第 6 章 图

1．选择题

（1）在一个图中，所有顶点的度数之和等于图的边数的（　　）倍。

A．1/2　　B．1　　C．2　　D．4

答案：C

（2）在一个有向图中，所有顶点的入度之和等于所有顶点的出度之和的（　　）倍。

A．1/2　　B．1　　C．2　　D．4

答案：B

解释：有向图所有顶点入度之和等于所有顶点出度之和。

（3）具有 n 个顶点的有向图最多有（　　）条边。

A．n　　B．n(n-1)　　C．n(n+1)　　D．n^2

答案：B

解释：有向图的边有方向之分，即为从 n 个顶点中选取 2 个顶点有序排列，结果为 n(n-1)。

（4）n 个顶点的连通图用邻接距阵表示时，该距阵至少有（　　）个非零元素。

A．n　　B．2(n-1)　　C．n/2　　D．n^2

答案：B

（5）G 是一个非连通无向图，共有 28 条边，则该图至少有（　　）个顶点。

A．7　　B．8　　C．9　　D．10

答案：C

解释：8 个顶点的无向图最多有 8*7/2=28 条边，再添加一个点即构成非连通无向图，故至少有 9 个顶点。

（6）若从无向图的任意一个顶点出发进行一次深度优先搜索可以访问图中所有的顶点，则该图一定是（　　）图。

A．非连通　　B．连通　　C．强连通　　D．有向

答案：B

解释：即从该无向图任意一个顶点出发有到各个顶点的路径，所以该无向图是连通图。

（7）下面（　　）算法适合构造一个稠密图 G 的最小生成树。

A．Prim 算法　　B．Kruskal 算法　　C．Floyd 算法　　D．Dijkstra 算法

答案：A

解释：Prim 算法适合构造一个稠密图 G 的最小生成树，Kruskal 算法适合构造一个稀疏图 G 的最小生成树。

（8）用邻接表表示图进行广度优先遍历时，通常借助（　　）来实现算法。

A．栈　　B. 队列　　C．树　　D．图

答案：B

解释：广度优先遍历通常借助队列来实现算法，深度优先遍历通常借助栈来实现算法。

（9）用邻接表表示图进行深度优先遍历时，通常借助（　　）来实现算法。

A．栈　　B. 队列　　C．树　　D．图

答案：A

解释：广度优先遍历通常借助队列来实现算法，深度优先遍历通常借助栈来实现算法。

（10）深度优先遍历类似于二叉树的（　　）。

A．先序遍历　　　B．中序遍历　　　　C．后序遍历　　　D．层次遍历

答案：A

（11）广度优先遍历类似于二叉树的（　　）。

A．先序遍历　　　B．中序遍历　　　　C．后序遍历　　　D．层次遍历

答案：D

（12）图的 BFS 生成树的树高比 DFS 生成树的树高（　　）。

A．小　　　　　　B．相等　　　　　　C．小或相等　　　D．大或相等

答案：C

解释：对于一些特殊的图，比如只有一个顶点的图，其 BFS 生成树的树高和 DFS 生成树的树高相等。一般的图，根据图的 BFS 生成树和 DFS 树的算法思想，BFS 生成树的树高比 DFS 生成树的树高小。

（13）已知图的邻接矩阵如图 6.30 所示，则从顶点 v_0 出发按深度优先遍历的结果是（　　）。

$$
\begin{array}{c}
v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6
\end{array}
\begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
$$

图 6.30　邻接矩阵

A. 0 2 4 3 1 5 6

B. 0 1 3 6 5 4 2

C. 0 1 3 4 2 5 6

D. 0 3 6 1 5 4 2

（14）已知图的邻接表如图 6.31 所示，则从顶点 v_0 出发按广度优先遍历的结果是（　　），按深度优先遍历的结果是（　　）。

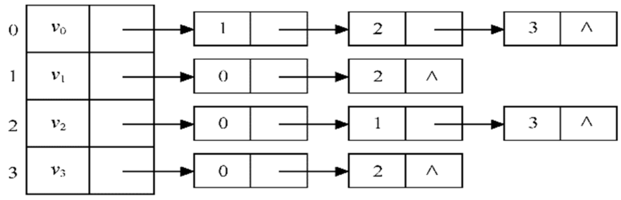


图 6.31　邻接表

A．0 1 3 2　　　　　B．0 2 3 1　　　　C．0 3 2 1　　　　　D．0 1 2 3

答案：D、D

（15）下面（　　）方法可以判断出一个有向图是否有环。

A．深度优先遍历　　　B．拓扑排序　　　C．求最短路径　　　D．求关键路径

答案：B

2．应用题

（1）已知图 6.32 所示的有向图，请给出：

① 每个顶点的入度和出度；

② 邻接矩阵；

③ 邻接表；

④ 逆邻接表。

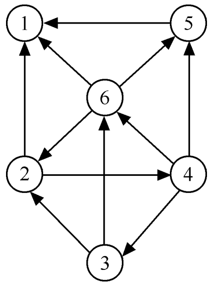


图 6.32　有向图

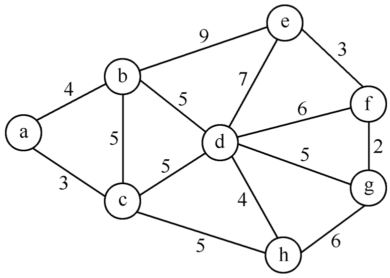


图 6.33　无向网

答案：

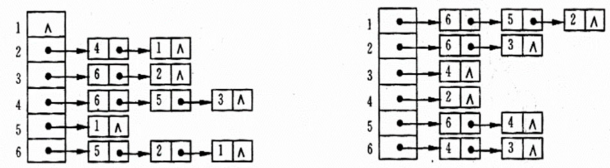
（1）

顶点	1	2	3	4	5	6
入度	3	2	1	1	2	2
出度	0	2	2	3	1	3

（2）邻接矩阵

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

（3）邻接表　　（4）逆邻接表



（2）已知如图 6.33 所示的无向网，请给出：

① 邻接矩阵；

② 邻接表；

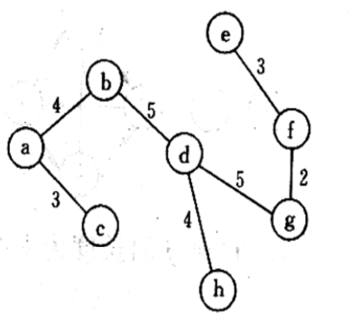
③ 最小生成树

答案：

①

$$\begin{bmatrix} \infty & 4 & 3 & \infty & \infty & \infty & \infty & \infty \\ 4 & \infty & 5 & 5 & 9 & \infty & \infty & \infty \\ 3 & 5 & \infty & 5 & \infty & \infty & \infty & 5 \\ \infty & 5 & 5 & \infty & 7 & 6 & 5 & 4 \\ \infty & 9 & \infty & 7 & \infty & 3 & \infty & \infty \\ \infty & \infty & \infty & 6 & 3 & \infty & 2 & \infty \\ \infty & \infty & \infty & 5 & \infty & 2 & \infty & 6 \\ \infty & \infty & 5 & 4 & \infty & \infty & 6 & \infty \end{bmatrix}$$

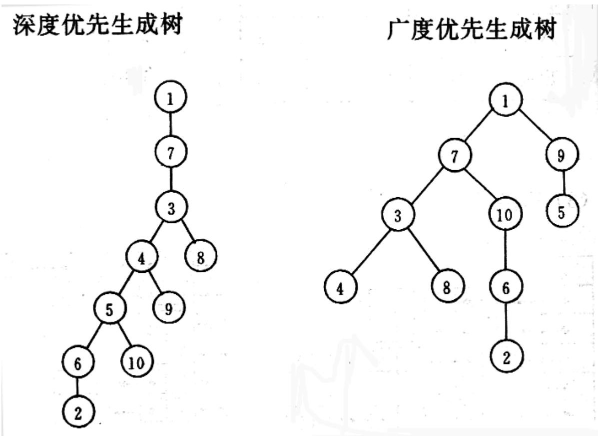
③



②

a → b 4 → c 3
b → a 4 → c 5 → d 5 → e 9
c → a 3 → b 5 → d 5 → h 5
d → b 5 → c 5 → e 7 → f 6 → g 5 → h 4
e → b 9 → d 7 → f 3
f → d 6 → e 3 → g 2
g → d 5 → f 2 → h 6

（3）已知图的邻接矩阵如图 6.34 所示。试分别画出自顶点 1 出发进行遍历所得的深度优先生成树和广度优先生成树。



（4）有向网如图 6.35 所示，试用迪杰斯特拉算法求出从顶点 a 到其他各顶点间的最短路径，完成表 6.9。

图 6.28　邻接矩阵

1	0	0	0	0	0	0	1	0	1	0
2	0	0	1	0	0	0	1	0	0	0
3	0	0	0	1	0	0	0	1	0	0
4	0	0	0	0	1	0	0	0	1	0
5	0	0	0	0	0	1	0	0	0	1
6	1	1	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	0	0	0	1
8	1	0	0	1	0	0	0	0	1	0
9	0	0	0	0	1	0	1	0	0	1
10	1	0	0	0	0	1	0	0	0	0

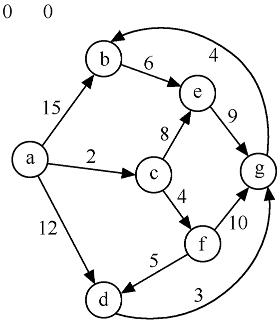


图 6.34　邻接矩阵　　　　图 6.35　有向网

表 6.9

D / 终点	i=1	i=2	i=3	i=4	i=5	i=6
b	15 (a,b)	15 (a,b)	15 (a,b)	15 (a,b)	15 (a,b)	**15 (a,b)**

c	**2** **(a,c)**					
d	12 (a,d)	12 (a,d)	11 (a,c,f,d)	**11** **(a,c,f,d)**		
e	∞	10 (a,c,e)	**10** **(a,c,e)**			
f	∞	**6** **(a,c,f)**				
g	∞	∞	16 (a,c,f,g)	16 (a,c,f,g)	**14** **(a,c,f,d,g)**	
S 终 点 集	{a,c}	{a,c,f}	{a,c,f,e}	{a,c,f,e,d}	{a,c,f,e,d,g}	{a,c,f,e,d,g,b}

（5）试对图 6.36 所示的 AOE-网：

① 求这个工程最早可能在什么时间结束；

② 求每个活动的最早开始时间和最迟开始时间；

③ 确定哪些活动是关键活动

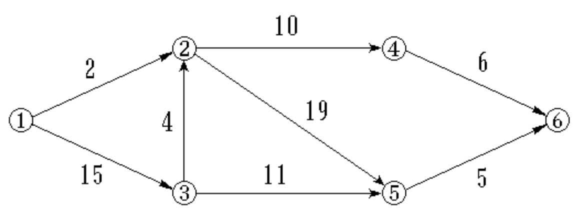


图 6.36　AOE-网

答案：按拓扑有序的顺序计算各个顶点的最早可能开始时间 *Ve* 和最迟允许开始时间 *Vl*。然后再计算各个活动的最早可能开始时间 e 和最迟允许开始时间 *l*，根据 *l* - *e* = 0? 来确定关键活动，从而确定关键路径。

	1 ∂	2 ÷	3 •	4 ≠	5 ≡	6 ≈
Ve	0	19	15	29	38	43
Vl	0	19	15	37	38	43

	<1, 2>	<1, 3>	<3, 2>	<2, 4>	<2, 5>	<3, 5>	<4, 6>	<5, 6>
e	0	0	15	19	19	15	29	38
l	17	0	15	27	19	27	37	38
-e	17	0	0	8	0	12	8	0

此工程最早完成时间为 43。关键路径为<1, 3><3, 2><2, 5><5, 6>

3．算法设计题

（1）分别以邻接矩阵和邻接表作为存储结构，实现以下图的基本操作：

① 增加一个新顶点 v，InsertVex(G, v);

② 删除顶点 v 及其相关的边，DeleteVex(G, v);

③ 增加一条边<v， w>，InsertArc(G, v, w);

④ 删除一条边<v， w>，DeleteArc(G, v, w)。

［算法描述］

假设图 G 为有向无权图，以邻接矩阵作为存储结构四个算法分别如下：

① 增加一个新顶点 v

```
Status Insert_Vex(MGraph &G, char v)//在邻接矩阵表示的图 G 上插入顶点 v
{
   if(G.vexnum+1)>MAX_VERTEX_NUM return INFEASIBLE;
   G.vexs[++G.vexnum]=v;
   return OK;
}//Insert_Vex
```

② 删除顶点 v 及其相关的边，

```
Status Delete_Vex(MGraph &G,char v)//在邻接矩阵表示的图 G 上删除顶点 v
{
n=G.vexnum;
if((m=LocateVex(G,v))<0) return ERROR;
G.vexs[m]<->G.vexs[n]; //将待删除顶点交换到最后一个顶点
for(i=0;i<n;i++)
{
G.arcs[m]=G.arcs[n];
G.arcs[m]=G.arcs[n]; //将边的关系随之交换
}
G.arcs[m][m].adj=0;
G.vexnum--;
return OK;
}//Delete_Vex
```

分析:如果不把待删除顶点交换到最后一个顶点的话,算法将会比较复杂,而伴随着大量元素的移动,时间复杂度也会大大增加。

③ 增加一条边<v，w>

```
Status Insert_Arc(MGraph &G,char v,char w)//在邻接矩阵表示的图 G 上插入边(v,w)
{
   if((i=LocateVex(G,v))<0) return ERROR;
   if((j=LocateVex(G,w))<0) return ERROR;
   if(i==j) return ERROR;
   if(!G.arcs[j].adj)
{
   G.arcs[j].adj=1;
   G.arcnum++;
}
      return OK;
}//Insert_Arc
```

④ 删除一条边<v，w>

```
Status Delete_Arc(MGraph &G,char v,char w)//在邻接矩阵表示的图 G 上删除边(v,w)
```

```
{
if((i=LocateVex(G,v))<0) return ERROR;
if((j=LocateVex(G,w))<0) return ERROR;
if(G.arcs[j].adj)
{
G.arcs[j].adj=0;
G.arcnum--;
}
return OK;
}//Delete_Arc
```

以邻接表作为存储结构，本题只给出 Insert_Arc 算法.其余算法类似。

```
Status Insert_Arc(ALGraph &G,char v,char w)//在邻接表表示的图 G 上插入边(v,w)
{
if((i=LocateVex(G,v))<0) return ERROR;
if((j=LocateVex(G,w))<0) return ERROR;
p=new ArcNode;
p->adjvex=j;p->nextarc=NULL;
if(!G.vertices.firstarc) G.vertices.firstarc=p;
else
{
for(q=G.vertices.firstarc;q->q->nextarc;q=q->nextarc)
if(q->adjvex==j) return ERROR; //边已经存在
q->nextarc=p;
}
G.arcnum++;
return OK;
}//Insert_Arc
```

（2）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。

[算法描述]

```
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;
SetEmpty(s);
Push(s,v);
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!=GetTop(s)) Push(s,w);     //图中有环时 w==GetTop(s)
        }
    }
}
```

（3）设计一个算法，求图 G 中距离顶点 v 的最短路径长度最大的一个顶点，设 v 可达其余各个顶点。

[题目分析]

利用 Dijkstra 算法求 v0 到其它所有顶点的最短路径，分别保存在数组 D[i]中，然后求出 D[i]中值最大的数组下标 m 即可。

[算法描述]

```
int ShortestPath_MAX(AMGraph G, int v0){
    //用 Dijkstra 算法求距离顶点 v0 的最短路径长度最大的一个顶点 m
    n=G.vexnum;                          //n 为 G 中顶点的个数
    for(v = 0; v<n; ++v){                //n 个顶点依次初始化
        S[v] = false;                    //S 初始为空集
        D[v] = G.arcs[v0][v];            //将 v0 到各个终点的最短路径长度初始化
        if(D[v]< MaxInt)   Path [v]=v0;  //如果 v0 和 v 之间有弧，则将 v 的前驱置为 v0
        else Path [v]=-1;                //如果 v0 和 v 之间无弧，则将 v 的前驱置为-1
      }//for
      S[v0]=true;                        //将 v0 加入 S
      D[v0]=0;                           //源点到源点的距离为 0
      /*开始主循环，每次求得 v0 到某个顶点 v 的最短路径，将 v 加到 S 集*/
      for(i=1;i<n; ++i){                 //对其余 n−1 个顶点，依次进行计算
        min= MaxInt;
        for(w=0;w<n; ++w)
          if(!S[w]&&D[w]<min)
              {v=w; min=D[w];}           //选择一条当前的最短路径，终点为 v
        S[v]=true;                       //将 v 加入 S
        for(w=0;w<n; ++w)                //更新从 v0 到 V−S 上所有顶点的最短路径长度
        if(!S[w]&&(D[v]+G.arcs[v][w]<D[w])){
              D[w]=D[v]+G.arcs[v][w];    //更新 D[w]
              Path [w]=v;                //更改 w 的前驱为 v
        }//if
    }//for
/*最短路径求解完毕，设距离顶点 v0 的最短路径长度最大的一个顶点为 m */
Max=D[0];
m=0;
```

```
for(i=1;i<n;i++)
 if(Max<D[i]) m=i;
return m;
}
```

（4）试基于图的深度优先搜索策略写一算法，判别以邻接表方式存储的有向图中是否存在由顶点 v_i 到顶点 v_j 的路径（i≠j）。

[题目分析]

引入一变量 level 来控制递归进行的层数

[算法描述]

```
int visited[MAXSIZE]; //指示顶点是否在当前路径上
int level＝1;//递归进行的层数
int exist_path_DFS(ALGraph G,int i,int j)//深度优先判断有向图 G 中顶点 i 到顶点 j
是否有路径,是则返回 1,否则返回 0
{
  if(i==j) return 1; //i 就是 j
  else
  {
    visited[i]=1;
    for(p=G.vertices[i].firstarc;p;p=p->nextarc，level--)
    { level++;
      k=p->adjvex;
      if(!visited[k]&&exist_path(k,j)) return 1;//i 下游的顶点到 j 有路径
}//for
  }//else
if (level==1)   return 0;
}//exist_path_DFS
```

（5）采用邻接表存储结构，编写一个算法，判别无向图中任意给定的两个顶点之间是否存在一条长度为为 k 的简单路径。

[算法描述]

```
int visited[MAXSIZE];
int exist_path_len(ALGraph G,int i,int j,int k)
//判断邻接表方式存储的有向图 G 的顶点 i 到 j 是否存在长度为 k 的简单路径
{if(i==j&&k==0) return 1; //找到了一条路径,且长度符合要求
 else if(k>0)
  {visited[i]=1;
   for(p=G.vertices[i].firstarc;p;p=p->nextarc)
    {l=p->adjvex;
     if(!visited[l])
        if(exist_path_len(G,l,j,k-1)) return 1; //剩余路径长度减一
    }//for
   visited[i]=0; //本题允许曾经被访问过的结点出现在另一条路径中
  }//else
 return 0; //没找到
}//exist_path_len
```