姓名： 毛翊尧 班级： A1611 学号： 31号

**实验四 图的表示与遍历**

一、实验目的

1、掌握图的邻接矩阵和邻接表表示

2、掌握图的深度优先和广度优先搜索方法

3、理解图的应用方法

二、实验预习

说明以下概念

1、深度优先搜索遍历：从根开始一个一个搜索。

2、广度优先搜索遍历：从根的邻接点出发依次访问。

3、拓扑排序：一个无指向的点开始排序。

4、最小生成树：最小权的生成树。

5、最短路径：路径权数最小。

三、实验内容和要求

1、阅读并运行下面程序，根据输入写出运行结果。

#include<stdio.h>

#define N 20

#define TRUE 1

#define FALSE 0

int visited[N];

typedef struct /\*队列的定义\*/

{

int data[N];

int front,rear;

}queue;

typedef struct /\*图的邻接矩阵\*/

{

int vexnum,arcnum;

char vexs[N];

int arcs[N][N];

}

graph;

void createGraph(graph \*g); /\*建立一个无向图的邻接矩阵\*/

void dfs(int i,graph \*g); /\*从第i个顶点出发深度优先搜索\*/

void tdfs(graph \*g); /\*深度优先搜索整个图\*/

void bfs(int k,graph \*g); /\*从第k个顶点广度优先搜索\*/

void tbfs(graph \*g); /\*广度优先搜索整个图\*/

void init\_visit(); /\*初始化访问标识数组\*/

void createGraph(graph \*g) /\*建立一个无向图的邻接矩阵\*/

{ int i,j;

char v;

g->vexnum=0;

g->arcnum=0;

i=0;

printf("输入顶点序列(以#结束)：\n");

while((v=getchar())!='#')

{

g->vexs[i]=v; /\*读入顶点信息\*/

i++;

}

g->vexnum=i; /\*顶点数目\*/

for(i=0;i<g->vexnum;i++) /\*邻接矩阵初始化\*/

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

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

printf("输入边的信息：\n");

scanf("%d,%d",&i,&j); /\*读入边i,j\*/

while(i!=-1) /\*读入i,j为－1时结束\*/

{

g->arcs[i][j]=1;

g->arcs[j][i]=1;

scanf("%d,%d",&i,&j);

}

}

void dfs(int i,graph \*g) /\*从第i个顶点出发深度优先搜索\*/

{

int j;

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

visited[i]=TRUE;

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

if((g->arcs[i][j]==1)&&(!visited[j]))

dfs(j,g);

}

void tdfs(graph \*g) /\*深度优先搜索整个图\*/

{

int i;

printf("\n从顶点%C开始深度优先搜索序列：",g->vexs[0]);

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

if(visited[i]!=TRUE)

dfs(i,g);

}

void bfs(int k,graph \*g) /\*从第k个顶点广度优先搜索\*/

{

int i,j;

queue qlist,\*q;

q=&qlist;

q->rear=0;

q->front=0;

printf("%c",g->vexs[k]);

visited[k]=TRUE;

q->data[q->rear]=k;

q->rear=(q->rear+1)%N;

while(q->rear!=q->front)

{

i=q->data[q->front];

q->front=(q->front+1)%N;

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

if((g->arcs[i][j]==1)&&(!visited[j]))

{

printf("%c",g->vexs[j]);

visited[j]=TRUE;

q->data[q->rear]=j;

q->rear=(q->rear+1)%N;

}

}

}

void tbfs(graph \*g) /\*广度优先搜索整个图\*/

{

int i;

printf("\n从顶点%C开始广度优先搜索序列：",g->vexs[0]);

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

if(visited[i]!=TRUE)

bfs(i,g);

}

void init\_visit() /\*初始化访问标识数组\*/

{

int i;

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

visited[i]=FALSE;

}

int main()

{

graph ga;

int i,j;

createGraph(&ga);

printf("无向图的邻接矩阵：\n");

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

{

for(j=0;j<ga.vexnum;j++)

printf("%3d",ga.arcs[i][j]);

printf("\n");

}

init\_visit();

tdfs(&ga);

init\_visit();

tbfs(&ga);

return 0;

}

根据右图的结构验证实验，输入：

ABCDEFGH#

0,1

0,2

A

B

C

D

E

F

G

H

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

0,5

1,3

1,4

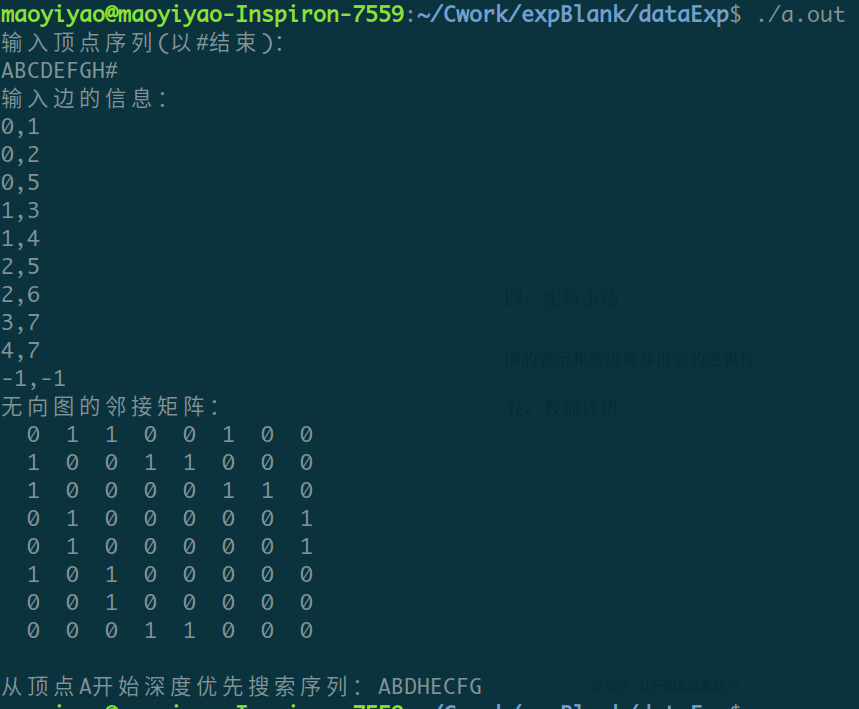
2,5

2,6

3,7

4,7

-1,-1

运行结果：2、阅读并运行下面程序，补充拓扑排序算法。

#include<stdio.h>

#include<malloc.h>

#define N 20

typedef struct edgenode{ /\*图的邻接表：邻接链表结点\*/

int adjvex; /\*顶点序号\*/

struct edgenode \*next; /\*下一个结点的指针\*/

}edgenode;

typedef struct vnode{ /\*图的邻接表：邻接表\*/

char data; /\*顶点信息\*/

int ind; /\*顶点入度\*/

struct edgenode \*link; /\*指向邻接链表指针\*/

}vnode;

void createGraph\_list(vnode adjlist[],int \*p)； /\*建立有向图的邻接表\*/

void topSort(vnode g[],int n)； /\*拓扑排序\*/

void createGraph\_list(vnode adjlist[],int \*p){ /\*建立有向图的邻接表\*/

int i,j,n,e;

char v;

edgenode \*s;

i=0;n=0;e=0;

printf("输入顶点序列(以#结束)：\n");

while((v=getchar())!='#')

{

adjlist[i].data=v; /\*读入顶点信息\*/

adjlist[i].link=NULL;

adjlist[i].ind=0;

i++;

}

n=i;

\*p=n;

/\*建立邻接链表\*/

printf("\n请输入弧的信息(i=-1结束)：i,j:\n");

scanf("%d,%d",&i,&j);

while(i!=-1){

s=(struct edgenode\*)malloc(sizeof(edgenode));

s->adjvex=j;

s->next=adjlist[i].link;

adjlist[i].link=s;

adjlist[j].ind++; /\*顶点j的入度加1\*/

e++;

scanf("%d,%d",&i,&j);

}

printf("邻接表:");

for(i=0;i<n;i++){ /\*输出邻接表\*/

printf("\n%c,%d:",adjlist[i].data,adjlist[i].ind);

s=adjlist[i].link;

while(s!=NULL){

printf("->%d",s->adjvex);

s=s->next;

}

}

}

void topSort(vnode g[],int n){ /\*拓扑排序\*/

}

int main(){

vnode adjlist[N];

int n,\*p;

p=&n;

createGraph\_list(adjlist,p);

return 0;

}

根据输入，输出有向图的拓扑排序序列。并画出有向图。输入:

ABCDEF#ABC

0,1

1,2

2,3

4,1

4,5

-1,-1

运行结果：

3、阅读并运行下面程序。

#include<stdio.h>

#define N 20

#define TRUE 1

#define INF 32766 /\*邻接矩阵中的无穷大元素\*/

#define INFIN 32767 /\*比无穷大元素大的数\*/

typedef struct

{ /\*图的邻接矩阵\*/

int vexnum,arcnum;

char vexs[N];

int arcs[N][N];

}

graph;

void createGraph\_w(graph \*g,int flag);

void prim(graph \*g,int u);

void dijkstra(graph g,int v);

void showprim();

void showdij();

/\*建带权图的邻接矩阵,若flag为1则为无向图，flag为0为有向图\*/

void createGraph\_w(graph \*g,int flag)

{

int i,j,w;

char v;

g->vexnum=0;

g->arcnum=0;

i=0;

printf("输入顶点序列(以#结束)：\n");

while((v=getchar())!='#')

{

g->vexs[i]=v; /\*读入顶点信息\*/

i++;

}

g->vexnum=i;

for(i=0;i<6;i++) /\*邻接矩阵初始化\*/

for(j=0;j<6;j++)

g->arcs[i][j]=INF;

printf("输入边的信息：\n");

scanf("%d,%d,%d",&i,&j,&w); /\*读入边(i,j,w)\*/

while(i!=-1) /\*读入i为－1时结束\*/

{

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

if(flag==1)

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

scanf("%d,%d,%d",&i,&j,&w);

}

}

void prim(graph \*g,int u)/\*出发顶点u\*/

{

int lowcost[N],closest[N],i,j,k,min;

for(i=0;i<g->vexnum;i++) /\*求其他顶点到出发顶点u的权\*/

{

lowcost[i]=g->arcs[u][i];

closest[i]=u;

}

lowcost[u]=0;

for(i=1;i<g->vexnum;i++) /\*循环求最小生成树中的各条边\*/

{ min=INFIN;

for(j=0;j<g->vexnum;j++) /\*选择得到一条代价最小的边\*/

if(lowcost[j]!=0&&lowcost[j]<min)

{

min=lowcost[j];

k=j;

}

printf("(%c,%c)--%d\n",g->vexs[closest[k]],g->vexs[k],lowcost[k]); /\*输出该边\*/

lowcost[k]=0; /\*顶点k纳入最小生成树 \*/

for(j=0;j<g->vexnum;j++) /\*求其他顶点到顶点k 的权\*/

if(g->arcs[k][j]!=0&&g->arcs[k][j]<lowcost[j])

{

lowcost[j]=g->arcs[k][j];

closest[j]=k;

}

}

}

void printPath(graph g,int startVex,int EndVex)

{

int stack[N],top=0; /\*堆栈\*/

int i,k,j;

int flag[N]; /\*输出路径顶点标志\*/

k=EndVex;

for (i=0;i<g.vexnum;i++) flag[i]=0;

j=startVex;

printf("%c",g.vexs[j]);

flag[j]=1;

stack[top++]=k;

while (top>0) /\*找j到k的路径\*/

{

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

{

if (path[k][i]==1 && flag[i]==0) /\*j到k的路径含有i顶点\*/

{

if (g.arcs[j][i]!=INF ) /\*j到i的路径含有中间顶点\*/

{

printf("-> %c(%d) ",g.vexs[i],g.arcs[j][i]);

/\*输出j到k的路径的顶点i\*/

flag[i]=1;

j=i;

k=stack[--top];

break;

}

else

{

if (i!=k) stack[top++]=i; /\*break;\*/

}

}

}

}

void dijkstra(graph g,int v){ /\*dijkstra算法求单源最短路径\*/

int path[N][N],dist[N],s[N];

int mindis,i,j,u,k;

for(i=0;i<g.vexnum;i++){

dist[i]=g.arcs[v][i];

s[i]=0;

for(j=0;j<g.vexnum;j++)

path[i][j]=0;

if(dist[i]<INF){

path[i][v]=1;

path[i][i]=1;

}

}

dist[v]=0;

s[v]=1;

for(i=0,u=1;i<g.vexnum;i++){

mindis=INFIN;

for(j=0;j<g.vexnum;j++)

if(s[j]==0)

if(dist[j]<mindis){

u=j;

mindis=dist[j];

}

s[u]=1;

for(j=0;j<g.vexnum;j++)

if((s[j]==0)&&dist[u]+g.arcs[u][j]<dist[j]){

dist[j]=dist[u]+g.arcs[u][j];

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

path[j][k]=path[u][k];

path[j][j]=1;

}

}

printf("\n顶点%c->到各顶点的最短路径\n",g.vexs[v]);

for(i=0;i<g.vexnum;i++){

printf("\n顶点%c->顶点%c：",g.vexs[v],g.vexs[i]);

if(dist[i]==INF||dist[i]==0)

printf("无路径");

else{

printf("%d ",dist[i]);

printf("经过顶点：");

printPath(g,v,i); /\*输出v到i的路径\*/

}

}

}

void showprim()/\*最小生成树prim算法演示\*/

{

graph ga;

createGraph\_w(&ga,1);

prim(&ga,0);

}

void showdij(){ /\*dijstra算法演示\*/

graph ga;

createGraph\_w(&ga,0);

dijkstra(ga,0);

}

int main(){

showprim(); /\*prim算法演示\*/

getchar();

showdij(); /\*dijstra算法演示\*/

return 0;

}

下面的输入分别验证prim算法和dijstra算法。输入实例的第一部分为无向图，求其最小生成树；输入的第二部分为有向图，求其最短路径。

最小生成树 最短路径

ABCDEF#

0,1,6

0,2,1

0,3,5

1,2,5

1,4,3

2,3,5

2,4,6

2,5,4

3,5,2

4,5,6

-1,-1,-1

ABCDEF#

0,2,10

0,5,100

0,4,30

1,2,5

2,3,50

3,4,20

3,5,10

4,3,20

4,5,60

-1,-1,-1

运行结果：（并画出两个图）

最小生成树 最短路径

1. 实验小结

清楚了有向图的邻接链表的画法，理清了有向图的遍历方法，了解深度搜索和广度搜索的差别，知道二者之间的共同点以及区别，不足之处是，对代码的编写还有困难，无法自己完整编写出程序。