



数据结构试题库及答案 - asd

Programming (Victory University)

数据结构试题及答案

一、单项选择题

- (1) 一个算法应该是 ()。
- A) 程序
B) 问题求解步骤的描述
C) 要满足五个基本属性
D) A 和 C
- (2) 算法指的是 ()。
- A) 计算机程序
B) 解决问题的计算方法
C) 排序算法
D) 解决问题的有限运算序列。
- (3) 与数据元素本身的形式、内容、相对位置、个数无关的是数据的 ()。
- A) 存储结构
B) 逻辑结构
C) 算法
D) 操作
- (4) 从逻辑上可以把数据结构分为 () 两大类。
- A) 动态结构、静态结构
B) 顺序结构、链式结构
C) 线性结构、非线性结构
D) 初等结构、构造型结构
- (5) 下列叙述中正确的是 ()。
- A) 一个逻辑数据结构只能有一种存储结构
B) 数据的逻辑结构属于线性结构，存储结构属于非线性结构
C) 一个逻辑数据结构可以有多种存储结构，且各种存储结构不影响数据处理的效率
D) 一个逻辑数据结构可以有多种存储结构，且各种存储结构影响数据处理的效率
- (6) 数据的基本单位是 ()。
- A) 数据项
B) 数据类型
C) 数据元素
D) 数据变量
- (7) 下列程序的时间复杂度为 ()。
- ```
i=0; s=0;
while (s<n)
{ i++; s=s+i; }
```
- A)  $O(\sqrt{n})$   
B)  $O(\sqrt{2n})$   
C)  $O(n)$   
D)  $O(n^2)$
- (8) 下列程序段的渐进时间复杂度为 ( )。
- ```
for( int i=1; i<=n; i++)
for( int j=1; j<= m; j++)
A[i][j] = i*j;
```
- A) $O(m^2)$
B) $O(n^2)$
C) $O(m*n)$
D) $(m+n)$
- (9) 程序段如下：
- ```
sum=0;
for(i=1; i<=n; i++)
for(j=1; j<=n; j++)
```

sum++;

其中  $n$  为正整数, 则最后一行的语句频度在最坏情况下是 ( )。

- A)  $O(n)$                       B)  $O(n \log n)$                       **C)  $O(n^3)$**                       D)  $O(n^2)$

(10) 在下面的程序段中, 对  $x$  的赋值语句的频度为 ( )。

```
for (i=1; i<=n; i++)
 for (j=1; j<=n; j++)
 x:=x+1;
```

- A)  $O(2n)$                       B)  $O(n)$                       **C)  $O(n^2)$**                       D)  $O(\log_2 n)$

(11) 程序段 for (i:=n-1; i>=1; i--)

```
for (j:=1; j<=i; j++)
 if (a[j]>a[j+1])
 { t=a[j]; a[j]=a[j+1]; a[j+1]=t; }
```

其中  $n$  为正整数, 则最后一行的语句频度在最坏情况下是 ( )。

- A)  $O(n)$                       B)  $O(n \log n)$                       C)  $O(n^3)$                       **D)  $O(n^2)$**

(12) 设有一个递归算法如下:

```
int fact(int n)
{ /* 大于等于 0 */
 if (n<=0) return 1;
 else return n*fact(n-1);
}
```

则计算 fact( $n$ ) 需要调用该函数的次数为 ( )。

- A)  $n$                       **B)  $n+1$**                       C)  $n+2$                       D)  $n-1$

(13) 下述程序段中语句①的频度是 ( )。

```
s=0;
for(i=1;i<=m;i++)
 for(j=0;j<=i;j++)
 s+=j;
```

- A)  $\frac{(m+1)(m-1)}{2}$                       B)  $\frac{m(m-1)}{2}$                       **C)  $\frac{(m+2)(m-1)}{2}$**                       D)  $\frac{m(m+1)}{2}$

(14) 若某线性表中最常用的操作是在最后一个元素之后插入一个元素和删除第一个元素, 则最节省运算时间的存储方式是 ( )。

- A) 单链表                      B) 仅有头指针的单循环链表  
C) 双链表                      **D) 仅有尾指针的单循环链表**

(1) 求循环链表中当前结点的后继和前驱的时间复杂度分别是 ( )。

- A)  $O(n)$  和  $O(1)$                       B)  $O(1)$  和  $O(1)$                       **C)  $O(1)$  和  $O(n)$**                       D)  $O(n)$  和  $O(n)$

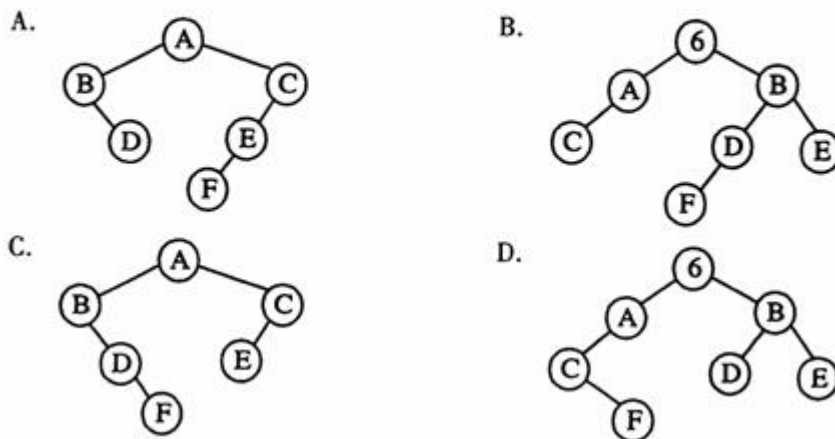


- A) head==NULL      B) head->next==NULL      C) head!=NULL      D) head->next==head
- (28) 某线性表中最常用的操作是在最后一个元素之后插入一个元素和删除第一个元素, 则采用 ( ) 存储方式最节省运算时间。
- A) 单链表      B) 仅有头指针的单循环链表  
C) 双链表      D) 仅有尾指针的单循环链表
- (29) 若允许表达式内多种括号混合嵌套, 则为检查表达式中括号是否正确配对的算法, 通常选用的辅助结构是 ( )。
- A) 栈      B) 线性表      C) 队列      D) 二叉排序树
- (30) 顺序栈 S 中 top 为栈顶指针, 指向栈顶元素所在的位置, elem 为存放栈的数组, 则元素 e 进栈操作的主要语句为 ( )。
- A) s.elem[top] = e; s.top = s.top + 1;      B) s.elem[top + 1] = e; s.top = s.top + 1;  
C) s.top = s.top + 1; s.elem[top + 1] = e;      D) s.top = s.top + 1; s.elem[top] = e;
- (31) 循环队列 sq 中, 用数组 elem[0..25] 存放数据元素, sq.front 指示队头元素的前一个位置, sq.rear 指示队尾元素的当前位置, 设当前 sq.front 为 20, sq.rear 为 12, 则当前队列中的元素个数为 ( )。
- A) 8      B) 16      C) 17      D) 18
- (32) 链式栈与顺序栈相比, 一个比较明显的优点是 ( )。
- A) 插入操作更加方便      B) 通常不会出现栈满的情况  
C) 不会出现栈空的情况      D) 删除操作更加方便
- (33) 一个递归的定义可以用递归过程求解, 也可以用非递归过程求解, 但单从运行时间来看, 通常递归过程比非递归过程 ( )。
- A) 较快      B) 较慢      C) 相同      D) 不定
- (34) 若已知一个栈的入栈序列是 1, 2, 3, 4, ..., n, 其输出序列为 p<sub>1</sub>, p<sub>2</sub>, p<sub>3</sub>, ..., p<sub>n</sub>, 若 p<sub>1</sub> = n, 则 p<sub>i</sub> 为 ( )。
- A) i      B) n = i      C) n - i + 1      D) 不确定
- (35) 一个栈的入栈序列是 a, b, c, d, e, 则栈的不可能的输出序列是 ( )。
- A) edcba      B) decba      C) dceab      D) abcde
- (36) 若进栈序列为 1, 2, 3, 4, 5, 6, 且进栈和出栈可以穿插进行, 则不可能出现的出栈序列是 ( )。
- A) 2, 4, 3, 1, 5, 6      B) 3, 2, 4, 1, 6, 5  
C) 4, 3, 2, 1, 5, 6      D) 2, 3, 5, 1, 6, 4
- (37) 对于栈操作数据的原则是 ( )。
- A) 先进先出      B) 后进先出      C) 后进后出      D) 不分顺序
- (38) 栈和队列的共同点是 ( )。

- This document is available free of charge on

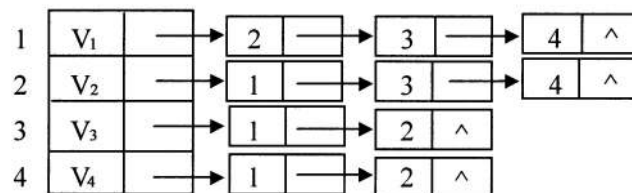
- A) 空或只有一个结点                      B) 高度等于其节点数  
C) 任一结点无左孩子                    D) 任一结点无右孩子
- (51) 含有 10 个结点的二叉树中, 度为 0 的结点数为 4, 则度为 2 的结点数为 ( )  
A)3                      B)4                      C)5                      D)6
- (52) 除第一层外, 满二叉树中每一层结点个数是上一层结点个数的 ( )  
A)1/2 倍                      B)1 倍                      C) 2 倍                      D) 3 倍
- (53) 对一棵有 100 个结点的完全二叉树按层编号, 则编号为 49 的结点, 它的父结点的编号为 ( )  
A)24                      B)25                      C)98                      D)99
- (54) 可以惟一地转化成一棵一般树的二叉树的特点是 ( )  
A)根结点无左孩子    B)根结点无右孩子    C)根结点有两个孩子    D)根结点没有孩子
- (55) 设高度为  $h$  的二叉树上只有度为 0 和度为 2 的结点, 则此类二叉树中所包含的结点数至少为 ( )。  
A)  $2h$                       B)  $2h-1$                       C)  $2h+1$                       D)  $h+1$
- (56) 在一棵度为 3 的树中, 度为 3 的节点个数为 2, 度为 2 的节点个数为 1, 则度为 0 的节点个数为 ( )。  
A) 4                      B) 5                      C) 6                      D) 7
- (57) 设森林  $F$  对应的二叉树为  $B$ , 它有  $m$  个结点,  $B$  的根为  $p$ ,  $p$  的右子树结点个数为  $n$ , 森林  $F$  中第一棵子树的结点数是 ( )。  
A) $m-n$                       B) $m-n-1$                       C)  $n+1$                       D) 条件不足, 无法确定
- (58) 将一株有 100 个节点的完全二叉树从上到下, 从左到右依次进行编号, 根节点的编号为 1, 则编号为 49 的节点的左孩子编号为 ( )。  
A) 98                      B) 89                      C) 50                      D) 没有孩子
- (59) 下列图示的顺序存储结构表示的二叉树是(A)

|   |   |   |   |   |   |   |   |   |   |    |    |    |
|---|---|---|---|---|---|---|---|---|---|----|----|----|
| 6 | A | B | C |   | D | E |   |   |   |    |    | F  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |



- (60) 树最适合用来表示 ( )。
- A) 有序数据元素                      B) 无序数据元素  
**C) 元素之间具有分支层次关系的数据**                      D) 元素之间无联系的数据
- (61) 在一个非空二叉树的中序遍历序列中, 根结点的右边 ( )。
- A) 只有右子树上的所有结点**                      B) 只有右子树上的部分结点  
 C) 只有左子树上的部分结点                      D) 只有左子树上的所有结点
- (62) 任何一棵二叉树的叶结点在先序、中序和后序遍历序列中相对次序 ( )。
- A) 不发生改变                      B) 发生改变                      C) 不能确定                      **D) 以上都不对**
- (63) 在有  $n$  个叶子结点的哈夫曼树中, 其结点总数为 ( )。
- A) 不确定                      B)  $2n$                       C)  $2n+1$                       **D)  $2n-1$**
- (64) 权值为  $\{1,2,6,8\}$  的四个结点构成的哈夫曼树的带权路径长是 ( )。
- A) 18                      B) 28                      C) 19                      **D) 29**
- (65) 对一个满二叉树,  $m$  个树叶,  $k$  个分枝结点,  $n$  个结点, 则 ( )。
- A)  $n=m+1$                       B)  $m+1=2n$                       C)  $m=k-1$                       **D)  $n=2k+1$**
- (66) 在含有  $n$  个顶点和  $e$  条边的无向图的邻接矩阵中, 零元素的个数为 ( )。
- A)  $e$                       B)  $2e$                       C)  $n^2-e$                       **D)  $n^2-2e$**
- (67) 若采用邻接矩阵存储一个  $n$  个顶点的无向图, 则该邻接矩阵是一个 ( )。
- A) 上三角矩阵                      B) 稀疏矩阵                      C) 对角矩阵                      **D) 对称矩阵**
- (68) 在一个图中, 所有顶点的度数之和等于所有边数的 ( ) 倍。
- A)  $1/2$                       B) 1                      **C) 2**                      D) 4
- (69) 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
- (70) 对于含  $n$  个顶点和  $e$  条边的图, 采用邻接矩阵表示的空间复杂度为 ( )。
- A)  $O(n)$                       B)  $O(e)$                       C)  $O(n+e)$                       **D)  $O(n^2)$**
- (71) 如果求一个连通图中以某个顶点为根的高度最小的生成树, 应采用 ( )。
- A) 深度优先搜索算法                      B) 广度优先搜索算法  
 C) 求最小生成树的 prim 算法                      D) 拓扑排序算法
- (72)  $n$  个顶点的连通图至少中含有 ( )。
- A)  $n-1$**                       B)  $n$                       C)  $n+1$                       D) 0
- (73)  $n$  个顶点的完全有向图中含有 ( )。
- A)  $n-1$  条有向边                      B)  $n$  条有向边                      C)  $n(n-1)/2$  条有向边                      **D)  $n(n-1)$  条有向边**

- (74) 假设一个有  $n$  个顶点和  $e$  条弧的有向图用邻接表表示, 则删除预某个顶点  $v_i$  相关的所有弧的时间复杂度是 ( )。
- A)  $O(n)$                       B)  $O(e)$                       C)  $O(n+e)$                       D)  $O(n \cdot e)$
- (75) 在无向图中定义顶点  $V_i$  域  $V_j$  之间的路径为从  $V_i$  到达  $V_j$  的一个 ( )。
- A) 顶点序列                      B) 边序列                      C) 权值总和                      D) 边的条数
- (76) 无向图  $G=(V,E)$ , 其中:  $V=\{a,b,c,d,e,f\}$ ,  $E=\{(a,b),(a,e),(a,c),(b,e),(c,f),(f,d),(e,d)\}$ , 对该图进行深度优先遍历, 得到的顶点序列正确的是 ( )。
- A) a,b,e,c,d,f                      B) a,c,f,e,b,d                      C) a,e,b,c,f,d                      D) a,e,d,f,c,b
- (77) 下面哪一方法可以判断出一个有向图是否有环 (回路)。
- A) 求节点的度                      B) 拓扑排序                      C) 求最短路径                      D) 求关键路径
- (78) 图的广度优先搜索类似于树的 ( ) 次序遍历。
- A) 先根                      B) 中根                      C) 后根                      D) 层次
- (79) 在图采用邻接表存储时, 求最小生成树的 Prim 算法的时间复杂度为 ( )。
- A)  $O(n)$                       B)  $O(n+e)$                       C)  $O(n^2)$                       D)  $O(n^3)$
- (80) 已知有向图  $G=(V,E)$ , 其中  $V=\{V_1,V_2,V_3,V_4,V_5,V_6,V_7\}$ ,  $E=\{<V_1,V_2>,<V_1,V_3>,<V_1,V_4>,<V_2,V_5>,<V_3,V_5>,<V_3,V_6>,<V_4,V_6>,<V_5,V_7>,<V_6,V_7>\}$ ,  $G$  的拓扑序列是 ( )。
- A)  $V_1,V_3,V_4,V_6,V_2,V_5,V_7$                       B)  $V_1,V_3,V_2,V_6,V_4,V_5,V_7$   
C)  $V_1,V_3,V_4,V_5,V_2,V_6,V_7$                       D)  $V_1,V_2,V_5,V_3,V_4,V_6,V_7$
- (81) 关键路径是事件结点网络中 ( )。
- A) 从源点到汇点的最长路径                      B) 从源点到汇点的最短路径  
C) 最长回路                      D) 最短回路
- (82) 有  $n$  个结点的有向完全图的弧数是 ( )。
- A)  $n^2$                       B)  $2n$                       C)  $n(n-1)$                       D)  $2n(n+1)$
- (83) 设图的邻接链表如题 12 图所示, 则该图的边的数目是 ( )。



83 题图

- A) 4                      B) 5                      C) 10                      D) 20
- (84) 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
- (85) 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。

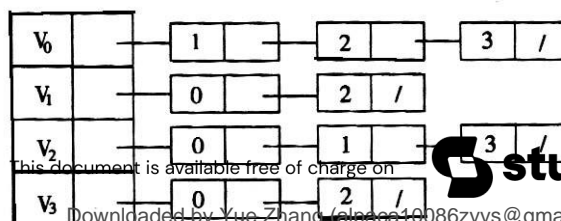
- A) 1/2                      B) 1                      C) 2                      D) 4
- (86) 有 8 个结点的无向图最多有 ( ) 条边。  
A) 14                      B) 28                      C) 56                      D) 112
- (87) 有 8 个结点的无向连通图最少有 ( ) 条边。  
A) 5                      B) 6                      C) 7                      D) 8
- (88) 有 8 个结点的有向完全图有 ( ) 条边。  
A) 14                      B) 28                      C) 56                      D) 112
- (89) 用邻接表表示图进行广度优先遍历时, 通常是采用 ( ) 来实现算法的。  
A) 栈                      B) 队列                      C) 树                      D) 图
- (90) 用邻接表表示图进行深度优先遍历时, 通常是采用 ( ) 来实现算法的。  
A) 栈                      B) 队列                      C) 树                      D) 图
- (91) 已知图的邻接矩阵, 根据算法思想, 则从顶点 0 出发按深度优先遍历的结点序列是 ( )。

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

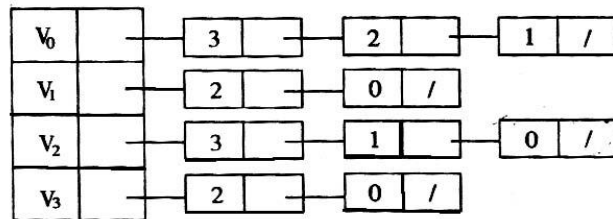
- A) 0 2 4 3 1 5 6                      B) 0 1 3 6 5 4 2                      C) 0 4 2 3 1 6 5                      D) 0 3 6 1 5 4 2

建议: 0 1 3 4 2 5 6

- (92) 已知图的邻接矩阵同上题 8, 根据算法, 则从顶点 0 出发, 按深度优先遍历的结点序列是 ( )。  
A) 0 2 4 3 1 5 6                      B) 0 1 3 5 6 4 2                      C) 0 4 2 3 1 6 5                      D) 0 1 3 4 2 5 6
- (93) 已知图的邻接矩阵同上题 8, 根据算法, 则从顶点 0 出发, 按广度优先遍历的结点序列是 ( )。  
A) 0 2 4 3 6 5 1                      B) 0 1 3 6 4 2 5                      C) 0 4 2 3 1 5 6                      D) 0 1 3 4 2 5 6  
(建议: 0 1 2 3 4 5 6)
- (94) 已知图的邻接矩阵同上题 8, 根据算法, 则从顶点 0 出发, 按广度优先遍历的结点序列是 ( )。  
A) 0 2 4 3 1 6 5                      B) 0 1 3 5 6 4 2                      C) 0 1 2 3 4 6 5                      D) 0 1 2 3 4 5 6
- (95) 已知图的邻接表如下所示, 根据算法, 则从顶点 0 出发按深度优先遍历的结点序列是 ( )。



- A) 1 3 2                      B) 0 2 3 1                      C) 0 3 2 1                      **D) 0 1 2 3**
- (96) 已知图的邻接表如下所示,根据算法,则从顶点 0 出发按广度优先遍历的结点序列是( )。



- A) 0 3 2 1**                      B) 0 1 2 3                      C) 0 1 3 2                      D) 0 3 1 2
- (97) 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
- (98) 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**
- (99) 任何一个无向连通图的最小生成树( )。
- A) 只有一棵**                      B) 一棵或多棵                      C) 一定有多棵                      D) 可能不存在
- (注, 生成树不唯一, 但最小生成树唯一, 即边权之和或树权最小的情况唯一)
- (100) 在分析折半查找的性能时常常加入失败节点, 即外节点, 从而形成扩充的二叉树。若设失败节点  $i$  所在层次为  $L_i$ , 那么查找失败到达失败点时所做的数据比较次数是( )。
- A)  $L_i+1$                       B)  $L_i+2$                       C)  $L_i-1$                       **D)  $L_i$**
- (101) 向一个有 127 个元素原顺序表中插入一个新元素并保存原来顺序不变, 平均要移动( )个元素。
- A) 8                      **B) 63.5**                      C) 63                      D) 7
- (102) 由同一组关键字集合构造的各棵二叉排序树( )。
- A) 其形态不一定相同, 但平均查找长度相同
- B) 其形态不一定相同, 平均查找长度也不一定相同**
- C) 其形态均相同, 但平均查找长度不一定相同
- D) 其形态均相同, 平均查找长度也都相同
- (103) 衡量查找算法效率的主要标准是( )。
- A) 元素的个数                      B) 所需的存储量                      **C) 平均查找长度**                      D) 算法难易程度
- (104) 适合对动态查找表进行高效率查找的组织结构是( )。

- A) 有序表                      B) 分块有序表                      **C) 二叉排序树**                      D) 快速排序
- (3) 能进行二分查找的线性表,必须以 (      )。
- A) 顺序方式存储, 且元素按关键字有序**  
 B) 链式方式存储, 且元素按关键字有序  
 C) 顺序方式存储, 且元素按关键字分块有序  
 D) 链式方式存储, 且元素按关键字分块有序
- 河 南大学考试墙 QQ: 2139034270
- (105) 为使平均查找长度达到最小,当由关键字集合{05,11,21,25,37,40,41,62,84}构建二叉排序树时,第一个插入的关键字应为 (      )
- A) 5                                  B)37                                  C) 41                                  D) 62
- (106) 对关键字序列(56,23,78,92,88,67,19,34)进行增量为3的一趟希尔排序的结果为 (      )。
- A) (19,23,56,34,78,67,88,92)                                  B) 23,56,78,66,88,92,19,34)  
 C) (19,23,34,56,67,78,88,92)                                  **D) (19,23,67,56,34,78,92,88)**
- (107) 用某种排序方法对关键字序列{35,84,21,47,15,27,68,25,20}进行排序时, 序列的变化情况如下:
- 20,15,21,25,47,27,68,35,84  
 15,20,21,25,35,27,47,68,84  
 15,20,21,25,27,35,47,68,84
- 则采用的方法是 (      )。
- A) 直接选择排序                  B) 希尔排序                      C) 堆排序                      **D) 快速排序**
- (108) 一组记录的排序码为(46,79,56,38,40,84), 则利用快速排序的方法, 以第一个记录为基准得到的第一次划分结果为 (      )。
- A) 38,40,46,56,79,84                  B) 40,38,46,79,56,84                  **C) 40,38,46,56,79,84**                  D) 40,38,46,84,56,79
- (109) 快速排序在最坏情况下的时间复杂度是 (      )
- A)  $O(n^2 \log_2 n)$                       **B)  $O(n^2)$**                       C)  $O(n \log_2 n)$                       D)  $O(\log_2 n)$
- (110) 下列排序算法中不稳定的 (      )。
- A) 直接选择排序                  B) 折半插入排序                  C) 冒泡排序                      **D) 快速排序**
- (111) 对待排序的元素序列进行划分, 将其分为左、右两个子序列, 再对两个子序列进行同样的排序操作, 直到子序列为空或只剩下一个元素为止。这样的排序方法是 (      )。
- A) 直接选择排序                  B) 直接插入排序                  **C) 快速排序**                      D) 冒泡排序
- (112) 将5个不同的数据进行排序, 至多需要比较 (      ) 次。
- A) 8                                  B) 9                                  **C) 10**                                  D) 25
- (113) 排序算法中, 第一趟排序后, 任一元素都不能确定其最终位置的算法是 (      )。
- A)选择排序                      B)快速排序                      C)冒泡排序                      **D)插入排序**
- (114) 排序算法中, 不稳定的排序是 (      )。

- A)直接插入排序      B)冒泡排序      C)堆排序      D)选择排序
- (115) 排序方法中,从未排序序列中依次取出元素与已排序序列(初始时空)中的元素进行比较,将其放入已排序序列的正确位置上的方法,称为( )。
- A) 希尔排序      B) 冒泡排序      C) 插入排序      D) 选择排序
- (116) 从未排序序列中挑选元素,并将其依次插入已排序序列(初始时空)的一端的方法,称为( )。
- A) 希尔排序      B) 归并排序      C) 插入排序      D) 选择排序
- (117) 对  $n$  个不同的排序码进行冒泡排序,在下列哪种情况下比较的次数最多。( )
- A) 从小到大排列好的      B) 从大到小排列好的
- C) 元素无序      D) 元素基本有序
- (118) 对  $n$  个不同的排序码进行冒泡排序,在元素无序的情况下比较的次数为( )。
- A)  $n+1$       B)  $n$       C)  $n-1$       D)  $n(n-1)/2$
- (119) 快速排序在下列哪种情况下最易发挥其长处。( )
- A) 被排序的数据中含有多个相同排序码
- B) 被排序的数据已基本有序
- C) 被排序的数据完全无序
- D) 被排序的数据中的最大值和最小值相差悬殊
- (120) 对有  $n$  个记录的表作快速排序,在最坏情况下,算法的时间复杂度是( )。
- A)  $O(n)$       B)  $O(n^2)$       C)  $O(n\log_2 n)$       D)  $O(n^3)$
- (121) 若一组记录的排序码为(46, 79, 56, 38, 40, 84),则利用快速排序的方法,以第一个记录为基准得到的一次划分结果为( )。
- A) 38, 40, 46, 56, 79, 84      B) 40, 38, 46, 79, 56, 84
- C) 40, 38, 46, 56, 79, 84      D) 40, 38, 46, 84, 56, 79
- (122) 下列关键字序列中,( )是堆。
- A) 16, 72, 31, 23, 94, 53      B) 94, 23, 31, 72, 16, 53
- C) 16, 53, 23, 94, 31, 72      D) 16, 23, 53, 31, 94, 72
- (123) 堆是一种( )排序。
- A) 插入      B) 选择      C) 交换      D) 归并
- (124) 堆的形状是一棵( )。
- A) 二叉排序树      B) 满二叉树      C) 完全二叉树      D) 平衡二叉树
- (125) 若一组记录的排序码为(46, 79, 56, 38, 40, 84),则利用堆排序的方法建立的初始堆为( )。
- A) 79, 46, 56, 38, 40, 84      B) 84, 79, 56, 38, 40, 46
- C) 84, 79, 56, 46, 40, 38      D) 84, 56, 79, 40, 46, 38
- (126) 下述几种排序方法中,要求内存最大的是( )。

- A) 插入排序      B) 快速排序      C) 归并排序      D) 选择排序
- (127) 有一组数据 (15, 9, 7, 8, 20, -1, 7, 4), 用堆排序的筛选方法建立的初始堆为 ( )。
- A) -1, 4, 8, 9, 20, 7, 15, 7      B) -1, 7, 15, 7, 4, 8, 20, 9
- C) -1, 4, 7, 8, 20, 15, 7, 9      D) A, B, C 均不对。
- (128) 51. 下列四个序列中, 哪一个堆 ( )。
- A) 75,65,30,15,25,45,20,10      B) 75,65,45,10,30,25,20,15
- C) 75,45,65,30,15,25,20,10      D) 75,45,65,10,25,30,20,15
- (129) 以下序列不是堆的是 ( )。
- A) (100,85,98,77,80,60,82,40,20,10,66)      B) (100,98,85,82,80,77,66,60,40,20,10)
- C) (10,20,40,60,66,77,80,82,85,98,100)      D) (100,85,40,77,80,60,66,98,82,10,20)
- (130) 快速排序方法在 ( ) 情况下最不利于发挥其长处。
- A) 要排序的数据量太大      B) 要排序的数据中含有多个相同值
- C) 要排序的数据个数为奇数      D) 要排序的数据已基本有序
- (131) 对关键码序列 28, 16, 32, 12, 60, 2, 5, 72 快速排序, 从小到大一次划分结果为 ( )。
- A) (2,5,12,16)26(60,32,72)      B) (5,16,32)28(60,32,72)
- C) (2,16,12,5)28(60,32,72)      D) (5,16,2,12)28(32,60,72)
- (132) 对下列关键字序列用快速排序法进行排序时, 速度最快的情形是 ( )。
- A) {21,25,5,17,9,23,30}      B) {25,23,30,17,21,5,9}
- C) {21,9,17,30,25,23,5}      D) {5,9,17,21,23,25,30}

## 二、填空题

(133) 数据结构是一门研究非数值计算的程序设计问题中计算机的 操作对象 以及它们之间的 关系 和运算等的学科。

(134) 数据结构被形式地定义为  $(D, R)$ , 其中  $D$  是 数据元素 的有限集合,  $R$  是  $D$  上的 关系 有限集合。

(135) 数据结构包括数据的 逻辑结构、数据的 存储结构 和数据的 运算 这三个方面的内容。

(136) 数据结构按逻辑结构可分为两大类, 它们分别是 线性结构 和 非线性结构。

(137) 线性结构中元素之间存在一对一关系, 树形结构中元素之间存在一对多关系, 图形结构中元素之间存在多对多关系。

(138) 在线性结构中, 第一个结点 没有 前驱结点, 其余每个结点有且只有 1 个前驱结点; 最后一个结点 没有 后续结点, 其余每个结点有且只有 1 个后续结点。

(139) 在树形结构中, 树根结点没有 前驱 结点, 其余每个结点有且只有 1 个前驱结点; 叶子结点没有 后续 结点, 其余每个结点的后续结点数可以任意多个。

(140) 在图形结构中, 每个结点的前驱结点数和后续结点数可以 任意多个。

(141)数据的存储结构可用四种基本的存储方法表示,它们分别是顺序 、 链式 、 索引 和 散列 。

(142)数据的运算最常用的有 5 种,它们分别是插入 、 删除、修改、 查找 、 排序。

(143)一个算法的效率可分为 时间 效率和 空间 效率。

(144)对于给定的  $n$  个元素,可以构造出的逻辑结构有\_\_\_\_\_ 集合,线性表,树,图四种。

(145)顺序映象的特点是借助元素在存储器中的\_\_\_\_\_ **相对位置**来表示数据元素之间的逻辑关系。非顺序映象的特点是借助是指示元素存储地址的\_\_\_\_\_ **指针**表示数据元素之间的逻辑关系。任何一个算法的设计取决于选定\_\_\_\_\_ **逻辑结构**,而算法的实现依赖于采用的\_\_\_\_\_ **存储结构**。

(146)数据类型是一组\_\_\_\_\_ **性质相同**的值集合以及定义在这个值集合上的一组操作的总称。

(147)数据对象是\_\_\_\_\_ **性质相同**的数据元素的集合,是数据的一个子集。

(148)如果操作不改变原逻辑结构的“值”,而只是从中提取某些信息作为运算结果,则称该类运算为\_\_\_\_\_型运算。 **引用**

(149)算法的\_\_\_\_\_ **健壮**特性是指做为一个好的算法,当输入的数据非法时,也能适当地做出正确反应或进行相应的处理,而不会产生一些莫名其妙的输出结果。

(150)算法分析不是针对实际执行时间的精确的算法执行具体时间的分析,而是针对算法中语句的\_\_\_\_\_ **执行次数**做出估计,从中得到算法执行时间的信息。

(151) $T(n)=O(f(n))$ ,它表示随问题规模  $n$  的增大算法的执行时间的增长率和  $f(n)$  的增长率\_\_\_\_\_ **相同**,称作算法的渐进时间复杂度,简称时间复杂度。

(152)若算法执行时所需要的辅助空间相对于输入数据量而言是个常数,则称这个算法为\_\_\_\_\_ **原地工作**,辅助空间为  $O(1)$ 。

(153)在带有头结点的单链表中  $L$  中,第一个元素结点的指针是\_\_\_\_\_。  **$L \rightarrow next$**

(154)在一个带头节点的单循环链表中,  $p$  指向尾结点的直接前驱,则指向头结点的指针  $head$  可用  $p$  表示为  $head =$ \_\_\_\_\_。  **$p \rightarrow next \rightarrow next$**

(155)设单链表的结点结构为  $(data, next)$ ,  $next$  为指针域,已知指针  $px$  指向单链表中  $data$  为  $x$  的结点,指针  $py$  指向  $data$  为  $y$  的新结点,若将结点  $y$  插入结点  $x$  之后,则需要执行以下语句:\_\_\_\_\_  **$py \rightarrow next = px \rightarrow next; px \rightarrow next = py$** 。

(156) 对于栈操作数据的原则是\_\_\_\_\_。 **后进先出**

(157) 设以数组  $A[m]$  存放循环队列的元素,其头尾指针分别为  $front$  和  $rear$ ,则当前队列中的元素个数为\_\_\_\_\_。  **$(rear - front + m) \% m$**

(158)若已知一个栈的入栈序列是  $1, 2, 3, 4, \dots, n$ , 其输出序列为  $p_1, p_2, p_3, \dots, p_n$ , 若  $p_1 = n$ , 则  $p_i$  为\_\_\_\_\_。  **$n - i + 1$**

(159) \_\_\_\_\_ **队列** 是被限定为只能在表的一端进行插入运算, 在表的另一端进行删除运算的线性表。

(160) 通常程序在调用另一个程序时, 都需要使用一个 \_\_\_\_\_ **栈** 来保存被调用程序内分配的局部变量。形式参数的存储空间以及返回地址。

(161) 栈下溢是指在 \_\_\_\_\_ **栈空** \_\_\_\_\_ 时进行出栈操作。

(162) 用 P 表示入栈操作, D 表示出栈操作, 若元素入栈的顺序为 1234, 为了得到 1342 出栈顺序, 相应的 P 和 D 的操作串为 \_\_\_\_\_ 。 **PDPPDPDD**

(163) 在具有 n 个单元的循环队列中, 队满共有 \_\_\_\_\_ **n-1** 个元素。

(164) \_\_\_\_\_ **队列** 是被限定为只能在表的一端进行插入运算, 在表的另一端进行删除运算的线性表。

(165) 循环队列的引入, 目的是为了克服 \_\_\_\_\_ **假溢出** 。

(166) 所谓稀疏矩阵指的是 \_\_\_\_\_ **非零元** 很少 ( $t \ll m \times n$ ) 且分布没有规律 。

(167) 在稀疏矩阵表示所对应的三元组线性表中, 每个三元组元素按 \_\_\_\_\_ **行** 为主序, \_\_\_\_\_ **列** 号为辅序的次序排列。

(168) 二维数组  $A_{m \times n}$  按行优先顺序存储在内存中, 元素  $a_{00}$  地址为  $\text{loc}(a_{00})$ , 每个元素在内存中占 d 个字节, 元素  $a_{ij}$  的地址计算公式为  $\text{loc}(a_{ij}) = \text{loc}(a_{00}) + (i \times n + j) \times d$  \_\_\_\_\_ 。

(169) 去除广义表  $LS = (a_1, a_2, a_3, \dots, a_n)$  中第 i 个元素, 由其余元素构成的广义表称为 LS 的 \_\_\_\_\_ **表尾** \_\_\_\_\_ 。

(170) 树内个结点的度 \_\_\_\_\_ **最大值** 称为树的度。

(171) 一个二叉树第 5 层节点最多有 \_\_\_\_\_ **16** 个。

(172) 已知完全二叉树 T 的第 5 层只有 7 个结点, 则该树共有 \_\_\_\_\_ **11** \_\_\_\_\_ 个叶子结点。

(173) 在一棵二叉树中, 度为零的结点的个数为  $N_0$ , 度为 2 的结点的个数为  $N_2$ , 则有  $N_0 = \text{_____} \text{ } **N_2 + 1** \text{_____}$  。

(174) 假设用于通信的电文由 8 个字母组成, 其频率分别为 7, 19, 2, 6, 32, 3, 27, 10。设计哈夫曼编码, 其中字母的编码长度最大是 \_\_\_\_\_ **5** 位。

(175) 一棵具有 257 个结点的完全二叉树, 它的深度为 \_\_\_\_\_ **9** 。

(176) 图的深度优先遍历序列 \_\_\_\_\_ **不是** 惟一的。

(177) 在图中, 任何两个结点之间都可能存在关系, 因此图的数据元素之间时一种 \_\_\_\_\_ **多对多** 的关系。

(178) 在有向图中, 以顶点 v 为终点的边的数目称为 v 的 \_\_\_\_\_ **入度** \_\_\_\_\_ 。

(179) 一个无向图有 n 个顶点, e 条边, 则所有顶点的度数之和为 \_\_\_\_\_ **2e** 。

(180) 图有 \_\_\_\_\_ **邻接矩阵** \_\_\_\_\_ 、 \_\_\_\_\_ **邻接表** \_\_\_\_\_ 等存储结构, 遍历图有 \_\_\_\_\_ **深度优先遍历** \_\_\_\_\_ 、 \_\_\_\_\_ **广度优先遍历** \_\_\_\_\_ 等方法。

(181)

(182) 有向图  $G$  用邻接表矩阵存储, 其第  $i$  行的所有非零元素之和等于顶点  $i$  的\_\_\_\_\_。

出度

(183) 如果  $n$  个顶点的图是一个环, 则它有\_\_\_\_\_ 棵生成树。  $n$  (以任意一顶点为起点, 得到  $n-1$  条边)

(184)  $n$  个顶点  $e$  条边的图, 若采用邻接矩阵存储, 则空间复杂度为\_\_\_\_\_。  $O(n^2)$

(185)  $n$  个顶点  $e$  条边的图, 若采用邻接表存储, 则空间复杂度为\_\_\_\_\_。  $O(n+e)$

(186) 设有一稀疏图  $G$ , 则  $G$  采用 \_\_\_\_\_ 邻接表 存储较省空间。

(187) 设有一稠密图  $G$ , 则  $G$  采用 \_\_\_\_\_ 邻接矩阵 存储较省空间。

(188) 图的逆邻接表存储结构只适用于 \_\_\_\_\_ 有向 图。

(189) 已知一个图的邻接矩阵表示, 删除所有从第  $i$  个顶点出发的方法是 \_\_\_\_\_ 将邻接矩阵的第  $i$  行全部置 0 。

(190) 图的深度优先遍历序列 \_\_\_\_\_ 不是 惟一的。

(191)  $n$  个顶点  $e$  条边的图采用邻接矩阵存储, 深度优先遍历算法的时间复杂度为 \_\_\_\_\_  $O(n^2)$ ; 若采用邻接表存储时, 该算法的时间复杂度为 \_\_\_\_\_  $O(n+e)$  。

(192)  $n$  个顶点  $e$  条边的图采用邻接矩阵存储, 广度优先遍历算法的时间复杂度为  $O(n^2)$ ; 若采用邻接表存储, 该算法的时间复杂度为 \_\_\_\_\_  $O(n+e)$  。

(193) 图的 BFS 生成树的树高比 DFS 生成树的树高 \_\_\_\_\_ 小或相等 。

(194) 用普里姆(Prim)算法求具有  $n$  个顶点  $e$  条边的图的最小生成树的时间复杂度为  $O(n^2)$ ; 用克鲁斯卡尔(Kruskal)算法的时间复杂度为 \_\_\_\_\_  $O(e \log 2e)$  。

(195) 若要求一个稀疏图  $G$  的最小生成树, 最好用 \_\_\_\_\_ 克鲁斯卡尔(Kruskal) 算法来求解。

(196) 若要求一个稠密图  $G$  的最小生成树, 最好用 \_\_\_\_\_ 普里姆(Prim) 算法来求解。

(197) 用 Dijkstra 算法求某一顶点到其余各顶点间的最短路径是按路径长度 \_\_\_\_\_ 递增 的次序来得到最短路径的。

(198) 拓扑排序算法是通过重复选择具有 \_\_\_\_\_ 0 个前驱顶点的过程来完成的。

(199) 在各种查找方法中, 平均查找长度与结点个数  $n$  无关的查找方法是 \_\_\_\_\_ 散列查找 。

(200) 散列法存储的基本思想是由 \_\_\_\_\_ 关键字的值 决定数据的存储地址。

(201) 大多数排序算法都有两个基本的操作: \_\_\_\_\_ 比较和移动 。

(202) 由于查找算法的基本运算是关键字之间的比较操作, 所以可用 \_\_\_\_\_ 平均查找长度 来衡量查找算法的性能。

(203) 查找有静态查找和动态查找, 当查找不成功时动态查找会 \_\_\_\_\_ 将查找关键字插入在表中。

(204) 顺序查找法中设置监视哨, 可以起到 \_\_\_\_\_ 防止越界 的作用。

- (205) 假设列表长度为  $n$ ，那么查找第  $i$  个数据元素时需进行\_\_\_\_\_  $n-i+1$  次比较。
- (206) 假设查找每个数据元素的概率相等，即  $P_i=1/n$ ，则顺序查找算法的平均查找长度为：\_\_\_\_\_  
\_\_\_\_\_  $ASL=(n+1)/2$ 。
- (207) 折半查找法又称为二分法查找法，这种方法要求待查找的列表必须是\_\_\_\_\_  
按关键字大小有序排列的顺序表。
- (208) 假定将长度为  $n$  的表分成  $b$  块，且每块含  $s$  个元素，则  $b=n/s$ 。又假定表中每个元素的查找概率相等，
- (209) 在有序表(12,24,36,48,60,72,84)中二分查找关键字 72 时所需进行的关键字比较次数为\_\_\_\_  
2。
- (210) 折半查找有序表 (4,6,12,20,28,38,50,70,88,100)，若查找表中元素 20，它将依次与表中元素\_\_\_\_\_  
28, 6, 12, 20 比较大小。
- (211) 在各种查找方法中，平均查找长度与结点个数  $n$  无关的查找方法是\_\_\_\_\_散列查找。
- (212) 散列法存储的基本思想是由\_\_\_\_\_关键字的值 决定数据的存储地址。
- (213) 当关键字集合很大时，关键字值不同的元素可能会映象到哈希表的同一地址上，即  $k_1 \neq k_2$ ，但  $H(k_1)=H(k_2)$ ，这种现象称为\_\_\_\_\_冲突。
- (214) 产生冲突现象的两个关键字称为该散列函数的\_\_\_\_\_同义词\_\_\_\_\_。
- (215) 在散列函数  $H(key)=key \text{ MOD } p$  中， $p$  应取\_\_\_\_\_素数。
- (216) 设哈希表长  $m=14$ ，哈希函数  $H(key)=key \text{ MOD } 11$ 。表中已有 4 个结点： $addr(15)=4$ ， $addr(38)=5$ ， $addr(61)=6$ ， $addr(84)=7$ ，其余地址为空。如用二次探测再散列处理冲突，关键字为 49 的结点的地址是\_\_\_\_\_9\_\_\_\_\_。
- (217) 希尔排序是属于\_\_\_\_\_插入排序的改进方法。
- (218) 给出一组关键字  $T=(20,4,34,5,16,33,18,29,2,40,7)$ ，要求从下到大进行排序，试给出快速排序（选一个记录为枢纽）第一趟排序结果\_\_\_\_\_。  
7,4,2,85,16,18,20,,29,33,40,34
- (219) 大多数排序算法都有两个基本的操作：\_\_\_\_\_比较和移动\_\_\_\_\_。
- (220) 在对一组记录 (54,38,96,23,15,72,60,45,83) 进行直接插入排序时，当把第 7 个记录 60 插入到有序表时，为寻找插入位置至少需比较\_\_\_\_\_次。  
6。
- (221) 在插入和选择排序中，若初始数据基本正序，则选用\_\_\_\_\_插入\_\_\_\_\_；若初始数据基本反序，则选用\_\_\_\_\_选择\_\_\_\_\_。
- (222) 在堆排序和快速排序中，若初始记录接近正序或反序，则选用\_\_\_\_\_堆排序\_\_\_\_\_；若初始记录基本无序，则最好选用\_\_\_\_\_快速排序\_\_\_\_\_。
- (223) 对于  $n$  个记录的集合进行冒泡排序，在最坏的情况下所需要的时间是\_\_\_\_\_  $O(n^2)$  \_\_\_\_\_。若对其进行快速排序，在最坏的情况下所需要的时间是\_\_\_\_\_  $O(n^2)$  \_\_\_\_\_。

(234) 对于  $n$  个记录的集合进行归并排序, 所需要的平均时间是  $O(n\log_2 n)$ , 所需要的附加空间是  $O(n)$ 。

7. 对于  $n$  个记录的表进行 2 路归并排序, 整个归并排序需进行  $\lceil \log_2 n \rceil$  趟 (遍)。

8. 设要将序列 (Q, H, C, Y, P, A, M, S, R, D, F, X) 中的关键码按字母序的升序重新排列, 则:

冒泡排序一趟扫描的结果是 H C Q P A M S R D F X Y;

初始步长为 4 的希尔 (shell) 排序一趟的结果是 P A C S Q H F X R D M Y;

二路归并排序一趟扫描的结果是 H Q C Y A P M S D R F X;

快速排序一趟扫描的结果是 F H C D P A M Q R S Y X;

堆排序初始建堆的结果是 A D C R F Q M S Y P H X。

9. 在堆排序、快速排序和归并排序中,

若只从存储空间考虑, 则应首先选取\_\_方法, 其次选取\_快速排序\_方法, 最后选取\_归并排序\_方法;

若只从排序结果的稳定性考虑, 则应 选取\_\_归并排序\_\_方法;

若只从平均情况下最快考虑, 则应选取\_\_堆排序、快速排序和归并排序\_\_方法;

若只从最坏情况下最快并且要节省内存考虑, 则应选取\_\_堆排序\_\_方法。

### 三、程序填空题

(235) 以下程序的功能是实现带附加头结点的单链表的头结点逆序连接, 请填空完善之。

```
void reverse(pointer h)
/* h 为附加头结点指针; */
{ pointer p,q;
 p=h->next; h->next=NULL;
 while((1)_____)
 { q=p; p=p->next; q->next=h->next; h->next=(2)_____; }
}
```

(1)p!=null // 链表未到尾就一直作

(2)q // 将当前结点作为头结点后的第一元素结点插入

(236) 下面是用 c 语言编写的对不带头结点的单链表进行就地逆置的算法, 该算法用 L 返回逆置后的链表的头指针, 试在空缺处填入适当的语句。

```
void reverse (linklist &L) {
 p=null; q=L;
 while (q!=null)
 { (1)_____; q->next=p; p=q; (2)_____; }
 (3)_____;
}
```

(1) L=L->next; // 暂存后继

(2)q=L;           //待逆置结点  
(3)L=p;           //头指针仍为L

(237) 以下算法的功能是用头插法建立单链表的算法，请填空完善之。

```
Linklist CreateFromHead()
{
 LinkList L;
 Node *s;
 char c;
 L=(Linklist)malloc(sizeof(Node)); /*为头结点分配存储空间*/
 _____ L->next=NULL ;
 While((c=getchar()) !='*')
 {
 s=(Node*)malloc(sizeof(Node)); /*为读入的字符分配存储空间*/
 s->data=c;
 _____ s->next=L->next
 _____ L->next=s
 }
 return L;
}
```

(238) 以下算法的功能是尾插法建链表，请填空完善之。

```
typedef struct Node /*结点类型定义*/
{
 char data;
 struct Node * next;
} Node, *LinkList; /* LinkList 为结构指针类型*/

Linklist CreateFromTail() /*将新增的字符追加到链表的末尾*/
{
 LinkList L;
 Node *r, *s;
 char c;
 L=(Node *)malloc(sizeof(Node)); /*为头结点分配存储空间*/
 L->next=NULL;
 r=L; /*r 指针指向链表的当前表尾，以便于做尾插入，其初值指向头结点*/
 while((_____ c=getchar()) !='$') /*
 当输入'$'时，建表结束*/
```

```

{ s=(Node*)malloc(sizeof(Node)); s->data=c;
 _____; r->next=s; r=s;
}

_____; r->next=NULL /*将最后一个结点的 next 链域置为空，表示
链表的结束*/

return L;

} /*CreateFromTail*/

```

(239) 下列算法在顺序表 L 中依次存放着线性表中的元素，在表中查找与 e 相等的元素，若 L.elem[i]=e, 则找到该元素，并返回 i+1，若找不到，则返回“-1”，请填空完善之。

```

int Locate(SeqList L, int e)
{ i=0; /*i 为扫描计数器，初值为 0，即从第一个元素开始比较*/
 while ((i<=L.last)&&(L.elem[i]!=e)) _____ i++;
 /*顺序扫描表，直到找到值为 key 的元素,或扫描到表尾而没找到*/
 if (_____ i<=L.last) return(i+1); /*若找到值为 e 的元素，
则返回其序号*/
 else return(-1); /*若没找到，则返回空序号*/
}

```

(240) 下列算法在顺序表 L 中第 i 个数据元素之前插入一个元素 e。插入前表长 n=L->last+1，i 的合法取值范围是  $1 \leq i \leq L->last+2$ ，请填空完善之。

```

void InsList(SeqList *L, int i, int e)
{ int k;
 if((i<1) || (i>L->last+2)) printf("插入位置 i 值不合法");
 if(L->last>=maxsize-1) printf("表已满无法插入");
 for(k=L->last; k>=i-1; k--) /*为插入元素而移动位置*/
 _____ L->elem[k+1]=L->elem[k];
 _____ L->elem[i-1]=e; /*在 C 语言数组中，第 i 个元素的下标为 i-1*/
 _____ L->last++;
}

```

(241) 下列算法是在顺序表 L 中删除第 i 个数据元素，并用指针参数 e 返回其值。i 的合法取值范围为  $1 \leq i \leq L->last+1$ ，请填空完善之。

```

int DelList(SeqList *L, int i, int *e)
{ int k;
 if((i<1)|| (i>_____ L->last+1)) printf("删除位置不合法！");
}

```

```

 _____ *e= L->elem[i-1]; /* 将删除的元素存放到 e
 所指向的变量中*/
 for(k=i;i<=L->last;k++)
 _____ L->elem[k-1]= L->elem[k] ; /*将
 后面的元素依次前移*/
 _____ L->last-- ;
}

```

#### 四、解答题

(242) 假设以数组 `seqn [m]` 存放循环队列的元素，设变量 `rear` 和 `quelen` 分别指示循环队列中队尾元素的位置和元素的个数。

- (1) 写出队满的条件表达式；
- (2) 写出队空的条件表达式；
- (3) 设 `m=40, rear=13, quelen=19`, 求队头元素的位置；
- (4) 写出一般情况下队头元素位置的表达式。

(1) `quelen == m`

(2) `quelen == 0`

(3) `( 13 - 19 + 40 ) % 40 = 34`

(4) `( rear - quelen + m ) % m`

(243) 已知一棵二叉树的中序序列为 `ABCDEFGF`，层序序列为 `BAFEGCD`，请画出该二叉树。

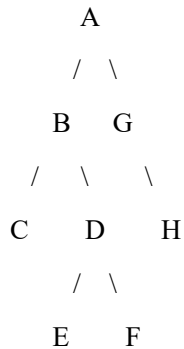
```

 B
 / \
 A F
 / \
 E G
 /
 C
 \

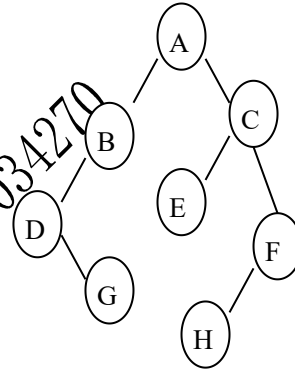
```

**D**

(244) 已知一棵二叉树的前序序列为 ABCDEFGH, 中序序列为 CBEDFAGH, 请画出该二叉树。



(245) 已知一棵二叉树如图所示。请分别写出按前序、中序、后序和层次遍历是得到的顶点序列。



前序: A,B,D,G,C,E,F,H

中序: D,G,B,A,E,C,H,F

后序: G,D,B,E,H,F,C,A

层次: A,B,C,D,E,F,G,H

(246) 已知一棵二叉树的前序序列为:

A,B,D,G,J,E,H,C,F,I,K,L 中序序列: D,J,G,B,E,H, A,C,K,I,L,F。

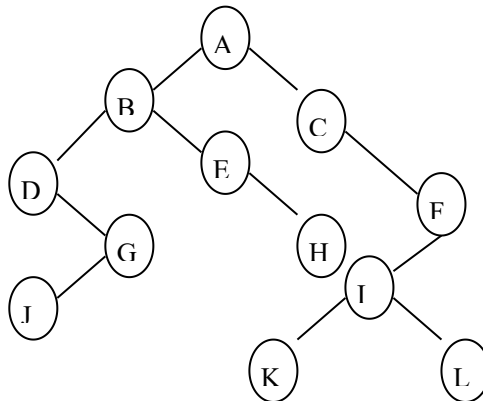
(1) 写出该二叉树的后序序列。

(2) 画出该二叉树;

(3) 求该二叉树的高度(假定空树的高度为-1)和度为 2、度为 1、及度为 0 的结点个数。

该二叉树的后序序列为: J,G,D,H,E,B,K,L,I,F,C,A。

该二叉树的形式如图所示:



该二叉树高度为：5。

度为 2 的结点的个数为：3。

度为 1 的结点的个数为：5。

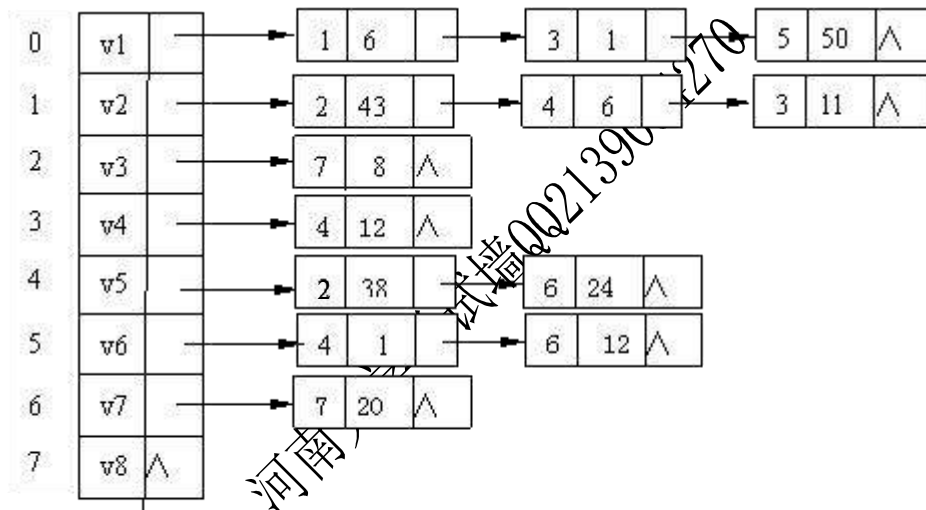
度为 0 的结点个数为：4。

(247)有一份电文中共使用 6 个字符:a,b,c,d,e,f,它们的出现频率依次为 2,3,4,7,8,9，试构造一棵哈夫曼树，并求其加权路径长度 WPL，字符 c 的编码。

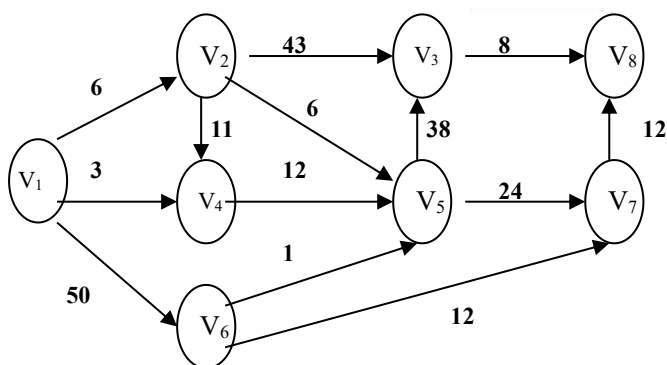
WPL=80 字符 c: 001 (不唯一)

(4) 下图是带权的有向图 G 的邻接表表示法。从结点 V1 出发，求出：

- 深度遍历图 G；
- G 的一个拓扑序列；
- 从结点 V1 到结点 V8 的最短路径。



题 29 图



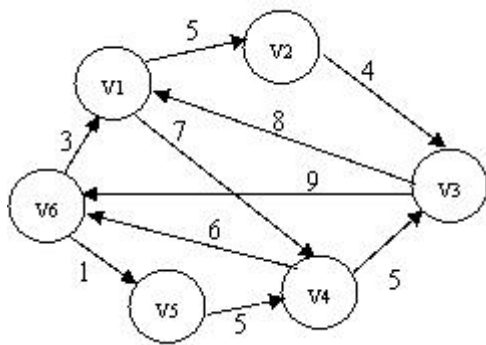
参考答案：v1,v2,v3,v8,v4,v5,v7,v6 (2 分)

v1,v2,v4,v6,v5,v3,v7,v8 (2 分)

V1 到结点 V8 的最短路径为：v1,v2,v3,v8 (2 分)

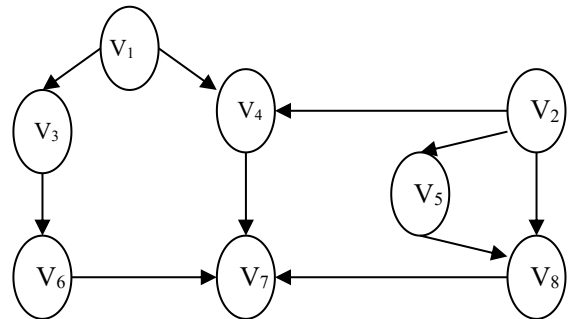
(248) 已知如图所示的有向图，请给出该图的：

- (1) 每个顶点的入度、出度；
- (2) 邻接矩阵；
- (3) 邻接表；



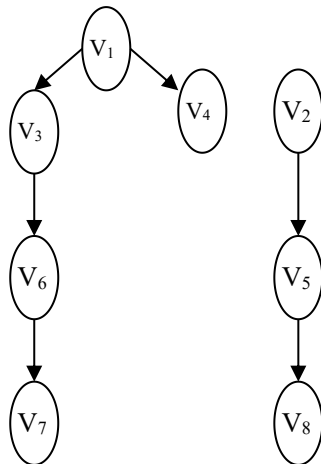
|    | v1       | v2       | v3       | v4       | v5       | v6       |
|----|----------|----------|----------|----------|----------|----------|
| v1 | $\infty$ | 5        | $\infty$ | 7        | $\infty$ | $\infty$ |
| v2 | $\infty$ | $\infty$ | 4        | $\infty$ | $\infty$ | $\infty$ |
| v3 | 8        | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 9        |
| v4 | $\infty$ | $\infty$ | 5        | $\infty$ | $\infty$ | 6        |
| v5 | $\infty$ | $\infty$ | $\infty$ | 5        | $\infty$ | $\infty$ |
| v6 | 3        | $\infty$ | $\infty$ | $\infty$ | 1        | $\infty$ |

(249) 对下面的有向图，从顶点  $V_1$  开始进行遍历，试画出遍历得到的 DFS 生成森林和 BFS 生成森林。

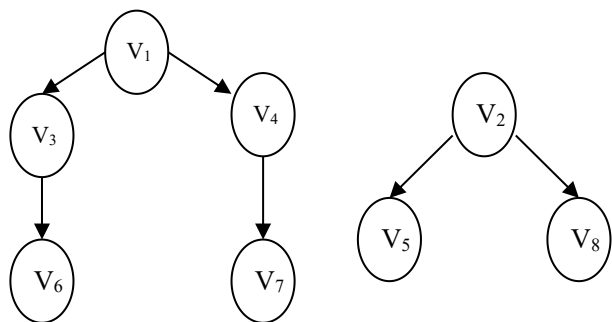


图全对给 4 分，错一个顶点扣 1 分，扣完为止。

遍历得到的 DFS 生成森林和 BFS 生成森林如下图：



DFS 生成森林



BFS 生成森林

(250)采用哈希函数  $H(k)=3*k \bmod 13$  并用线性探测开放地址法处理冲突，在数列地址空间  $[0..12]$  中对关键字序列 22,41,53,46,30,13,1,67,51

- (1) 构造哈希表（画示意图）；
- (2) 装填因子；等概率下
- (3) 成功的和 (4) 不成功的平均查找长度

1)

|      |    |    |   |    |   |   |    |    |    |   |    |    |    |
|------|----|----|---|----|---|---|----|----|----|---|----|----|----|
| 散列地址 | 0  | 1  | 2 | 3  | 4 | 5 | 6  | 7  | 8  | 9 | 10 | 11 | 12 |
| 关键字  | 13 | 22 |   | 53 | 1 |   | 41 | 67 | 46 |   | 51 |    | 30 |
| 比较次数 | 1  | 1  |   | 1  | 2 |   | 1  | 2  | 1  |   | 1  |    | 1  |

(2) 装填因子  $=9/13=0.7$

(3)  $ASL_{succ} = 11/9$

(4)  $ASL_{unsucc} = 29/13$

(251) 设有一组关键字 {9,01,23,14,55,20,84,27}，采用哈希函数： $H(key) = key \bmod 7$ ，表长为 10，用开放地址法的二次探测再散列方法  $H_i = (H(key) + d_i) \bmod 10 (d_i = 1, 2, 3, \dots)$  解决冲突。要求：对该关键字序列构造哈希表，并计算查找成功的平均查找长度。

|      |    |    |   |    |    |    |    |    |   |   |
|------|----|----|---|----|----|----|----|----|---|---|
| 散列地址 | 0  | 1  | 2 | 3  | 4  | 5  | 6  | 7  | 8 | 9 |
| 关键字  | 14 | 01 | 9 | 23 | 84 | 27 | 55 | 20 |   |   |
| 比较次数 | 1  | 1  | 1 | 2  | 3  | 4  | 1  | 2  |   |   |

平均查找长度： $ASL_{succ} = (1+1+1+2+3+4+1+2) / 8 = 15/8$

以关键字 27 为例： $H(27) = 27 \% 7 = 6$ （冲突） $H_1 = (6+1) \% 10 = 7$ （冲突）

$H_2 = (6+2^2) \% 10 = 0$ （冲突） $H_3 = (6+3^2) \% 10 = 5$  所以比较了 4 次。

(252) 对于给定的一组记录的关键字 { 23,13,17,21,30,60,58,28,30,90 }，试分别写出冒泡排序、快速排序、堆排序、归并排序第一趟排序后的结果。

冒泡排序 13,23,17,21,,28,30,60,58,30\*, 90

快速排序：(21,13,17,) 13,( 30,60,58,28,30\*,90 )

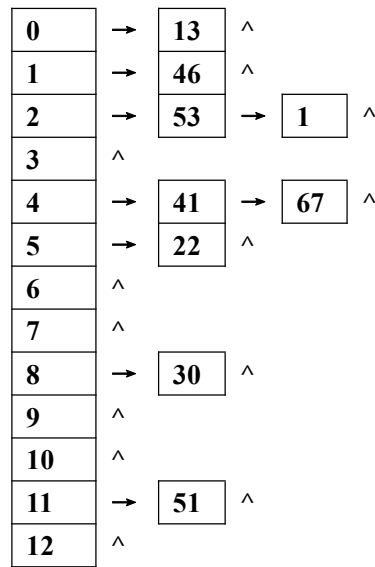
堆排序： 13,21,17,23,30,60,58,28,30\*,90,

归并排序按层遍历：(13 23) (17 21) (30 60) (28 58) (30\* 90)

(253)采用哈希函数  $H(k)=2*k \bmod 13$  并用链地址法处理冲突，在数列地址空间  $[0..12]$  中对关键字序列 22,41,53,46,30,13,1,67,51 进行下列工作：

- (a) 构造哈希表（画示意图）；
- (b) 等概率下成功的和不成功的平均查找长度。

参考答案：链地址表全对给 8 分。错一个结点扣 1 分，扣完为止。



ASLsucc=(7+4)/13=11/9 (1 分)

ASLunsucc=(5+4)/13=9/13 (1 分)

#### 四、算法设计题(10 分)

(254) 阅读下列递归算法，写出非递归方法实现相同功能的 C 程序。

```
void test(int &sum)
{ int x;
 scanf(x);
 if(x=0) sum=0 else {test(sum); sum+=x;}
 printf(sum);
}
```

#include<stdio.h> (1 分)

void main() (1 分)

{ int x,sum=0,top=0,s[]; (1 分)

scanf("%d",&x)

while (x<>0)

{ s[++top]:=a; scanf("%d",&x); }(3 分)

while (top)

sum+=s[top--]; (3 分)

printf("%d",sum); (1 分)

}

(255) 试写出把图的邻接矩阵表示转换为邻接表表示的算法。

设图的邻接矩阵为  $g[n][n]$  (针对无向图), 定义邻接表节点的类型为

```
struct edgenode
```

```
{ int adjvex;
```

```
 edgenode next;
```

```
}
```

```
typedef edgenode *adjlist[n];
```

```
void matritolist (int g[][], adjlist gl, int n)
```

```
{ edgenode *p, *q;
```

```
 for (int i=0; i<n; i++) gl[i]=null;
```

```
 for (int i=0; i<n; i++)
```

```
 for (int j=0; j<n; j++)
```

```
 { if (g[i][j]!=0)
```

```
 p = (edgenode *) malloc(sizeof(edgenode));
```

```
 p->adjvex=j;
```

```
 p->next=null;
```

```
 if (gl[i]==null) { gl[i]=p; q=p; }
```

```
 else (q->next=p; q=p; }
```

```
 }
```

```
}
```

(256) 阅读算法并根据输入数据画出链表。

```
linklist createlistr1()
```

```
{ char ch;
```

```
 linklist head=(listnode*)malloc(sizeof(listnode));
```

```
 listnode *p, *r;
```

```
 r=head;
```

```
 while((ch=getchar())!= '\n')
```

```
 { p=(listnode*)malloc(sizeof(listnode));
```

```
 while (p) p=(listnode*)malloc(sizeof(listnode));
```

```
 p->data=ch; r->next=p; r=p;
```

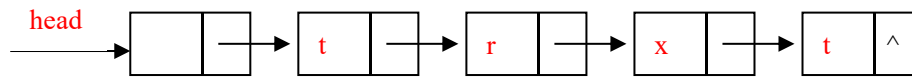
```
 }
```

```
 r->next=NULL;
```

```
 return(head);
```

}

输入数据为: text↵



(257) 阅读算法并指出下列各段程序完成的功能。

```

void add_poly(Lnode *pa, Lnode *pb)
{
 Lnode *p, *q, *u, *pre; int x;
 p=pa->next; q=pb->next; pre=pa;
 while((p!=NULL) && (q!=NULL))
 {
 if (p->exp < q->exp)
 {
 pre=p; p=p->next;
 }
 else if (p->exp == q->exp)
 {
 x=p->coef+q->coef;
 if (x!=0) { p->coef=x; pre=p; }
 else { pre->next=p->next; free(p); }
 p=pre->next; q=q->next;
 free(u);
 }
 else
 {
 u=q->next; q->next=p; pre->next=q;
 pre=q; q=u;
 }
 }
 if (q!=NULL) pre->next=q;
 free(pb);
}

```

两个多项式相加

(258) 阅读下面的程序，说明程序的具体功能。

```

typedef int elementype
typedef struct node
{
 elementype data;
 struct node *next;
}linklist;

```

```

void function(linklist *head, elemtype x)
{
 linklist *q, *p;
 q=head;
 p=q->next;
 while (p !=NULL) && (p->data != x)
 {
 q=p; p=p->next;
 }
 if (q==NULL) printf ("there is no this node ::\n");
 else { q->next =p->next ; free (p); }
}

```

该程序的功能是：在带头结点的单链表中，删除单链表中值为  $x$  的数据元素。

(259) 阅读下面的程序，说明程序的具体功能。

```

void function()
{
 initstack(s);
 scanf ("%d",&n);
 while(n)
 {
 push(s,n%8); n=n/8;
 }
 while(! Stackempty(s))
 {
 pop(s,e); printf("%d",e);
 }
}

```

该程序的功能是：10 进制数转换为 8 进制

(260) 阅读下面的程序，说明程序的具体功能。

```

void print(int w)
{
 int i;
 if (w!=0)
 {
 print(w-1);
 for(i=1;i<=w;++i)
 printf("%3d,",w);
 printf("\n");
 }
}

```

运行结果：

1,

2, 2,

3, 3, 3,

(261) 阅读下面的程序，分别说明程序中四个 for 循环和语句 ++cpot[col]; 的具体功能。

```

void FastTransposeSMatrix(Matrix M, Matrix &T)
{
 T.mu=M.nu; T.nu=M.mu; T.tu=M.tu;
 if (T.tu)
 {
 for (col=1;col<=M.nu;++col) num[col]=0;
 for (t=1;t<=M.tu;++t) ++num[M.item[t].j];
 cpot[1]=1;
 for (col=2;col<=M.nu;++col)
 cpot[col]=cpot[col-1]+num[col-1];
 for (p=1;p<=M.tu;++p)
 {
 col=M.item[p].j;
 q=cpot[col];
 T.item[q].i=M.data[p].i;
 T.item[q].j=M.item[p].j;
 T.item[q].e=M.data[p].e;
 ++cpot[col];
 }
 }
 return OK;
}

```

第一个 for 循环：初始化每一列中非零元素的个数为 0

第二个 for 循环，计算每一列中非零元素的个数；

第三个 for 循环，计算每一列的第一个元素的首地址；

第四个 for 循环，转置过程；

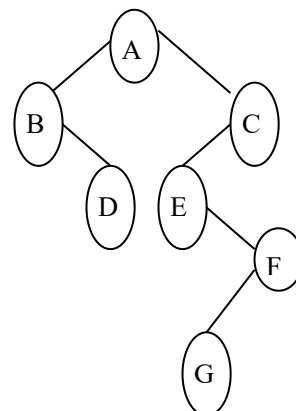
++cpot[col]：语句的功能是当每一列进行一次转置后，其位置向后加 1。

(262) 已知二叉树的二叉链表存储表示，指出建立如图所示树的结构时输入的字符序列。

```

typedef struct BiTNode
{
 char data;
 struct BiTNode *lchild,*rchild;
} BiTNode;
void CreateBiTree(BiTNode *T)

```



```

{ char ch;
 scanf(&ch);
 if(ch== ' ') T=NULL;
 else{ T=(BiTNode *)malloc(sizeof(BiTNode));
 while (T==NULL)
 T=(BiTNode *)malloc(sizeof(BiTNode));
 T->data=ch;
 CreateBiTree(T->lchild);
 CreateBiTree(T->rchild);
 }
}

```

AB\_D\_\_CE\_FG\_\_↵

(263) 已知二叉树的二叉链表存储表示，写出中序遍历的递归算法。

```

typedef struct BiTNode
{
 char data;
 struct BiTNode *lchild,*rchild;
} BiTNode,*BiTree;

```

```

void inorder(BiTNode *p)
{ if (p!=NULL)
 { inorder(p->lchild);
 printf("%c",p->data);
 inorder(p->rchild);
 }
}

```

(264) 阅读下面的程序，分别指出程序中三个 for 循环、整个程序以及语句 `scanf("%d,%d",&G.vexnum,&G.arcnum);` 的功能。

```

void funcgraph(MGraph &G)
{ int i,j,k,w; char v1,v2;
 printf("Input vexnum & arcnum:");
 scanf("%d,%d",&G.vexnum,&G.arcnum);
 printf("Input Vertices:");
 for (i=0;i<G.vexnum;i++)

```

```

scanf("%c",&G.vexs[i]);
for (i=0;i<G.vexnum;i++)
 for (j=0;j<G.vexnum;j++)
 G.arcs[i][j]=0;
for (k=0;k<G.arcnum;k++)
{
 printf("Input Arcs(v1,v2 & w):\n");
 scanf("%c%c,%d",&v1,&v2,&w);
 i=LocateVex(G,v1);
 j=LocateVex(G,v2);
 G.arcs[i][j]=w; G.arcs[j][i]=w;
}
}

```

第一个 for 循环：将图中的顶点输入到数组 G.vexs[i];

第二个 for 循环，初始化邻接矩阵；

第三个 for 循环，将图中边信息存入数组 G.vexs[i] 中；

本程序的功能是：创建图的邻接矩阵；

scanf("%d,%d",&G.vexnum,&G.arcnum); : 语句的功能输入定点数和图中的边数。

设哈希 (Hash) 表的地址范围为 0~17, 哈希函数为:  $H(K) = K \text{ MOD } 16$ 。

K 为关键字, 用线性探测法再散列法处理冲突, 输入关键字序列:

(10, 24, 32, 17, 31, 30, 46, 47, 40, 63, 49)

造出 Hash 表, 试回答下列问题:

- (1) 画出哈希表的示意图;
- (2) 若查找关键字 63, 需要依次与哪些关键字进行比较?
- (3) 若查找关键字 60, 需要依次与哪些关键字比较?
- (4) 假定每个关键字的查找概率相等, 求查找成功时的平均查找长度。

解: (1) 画表如下:

|    |    |    |    |   |   |   |   |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|---|---|---|---|----|----|----|----|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4 | 5 | 6 | 7 | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 32 | 17 | 63 | 49 |   |   |   |   | 24 | 40 | 10 |    |    |    | 30 | 31 | 46 | 47 |

(2) 查找 63, 首先要与  $H(63)=63\%16=15$  号单元内容比较, 即 63 vs 31, no;

然后顺移, 与 46, 47, 32, 17, 63 相比, 一共比较了 6 次!

(3) 查找 60, 首先要与  $H(60)=60\%16=12$  号单元内容比较, 但因为 12 号单元为空 (应当有空标记), 所以应当只比较这一次即可。

(4) 对于黑色数据元素, 各比较 1 次; 共 6 次;

对红色元素则各不相同, 要统计移位的位数。“63”需要 6 次, “49”需要 3 次, “40”需要 2 次, “46”需要 3 次, “47”需要 3 次,

所以  $ASL=1/11 (6+2+3 \times 3) = 17/11 = 1.5454545454 \approx 1.55$