

9.11 选择11

(11) 创建一个包括 n 个结点的有序单链表的时间复杂度是 ()。

- A. $O(1)$
- C. $O(n^2)$

- B. $O(n)$
- D. $O(n \log_2 n)$

C

创建单链表的时间复杂度为 $O(n)$ ，创建有序单链表，每生成一个新结点都要与已有结点比较来确定插入位置，第 i 个结点最坏情况下要比较 i 次，所以时间复杂度为 $O(n^2)$

考点：有序链表的创建

同理，链表、栈、队列等数据结构的基础操作的时间复杂度也要掌握。

9.29 选择9

(9) 若一个栈以向量 $V[1..n]$ 存储，初始栈顶指针 top 设为 $n+1$ ，则元素 x 进栈的正确操作是()。

- A. $top++; V[top]=x;$
- B. $V[top]=x; top++;$
- C. $top--; V[top]=x;$
- D. $V[top]=x; top--;$

C

初始栈顶指针是 $n+1$ ，说明元素从 V 的高端地址入栈，则 top 需要先下移变为 n ($top--$)，之后将元素存储到 $V[n]$ 中，所以选 C

考点：栈的结构，入栈的操作

同理，进出栈、队列、取栈顶元素等基础操作需要掌握

10.20 选择7

(7) 设有数组 $A[i,j]$, 数组的每个元素长度为 3 字节, i 的值为 1~8, j 的值为 1~10, 数组从内存首地址 BA 开始顺序存放, 当用以列为主存放时, 元素 $A[5,8]$ 的存储首地址为()。

- A. BA + 141
- B. BA + 180
- C. BA + 222
- D. BA + 225

B

i 的值为 1~8, j 的值是 1~10, 可以知道数组是 8*10 的大小。列为主存放, 对元素 a_{ij} , 则前 $j-1$ 列都已经存满 (j 从 1 开始), 则 $A[5,8]$ 的首地址为 $BA + 7*8*3 + 4*3 = BA + 180$ 。

注意 5 表示的是第 5 行, 8 表示第 8 列。 i 虽然是行标, 但 i 的范围大小表示了列的大小, 同理对 j 。

| | | | | |
|--|--|--|--|--|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

行为主存放

| | | | | |
|--|--|--|--|--|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

列为主存放

10.20选择11

- (11) 设二维数组 $A[1..m, 1..n]$ (即 m 行 n 列) 按行存储在数组 $B[1..m \times n]$ 中, 则二维数组元素 $A[i,j]$ 在一维数组 B 中的下标为 ()。
- A. $(i-1) \times n + j$
 - B. $(i-1) \times n + j - 1$
 - C. $i \times (j-1)$
 - D. $j \times m + i - 1$

A

按行存储, 说明前*i-1*行是满的, 每行有*n*个元素, $A[i, j]$ 前共有
 $n * (i-1) + j-1$ 个元素, 下标从1开始, 则 $A[i,j]$ 的下标是 $n*(i-1)+j$ 。

第1个元素前有0个元素, 下标是1。

与上一题类似

10.20 选择14, 15

(14) 广义表((a,b,c,d))的表头是()，表尾是()。

- A. a
- B. ()
- C. (a, b, c, d)
- D. (b, c, d)

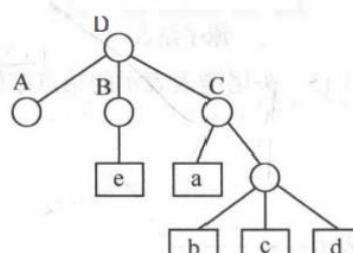
(15) 设广义表 $L = ((a, b, c))$ ，则 L 的长度和深度分别为()。

- A. 1 和 1
- B. 1 和 3
- C. 1 和 2
- D. 2 和 3

广义表一般记作

$$LS = (a_1, a_2, \dots, a_n)$$

其中， LS 是广义表 (a_1, a_2, \dots, a_n) 的名称， n 是其长度。在线性表的定义中， a_i ($1 \leq i \leq n$) 只限于是单个元素。而在广义表的定义中， a_i 可以是单个元素，也可以是广义表，分别称为广义表 LS 的原子和子表。习惯上，用大写字母表示广义表的名称，用小写字母表示原子。



(1) $A = ()$ —— A 是一个空表，其长度为零。

(2) $B = (e)$ —— B 只有一个原子 e ，其长度为 1。

(3) $C = (a, (b, c, d))$ —— C 的长度为 2，两个元素分别为原子 a 和子表 (b, c, d) 。

(4) $D = (A, B, C)$ —— D 的长度为 3，3 个元素都是广义表。显然，将子表的值代入后，则有

$D = ((), (e), (a, (b, c, d)))$ 。

图 4.14 广义表的图形表示

14. C,B 广义表((a,b,c,d))的表头为子表 (a, b, c, d) ，表尾为除去表头之外，由其余元素构成的表，表尾一定是个广义表， $((a, b, c, d))$ 去除 (a, b, c, d) 只剩下空表()

15. C 广义表长度是指广义表所含元素个数，就像是集合个数，深度是指括号的层数。

11.3 选择8

(8) 在一棵度为 4 的树 T 中，若有 20 个度为 4 的结点，10 个度为 3 的结点，1 个度为 2 的结点，10 个度为 1 的结点，则树 T 的叶结点个数是（ ）。

- A. 41
 - B. 82
 - C. 113
 - D. 122
-

B

设叶子结点为n，则一共有 $m=n+20+10+1+10=n+41$ 个结点，树的分支个数为 $B=20*4+10*3+1*2+10*1=122$ ，因为 $m=B+1$,所以 $n=82$

$m=B+1$: 对于树，除根节点外，每个结点都有一个父节点，所以计算所有度的大小能计算除所有有父结点的结点个数B，总结点数量即为B+1

11.26选择12

(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

(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

B

n 个顶点连通图至少有 $n-1$ 条边，无向图中每条边关联两个顶点，每条边存储两次。比如边*i-j*，则需要存 $M[i,j]$ 和 $M[j,i]$ ，因此至少有 $2(n-1)$ 个非零元素

C

一个 n 个结点连通无向图最多有 $\frac{n(n-1)}{2}$ 条边，若让顶点最少，则每个连通片都完全连通。

8顶点的连通无向图最多有 $8*7/2=28$ 条边，再添加一个点构成非连通无向图。

有向图 VS 无向图

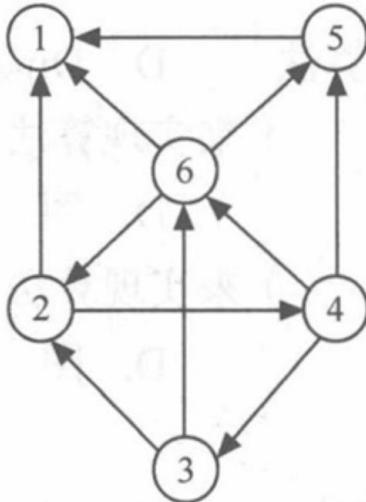


图 6.32 有向图

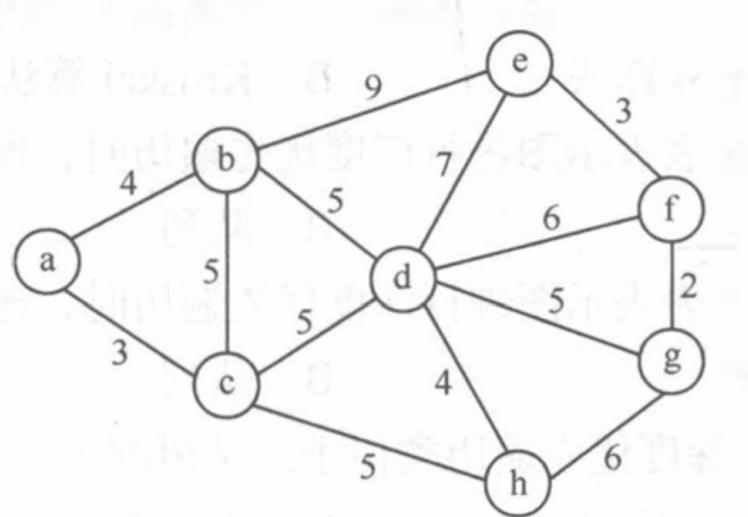


图 6.33 无向网

入度、出度区别

邻接点的区别。在深度优先遍历时，在有向图中，从顶点6出发，下一步只能到达2或者5，不能到达3和4。有向边是单向到达。

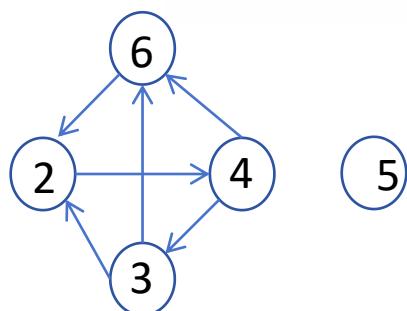
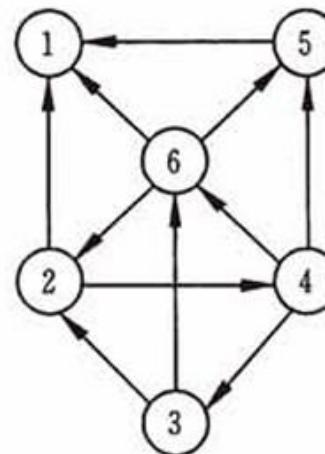
边的权重/顶点距离的区别

邻接矩阵/邻接表的区别

强连通分量

◆7.1① 已知如右图所示的有向图,请给出该图的

- (1) 每个顶点的入/出度;
- (2) 邻接矩阵;
- (3) 邻接表;
- (4) 逆邻接表;
- (5) 强连通分量。



(11) **强连通图**和**强连通分量**: 在有向图 G 中, 如果对于每一对 $v_i, v_j \in V, v_i \neq v_j$, 从 v_i 到 v_j 和从 v_j 到 v_i 都存在路径, 则称 G 是**强连通图**。有向图中的极大**强连通子图**称作有向图的**强连通分量**。例如图 6.1(a) 中的 G_1 不是**强连通图**, 但它有两个**强连通分量**, 如图 6.4 所示。

深度优先算法

注意此处是访问第一个未被访问的邻接点
而不是所有未被访问的邻接点

1. 深度优先搜索遍历的过程

深度优先搜索 (Depth First Search, DFS) 遍历类似于树的先序遍历, 是树的先序遍历的推广。

对于一个连通图，深度优先搜索遍历的过程如下。

(1) 从图中某个顶点 v 出发, 访问 v 。

(2) 找出刚访问过的顶点的第一个未被访问的邻接点，访问该顶点。以该顶点为新顶点，重复此步骤，直至刚访问过的顶点没有未被访问的邻接点为止。

(3) 返回前一个访问过的且仍有未被访问的邻接点的顶点, 找出该顶点的下一个未被访问的邻接点, 访问该顶点。

(4) 重复步骤(2)和(3), 直至图中所有顶点都被访问过, 搜索结束。

以图 6.17(a) 中所示的无向图 G_4 为例, 深度优先搜索遍历图的过程如图 6.17(b) 所示^①。具体过程如下。

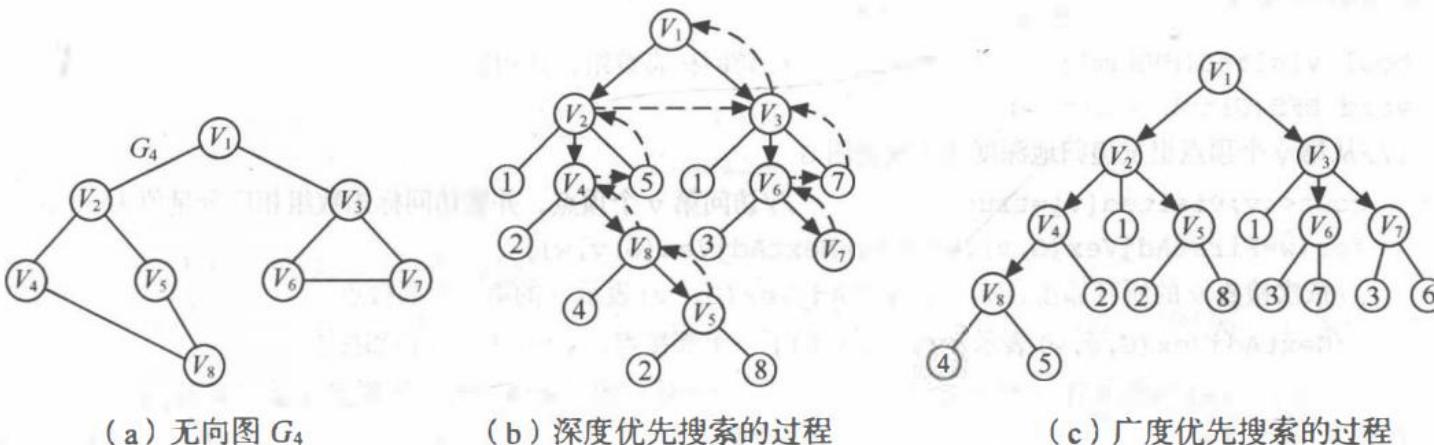
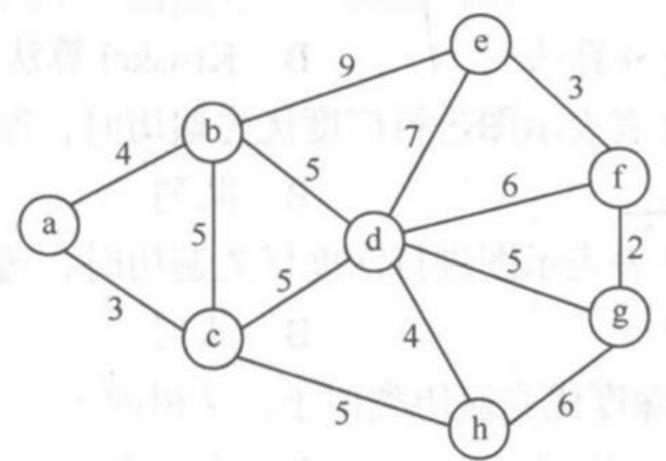


图 6.17 遍历图的过程

深度优先算法

非递归的一种实现方式

```
22 void DFS(Graph G, int v){  
23     InitStack(S);  
24     Push(S, v);  
25     visited[v] = true;  
26     while(!IsEmpty(S)){  
27         Pop(S, k);  
28         if(!visited[k]){// k未被访问  
29             visited[k] = true; // 访问第k个顶点  
30             p=G.vertices[k].firstarc;  
31             // 遍历所有邻接点  
32             while(p){  
33                 w = p->adjvex;  
34                 // 如果该邻接点未被访问则入栈  
35                 if(!visited[w]){  
36                     Push(S, w);  
37                 }  
38                 p=p->nextarc;  
39             }  
40         }  
41     }  
42 }
```



很多人写成了

```
if(!visited[w]) {  
    Push(S, w)  
    visited[w] = True  
}
```



把所有相邻结点的visited值都
置为True

熟悉dfs算法，那么按照这个算法也就很容易得到深度优先生成树

- 审题：是应用题还是算法题。