数据结构课程设计报告

设计题目： 图的算法实现

班 级：

学 号：

姓 名：

南京农业大学计算机系

**数据结构课程设计报告内容**

# 课程设计题目

图的算法实现

【基本要求】

* 1. 建立一文件，将图的信息存在此文件中；
  2. 从此文件读入图的信息，建立邻接矩阵或邻接表；
  3. 实现 Prim、Kruskal、Dijkstra 和拓扑排序算法。

# 算法设计思想

1. 因为用 Prim、Kruskal 算法生成最小生成树，用 Dijkstra 算法求最小路径要求图是网，而 Prim 算法不支持有向图，拓扑排序算法要求是有向图，为使程序建图算法简单，所以在 Prim 算法中建立无向网，其余方法建立有向网。

同时稠密图在实际中并不多见（而有向网的邻接矩阵也不能折半压缩），故选用邻接表形式建图。

1. 从文件读入图的信息：使用文件指针，用 fopen 函数打开文件，用‘\n’作为间断点，按行读取数据，其中第一行为顶点个数和弧的个数，第二行为各顶点内容， 其余行每行为一条边的起点、终点、权值，如果建立有向图，则起点对应弧尾，终点对应弧头。最后，用 fclose 函数关闭文件。

(3).Prim 算法：Prim 算法的内容为假设 N=(V,{E})是连通网，TE 是 N 上最小生成树中边的集合。算法从 U={u0} (u0∈V)，TE={ }，重复执行下述操作：在所有 u∈U， v∈V-U 的边 (u,v) ∈E 中找—条代价最小的边(u0,v0)并入集合 TE，同时 v0 并入U，直至 U = V 为止。此时 TE 中必有 n-1 条边，T=(V,{TE})为 N 的最小生成树。

其本质是初始有一个只含初始点的顶点集合 V，然后每次从两边顶点一个在 V，一个不在 V 的弧中找出权值最小的一个，将此弧记录，并把不在 V 的弧加入 V。要用如下辅助数组实现：

typedef struct

{

int adjvex; int lowcost;

}minside[MVNum];

将顶点标号为0,1,2 ……对应此数组的[0],[1],[2]……

代表一边弧尾在V中，弧头为该点的弧中权值最小的弧，adjvex为该弧尾的标号，

lowcost为权值。

用循环结构遍历数组，每次找出权值（lowcost）大于0且最小的弧，（并输出显示），把弧头对应的lowcost赋为0，意为将弧头加入V，找最小弧时不在考虑。

然后因为有新的点加入V，遍历图的邻接表中此点指向其他点的弧，将权值和已记录的各个lowcost做对比，若新的权值更小就把新的弧记入该数组，覆盖旧值。

当所有顶点都在V中时（即循环了“顶点个数-1”次），最小生成树建成，程序完成。

(4).Kruskal算法：Kruskal算法的内容是假设连通网N=(V,{E})，则令最小生成树的初始状态为只有n个顶点而无边的非连通图T=(V,{ })，图中每个顶点自成一个连通分量。在E中选择代价最小的边，若该边依附的顶点落在T中不同的连通分量上，则将此边加入到T中，否则舍去此边而选择下一条代价最小的边。依次类推，直至T中所有顶点都在同一连通分量上。

其本质是将各个顶点设为单独的连通分量，每次在弧头、弧尾属于不同连通分量的弧中找出权值最小的，记录，并将这两个连通分量合并，直到所有的点都在同一个连通分量时，最小生成树建成。在程序中，用如下辅助数组实现：

typedef struct edge

{

int start; int end; int weight;

}edges[30];

flag[MVNum]；（MVNum为最大顶点数）

将点标号为0,1,2 ……对应此flag数组的[0],[1],[2]……

其中，edges结构体数组用于记录所有的边，建生成树前先根据权值从小到大进行排序，这样就无需遍历寻找权值最小弧了。

Flag数组用于标记，属于同一连通分量的顶点对应的值相同，初始设置各点对应的值等于其下标（各不相同）

从头遍历edges结构体数组，如果此边两点在flag[]中对应的点不同，就把此弧记录（并输出显示），然后遍历flag数组，把所有与弧头对应值相同的点的值改成弧未对应值（将弧头所在连通分量并入弧尾所在连通分量），遍历结束，最小生成树建成，程序完成。

1. Dijkstra算法：算法思想为根据所给的顶点，录入所有以其为弧尾的弧，弧的权值为两点距离（若没有弧则设距离为无限大），每次找出其中距离最小的弧，即为给

定点与该点的最短距离，在之后的权值比较中排除该点。然后看以此点为弧尾，未排除点为弧头的弧，若（此弧权值+已求得的最短距离）小于记录的给定点到此未排除点的距离，则更新该数据。直到所有的点均被排除，便求得了给定点与所有点的最短距离，从相关结构中可调取查看。在程序中，用如下辅助数组实现：

typedef struct dijkstra\_set

{

int last; int length; int flag;

}d\_set[10]；

将点标号为0,1,2 ……对应此数组的[0],[1],[2]…… Last存放给定点和此点已知最短路径中，此点的前驱, Length存放已知最短路径的长度

Flag为标记，初始flag=0，flag==1意为该点已被排除，寻找最小弧时不再考虑。

用循环进行。每次循环遍历d\_set，找出flag==0且length最小的，遍历此点（设为x） 对应的邻接表，如果有点（设为y），使得(x->y的权值+x对应的最短距离length)<y 已记录的最短距离length，那就把y对应的length换成新的最小值，同时把y对应的前驱改成x，最多循环‘顶点个数-1’次，程序完成。

1. 拓扑排序算法：算法是由某个集合的一个偏序得到该集合上的一个全序的操作， 实际算法为先输出一个入度为0的点，后删除此点和以此点为弧尾的弧，对应弧头入度-1，重复此过程直到没有入度为0的点。在程序中，用如下辅助数组实现： struct SqStack

{

SElemType \*base; SElemType \*top; int stacksize;

};

indegree[20]

用栈SqStack使程序进行，数组indegree用于记录各点的入度先把入度为0的一个点入栈，然后将数据出栈至栈空。

（其中每次出栈时遍历出栈点对应的邻接表，所有以该点为弧尾，本身为弧头

的点在indegree数组中对应的入度-1，若-1后入度为0，则将其入栈。）

另外可以再设置一个计数器变量，初始为0，每次出栈+1，若最终程序停止时计数器变量<顶点个数，则说明此图有环。

# 程序结构

主程序:

展示界面，提供选择->接受选择->根据选择导入数据，建表

->根据选择调用对应的算法函数主程序流程图:

主程序开始

Menu 函数展示目录



输入选项

3 号选项

(dijkstra 算法)

ShortPath\_Dijkstra

函数

4 号选项

(拓扑排序算法)

TopologicalSort

函数

2 号选项

(kruskal 算法)

MiniSpanTree\_KRU SKAL

函数

0 号选项关闭程序

1 号选项

(prim 算法)

MiniSpanTree\_PRIM

函数

Switch 函数

根据选项选择算法

CreateGraph 函数建表

Display 函数展示

主程序结束

调用关系：

main() Menu() //展示目录

CreateGraph() //建表Display() //展示导入的表

Switch() MiniSpanTree\_PRIM() FindArc() //寻找符合条件的最小边MiniSpanTree\_KRUSKAL() InitEdges() //初始化辅助数组，存储所有边

SortArc() //将所有边按权值从小到大排序ShortPath\_Dijkstra() FindMinPath() //寻找符合条件的一条最短路径TopologicalSort() InitStack() // 初始化辅助用的栈

Push() //入栈顶点标号Pop() //出栈顶点标号

StackEmpty() //判断是否栈空

# 实验结果与分析

## 用户使用说明

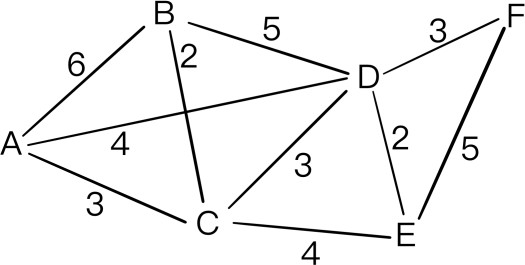
* + 1. 运行工程程序，可以看到一个选项界面
    2. 输入选择的选项，按 Enter 键，会提示输入要导入的图文件的选项（目前支持 1-4，注意：建议选择和功能序号一样的选项，因为每个图文件的内容都是针对相应的算法赋值的，选用不对应的图文件，最后的结果不会出错， 但可能因为图的数据不搭导致结果怪异，阅读性差）
    3. 输入选项，按 Enter 键，便会看到生成的图和应用相应算法的结果
    4. 在结果界面再按 Enter 键，画面会被重置，重新显示初始界面等待输入选项
    5. 当输入 0 号选项时，程序结束，自动关闭。

## 测试结果

测试数据：文件 GraphData1.txt---GraphData4.txt 对应图文件 1-4

（每个图文件的文本第一行为顶点个数，边个数；第二行为各顶点名称，其余行每行为一条边的起点、终点、权值）

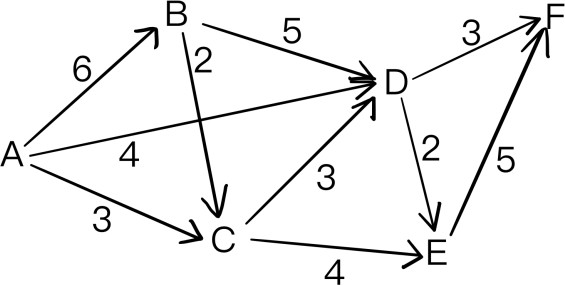
**图文件 1**： （prim 算法中，建立无向图；其余算法中，建立有向图；下同。）

6,10

a b c d e f a,b,6

a,c,3

a,d,4

b,c,2

b,d,5

c,d,3

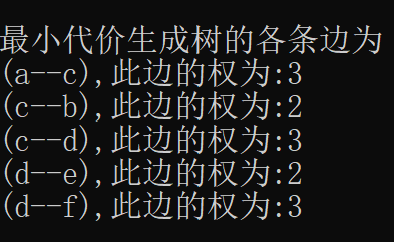
c,e,4

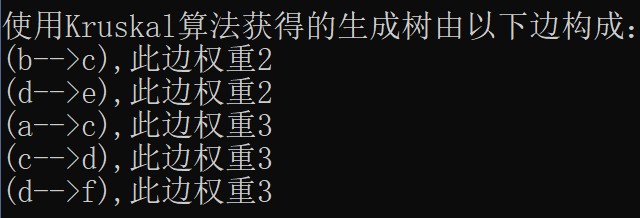
d,e,2

d,f,3

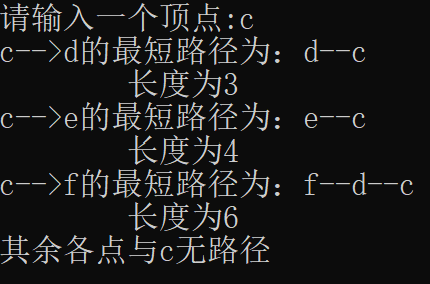
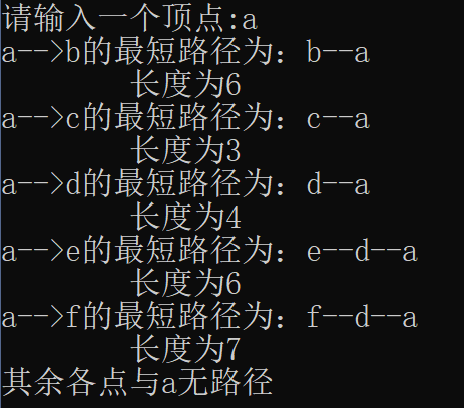
e,f,5

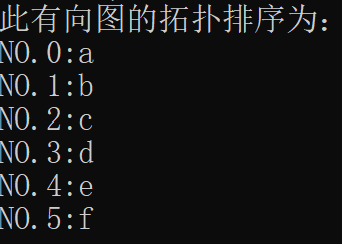
GraphData1.txt 所存的图

Prim 算法：

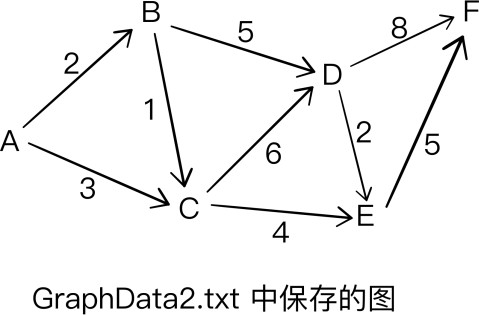
Kruskal 算法：

Dijkstra 算法：（为简洁只列出 a,c 两点的最短路径）



拓扑排序算法：

**图文件 2**： 6,9

a b c d e f

a,b,2

a,c,3

b,c,1

b,d,5

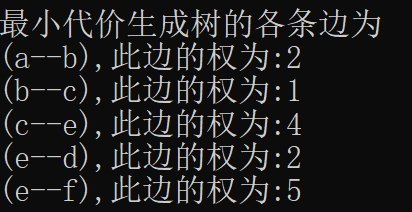
c,d,6

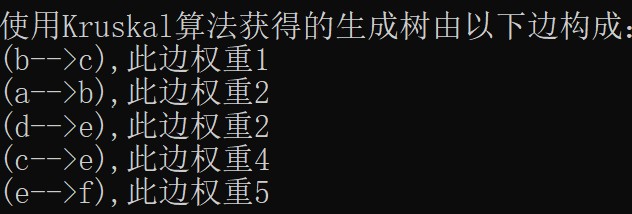
c,e,4

d,e,2

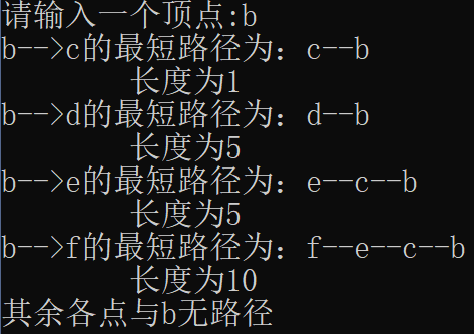
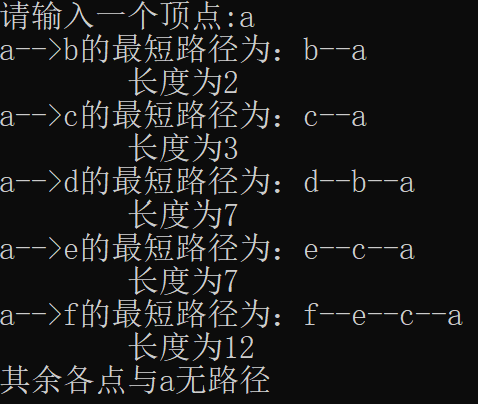
d,f,8

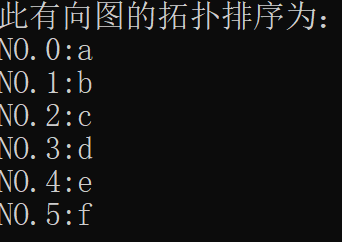
e,f,5

Prim 算法：

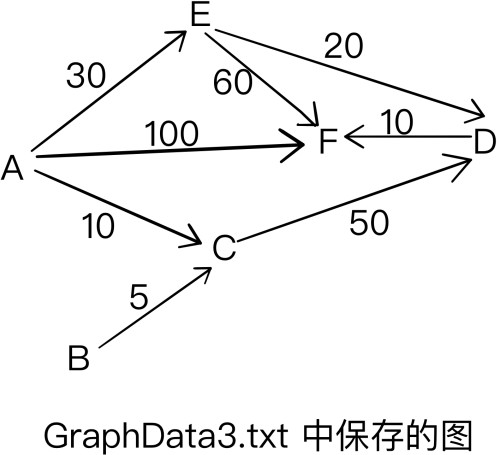
Kruskal 算法：

Dijkstra 算法：（为简洁只列出 a,b 两点的最短路径）



拓扑排序算法：

**图文件 3：** 6,8

a b c d e f

a,c,10

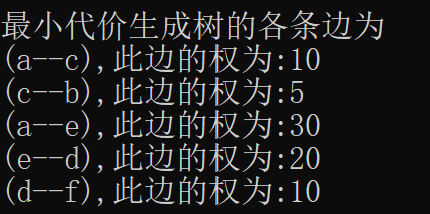
a,e,30 a,f,100

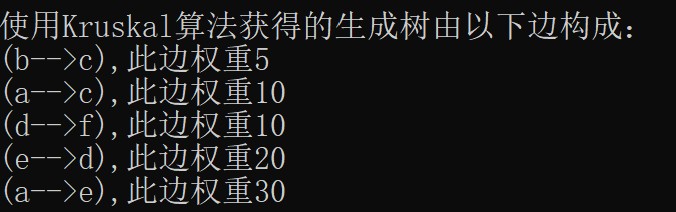
b,c,5

c,d,50

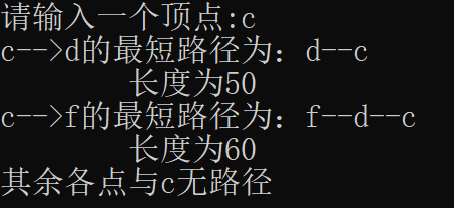
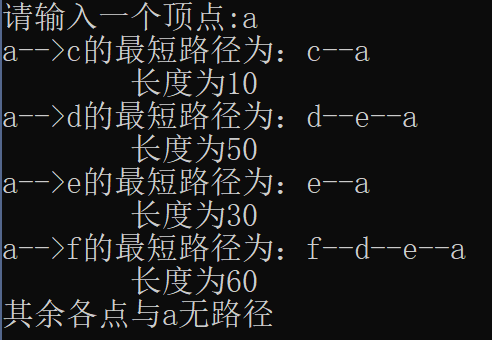
d,f,10

e,d,20

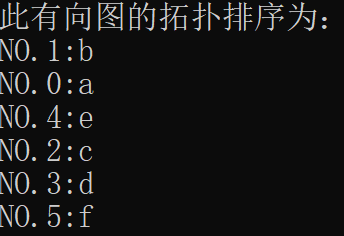
e,f,60 Prim 算法：

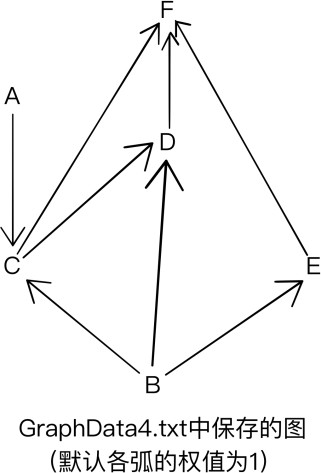
Kruskal 算法：

Dijkstra 算法：（为简洁只列出 a,c 两点的最短路径）



拓扑排序算法：

**图文件 4：** 6,8

a b c d e f

a,c,1

b,c,1

b,d,1

b,e,1

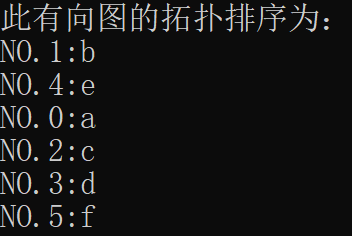
c,d,1

c,f,1

d,f,1

e,f,1

因为各弧的权值为 1，Prim 算法、Kruskal 算法、Dijkstra 算法意义较小，故不再列出。

拓扑排序算法：

以上列出了每个图文件在 4 种算法下的大部分结果，经检验全部正确。若要检验其他结果，可自行运行程序，对照所给图进行验证。

## 调试分析

①遇到的问题：我最初在 prim 算法中使用的是有向图，在调试过程中，我发现 prim 算法建立的有可能不是最小的生成树，经过我的分析，这是因为 prim 算法是根据最初的一点逐渐拓展的，而在有向图中，这会导致有些弧尾不是初始点、而权值较小的弧被忽略，经过进一步的查证，我确定了 prim 算法是不适用于有向图的，因此， 我改进了建图的函数，令之在为 prim 算法建图时建立无向图，对于其他算法，则还是建立有向图。

②算法的时间复杂度（设图文件有 V 个顶点，E 条边）： 建表函数(CreateGraph)：O(V+E)

展示表函数(Display)：O(V+E)（有向图）; O(V+2E)（无向图）;

销毁表函数(DestroyGraph)：O(V) 定位顶点函数(LocateVex)：O(V) Prim 算法：O(E\*logV)（邻接表）

Kruskal 算法： O(E\*logV)（稀疏图）; O(E\*logE)（稠密图）; Dijkstra 算法：O(V^2)

拓扑排序算法：O(V+E)（有向图）

③算法改进设想：定位顶点的函数 LocateVex ( )目前采用的是直接查找，可以效率较高的折半查找以适应更大的数据量；在 Kruskal 算法中，按权值排序所有边的方法可由冒泡排序改进为效率更高的排序方法，如快速排序或堆排序。

# 总结（收获与体会）

经过本次课设，我对图的结构及其相关算法有了更加清晰的认识，尤其是Kruskal 算法和 Dijkstra 算法，这两个算法只在理论课上，之前没有上机操作过， 这也让我认识到数据结构课自己多上机练习的重要意义：只有自主的完成代码， 才能说是对相关算法有了比较清晰的认识。

给我印象很深的是基础算法的选择，图分为有向图和无向图，又分为带权和不带权，图的结构又分为邻接矩阵和邻接表。所以在建什么格式的图，用什么结构建图的问题上想了很多，最后为尽可能满足题干，节约空间和提高效率，而选择去建邻接表结构的有向网，但我在这里犯了一个错，没有意识到 prim 算法不适用于有向图，这点为之后的调试工作带来了一定的困难，更使我意识到了分析问题、选取正确算法也应该严谨认真。这种对于算法、结构的选择在之前的学习中很少，我也因此体会到了根据具体任务，分析时间效率、空间效率和算法难易度的重要性和必要性。

我还意识到我个人的程序调试能力有待加强，对于这样一个稍大的程序，难免会出现各种各样的错误，这时快速的发现、分析问题就显得很重要，因此断点的位置、数量就很重要了。我发现，在控制循环的变量和循环中的变量改变的地方加上断点，效果比较好，因为这样可以逐步分析循环和处理数据每一步是否正常。

我感受到对于数据结构这门学科，我们的理论积累还远远不够，如与外部文件相关的操作、图的高级算法等；上机实践练习也很必要，只学习理论知识难以提高实际工作的水平。而这些，都需要我们自己努力去达成。

# 六． 源程序

文件名：

GraphAlgorithm.h

main.cpp、InitGraph.cpp、Prim.cpp、Kruskal.cpp、Dijkstra.cpp、TopologicalSort.cpp

GraphAlgorithm.h

#include<stdio.h> #include<stdlib.h> #include<limits.h>

#define MVNum 10 //定义的节点最大数量#define STACK\_INIT\_SIZE 10

#define STACK\_INCREMENT 2

#define ERROR 0

#define NULL 0

#define OVERFLOW -2

#define INFINITY INT\_MAX //定义 INT\_MAX 为无限大

typedef int SElemType; typedef char VertexType; typedef int ArcType; typedef struct ArcNode

{

int adjvex;//此弧的弧头int weight;//此弧的权值struct ArcNode \*next;

}ArcNode;//弧节点typedef struct VNode

{

char data;//顶点的内容(名称)

ArcNode \*firstarc;//指向此顶点指向的一条弧的弧节点

}VNode,AdjList[MVNum];//图的顶点数组struct ALGraph

{

AdjList vertices;//顶点数组

int vexnum,arcnum;//顶点数量.弧的数量

};//图的邻接表结构typedef struct

{

int adjvex;//最近的邻居的标号int lowcost;//对应弧的权值

}minside[MVNum];//prim 算法所用数组,用于存放每个顶点最近的邻居,和对应弧的权值

typedef struct edge

{

int start; //此弧弧尾int end; //此弧弧头int weight;//此弧权值

}edges[30];//kruskal 算法所用数组,用于存放和标记每个弧的数据

typedef struct dijkstra\_set

{

int last;//对应点和所给点最短路径中,对应点的前驱的标号int length;//此最短路径的长度

int flag;//标记(详情见相关函数)

}d\_set[10];//dijkstra 算法所用数组,用于存放和标记各点的最短路径

struct SqStack

{

SElemType \*base; SElemType \*top; int stacksize;

};//拓扑排序算法所用链栈

int LocateVex(ALGraph G,char u);//基础函数,返回所给顶点的标号(标号从 0 开始)

void CreateGraph(ALGraph &G,int choice);//建表函数void Display(ALGraph &G);//展示表函数

void DestroyGraph(ALGraph &G);//销毁表函数,用于一个功能完成后对表的销毁

int FindArc(minside SZ,ALGraph G);//prim 算法使用函数,每次返回一条符合要求的弧的弧头的标号

void MiniSpanTree\_PRIM(ALGraph G);//prim 算法主函数

void InitEdges(edges a,ALGraph G);//kruskal 算法使用函数,初始化用于记录和标记各弧的数组

void SortArc(edges a,ALGraph G);//kruskal 算法使用函数,以各弧权值为比较,用冒泡排序把所有的弧排序

void MiniSpanTree\_KRUSKAL(ALGraph G);//kruskal 算法主函数

int FindMinPath(ALGraph G,d\_set set);//dijkstra 算法使用函数,每次返回一条符合要求的弧的弧头的标号

void ShortPath\_Dijkstra(ALGraph G);//dijkstra 算法主函数

void InitStack(SqStack &S);//拓扑排序算法使用函数,初始化要用到的栈void Push(SqStack &S,SElemType e);//拓扑排序算法使用函数,入栈函数void Pop(SqStack &S,SElemType &e);//拓扑排序算法使用函数,出栈函数

int StackEmpty(SqStack S);//拓扑排序算法使用函数,判断栈是否为空的函数void TopologicalSort(ALGraph G);//拓扑排序算法主函数

//补充:在本程序中使用的数组,若无特别注明,则顶点标号(从0 开始)与数组的下标对应,如: a 是第一个顶点,是 0 号, 在辅助数组 a[n] 中使用a[0];

main.cpp

#include"GraphAlgorithm.h" void menu()//目录展示函数

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*欢迎使用图的算法演示程序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

|  |  |  |
| --- | --- | --- |
| printf(" | 1.用 Prim 算法建立图的最小生成树 | \n\n"); |
| printf(" | 2.用Kruskal 算法建立图的最小生成树 | \n\n"); |
| printf(" | 3.用Dijkstra 算法获得最短路径 | \n\n"); |
| printf(" | 4.图的拓扑排序 | \n\n"); |
| printf(" | 0.退出程序 | \n"); |

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

}

int main()

{

int i,choice; ALGraph G;

while(1)

{

if(i!=0) menu(); fflush(stdin);

printf("请输入你要选择的功能序号：");//读入选项scanf("%d",&choice);

while(choice<0||choice>4)//检查输入的选项,若不合要求则要求重新输入

{

printf("选项输入错误,请重新输入要选择的功能序号"); scanf("%d",&choice);

}

if(choice)//如果不是 0 号选项(关闭程序),则开始导入图的数据

{

CreateGraph(G,choice);//建图函数

printf("\n 图的文件导入成功，图的内容如下(邻接表形式):\n"); Display(G);//展示建成的图

}

switch (choice)//根据选项执行相应的算法函数

{

prim 算法

kruskal 算法

dijkstra 算法

case 1: MiniSpanTree\_PRIM(G); system("pause"); system("CLS"); i=1; break;//1 号选项,执行case 2: MiniSpanTree\_KRUSKAL(G); system("pause"); system("CLS"); i=1; break;//2 号选项,执行case 3: ShortPath\_Dijkstra(G); system("pause"); system("CLS"); i=1; break;//3 号选项,执行

序算法

case 4: TopologicalSort(G); system("pause"); system("CLS"); i=1; break;//4 号选项,执行拓扑排

case 0: return 0;//0 号选项，关闭程序

default: printf("ERROR\n" );

fflush(stdin);

}

}

system("pause");

}

InitGraph.cpp

#include"GraphAlgorithm.h"

int LocateVex(ALGraph G,char u)//定位顶点,根据顶点名返回其标号(从 0 号开始)

{

int i; for(i=0;i<G.vexnum;i++)

if(u==G.vertices[i].data) return i;

return -1;

}

void CreateGraph(ALGraph &G,int choice)//建表

{

int i,j,wei;

char k,v1,v2,filename[]={"GraphData?.txt"};//初始化文件源名称ArcNode \*e,\*p;

FILE \*fp;//文件指针

printf("请选择要导入的数据源（1 - 4），数据源的内容请自行打开对应的 txt 文件查看：\n(注意：建议选择与功能序号相同的数字，以获得最佳效果)\n");

fflush(stdin);

scanf("%c",&k);//获得要导入的文件选项filename[9]=k;//根据选项更改文件源名称if((fp=fopen(filename,"r"))==NULL)//用函数打开文件

{

printf("\n 导入数据失败！\n"); exit(0);

}

fscanf(fp,"%d,%d\n",&G.vexnum,&G.arcnum);//读取图的顶点数目.弧的数目fflush(stdin);

for(i=0;i<G.vexnum;i++)//读取各顶点的名称

{

fscanf(fp,"%c ",&G.vertices[i].data); G.vertices[i].firstarc=NULL;//初始化各顶点无弧相连

}

fflush(stdin); for(k=0;k<G.arcnum;k++)

{

fscanf(fp,"%c,%c,%d\n",&v1,&v2,&wei);//每次读入一行,为一条弧的弧尾,弧头,权值fflush(stdin);

i=LocateVex(G,v1); j=LocateVex(G,v2);

if(G.vertices[i].firstarc==NULL)//以邻接表形式建立表

{

e=(ArcNode \*)malloc(sizeof(ArcNode)); e->adjvex=j;

e->weight=wei; e->next=NULL;

G.vertices[i].firstarc=e;

}

else

{

p=G.vertices[i].firstarc; while(p->next!=NULL)

p=p->next;

e=(ArcNode \*)malloc(sizeof(ArcNode)); e->adjvex=j;

e->weight=wei; e->next=NULL;

p->next=e;

}

if(choice==1)

{

有向图）

}

else

}

if(G.vertices[j].firstarc==NULL)//如果图用于 prim 算法，则建立无向图，否则建立有向图（prim 不适用于

{

e=(ArcNode \*)malloc(sizeof(ArcNode)); e->adjvex=i;

e->weight=wei; e->next=NULL;

G.vertices[j].firstarc=e;

}

{

p=G.vertices[j].firstarc; while(p->next!=NULL)

p=p->next;

e=(ArcNode \*)malloc(sizeof(ArcNode)); e->adjvex=i;

e->weight=wei; e->next=NULL; p->next=e;

}

fclose(fp);//关闭文件

}

void Display(ALGraph &G)//展示建好的图

{

int i; ArcNode \*p;

for(i=0;i<G.vexnum;i++)

{

printf("结点%d:%c,指向的结点有：",i+1,G.vertices[i].data); p=G.vertices[i].firstarc;

while(p!=NULL)

{

printf("%c(此边权值%d)\t",G.vertices[p->adjvex].data,p->weight); p=p->next;

}

printf("\n");

}

}

void DestroyGraph(ALGraph &G)//销毁图

{

int i;

for(i=0;i<G.vexnum;i++)//所有值为 0 或指向 NULL

{

G.vertices[i].data=0; G.vertices[i].firstarc=NULL;

}

G.vexnum=0; G.arcnum=0;

}

Prim.cpp

#include"GraphAlgorithm.h"

int FindArc(minside SZ,ALGraph G)

{

int i=1,goal,min;

while(!SZ[i].lowcost)//跳过表头已经排除的点i++;

goal=i;//将第一个点作为初始的最小项 min=SZ[i].lowcost;//初始化 min 为第一个可取的点的lowcost

for(;i<G.vexnum;i++)//遍历顶点

if(SZ[i].lowcost>0&&SZ[i].lowcost<min)//若此顶点没有被排除且其对应的权值在比此时的最小项小,更新最小项

{

min=SZ[i].lowcost; goal=i;

}

return goal;//返回最小项的标号

}

void MiniSpanTree\_PRIM(ALGraph G)

{

int i,j; ArcNode \*k;

minside closedge; //定义辅助数组for(i=0;i<G.vexnum;i++)//初始化辅助数组,

{

closedge[i].adjvex=0; closedge[i].lowcost=INT\_MAX;//初始化权值为无限

}

for(k=G.vertices[0].firstarc;k;k=k->next)

closedge[k->adjvex].lowcost=k->weight;//将真正和 0 号顶点被所连的顶点对应的权值赋为真实值

closedge[0].lowcost=0;//排除将 0 号顶点自身printf("\n 最小代价生成树的各条边为\n"); for(i=1;i<G.vexnum;i++)

{

j=FindArc(closedge,G);//返回下一个符合条件的结点的标号printf("(%c--%c),此边的权

为:%d\n",G.vertices[closedge[j].adjvex].data,G.vertices[j].data,closedge[j].lowcost);//显示该边

closedge[j].lowcost=0;//排除该点

for(k=G.vertices[j].firstarc;k;k=k->next)//遍历含有刚排除的点的边,若有权值更小的边符合要求,则更新辅助数

组

if(k->weight<closedge[k->adjvex].lowcost)//比较权值

{

closedge[k->adjvex].adjvex=j;//更新最小邻居标号closedge[k->adjvex].lowcost=k->weight;//更新最小权值

}

}

printf("\n");

DestroyGraph(G);//销毁此表，防止为下一个功能建表时出错

}

Kruskal.cpp

#include"GraphAlgorithm.h"

void InitEdges(edges a,ALGraph G)//建立保存各弧的辅助数组

{

int i,j=0; ArcNode \*p;

for(i=0;i<G.vexnum;i++)//遍历顶点数组,录入每个弧的弧头、弧尾、权值

{

p=G.vertices[i].firstarc; if(p)

{

if((i==a[j].start&&p->adjvex==a[j].end)||(i==a[j].end&&p->adjvex==a[j].start))

{

}

else

for(;p;p=p->next)

{

a[j].start=i; a[j].end=p->adjvex; a[j].weight=p->weight; j++;

}

}

}

}

void SortArc(edges a,ALGraph G)//将所有弧按权值排序

{

int i,j; edge k;

for(i=0;i<G.arcnum-1;i++)//冒泡排序for(j=0;j<G.arcnum-1-i;j++)

if(a[j].weight>a[j+1].weight)

{

k=a[j]; a[j]=a[j+1]; a[j+1]=k;

}

}

void MiniSpanTree\_KRUSKAL(ALGraph G)

{

int i,j,k,flag[MVNum]; edges edge\_set;

InitEdges(edge\_set,G);//初始化辅助数组printf("\n 所有边按权值排列为：\n");

for(i=0;i<G.arcnum;i++)

printf("%d---%d ,权重%d\n",edge\_set[i].start,edge\_set[i].end,edge\_set[i].weight);

SortArc(edge\_set,G);//将所有弧按权值排序

for(i=0;i<G.vexnum;i++)//初始化每个顶点的标记为自身的标号,标号不同说明不在同一个连通分量,标号相同的在同一个连通分量.

flag[i]=i;

printf("\n 所有边按权值排列为：\n"); for(i=0;i<G.arcnum;i++)

printf("%d---%d ,权重%d\n",edge\_set[i].start,edge\_set[i].end,edge\_set[i].weight); printf("\n 使用 Kruskal 算法获得的生成树由以下边构成：\n");

for(j=0;j<G.arcnum;j++)//遍历所有弧,因为已经从小到大排序,所以每次遍历到的一定是权值最小的

{

符合要求

if(flag[(edge\_set[j].start)]!=flag[(edge\_set[j].end)])//如果此弧的尾和头不在同一个连通分量(标记不同),则

{

printf("(%c-->%c),此边权

重%d\n",G.vertices[(edge\_set[j].start)].data,G.vertices[(edge\_set[j].end)].data,edge\_set[j].weight);//展示此弧

for(k=0;k<G.vexnum;k++)

if((flag[k]==flag[(edge\_set[j].end)])&&(k!=edge\_set[j].end))//将尾和头所在的两个连通分量合并,

所有涉及到的点标记统一

flag[k]=flag[(edge\_set[j].start)]; flag[edge\_set[j].end]=flag[edge\_set[j].start];

}

}

printf("\n");

DestroyGraph(G);//销毁此表，防止为下一个功能建表时出错

}

Dijkstra.cpp

#include"GraphAlgorithm.h"

int FindMinPath(ALGraph G,d\_set set)

{

int i,min=0;

for(i=1;i<G.vexnum;i++)//遍历辅助数组,在未排除的点中选出最短的路径if(set[i].length<set[min].length&&set[i].flag==0)

min=i;

return min;//返回该路径弧头的标号

}

void ShortPath\_Dijkstra(ALGraph G)

{

int i,j,k,in; char c;

ArcNode \*p;

d\_set set;//辅助数组printf("请输入一个顶点:"); scanf("%c",&c);//获得起点

i=LocateVex(G,c);//获得起点标号in=i;

set[i].flag=1;//在辅助数组中排除起点for(j=0;j<G.vexnum;j++)//初始化辅助数组

{

set[j].last=INFINITY;//无前驱set[j].length=INFINITY;//距离无限

set[j].flag=0;//0 为未排除,1 为排除

}

p=G.vertices[i].firstarc;//遍历所有以起点为弧尾的弧,将之录入while(p)

{

set[p->adjvex].last=i;//这些弧的前驱为起点set[p->adjvex].length=p->weight;//距离为权值p=p->next;

}

for(k=1;k<G.vexnum;k++)

{

i=FindMinPath(G,set);//寻找此时辅助数组中距离最小的路径,返回弧头标号if(set[i].length!=INFINITY)

{

p=G.vertices[i].firstarc;//遍历以返回值为弧尾的弧while(p)

{

if(((p->weight+set[i].length)<set[p->adjvex].length)&&set[p->adjvex].flag==0)//若新的路径短于已知

路径,更新辅助数组

}

{

set[p->adjvex].length=p->weight+set[i].length; set[p->adjvex].last=i;

}

p=p->next;

}

set[i].flag=1;//此点的最短路径得到了,排除此点

}

for(i=0;i<G.vexnum;i++)

{

if(i!=in&&set[i].length!=INFINITY)//输出得到的所有的最短路径

{

printf("%c-->%c 的最短路径为：",c,G.vertices[i]); printf("%c--",G.vertices[i]);

for(j=set[i].last;j!=in;j=set[j].last)//根据前驱不断回退,直到遇到起点printf("%c--",G.vertices[j].data);

printf("%c\n",c);

printf("\t 长度为%d\n",set[i].length);

}

}

printf("其余各点与%c 无路径\n\n",c); DestroyGraph(G);//销毁此表，防止为下一个功能建表时出错

}

TopologicalSort.cpp

#include"GraphAlgorithm.h"

void InitStack(SqStack &S) //初始化栈

{

S.base=(SElemType \*)malloc(STACK\_INIT\_SIZE\*sizeof(SElemType)); if(!S.base) exit(OVERFLOW);

S.top=S.base; S.stacksize=STACK\_INIT\_SIZE;

}

void Push(SqStack &S,SElemType e) //入栈

{

if(S.top-S.base==S.stacksize)

{

S.base=(SElemType \*)realloc(S.base,(STACK\_INIT\_SIZE+STACK\_INCREMENT)\*sizeof(SElemType)); if(!S.base) exit(OVERFLOW);

S.top=S.base+S.stacksize; S.stacksize+=STACK\_INCREMENT;

}

\*S.top=e; S.top++;

}

void Pop(SqStack &S,SElemType &e) //栈顶元素出栈

{

if(S.top==S.base) exit(ERROR); S.top--;

e=\*S.top;

}

int StackEmpty(SqStack S) // 判断栈是否为空

{

if(S.top==S.base) return 1;

else return 0;

}

void TopologicalSort(ALGraph G)

{

int i,k,count=0,indegree[20]={0}; ArcNode \*p;

SqStack S; InitStack(S);

printf("此有向图的拓扑排序为：\n"); for(i=0;i<G.vexnum;i++)//遍历结点

{

p=G.vertices[i].firstarc; while(p!=NULL)//遍历每个结点的链

{

indegree[p->adjvex]++;//遇到的 adjvex 代表这个结点被用作了一次终点,入度+1 p=p->next;

}

}

for(i=0;i<G.vexnum;i++)

if(indegree[i]==0) Push(S,i);//将已经入度为 0 的点,即整个工程的起点入栈

while(!StackEmpty(S))

{

Pop(S,i);

printf("NO.%d:%c\n",i,G.vertices[i].data);//输出栈中入度为 0 的对象

count++;//计数器+1

for(p=G.vertices[i].firstarc;p!=NULL;p=p->next)//排除刚输出的对象,再遍历一遍它的链

{

k=p->adjvex;

if(!(--indegree[k])) Push(S,k);//遇到的结点入度-1(少了刚被排除的点和这个点的那条弧;若出现新的入度为 0 的点,入栈等待输出)

}

}

if(count<G.vexnum) printf("此有向图有回路\n");//若最后计数器的值小于结点数,则说明有环使输出提前结束了printf("\n");

DestroyGraph(G);//销毁此表，防止为下一个功能建表时出错

}