+1]

int edg[MaxSize][MaxSize];

}Graph;

//邻接矩阵法图的生成函数

void CreateGraph( Graph \*g )

{

int i = 0;

int j = 0;

int VertexNum;

VertexType Ver; //顶点名称ABCDEF

printf("请输入图的顶点:\n");

while( '\n' != (Ver=getchar()) )

g->ver[i++] = Ver; //p->a，其中p是指向一个结构体的指针，a是这个结构体类型的一个成员。表达式p->a引用了指针p指向的结构体的成员a。

g->ver[i] = '\0'; //终止符号

VertexNum = strlen(g->ver);//顶点数6

printf("请输入相应的的邻接矩阵:\n");

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

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

scanf("%d", &g->edg[i][j]); //输入每条边的权值

}

//打印图的结点标识符和邻接矩阵

void PrintGraph( Graph g )

{

int i, j;

int VertexNum = strlen(g.ver);//与strlen(g->ver)有区别吗

printf("图的顶点为:\n");

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

printf("%c ", g.ver[i]);

printf("\n");

printf("图的邻接矩阵为:\n");

for( i=0; i<VertexNum; i++ ) {

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

printf("%d ", g.edg[i][j]);

printf("\n");

}

}

//求图的顶点数

int CalVerNum( Graph g )

{

return strlen(g.ver);

}

//将不邻接的顶点之间的权值设置为MAX

void SetWeight( Graph \*g )

{

for( int i=0; i<CalVerNum(\*g); i++ )

for( int j=0; j<CalVerNum(\*g); j++ )

if( 0 == g->edg[i][j] )

g->edg[i][j] = MAX;

}

//运用prim算法求最小生成树

void prim( Graph g, int VerNum, int \*parent )

{

int i, j, k;

int lowcost[MaxSize];

int closest[MaxSize], used[MaxSize];

int min;

for( i=0; i<VerNum; i++ ) { //对辅助数组lowcost和closest进行初始化

lowcost[i] = g.edg[0][i];

closest[i] = 0;

used[i] = 0; //used[i] == 0 表示i顶点在U中，反之，在V-U中。

parent[i] = -1;

}

used[0] = 1; //第一步将编号为0的顶点放入U中，也可以是其他顶点

for( i=0; i<VerNum-1; i++ ) {

j = 0;

min = MAX;

for( k=1; k<VerNum; k++ ) //找到V-U中的与U中顶点组成的最小权值的边的顶点编号

if( (0==used[k]) && (lowcost[k]<min) ) {

min = lowcost[k];

j = k;

}

parent[j] = closest[j];

used[j] = 1; //将j顶点加入U中

for( k=0; k<VerNum; k++ ) //由于j顶点加入U中，更新lowcost和closest数组中的元素，检测V-U中的顶点到j顶点的权值是否比j加入U之前的lowcost数组的元素小

if( (0==used[k]) && (g.edg[k][j]<lowcost[k]) ) {

lowcost[k] = g.edg[k][j];

closest[k] = j; //closest数组保存的是U中到V-U中最小权值的顶点编号

}

}

}

//打印最小生成树的边和MST的权值

void PrintMST( Graph g, int \*parent )

{

int VerNum = CalVerNum( g );

int weight = 0;

printf("MST的边为:\n");

for( int i=1; i<VerNum; i++ ) { //VerNum-1条边

printf("%c--%c\n", g.ver[parent[i]], g.ver[i] );

weight+=g.edg[parent[i]][i];

}

printf("MST的权值为:%d\n", weight);

}

int main()

{

Graph g;

int parent[20];

CreateGraph ( &g );

PrintGraph( g );

SetWeight( &g );

prim( g, CalVerNum(g), parent );

PrintMST( g, parent );

return 0;

}