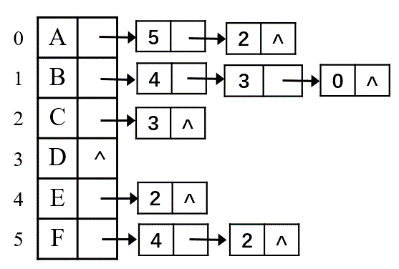
哈尔滨工业大学(深圳)2023年春《数据结构》

第三次作业 图型结构 参考答案

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学号 |  | 姓名 |  | 成绩 |  |

**1、简答题**

1-1已知图的邻接表如图所示，给出以顶点A为起点的一次深度优先（先深，DFS）和广度优先（先广，BFS）的搜索序列,并给出相应的生成树/森林。

【参考答案】

DFS：AFECDB

先深生成森林：(A-F，F-E，E-C，C-D)，(B)

BFS：AFCEEB

先广生成森林：(A-F，A-C，F-E，C-D)，(B)

1-2对一个图进行遍历可以得到不同的遍历序列，那么导致得到的遍历序列不唯一的因素有哪些?

【参考答案】

遍历不唯一的因素有：

1. 遍历起始点的顶点不同；
2. 存储结构不同；
3. 在邻接表情况下邻接点的顺序不同。

1-3 已知有6个顶点（顶点编号为0 ~ 5）的有向带权图*G*，其邻接矩阵*A*为上三角矩阵，按行为主序（行优先）保存在如下的一维数组中。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 6 | ∞ | ∞ | ∞ | 5 | ∞ | ∞ | ∞ | 4 | 3 | ∞ | ∞ | 3 | 3 |

（1）写出图*G*的邻接矩阵*A*；

（2）画出有向带权图*G*；

（3）给出一个拓扑序列；

（4）求图*G*的关键路径，并计算该关键路径的长度。

【参考答案】

（1）图*G*的邻接矩阵A如下：



（2）图*G*如下：

4

5

4

6

3

3

3

（3）拓扑序列：0-1-2-3-4-5

（4）下图中双线箭头所标识的4个活动组成图*G*的关键路径。

4

5

4

6

3

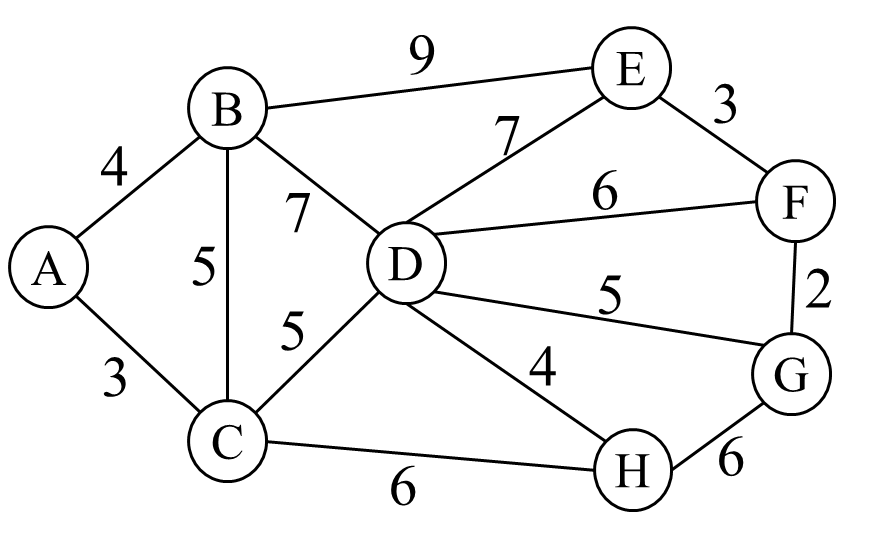
3

3

图*G*的关键路径的长度为16。

．

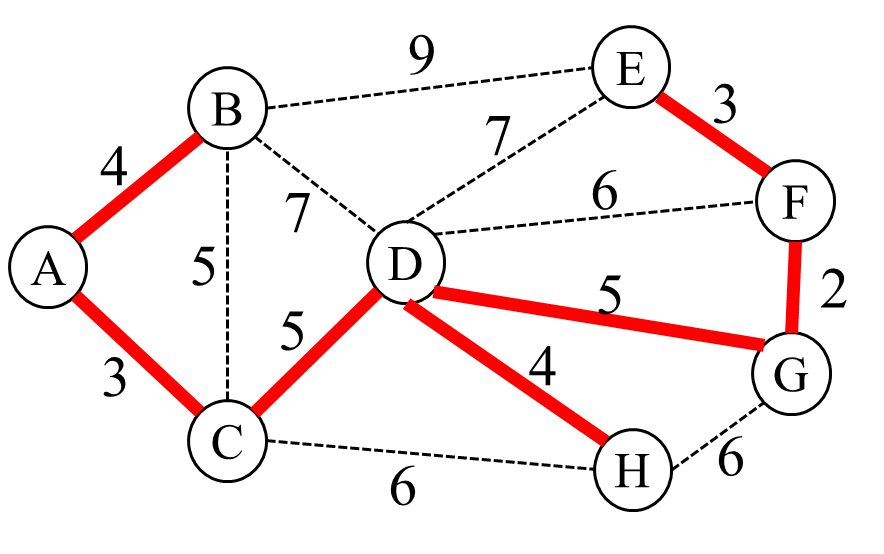
* 1. 无向网如下图所示，回答下列为题：



(1)按照Prim算法求其最小生成树，填表完成求解过程；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Prim算法求解过程(1) ∑w=26** | | | |
| No. | U | V-U | 本轮需考察的边及其权值 | 最近邻/A |
| 1 | {A} | {BCDEFGH} | AC:3,AB:4 | C |
| 2 | {AC} | {BDEFGH} | AB:4,CB:5,CD:5,CH:6 | B |
| 3 | {ACB} | {DEFGH} | CD:5,CH:6,BD:7,BE:9 | D |
| 4 | {ACBD} | {EFGH} | BE:9,DE:7,DF:6,DG:5DH:4,CH5 | H |
| 5 | {ACBDH} | {EFG} | BE:9,DE:7,DF:6,DG:5,HG:6 | G |
| 6 | {ACBDHG} | {EF} | BE:9,DE:7,DF:6,GF:2, | F |
| 7 | {ACBDHGF} | {E} | BE:9,DE:7,EF:3 | E |
| 8 | {ACBDHGFE} | {} |  | --- |

所求最小生成树如下图所示

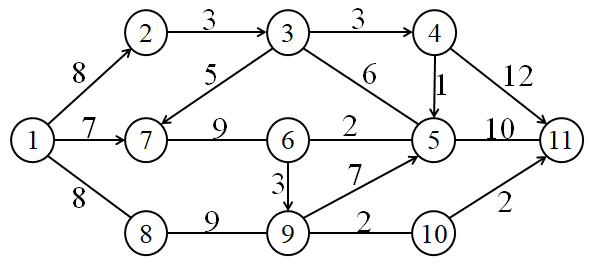


(2)按Kruskal算法求其最小生成树，按顺序给出边。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **FG:2** | **AC:3** | **EF:3** | **AB:4** | **DH:4** | **CD:5** | **DG:5** |

注：2和3，4和5，6和7位置均可互换。

1-5 求混合（有向和无向混合）图单源最短路径,填写表格完成Dijkstra算法各步骤。设源点为顶点①。



|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Dijkstra算法求单源最短路径** | | | | | | | | | | | | |
| **No** | **S** | **w** | **D[2]** | **D[3]** | **D[4]** | **D[5]** | **D[6]** | **D[7]** | **D[8]** | **D[9]** | **D[10]** | **D[11]** |
| **0** | **{1}** | **-** | **8** | **∞** | **∞** | **∞** | **∞** | **7** | **8** | **∞** | **∞** | **∞** |
| **1** | **{1,7}** | **7** | **8** | **∞** | **∞** | **∞** | **16** | **--** | **8** | **∞** | **∞** | **∞** |
| **2** | **{1,7,2}** | **2** | **--** | **11** | **∞** | **∞** | **16** | **--** | **8** | **∞** | **∞** | **∞** |
| **3** | **{1,7,2,8}** | **8** | **--** | **11** | **∞** | **∞** | **16** | **--** | **--** | **17** | **∞** | **∞** |
| **4** | **{1,7,2,8,3}** | **3** | **--** | **--** | **14** | **17** | **16** | **--** | **--** | **17** | **∞** | **∞** |
| **5** | **{1,7,2,8,3,4}** | **4** | **--** | **--** | **--** | **15** | **16** | **--** | **--** | **17** | **∞** | **26** |
| **6** | **{1,7,2,8,3,4,5}** | **5** | **--** | **--** | **--** | **--** | **16** | **--** | **--** | **17** | **∞** | **25** |
| **7** | **{1,7,2,8,3,4,5,6}** | **6** | **--** | **--** | **--** | **--** | **--** | **--** | **--** | **17** | **∞** | **25** |
| **8** | **{1,7,2,8,3,4,5,6,9}** | **9** | **--** | **--** | **--** | **--** | **--** | **--** | **--** | **--** | **19** | **25** |
| **9** | **{1,7,2,8,3,4,5,6,9,10}** | **10** | **--** | **--** | **--** | **--** | **--** | **--** | **--** | **--** | **--** | **21** |
| **10** | **{1,7,2,8,3,4,5,6,9,10,11}** | **11** | **8** | **11** | **14** | **15** | **16** | **7** | **8** | **17** | **19** | **21** |

**2、算法设计**

1. 采用C或C++语言设计数据结构；
2. 给出算法的基本设计思想；
3. 根据设计思想，采用C或C++语言描述算法，关键之处给出注释；
4. 说明你所设计算法的时间复杂度和空间复杂度。
   1. 图的存储结构实践。

自定义图的邻接矩阵和邻接表两种存储结构。以下两项任选其一：

1. 创建图的邻接矩阵，设计算法自动生成邻接表，或：
2. 创建图的邻接表，设计算法自动生成邻接矩阵。

要求能够打印图的邻接矩阵和邻接表，进行验证。

（1）数据结构

#define MAX\_VERTEX\_NUM 10

typedef enum { DG, DN, AG, AN } GraphKind ;

typedef int VertexType;

typedef struct ArcNode {

int adjvex;

struct ArcNode \*nextarc;

} ArcNode ;

typedef struct Vnode {

VertexType data ;

ArcNode \*firstarc ;

} Vnode, AdjList[MAX\_VERTEX\_NUM] ;

typedef struct {

AdjList vertices ;

int vexnum ;

GraphKind kind ;

} ALGraph; //邻接表

typedef struct ArcCell {

int adj ; //0/1邻接关系 或 wij

} ArcCell,AdjMatrix[ MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM] ;

typedef struct {

VertexType vex[MAX\_VERTEX\_NUM] ;

AdjMatrix arcs ;

int vexnum , arcnum ;

GraphKind kind;

}MGraph ; //邻接矩阵

（2）设计思想

首先将邻接矩阵清零G.arcs[i][j].adj=0，i,j=1…n

依次取每个表头结点标号i，遍历邻接表中每个链表，取到邻接点位置j=p->adjvex，然后将G.arcs[i][j].adj=1。

（3）源代码

void DisplayALGraph(ALGraph G) //打印图的邻接表

{

int i;

ArcNode \*p;

printf("邻接表如下：\n");

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

{

printf("顶点%3d -- ",G.vertices[i].data);

p=G.vertices[i].firstarc;

while(p)

{

printf("%4d ",p->adjvex);

p=p->nextarc;

}

printf("^\n");

}

}

void AlgToM(ALGraph G1,MGraph \*G2)

{ //邻接表转邻接矩阵

int i,j;

(\*G2).kind=G1.kind;

(\*G2).vexnum=G1.vexnum;

(\*G2).arcnum=0;

ArcNode \*p;

for(i=1;i<=G1.vexnum;i++)

(\*G2).vex[i]=G1.vertices[i].data; //传递顶点信息

for(i=1;i<=G1.vexnum;i++)

for(j=1;j<=G1.vexnum;j++)

(\*G2).arcs[i][j].adj=0; //邻接矩阵清零

for(i=1;i<=G1.vexnum;i++) //转换

{

p=G1.vertices[i].firstarc;

while(p)

{

(\*G2).arcs[i][p->adjvex].adj =1;

(\*G2).arcnum++;

p=p->nextarc;

}

}

}

void DisplayMGraph(MGraph \*G2) //打印图的邻接矩阵

{

int i,j;

for(i=1;i<=(\*G2).vexnum;i++)

{

for(j=1;j<=(\*G2).vexnum;j++)

printf("%4d",(\*G2).arcs[i][j].adj);

printf("\n");

}

}

（4）算法分析

时间复杂度T(n)=O(n)，n为顶点个数。

* 1. 设具有n个顶点的有向图用邻接表存储（不存在逆邻接表）。试写出计算所有顶点入度的算法，将每个顶点的入度值分别存入一维数组 int Indegree[n]中。

【参考答案】

1. 存储结构如题2-1所示。
2. 设计思想

设结点i的如度为InDegree[i]，初始均为0，i=1…n；

遍历邻接表的每个邻接点i，执行InDegree[i]++。

1. 程序代码

void GetInDegree(ALGraph G,int InDegree[])

{

int i;

ArcNode \*p;

for(i=1;i<=G.vexnum;i++) InDegree[i]=0;

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

{ p=G.vertices[i].firstarc;

while(p)

{

InDegree[p->adjvex]++;

p=p->nextarc;

}

}

}

1. 算法分析

T(n)=O(n)

* 1. 一个连通图采用邻接表作为存储结构，设计一个算法，实现从顶点v出发的深度优先遍历的非递归过程。

【参考答案】

（1）数据结构

设立一个顶点栈S，采用栈和图的ADT操作。

（2）设计思想

先将遍历起点进栈；

当栈非空时：弹出栈顶顶点v，访问并标记该顶点v，找到与该顶点v相关联的每一个顶点w，如果w未被访问过且与当前栈顶顶点不同则进栈保留。

直到栈空为止。

（3）算法ADT实现

Void DFSn(Graph G,int v)

{ //从第v个顶点出发非递归实现深度优先遍历图G

Stack S;

InitStack(S); //初始栈S为空

Push(S,v); //起点v进栈

While(!StackEmpty(S))

{ //栈空时第v个顶点所在的连通分量已遍历完

Pop(S,k);

If(!visited[k])

{ visited[k]=TRUE;

VisitFunc(k); //访问第k个顶点

//将第k个顶点的所有邻接点进栈

for(w=FirstAdjVex(G,k);w;w=NextAdjVex(G,k,w))

{

if(!visited[w]&&w!=Top(s)) Push(S,w);

//图中有环时w==Top(S)

}

}

}

（4）算法分析

时间复杂度T(n)=O(n)；空间复杂度S(n)=O(n)

* 1. 采用链接表存储结构，编写一个判别无向图中任意给定的两个顶点(u，v)之间是否存在一条长度为k的简单路径

【参考答案】

（1）数据结构

邻接表（详细解释见课件）

#define MAX\_VERTEX\_NUM 10

typedef enum { DG, DN, AG, AN } GraphKind ;

typedef int VertexType;

typedef struct ArcNode {

int adjvex;

struct ArcNode \*nextarc;

} ArcNode ;

typedef struct Vnode {

VertexType data ;

ArcNode \*firstarc ;

} Vnode, AdjList[MAX\_VERTEX\_NUM] ;

typedef struct {

AdjList vertices ;

int vexnum ;

GraphKind kind ;

} ALGraph;

int visited[MAX\_VERTEX\_NUM]; //访问标记

（2）算法设计思想

采用深度优先搜索策略，以u为起点进行深度优先遍历，若第一个邻接点存在，从第一个另结点出发继续判断是否有到v的长度为k-1的路径。

（3）算法描述

int FirstAdjVex(ALGraph G,VertexType v)

{ //返回值为图G中与顶点v邻接的第一个临界点，0为没有邻接点

if(G.vertices[v].firstarc)

return(G.vertices[v].firstarc->adjvex);

else

return(0);

}

int NextAdjVex(ALGraph G,VertexType v,VertexType w)

{ //返回值为图G中与顶点v邻接的w之后的邻接点，0为武侠一个邻接点

ArcNode \*p;

p=G.vertices[v].firstarc;

while(p!=NULL&&p->adjvex!=w)

p=p->nextarc;

if(p)

return(0);

else

if(p->nextarc)

return(p->nextarc->adjvex);

else

return(0);

}

int exist\_path\_len(ALGraph G,int i,int j,int k)

{ //判断是否有从顶u到顶点v长度为k的路径，深度优先搜索

int temp;

ArcNode \*p;

if(i==j&&k==0)

return 1;

else if(k>0)

{

visited[i]=1;

for(p=G.vertices[i].firstarc;p;p=p->nextarc)

{

temp=p->adjvex;

if(!visited[temp]&&exist\_path\_len(G,temp,j,k-1))

return 1;

}

visited[i]=0;

}

return 0;

}

（4）算法分析

时间复杂度T(n)=O(k)，空间复杂度S(n)=O(log2k)。