

# 课程题库

1、数据结构通常是研究数据的（ ）及它们之间的相互联系。

A、 存储结构和逻辑结构

B、 存储和抽象

C、 联系和抽象

D、 联系与逻辑

答案： A

解析：数据结构通常是研究 3 方面的内容：数据的逻辑结构、数据的存储结构以及它们之间的相互联系。

2、在逻辑上可以把数据结构分成：（ ）。

A、 动态结构和静态结构

B、 紧凑结构和非紧凑结构

C、 线性结构和非线性结构

D、 内部结构和外部结构

答案： C

解析： 在逻辑上可以把数据结构分成线性结构和非线性结构。

3、 数据在计算机存储器内表示时，物理地址和逻辑地址相同并且是连续的，称之为（ ）。

A、 存储结构

B、 逻辑结构

C、 顺序存储结构

D、 链式存储结构

答案： C

解析： 数据在计算机存储器内表示时，物理地址和逻辑地址相同并且是连续的，称之为顺序存储结构。

4、 非线性结构中的每个结点（ ）。

- A、 无直接前趋结点
- B、 无直接后继结点
- C、 只有一个直接前趋结点和一个直接后继结点
- D、 可能有多个直接前趋结点和多个直接后继结点

答案： D

解析： 非线性结构中的每个结点是多对多的关系。

5、链式存储的存储结构所占存储空间（ ）。

- A、 分两部分，一部分存放结点的值，另一部分存放表示结点间关系的指针
- B、 只有一部分，存放结点的值
- C、  
只有一部分，存储表示结点间关系的指针
- D、 分两部分，一部分存放结点的值，另一部分存放结点所占单元素

答案： A

解析： 链式存储的存储结构每个结点所占存储空间为 data 和 next。

6、 以下任何两个结点之间都没有逻辑关系的是（ ）。

A、 图形结构

B、 线性结构

C、 树形结构

D、 集合

答案： D

解析： 集合任何两个结点之间都没有逻辑关系。

7、 下列四种基本逻辑结构中，数据元素之间关系最弱的是（ ）。

A、 集合

B、 线性结构

C、 树形结构

D、 图形结构

答案： A

解析： 集合任何两个结点之间都没有逻辑关系。

8、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、 逻辑结构

B、 存储结构

C、 逻辑实现

D、 存储实现

答案： A

解析： 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逻辑结构。

9、每一个存储结点只含有一个数据元素，存储结点存放在连续的存储空间，另外有一组指明结点存储位置的表，该存储方式是（ ）存储方式。

A、 顺序

B、 链式

C、 索引

D、 散列

答案： C

解析： 索引存储包含索引表和结点表。

10、 根据二元组关系， 下列属于( )数据结构。

**S= (D, R) , 其中:**

**D={a, b, c, d, e},**

**R={ }**

A、 集合

B、 线性结构

C、 树形结构

D、 图形结构

答案： A

解析： 集合

11、根据二元组关系，下列属于( )数据结构。

$S = (D, R)$ ，其中：

$D = \{50, 25, 64, 57, 82, 36, 75, 55\}$ ,

$R = \{ \langle 50, 25 \rangle, \langle 50, 64 \rangle, \langle 25, 36 \rangle, \langle 64, 57 \rangle, \langle 64, 82 \rangle, \langle 57, 55 \rangle, \langle 57, 75 \rangle \}$

A、 集合

B、 线性结构

C、 树形结构

D、 图形结构

答案： C

解析： 表示层次关系。

12、根据二元组关系，下列属于( )数据结构。

$S = (D, R)$ ，其中：

$D = \{1, 2, 3, 4, 5, 6\}$ ,

$R = \{ (1, 2), (2, 3), (2, 4), (3, 4), (3, 5), (3, 6), (4, 5), (4, 6) \}$

- A、 集合
- B、 线性结构
- C、 树形结构
- D、 图形结构

答案： D

解析： 一对多关系。

13、 算法分析的两个主要方面是（ ）。

- A、 空间复杂性和时间复杂性
- B、 正确性和简明性
- C、 可读性和文档性
- D、 数据复杂性和程序复杂性



答案： A

解析： 算法分析的两个主要方面是时间和空间效率。

14、 算法在发生非法操作时可以作出处理的特性称为算法的（ ）。

A、 正确性

B、 易读性

C、 健壮性

D、 高效性

答案： C

解析： 算法在发生非法操作时可以作出处理的特性称为算法的健壮性。

15、 下列时间复杂度中最坏的是（ ）。

A、  $O(1)$

B、  $O(n)$

C、  $O(\log_2 n)$

D、  $O(n^2)$

答案： D

解析：  $O(1) < O(\log_2 n) < O(n) < O(n^2)$ 。

16、 下列算法的时间复杂度是（ ）。

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

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

```
        c[i][j]=i+j;
```

A、  $O(1)$

B、  $O(n)$

C、  $O(\log_2 n)$

D、  $O(n^2)$

答案： D

解析：  $n*n$ 。

17、下列算法的时间复杂度是（ ）。

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

```
for (j=0;j<m;j++)
```

```
A[i][j];
```

A、  $O(1)$

B、  $O(n*m)$

C、  $O(\log_2 n)$

D、  $O(n^2)$

答案： B

解析：  $n*m$ 。

18、下列算法的时间复杂度是（ ）。

```
T=A; A=B; B=T;
```

A、  $O(1)$

B、  $O(n)$

C、  $O(\log_2 n)$

D、  $O(n^2)$

答案： A

解析： 常量时间。

19、 下列算法的时间复杂度是（ ）。

```
s1(int n)  
  
{ int p=1,s=0;  
  
for (i=1;i<=n;i++)  
  
    { p*=i;s+=p; }  
  
return(s);  
  
}
```

A、  $O(1)$

B、  $O(n)$

C、  $O(\log_2 n)$

D、  $O(n^2)$

答案： B

解析： 线性时间。

20、 下列算法的时间复杂度是（ ）。

```
s2(int n)
```

```
    x=0;
```

```
    y=0;
```

```
    for (k=1;k<=n;k++)
```

```
        x++;
```

```
    for (i=1;i<=n;i++)
```

```
        for (j=1;j<=n;j++)
```

```
            y++;
```

A、  $O(1)$

B、  $O(n)$

C、  $O(\log_2 n)$

D、  $O(n^2)$

答案： D

解析： 并列循环取大的。

21、数据的逻辑结构与数据元素本身的内容和形式无关。（）

答案： 正确

解析： 数据的逻辑结构与数据元素本身的内容和形式无关。

22、数据元素是数据的最小单位。（）

答案： 错误

解析： 数据项是数据的最小单位。

23、数据的逻辑结构和数据的存储结构一定是相同的。（）

答案： 错误

解析： 概念不同。

24、数据的存储结构是数据的逻辑结构的存储映像。（）

答案： 正确

解析：数据的存储结构是数据的逻辑结构的存储表示。

25、数据的逻辑结构是依赖于计算机的。（ ）

答案： 错误

解析：数据的存储结构是依赖于计算机的。

26、数据的物理结构是指数据在计算机内实际的存储形式。（ ）

答案： 正确

解析：数据的物理结构是指数据在计算机内实际的存储形式。

27、数据逻辑结构除了集合以外，还包括：线性结构、树形结构图形结构。（ ）

答案： 正确

解析：4类逻辑结构。

28、数据的存储结构形式包括：顺序存储、链式存储、索引存储和散列存储。（ ）

答案： 正确

解析：4类存储结构。

29、线性结构中的元素之间存在一对一的关系。（ ）

答案： 正确

解析：线性结构中的元素是一一对应的关系。

30、树形结构结构中的元素之间存在多对多的关系。（ ）

答案： 错误

解析：树形结构结构中的元素之间存在一对多的关系。

31、图形结构的元素之间存在一对多的关系。（ ）

答案： 错误

解析：图形结构的元素之间存在多对多的关系。

32、数据结构被定义为 $(D, R)$ ，其中  $D$  是数据的有限集合， $R$  是  $D$  上的关系的有限集合。（ ）

答案： 正确

解析：数据结构被定义为 $(D, R)$ ，其中  $D$  是数据的有限集合， $R$  是  $D$  上的关系的有限集合。



33、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的学科。( )

答案： 正确

解析：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的学科。

34、算法可以是一个无穷指令的集合。( )

答案： 错误

解析：算法是一个有穷指令的集合。

35、算法效率的度量可以分为事先估算法和事后统计法。( )

答案： 正确

解析：算法效率的度量可以分为事先估算法和事后统计法。

36、一个算法的时间复杂性是算法输入规模的函数。( )

答案： 正确

解析：一个算法的时间复杂性是算法输入规模的函数。

37、算法的空间复杂度是指该算法执行时所耗费的存储空间，它不是该算法求解问题规模  $n$  的函数。( )

答案： 错误

解析：算法的空间复杂度是指该算法执行时所耗费的存储空间，它是该算法求解问题规模  $n$  的函数。

38、若一个算法中的语句频度之和为  $T(n) = 8n + 5n\log_2 n$ ，则算法的时间复杂度为  $O(n\log_2 n)$ 。( )

答案： 正确

解析：  $O(n\log_2 n) > O(n)$ 。

39、若一个算法中的语句频度之和为  $T(n) = 9n + n\log_2 n + 2n^2$ ，则算法的时间复杂度为  $O(n^2)$ 。( )

答案： 正确

解析：  $O(n^2) > O(n\log_2 n) > O(n)$ 。

40、代码 `i=1; while(i<=n) i=i*3;`的时间复杂度为  $O(n^2)$ 。( )

答案： 错误

解析：为  $O(\log_3 n)$ 。

41、线性表是( ) 。

- A、 一个有限序列，可以为空；
- B、 一个有限序列，不能为空；
- C、 一个无限序列，可以为空；
- D、 一个无序序列，不能为空。

答案： A

解析：线性表的定义，一个线性表是  $n$  个具有相同特性的数据元素的有限序列，可以为空。

42、 顺序表的特点是( ) 。

- A、 表中元素的个数为表长
- B、 按顺序方式存储数据元素
- C、 逻辑结构中相邻的结点在存储结构中仍相邻

D、 按表中元素的次序存储

答案： B

解析： 顺序表的定义， 类比数组的理解， 其存储单元的地址都是连续的。

43、 对顺序存储的线性表， 设其长度为  $n$ ， 在任何位置上插入或删除操作都是等概率的。 插入一个元素时平均要移动表中的（ ） 个元素。

A、  $n/2$

B、  $(n+1)/2$

C、  $(n-1)/2$

D、  $n$

答案： A

解析： 当对  $n$  个元素进行插入操作时， 有  $n+1$  个位置可以进行插入， 如下所示。

. 1. 2. 3. 4. — — .n.

而在每个位置插入时需要移动的元素个数分别为  $n, n-1, n-2, \dots, 1, 0$ ， 所以， 总共需要移动的元素个数为  $(1+2+3+4+\dots+n)=n*(n+1)/2$ 。 故平均需要移动的元素个数为  $n*(n+1)/2(n+1)=n/2$ ;

44、顺序表有 5 个元素，设在任何位置上插入元素是等概率的，则在表中插入一个元素时

所需移动元素的平均次数为（ ）。

- A、 3
- B、 2
- C、 2.5
- D、 5

答案： C

解析： 在顺序表中插入元素时，元素的平均移动个数为  $n/2$ 。

45、在一个长度为  $n$  的顺序存储的线性表中，向第  $i$  个元素 ( $1 \leq i \leq n+1$ ) 之前插入一个新元素时，需要从后向前依次移（ ）个元素。

- A、  $n-i$
- B、  $n-i+1$
- C、  $n-i-1$
- D、  $i$

答案： B

解析： 在顺序表中插入元素时，元素的平均移动个数为  $n/2$ 。

46、在一个长度为  $n$  的顺序存储的线性表中，删除第  $i$  个元素 ( $1 \leq i \leq n+1$ ) 时，需要从前向后依次前移（ ）个元素。

A、  $n-i$

B、  $n-i+1$

C、  $n-i-1$

D、  $i$

答案： A

解析：在顺序表中插入元素时，元素的平均移动个数为  $n/2$ 。

47、设顺序表有 9 个元素，则在第 3 个元素前插入一个元素所需移动元素的个数为（ ）。

A、 9

B、 4.5

C、 7

D、 6

答案： C

解析：插入元素后需要将包括插入位置的元素以及顺序表中的后续元素依次向后移动。

48、设顺序表有 19 个元素，第一个元素的地址为 200，且每个元素占 3 个字节，则第 14 个

元素的存储地址为（ ）。

- A、 236
- B、 239
- C、 242
- D、 245

答案： B

解析：在顺序表中元素的地址计算方法是：首地址+前面元素的个数\*每个元素的占用的地址。

49、以下关于线性表的说法不正确的是( )。

- A、 线性表中的数据元素可以是数字、字符、记录等不同类型。
- B、 线性表中包含的数据元素个数不是任意的。
- C、 线性表中的每个结点都有且只有一个直接前趋和直接后继。
- D、 存在这样的线性表：表中各结点都没有直接前趋和直接后继。

答案： C

解析：线性表的逻辑结构中存在没有后继结点的结点。

50、线性表的顺序存储结构是一种( )的存储结构。

- A、 随机存取

B、 顺序存取

C、 索引存取

D、 散列存取

答案： A

解析： 顺序表可以理解为数组，数组中的任一元素都是能够随机存取的。

51、 设顺序表共有  $n$  个元素，用数组 `elem` 存储，实现在第  $i$  个元素之前插入一个元素  $e$  的操

作，下列语句正确的为（ ）。

A、 `FOR j=n DOWNTO i DO elem[j]=elem[j+1];`  
`elem[i]=e;`

B、 `FOR j=i TO n DO elem[j]=elem[j+1];`  
`elem[i]=e;`

C、 `FOR j=i TO n DO elem[j+1]=elem[j];`  
`elem[i]=e;`

D、 `FOR j=n DOWNTO i DO elem[j+1]=elem[j];`  
`elem[i]=e;`



答案： D

解析：插入元素后需要将包括插入位置的元素以及顺序表中的后续元素依次向后移动。

52、设顺序表的第 5 个元素的存储地址为 200，且每个元素占一个存储单元，则第 14 个元素

的存储地址为（ ）。

A、 208

B、 209

C、 210

D、 214

答案： B

解析：在顺序表中元素的地址计算方法是：首地址+前面元素的个数\*每个元素的占用的地址。

53、 对于用一维数组  $d[0..n-1]$  顺序存储的线性表，其算法的时间复杂度为  $O(1)$  的操作是（ ）

A、 将  $n$  个元素从小到大排序

B、 从线性表中删除第  $i$  个元素 ( $1 \leq i \leq n$ )

C、 查找第  $i$  个元素 ( $1 \leq i \leq n$ )

D、

在线性表中第  $i$  个元素之后插入一个元素

答案： C

解析： A 操作的时间复杂度为  $O(n^2)$ ；

B 操作需要移动元素，时间复杂度为  $O(n)$ ；

C 操作可以直接由  $d[i-1]$  得到，时间复杂度为  $O(1)$ ；

D 操作需要移动元素，时间复杂度为  $O(n)$ 。

54、在顺序表中，只要知道( )，就可在相同时间内求出任一结点的存储地址。

A、 基地址

B、 结点大小

C、 向量大小

D、 基地址和结点大小

答案： D

解析： 访问数组中的任意一个元素，只需要知道数组的基地址和该元素在数组中的相对位置，也就是节点大小。

55、在  $n$  个结点的顺序表中，算法的时间复杂度是  $O(1)$  的操作是（ ）。

A、 访问第  $i$  个结点 ( $1 \leq i \leq n$ ) 和求第  $i$  个结点的直接前驱 ( $2 \leq i \leq n$ )

B、 在第  $i$  个结点后插入一个新结点 ( $1 \leq i \leq n$ )

C、 删除第  $i$  个结点 ( $1 \leq i \leq n$ )

D、 将  $n$  个结点从小到大排序

答案： A

解析：访问数组中的任意一个元素，只需要知道数组的基地址和该元素在数组中的相对位置，也就是节点大小。

56、用数组表示线性表的优点是（ ）。

A、 便于插入和删除操作

B、 便于随机存取

C、 可以动态的分配存储空间

D、 不需要占用一片相邻的存储空间

答案： B

解析：顺序表的特点就是可以随机访问表中的任意一个元素，这也是采用数组来表示线性表的主要原因。

57、下面关于线性表的叙述错误的是（ ）。

A、 用数组表示，表中诸元素的存储位置是连在一起的

B、 用链表表示，便于插入和删除操作

C、 用链表表示，不需要占用一片相邻的存储空间

D、 插入和删除操作仅允许在表的一端运行

答案： D

解析：线性表如果用顺序表的方式来表达，则可以任意访问表中的任意元素，也可以自任意位置进行插入和删除操作。

58、线性表的顺序存储结构是一种（ ）的存储结构。

A、 随机存取

B、 顺序存取

C、 索引存取

D、 HASH 存取

答案： A

解析： 顺序存储结构是一种随机存取结构。

59、在下面关于线性表的叙述中,选出错误的一项 ( )

A、 采用顺序存储的线性表,必须占用一片连续的存储单元

B、 采用顺序存储的线性表,便于进行插入和删除操作

C、 采用链接存储的线性表,不必占用一片连续的存储单元

D、 采用链接存储的线性表,便于进行插入和删除操作

答案： B

解析： 采用链接存储的线性表,便于进行插入和删除操作

60、线性表采用链式存储时，其地址(     ) 。

- A、 必须是连续的；
- B、 部分地址必须是连续的；
- C、 一定是不连续的；
- D、 连续与否均可以。

答案： D

解析：链表的存储结构中，各节点间的存储空间没有特定的连续与否的要求。

61、用链表表示线性表的优点是 (    ) 。

- A、 便于随机存取
- B、 花费的存储空间较顺序存储少
- C、 便于插入和删除
- D、 数据元素的物理顺序与逻辑顺序相同

答案： C

解析：链表优点是便于插入和删除。

62、某链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最后一个元素，则采用( )存储方式最节省运算时间。

A、 单链表

B、 双链表

C、 单循环链表

D、 带头结点的双循环链表

答案： D

解析：链表优点是便于插入和删除。

63、循环链表的主要优点是( ) 。

A、 不再需要头指针了

B、 已知某个结点的位置后，能够容易找到他的直接前趋

C、 在进行插入、删除运算时，能更好的保证链表不断开

D、 从表中的任意结点出发都能扫描到整个链表

答案： D

解析：循环链表优点是解决了假溢出的问题，并且可以在表中的任意节点出发访问任意节点。

64、单链表中，增加一个头结点的目的是为了（ ）。

A、 使单链表至少有一个结点

B、 标识表结点中首结点的位置

C、 方便运算的实现

D、 说明单链表是线性表的链式存储

答案： C

解析：单链表头节点的特点是方便在编码中寻找该链表。



65、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

答案： D

解析：尾指针便于寻找最后一个元素，对于最后一个元素的插入和删除操作来说，效率最高。

66、若某线性表中最常用的操作是取第  $i$  个元素和找第  $i$  个元素的前趋元素，则采用（ ）存储方式最节省运算时间（ ）。

- A、 单链表
- B、 顺序表
- C、 双链表

D、 单循环链表

答案： B

解析： 找前驱元素的操作，最好的是通过顺序表的方式进行存储。

67、一个向量(一种顺序表)第一个元素的存储地址是 100，每个元素的长度为 2，则第 5 个元素的地址是\_\_\_\_\_。

A、 110

B、 108

C、 100

D、 120

答案： B

解析： 第 5 个元素的地址= $100+2*(5-1)=108$ 。

68、不带头结点的单链表 head 为空的判定条件是\_\_\_\_\_。

A、  $head == NULL$ ;

B、  $head \rightarrow next == NULL$ ;

C、  $head \rightarrow next == head$ ;

D、  $head != NULL$ ;

答案： A

解析： 标准的判断链表为空的基本代码。

69、在循环双链表的 p 所指结点之后插入 s 所指结点的操作是\_\_\_\_\_。

A、 p->right=s; s->left=p; p->right->left=s; s->right=p->right;

B、 p->right=s; p->right->left=s; s->left=p; s->right=p->right;

C、 s->left=p; s->right= p->right; p->right=s; p->right->left=s;

D、 s->left=p; s->right=p->right; p->right->left=s; p->right=s;

答案： D

解析： 插入节点，首先更新插入节点的指针域，然后断链，并更新原来节点的相应指针域。

70、在一个单链表中，已知 q 所指结点是 p 所指结点的前驱结点，若在 q 和 p 之间插入 s 结点，则执行\_\_\_\_\_。

A、 s->next=p->next; p->next=s;

B、 p->next=s->next; s->next=p;

C、 `q->next=s; s->next=p;`

D、 `p->next=s; s->next=q;`

答案： C

解析：插入节点，首先更新插入节点的指针域，然后断链，并更新原来节点的相应指针域，注意与循环链表的插入操作的区别。

71、以下说法错误的是（ ）。

A、 求表长、定位这两种运算在采用顺序存储结构时实现的效率不比采用链式存储结构时实现的效率低

B、 顺序存储的线性表可以随机存取

C、 由于顺序存储要求连续的存储区域，所以在存储管理上不够灵活

D、 线性表的链式存储结构优于顺序存储结构。

答案： D

解析：在线性表中，链式存储结构和顺序存储结构各有各的优点。

72、(1) 静态链表既有顺序存储的优点，又有动态链表的优点。所以，它存取表中第  $i$  个元

素的时间与  $i$  无关。

(2) 静态链表中能容纳的元素个数的最大数在表定义时就确定了，以后不能增加。

(3) 静态链表与动态链表在元素的插入、删除上类似，不需做元素的移动。

以上错误的是 ( )

A、 (1) , (2)

B、 (1)

C、 (1) , (2) , (3)

D、 (2)

答案： B

解析：静态链表的是在顺序表中模拟了动态链表的逻辑结构，但在插入和删除的过程中不需要和顺序表一样进行数据元素的移动。

73、在一个以  $h$  为头的单循环链中， $p$  指针指向链尾的条件是 ( )

A、  $p.\text{next}=h$

B、 p.next=NULL

C、 p.next.next=h

D、 p.data=-1

答案： A

解析：在单循环链表中，尾节点的 next 指针域指向的是链表的头节点，因此选 A。

74、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

答案： D

解析：在单循环链表中，尾指针能够很快的找到链表中的第一个节点和链表中的最后一个节点，因此非常适合在最后一个元素之后插入和删除第一个元素。

75、静态链表中指针表示的是（ ）

A、 内存地址

B、 数组下标

C、 下一元素地址

D、 左、右孩子地址

答案： C

解析：根据静态链表的定义，指针指向的是链表中的下一个元素的地址，该地址用的是数组下标来进行表示。

76、单链表的存储密度 （）

A、 大于 1；

B、 等于 1；

C、 小于 1;

D、 不能确定

答案： C

解析：存储密度的概念，等于链表节点中的数据域占用的空间大小除以整个节点占用的存储空间大小，因此是小于 1。

77、在一个单链表中，若要在 p 所指向的结点之后插入一个新结点，则需要相继修改( )个指针域的值。

A、 1

B、 2

C、 3

D、 4

答案： B

解析：根据单链表的插入过程，插入一个节点是，需要修改前驱结点和插入结点两个结点的指针域。

78、p 指向线性链表中的某一结点，则在线性链表的表尾插入结点 S 的语句序列是( )。

A、 while(p->next!=NULL) p=p->next; p->next=s; s->next=NULL;

B、 while(p!=NULL) p=p->next; p->next=s; s->next=NULL;



C、 while(p->next!=NULL) p=p->next; s->next=p;p->next=NULL;

D、 while(p!=NULL) p=p->next->next;->next;p->next=s;s->next=p

答案： A

解析： 先通过 while 循环找到表尾结点， 然后进行新结点的插入操作。

79、 线性表的逻辑顺序与存储顺序总是一致的。（ ）

答案： 错误

解析： 采用不同的存储方式的时候， 逻辑顺序并不一定与存储顺序一致， 例如静态链表中， 就有可能是逻辑顺序排在后面的元素而其存储顺序却在前面。

80、 顺序存储的线性表可以按序号随机存取。（ ）

答案： 正确

解析： 顺序表的特点就是按数组下标随机存取。

81、 顺序表的插入和删除一个数据元素， 每次操作平均只有近一半的元素需要移动。（ ）

答案： 正确

解析： 顺序表的插入和删除的平均元素移动次数为  $n/2$ 。

82、线性表中的元素可以是各种各样的，但同一线性表中的数据元素具有相同的特性，因此是属于同一数据对象。（ ）

答案： 正确

解析：线性表的定义，所有数据元素具有相同的特性，属于统一数据对象。

83、在线性表的顺序存储结构中，逻辑上相邻的两个元素在物理位置上并不一定紧邻。（ ）

答案： 错误

解析：顺序表中两个逻辑上相邻的元素一定是在物理位置上相邻的。

84、在线性表的链式存储结构中，逻辑上相邻的元素在物理位置上不一定相邻。（ ）

答案： 正确

解析：链表结构中，每个结点的物理位置不是相邻的，因此逻辑上相邻的元素在物理位置上不一定相邻。

85、线性表的链式存储结构优于顺序存储结构。（ ）

答案： 错误

解析：线性表的链式存储结构和顺序存储结构各有各的优点，根据具体应用的不同选择不同的存储结构，因此不能说链式存储优于顺序存储结构。

86、在线性表的顺序存储结构中，插入和删除时，移动元素的个数与该元素的位置有关。（ ）

答案： 正确

解析：在顺序表的插入和删除操作中，没插入和删除一个元素的时候，需要进行后续元素的后移和前移操作，操作的个数跟操作元素的位置有关。

87、线性表的链式存储结构是用一组任意的存储单元来存储线性表中数据元素的。（ ）

答案： 正确

解析：链式存储结构中，各结点间是由指针进行连接的，因此每个结点的存储单元是任意随机分配的。

88、在单链表中，要取得某个元素，只要知道该元素的指针即可，因此，单链表是随机存取的存储结构。（ ）

答案： 错误

解析：要找到单链表中某元素的指针，需要从链表的头节点开始进行遍历才能找到结点，因此不是随机存取的存储结构，任何结点的存取都需要从头节点开始进行遍历得到。

89、链表中的头结点仅起到标识的作用。( )

答案： 错误

解析：链表的头结点的主要作用在于快速找到链表中的第一个结点，便于链表的遍历。

90、顺序存储结构的主要缺点是不利于插入或删除操作。( )

答案： 正确

解析：在顺序表中，每次的插入和删除操作都需要对相关结点的后续结点进行移动操作，其所消耗的时间复杂度较大。

91、线性表采用链表存储时，结点和结点内部的存储空间可以是不连续的。  
( )

答案： 错误

解析：结点的内部空间必须是连续的。

92、顺序存储方式插入和删除时效率太低，因此它不如链式存储方式好。  
( )

答案： 错误

解析：顺序表的缺点是结点的插入和删除效率低，但它相对于链式存储而言，可以快速的访问表中的任意一个结点，而不需要从第一个结点遍历访问的优势。

93、对任何数据结构链式存储结构一定优于顺序存储结构。( )

答案： 错误

解析：顺序表的缺点是结点的插入和删除效率低，但它相对于链式存储而言，可以快速的访问表中的任意一个结点，而不需要从第一个结点遍历访问的优势。

94、顺序存储方式只能用于存储线性结构。( )

答案： 错误

解析：顺序表的缺点是结点的插入和删除效率低，但它相对于链式存储而言，可以快速的访问表中的任意一个结点，而不需要从第一个结点遍历访问的优势。

95、集合与线性表的区别在于是否按关键字排序。( )

答案： 错误

解析：集合中的元素是无序的，这是它和线性表之间的主要区别。但线性表的元素之间不一定是按照关键字的顺序进行排序的。

96、所谓静态链表就是一直不发生变化的链表。( )

答案： 错误

解析：静态链表是在数组中实现的链表逻辑结构，而不是用指针连接每个结点，他不是一直不发生变化的链表。

97、线性表的特点是每个元素都有一个前驱和一个后继。( )

答案： 错误

解析：非循环线性表的头结点没有前驱结点，尾结点没有后继结点。

98、取线性表的第  $i$  个元素的时间同  $i$  的大小有关。( )

答案： 错误

解析：线性表中第  $i$  各元素的存取和结点的大小没有关系，只和其在线性表中所处的位置有关系。

99、循环链表不是线性表。( )

答案： 错误

解析：循环链表是采用链表方式实现的一种特殊的线性表。

100、线性表只能用顺序存储结构实现。( )

答案： 错误

解析：线性表有两种实现方式，一种是顺序存储结构，一种是链式存储结构。

101、线性表就是顺序存储的表。(            )

答案： 错误

解析：线性表有两种实现方式，一种是顺序存储结构，一种是链式存储结构。

102、为了很方便的插入和删除数据，可以使用双向链表存放数据。(            )

答案： 正确

解析：双向链表的特点是很方便的找到处理结点的前驱和后继结点，这对结点位置的数据插入和删除带来了便利性。

103、顺序存储方式的优点是存储密度大，且插入、删除运算效率高。  
(            )

答案： 错误

解析：顺序存储方式的有点是存储密度大，能快速的访问某个元素，但相对链式存储而言，元素的插入、删除效率较低。

104、链表是采用链式存储结构的线性表,进行插入、删除操作时，在链表中比在顺序存储结构中效率高。(            )

答案： 正确

解析：链式存储结构相对于顺序表而言，其进行插入和删除操作时，效率非常高。

105、线性表采用链表存储时，结点和结点内部的存储空间可以是不连续的。（ ）

答案： 错误

解析：结点的内部空间必须是连续的。

106、在具有头结点的链式存储结构中，头指针指向链表中的第一个数据结点。（ ）

答案： 错误

解析：头结点不是链表的第一个数据结点。

107、顺序存储的线性表可以随机存取。（ ）

答案： 正确

解析：顺序存储的线性表是可以随机存取的。

108、在单链表中，要访问某个结点，只要知道该结点的指针即可；因此，单链表是一种随机存储结构。（ ）



答案： 错误

解析：单链表种结点的访问，必须从头结点开始通过遍历的方式对链表种的结点进行访问。

109、在线性表的顺序存储结构中，插入和删除元素时，移动元素的个数与该元素的位置有关。（ ）

答案： 正确

解析：顺序表的顺序存储结构再插入和删除元素时，需要进行后续元素的前后移动，因此移动的次数和元素的位置有关。

110、顺序存储结构属于静态结构，链式结构属于动态结构。（ ）

答案： 正确

解析：参考顺序存储和链式存储的实现方式。

111、链表的每个结点中都恰好包含一个指针。（ ）

答案： 错误

解析：链表中的结点可含多个指针域，分别存放多个指针。例如，双向链表中的结点可以含有两个指针域，分别存放指向其直接前趋和直接后继结点的指针。

112、链表的物理存储结构具有同链表一样的顺序。（ ）

答案： 错误

解析：链表的存储结构特点是无序，而链表的示意图有序。

113、链表的删除算法很简单，因为当删除链中某个结点后，计算机会自动地将后续的各个单元向前移动。（ ）

答案： 错误

解析：链表的结点不会移动，只是指针内容改变。

114、线性表的每个结点只能是一个简单类型，而链表的每个结点可以是一个复杂类型。（ ）

答案： 错误

解析：混淆了逻辑结构与物理结构，链表也是线性表！且即使是顺序表，也能存放记录型数据。

115、顺序表结构适宜于进行顺序存取，而链表适宜于进行随机存取。（ ）

答案： 错误

解析：错，正好说反了。顺序表才适合随机存取，链表恰恰适于“顺藤摸瓜”。

116、顺序存储方式的优点是存储密度大，且插入、删除运算效率高。（ ）

答案： 错误

解析：错，前半正确，但后半说法错误，那是链式存储的优点。顺序存储方式插入、删除运算效率较低，在表长为  $n$  的顺序表中，插入和删除一个数据元素，平均需移动表长一半个数的数据元素。

117、线性表在物理存储空间中也一定是连续的。 （ ）

答案： 错误

解析：错，线性表有两种存储方式，顺序存储和链式存储。后者不要求连续存放。

118、线性表在顺序存储时，逻辑上相邻的元素未必在存储的物理位置次序上相邻。 （ ）

答案： 错误

解析：错误。线性表有两种存储方式，在顺序存储时，逻辑上相邻的元素在存储的物理位置次序上也相邻。

119、有一个单链表（不同结点的数据域值可能相同），其头指针为 head，编写一个函数计算数据域为  $x$  的结点个数。

提示：本题是遍历通过该链表的每个结点，每遇到一个结点，结点个数加 1，结点个数存储在变量  $n$  中。

答案：

```

答案: int count (head, x)

{

/*本题中 head 为链头指针，不含头结点*/

node *p;

int n=0;

p=head;

while (p!=NULL)

{

if (p->data==x) n++;

p=p->next;

}

return(n);

}

```

解析：从链表的头结点开始，依次进行遍历，每次遍历查找结点的 data 域，如果等于要查找的值则计数，否则遍历下一个元素。

120、假设有两个按元素值递增次序排列的线性表，均以单链表形式存储。请编写算法将这两

个单链表归并为一个按元素值递减次序排列的单链表，并要求利用原来两个单链表的结点存放归并后的单链表。

答案:

答案:

LinkedList Union(LinkedList la, lb) //la, lb 分别是带头结点的两个单链表的头指针, 链表中的元素值按递增序排列, 本算法将两链表合并成一个按元素值递减次序排列的单链表。

```
{ pa=la->next; pb=lb->next; // pa, pb 分别是链表 la 和 lb 的工作指针
```

```
    la->next=null; // la 作结果链表的头指针, 先将结果链表初始化为空。
```

```
    while(pa!=null && pb!=null) // 当两链表均不为空时作
```

```
        if(pa->data<=pb->data)
```

```
            { r=pa->next; // 将 pa 的后继结点暂存于 r。
```

```
                pa->next=la->next; // 将 pa 结点链于结果表中, 同时逆置。
```

```
                la->next=pa;
```

```
                pa=r; } // 恢复 pa 为当前待比较结点。
```

```
            else
```

```
                {r=pb->next; // 将 pb 的后继结点暂存于 r。
```

```
                    pb->next=la->next; // 将 pb 结点链于结果表中, 同时逆置。
                    la->next=pb;
```

```
                    pb=r; // 恢复 pb 为当前待比较结点。
```

```
                }
```

```
                while(pa!=null) // 将 la 表的剩余部分链入结果表, 并逆置。
```

```

        {r=pa->next; pa->next=la->next; la->next=pa; pa=r; }
        while(pb!=null)

                {r=pb->next; pb->next=la->next; la->next=pb;

pb=r; }

}

```

解析：因为两链表已按元素值递增次序排列，将其合并时，均从第一个结点起进行比较，将小的链入链表中，同时后移链表工作指针。该问题要求结果链表按元素值递减次序排列。故在合并的同时，将链表结点逆置。

121、知 L1、L2 分别为两循环单链表的头结点指针，m,n 分别为 L1、L2 表中数据结点个数。要求设计一算法，用最快速度将两表合并成一个带头结点的循环单链表。

答案：

LinkedList Union(LinkedList L1,L2;int m,n) //L1 和 L2 分别是两循环单链表的头结点的指针，m 和 n 分别是 L1 和 L2 的长度。 // 本算法用最快速度将 L1 和 L2 合并成一个循环单链表。

```

{

if(m<0||n<0)

{printf(“表长输入错误\n”); exit(0);}

if(m<n)                                // 若 m<n, 则查 L1 循环单链表的
最后一个结点。        { if(m==0)return(L2); //L1 为空表。

else {p=L1;

while(p->next!=L1) p=p->next; // 查最后一个元素结点。

```

```
        p->next=L2->next; // 将 L1 循环单链表的元素结点插入到 L2
        的第一元素结点前。
```

```
        L2->next=L1->next;
```

```
        free(L1); // 释放无用头结点。
```

```
    }
```

```
    } // 处理完  $m < n$  情况
```

```
    else // 下面处理 L2 长度小于等于 L1 的情况
```

```
    {if (n==0)
```

```
        return(L1); // L2 为空表。
```

```
    else {p=L2;
```

```
        while(p->next!=L2)    p=p->next; // 查最后元素结点。
```

```
        p->next=L1->next; // 将 L2 的元素结点插入到 L1 循环单
        链表的第一元素结点前。
```

```
        L1->next=L2->next;
```

```
        free(L2); // 释放无用头结点。
```

```
    }
```

```
    } // 算法结束。
```

解析：循环单链表 L1 和 L2 数据结点个数分别为  $m$  和  $n$ ，将二者合成一个循环单链表时，需要将一个循环链表的结点（从第一元素结点到最后一个结点）插入到另一循环链表的第一元素结点前即可。题目要求“用最快速度将两表合并”，因此应找结点个数少的链表查其尾结点。

122、已知不带头结点的线性链表 list，链表中结点构造为（data、link），其中 data 为数据域，link 为指针域。请写一算法，将该链表按结点数据域的值的从小到大小重新链接。要求链接过程中不得使用除该链表以外的任何链结点空间。

答案：

LinkedList LinkListSort(LinkedList list) //list 是不带头结点的线性链表，链表结点构造为 data 和 link 两个域，data 是数据域，link 是指针域。本算法将该链表按结点数据域的值的从小到大小重新链接。

```
{p=list->link;          //p 是工作指针，指向待排序的当前元素。
```

```
    list->link=null; //假定第一个元素有序，即链表中现只有一个  
    结点。                                while(p!=null)
```

```
        {r=p->link;          //r 是 p 的后继。
```

```
        q=list;
```

```
            if(q->data>p->data) //处理待排序结点 p 比第一个元  
            素结点小的情况。                {p->link=list;
```

```
            list=p; //链表指针指向最小元素。
```

```
        }
```

```
        else //查找元素值最小的结点。
```

```
            {while(q->link!=null&&q->link->data<p->data)
```

```
            q=q->link;
```

```
            p->link=q->link; //将当前排序结点链入有序链表中。
```

```
            q->link=p;
```

```
        }
```

```
        p=r; //p 指向下个待排序结点。
```



```
}  
  
}
```

解析：本题实质上是一个排序问题，要求“不得使用除该链表结点以外的任何链结点空间”。链表上的排序采用直接插入排序比较方便，即首先假定第一个结点有序，然后，从第二个结点开始，依次插入到前面有序链表中，最终达到整个链表有序。

算法时间复杂度的分析与用顺序存储结构时的情况相同。但顺序存储结构将第  $i$  ( $i > 1$ ) 个元素插入到前面第 1 至第  $i-1$  个元素的有序表时，是将第  $i$  个元素先与第  $i-1$  个元素比较。而在链表最佳情况均是和第一元素比较。两种存储结构下最佳和最差情况的比较次数相同，在链表情况下，不移动元素，而是修改结点指针。

123、 已知非空线性链表由 `list` 指出，链结点的构造为 (**data,link**) .请写一算法，将链表中数据域值最小的那个链结点移到链表的最前面。要求：不得额外申请新的链结点。

答案：

```
LinkedList delinsert (LinkedList list)  
  
    //list 是非空线性链表，链结点结构是 (data, link)，data 是数据域，  
    link 是链域。  
  
    // 本算法将链表中数据域值最小的那个结点移到链表的最前面。  
  
    { p=list->link; //p 是链表的工作指针  
  
        pre=list;           //pre 指向链表中数据域最小值结点的前驱。  
  
        q=p;                 //q 指向数据域最小值结点，初始假定是第一结点
```

```

while (p->link!=null)

    {if (p->link->data<q->data)

        {pre=p; q=p->link; } //找到新的最小值结点;

        p=p->link;

    }

if (q!=list->link) //若最小值是第一元素结点，则不需再操作

{pre->link=q->link; //将最小值结点从链表上摘下;

q->link= list->link; //将 q 结点插到链表最前面。

list->link=q;

}

} // 算法结束

```

解析：本题要求将链表中数据域值最小的结点移到链表的最前面。首先要查找最小值结点。将其移到链表最前面，实质上是将该结点从链表上摘下（不是删除并回收空间），再插入到链表的最前面。

124、假设一个单循环链表，其结点含有三个域 **pre**、**data**、**link**。其中 **data** 为数据域；**pre** 为指针域，它的值为空指针（NIL）；**link** 为指针域，它指向后继结点。请设计算法，将此表改成双向循环链表。

答案：

```

void    StoDouble (LinkedList  la)

```

//la 是结点含有 pre, data, link 三个域的单循环链表。其中 data 为数据域, pre 为空指针域, link 是指向后继的指针域。本算法将其改造成双向循环链表。

```
{  
  
    while (la->link->pre==null)  
  
    {  
  
        la->link->pre=la;        //将结点 la 后继的 pre 指针指向 la。  
  
        la=la->link;            //la 指针后移。  
  
    }  
  
}    //算法结束。
```

解析：将具有两个链域的单循环链表，改造成双向循环链表，关键是控制给每个结点均置上指向前驱的指针，而且每个结点的前驱指针置且仅置一次。算法中没有设置变量记住单循环链表的起始结点，至少省去了一个指针变量。当算法结束时，la 恢复到指向刚开始操作的结点，这是本算法的优点所在。

125、已知递增有序的单链表 A, B 分别存储了一个集合，请设计算法以求出两个集合 A 和 B 的差集 A-B（即仅由在 A 中出现而不在 B 中出现的元素所构成的集合），并以同样的形式存储，同时返回该集合的元素个数。

答案：

```
void    Difference (LinkedList    A, B, *n)
```

//A 和 B 均是带头结点的递增有序的单链表，分别存储了一个集合，本算法求两集合的差集，存储于单链表 A 中，\*n 是结果集合中元素个数，调用时为 0

```

    {p=A->next;                                     //p 和 q 分别是链表
A 和 B 的工作指针。

    q=B->next;

    pre=A;                                           //pre 为 A 中 p 所指结点的前驱结点的指
针。

    while (p!=null && q!=null)

        if (p->data<q->data) {pre=p; p=p->next; *n++; } // A 链表
中当前结点指针后移。

        else if (p->data>q->data) q=q->next;         //B 链表中当
前结点指针后移。

        else {pre->next=p->next;
// 处理 A, B 中元素值相同的结点, 应删除。

                u=p;    p=p->next;    free (u) ; }    //删除结点

```

解析：求两个集合 A 和 B 的差集 A-B，即在 A 中删除 A 和 B 中共有的元素。由于集合用单链表存储，问题变成删除链表中的结点问题。因此，要记住被删除结点的前驱，以便顺利删除被删结点。两链表均从第一元素结点开始，直到其中一个链表到尾为止。

126、已知一个单链表中每个结点存放一个整数，并且结点数不少于 2，请设计算法以判断该链表中第二项起的每个元素值是否等于其序号的平方减去其前驱的值，若满足则返回 **true**，否则返回 **false**。

答案：

```

int Judge (LinkedList la)

//la 是结点的元素为整数的单链表。本算法判断从第二结点开始，每个元素值是否等于其序号的平方减去其前驱的值，如是返回 true；否则，返回 false。

{

    p=la->next->next; //p 是工作指针，初始指向链表的第二项。

    pre=la->next;      //pre 是 p 所指结点的前驱指针。

    i=2;                //i 是 la 链表中结点的序号，初始值为 2。

    while (p!=null)

        if (p->data==i*i-pre->data)

            {i++; pre=p; p=p->next; } // 结点值间的关系符合题目要求

        else break;

// 当前结点的值不等于其序号的平方减去前驱的值。

    if (p!=null)

        return (false); // 未查到表尾就结束

    了。

    else

        return (true); // 成功返回。

} // 算法结束。

```

解析：本题要求对单链表结点的元素值进行运算，判断元素值是否等于其序号的平方减去其前驱的值。这里主要技术问题是结点的序号和前驱及后继指针的正确指向。

127、两个整数序列  $A=a_1, a_2, a_3, \dots, a_m$  和  $B=b_1, b_2, b_3, \dots, b_n$  已经存入两个单链表中，设计一个算法，判断序列  $B$  是否是序列  $A$  的子序列。

答案：

```
int    Pattern (LinkedList  A, B)

//A 和 B 分别是数据域为整数的单链表，本算法判断 B 是否是 A 的子序列。如
//是，返回 1；否则，返回 0 表示失败。

{

    p=A;          //p 为 A 链表的工作指针，本题假定 A 和 B 均无头结点。

    pre=p;        //pre 记住每趟比较中 A 链表的开始结点。

    q=B;          //q 是 B 链表的工作指针。

    while (p  &&  q)

        if (p->data==q->data)

            {p=p->next; q=q->next; }

        else

            {pre=pre->next; p=pre;    //A 链表新的开始比较结点。

            q=B; }                      //q 从 B 链表第一
结点开始。

        if (q==null)

            return (1) ;                //B 是 A 的子序列。

        else

            return (0) ;                //B 不是 A 的子序列。
}
```

```
} // 算法结束。
```

解析：本题实质上是一个模式匹配问题，这里匹配的元素是整数而不是字符。因两整数序列已存入两个链表中，操作从两链表的第一个结点开始，若对应数据相等，则后移指针；若对应数据不等，则 A 链表从上次开始比较结点的后继开始，B 链表仍从第一结点开始比较，直到 B 链表到尾表示匹配成功。A 链表到尾 B 链表未到尾表示失败。操作中应记住 A 链表每次的开始结点，以便下趟匹配时好从其后继开始。

128、已知 L 为没有头结点的单链表中第一个结点的指针，每个结点数据域存放一个字符，该字符可能是英文字母字符或数字字符或其它字符，编写算法构造三个以带头结点的单循环链表表示的线性表，使每个表中只含同一类字符。（要求用最少的时间和最少的空间）

答案：

```
void    OneToThree (LinkedList L, la, ld, lo)
```

//L 是无头结点的单链表第一个结点的指针，链表中的数据域存放字符。本算法将链表 L 分解成含有英文字母字符、数字字符和其它字符的带头结点的三个循环链表。

```
{
```

```
    la= (LinkedList) malloc (sizeof (LNode) );    // 建立三个链表的头结点
```

```
    ld= (LinkedList) malloc (sizeof (LNode) );
```

```
    lo= (LinkedList) malloc (sizeof (LNode) );
```

```
    la->next=la; ld->next=ld; lo->next=lo; // 置三个循环链表为空表
```

```
    while (L!=null) // 分解原链表。
```

```

        {

            r=L;

            L=L->next;    //L 指向待处理结点的后继

            if (r->data>= 'a' && r->data<= 'z' || r->data>= 'A' && r->data<= 'Z' )

                {r->next=la->next;    la->next=r; }                // 处理
字母字符。

            else if (r->data>= '0' && r->data<= '9' )

                {r->next=ld->next; ld->next=r; }                // 处理数字
字符

            else

                {r->next=lo->next; lo->next=r; }                // 处理其它符号。

        } // 结束 while (L!=null) 。

    } // 算法结束

```

解析：将一个结点数据域为字符的单链表，分解成含有字母字符、数字字符和其它字符的三个循环链表，首先要构造分别含有这三类字符的表头结点。然后从原链表第一个结点开始，根据结点数据域是字母字符、数字字符和其它字符而分别插入到三个链表之一的链表。注意不要因结点插入新建链表而使原链表断链。另外，题目并未要求链表有序，插入采用“前插法”，每次插入的结点均成为所插入链表的第一元素的结点即可。算法中对 L 链表中每个结点只处理一次，时间复杂度  $O(n)$ ，只增加了必须三个表头结点，符合题目“用最少的时间和最少的空间”的要求。

129、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

答案： B

解析：根据栈结构的定义，栈操作数据的顺序是后进先出的结构。

130、一个栈的输入序列为  $123\cdots n$ ，若输出序列的第一个元素是  $n$ ，输出第  $i$  ( $1 \leq i \leq n$ ) 个元素是 ( )。

A、 不确定

B、  $n-i+1$

C、  $i$

D、  $n-i$

答案： B

解析：应为栈的操作是后进先出的结构，并且第一个输出的是  $n$ ，则是按照逆序输出，则第  $i$  个输出的是  $n-i+1$

131、若一个栈的输入序列为  $1, 2, 3, \dots, n$ ，输出序列的第一个元素是  $i$ ，则第  $j$  个输出元素是（ ）。

A、  $i-j-1$

B、  $i-j$

C、  $j-i+1$

D、 不确定的

答案： D

解析：第一个输出的元素为  $i$ ，则后续的操作中无法确定是新的元素入栈还是栈中的元素出栈，因此无法确定第  $j$  个元素的输出序列。

132、若已知一个栈的入栈序列是  $1, 2, 3, \dots, n$ ，其输出序列为  $p_1, p_2, p_3, \dots, p_n$ ，若  $p_n$  是  $n$ ，则  $p_i$  是（ ）。

A、  $i$

B、  $n-i$

C、  $n-i+1$

D、 不确定

答案： D

解析：当栈里面的输出序列中，最后一个元素的序号是  $n$ ，同样不能保证前面的元素是按照顺序输入或输出的，其输出的序列也可以是任意的，只需要保证最后一次输出的是  $n$  元素就可以。

133、有六个元素 6，5，4，3，2，1 的顺序进栈，问下列哪一个不是合法的出栈序列？（ ）

A、 5 4 3 6 1 2

B、 4 5 3 1 2 6

C、 3 4 6 5 2 1

D、 2 3 4 1 5 6

答案： C

解析：按照先进后出的原则，当第一个输出的元素是 3 的时候，其前面的元素一定都在栈中，因此，不可能出现输出的时候 6 在 5 的前面输出。

134、设栈的输入序列是 1，2，3，4，则（ ）不可能是其出栈序列。

A、 1，2，4，3，

B、 2, 1, 3, 4,

C、 1, 4, 3, 2,

D、 4, 3, 1, 2,

答案： D

解析：按照先进后出的原则，当第一个输出的元素是 4 的时候，其前面的元素一定都在栈中，因此，不可能出现输出的时候 1 在 2 的前面输出。

135、一个栈的输入序列为 1 2 3 4 5，则下列序列中不可能是栈的输出序列的是（ ）。

A、 2 3 4 1 5

B、 5 4 1 3 2

C、 2 3 1 4 5

D、 1 5 4 3 2

答案： B

解析：5 第一个输出的时候，其前面的元素一定都是按顺序输入到栈中的，因此不可能会出现先输出 1 再输出 3 和 2 的情况。

136、设一个栈的输入序列是 1, 2, 3, 4, 5, 则下列序列中, 是栈的合法输出序列的是 ( )。

A、 5 1 2 3 4

B、 4 5 1 3 2

C、 4 3 1 2 5

D、 3 2 1 5 4

答案: D

解析: 5 输出的时候, 其前面的元素一定都是按照顺序输入或者已经输出了, 因此, 不可能出现 5 输出后 4 再输出的情况。

137、某堆栈的输入序列为 a, b, c, d, 下面的四个序列中, 不可能是它的输出序列的是 ( )。

A、 a, c, b, d

B、 b, c, d, a

C、 c, d, b, a

D、 d, c, a, b

答案： D

解析： d 第一个输出，因此在输入序列中 d 之前的元素一定都在栈中，因此按照后进先出的原则，a 元素不可能在 b 元素之前输出。

138、设 abcdef 以所给的次序进栈，若在进栈操作时，允许退栈操作,则下面得不到的序列为（ ）。

A、 fedcba

B、 bcafed

C、 dcefba

D、 cabdef

答案： D

解析： 因为 c 第一个输出，则按照输入顺序，a, b 一定在栈中，根据后入先出的原则，a 不可能在 b 元素之前输出。

139、设有三个元素 X, Y, Z 顺序进栈（进的过程中允许出栈），下列得不到的出栈排列是( )。

A、 XYZ

B、 YZX

C、 ZXY

D、 ZYX

答案： C

解析： 因为 Z 第一个输出，则按照输入顺序，XY 必然在栈中，根据后入先出的原则，X 不可能在 Y 之前输出。

140、输入序列为 ABC，可以变为 CBA 时，经过的栈操作为（ ）

A、 push, pop, push, pop, push, pop

B、 push, push, push, pop, pop, pop

C、 push, push, pop, pop, push, pop

D、 push, pop, push, push, pop, pop

答案： B

解析：按照后进先出的原则，根据输入序列和输出序列的顺序，只有 B 选项的操作满足条件。

141、若一个栈以向量  $V[1..n]$  存储，初始栈顶指针  $top$  为  $n+1$ ，则下面  $x$  进栈的正确操作是( )。

A、  $top:=top+1; \quad V[top]:=x$

B、  $V[top]:=x; \quad top:=top+1$

C、  $top:=top-1; \quad V[top]:=x$

D、  $V[top]:=x; \quad top:=top-1$

答案： C

解析：栈顶指针初始指向的是数组的最后一个元素的后一位，因此是按照从后往前的顺序进栈和出栈操作的。因此选择 C 选项，先操作栈顶下标，然后将元素存于相应位置。

142、若栈采用顺序存储方式存储，现两栈共享空间  $V[1..m]$ ， $top[i]$  代表第  $i$  个栈 ( $i=1,2$ ) 栈顶，栈 1 的底在  $v[1]$ ，栈 2 的底在  $v[m]$ ，则栈满的条件是 ( )。



A、  $|\text{top}[2] - \text{top}[1]| = 0$

B、  $\text{top}[1] + 1 = \text{top}[2]$

C、  $\text{top}[1] + \text{top}[2] = m$

D、  $\text{top}[1] = \text{top}[2]$

答案： B

解析：由题意得，两个栈共享一个数组，入栈操作都是从两头往中间的方向进行，因此，当符合 B 选项的时候，则代表两个栈所占的存储空间已经占满了整个数组，也意味着栈满。

143、栈在（ ）中应用。

A、 递归调用

B、 子程序调用

C、 表达式求值

D、 A, B, C

答案： D

解析： 栈可以应用于递归，子程序调用以及表达式求值中。

144、一个递归算法必须包括（ ）。

A、 递归部分

B、 终止条件和递归部分

C、 迭代部分

D、 终止条件和迭代部分

答案： B

解析： 一个递归算法必须包括终止条件和递归部分。

145、执行完下列语句段后，i 值为：（ ）

```
int f(int x)
```

```
{ return ((x>0) ? x* f(x-1):2);}
```

```
int i ;
```

```
i =f(f(1));
```

- A、 2
- B、 4
- C、 8
- D、 无限递归

答案： B

解析： 计算的是  $2*2=4$

146、表达式  $a*(b+c)-d$  的后缀表达式是( )

- A、  $abcd*+-$
- B、  $abc+*d-$
- C、  $abc*+d-$
- D、  $-+*abcd$

答案： B

解析： 根据后缀表达式的规则，得到 B 选项是正确答案。

147、表达式  $3*2^{(4+2*2-6*3)}-5$  求值过程中当扫描到 6 时，对象栈和算符栈为( )，其中 ^ 为乘幂。

- A、 3, 2, 4, 1, 1;  $(*^(+*-$

B、 3, 2, 8; (\*^(-

C、 3, 2, 4, 2, 2; (\*^(-

D、 3, 2, 8; (\*^(-

答案： D

解析：根据四则表达式的栈的实现过程，D 选项是正确的。

148、设计一个判别表达式中左，右括号是否配对出现的算法，采用（ ）数据结构最佳。

A、 线性表的顺序存储结构

B、 队列

C、 线性表的链式存储结构

D、 栈

答案： D

解析：参考四则表达式的栈的实现中左右括号的匹配原则。

149、用链接方式存储的队列，在进行删除运算时（ ）。

A、 仅修改头指针

B、 仅修改尾指针

C、 头、尾指针都要修改

D、 头、尾指针可能都要修改

答案： D

解析：队列删除运算时，从队头进行删除操作，但在链队中，当队列中只有一个元素时，删除该元素的操作需要同时修改头尾指针。

150、用不带头结点的单链表存储队列时,其队头指针指向队头结点,其队尾指针指向队尾结点，则在进行删除操作时( )。

A、 仅修改队头指针

B、 仅修改队尾指针

C、 队头、队尾指针都要修改

D、 队头, 队尾指针都可能要修改

答案： D

解析：队列删除运算时，从队头进行删除操作，但在链队中，当队列中只有一个元素时，删除该元素的操作需要同时修改头尾指针。

151、递归过程或函数调用时，处理参数及返回地址，要用一种称为（ ）的数据结构。

A、 队列

B、 多维数组

C、 栈

D、 线性表

答案： C

解析：在递归调用时，调用函数是先执行，执行后才执行本层的函数，因此是一种逻辑上的后调用先执行的顺序，因此采用栈比较合适。

152、假设以数组  $A[m]$  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（ ）。

A、  $(\text{rear}-\text{front}+\text{m})\% \text{m}$

B、  $\text{rear}-\text{front}+1$

C、  $(\text{front}-\text{rear}+\text{m})\% \text{m}$

D、  $(\text{rear}-\text{front})\% \text{m}$

答案： A

解析： 参考循环队列的头尾指针及相应的逻辑结构，A 选项尾正确选项。

153、循环队列 A[0..m-1]存放其元素值，用 **front** 和 **rear** 分别表示队头和队尾，则当前队列中的元素数是( )。

A、  $(\text{rear}-\text{front}+\text{m})\% \text{m}$

B、  $\text{rear}-\text{front}+1$

C、  $\text{rear}-\text{front}-1$

D、  $\text{rear}-\text{front}$

答案： A

解析： 参考循环队列的头尾指针及相应的逻辑结构，A 选项尾正确选项。

154、循环队列存储在数组  $A[0..m]$  中，则入队时的操作为（ ）。

A、  $rear=rear+1$

B、  $rear=(rear+1) \bmod (m-1)$

C、  $rear=(rear+1) \bmod m$

D、  $rear=(rear+1) \bmod (m+1)$

答案： D

解析： 队列的入队操作是在对位进行插入操作，在循环链表中先通过队尾下标找到新元素所插入的位置，因为数组中一共有  $m+1$  个元素，因此是对  $m+1$  取余。

155、若用一个大小为 6 的数组来实现循环队列，且当前 **rear** 和 **front** 的值分别为 0 和 3，当从队列中删除一个元素，再加入两个元素后，**rear** 和 **front** 的值分别为多少？（ ）

A、 1 和 5



B、 2 和 4

C、 4 和 2

D、 5 和 1

答案： B

解析： 参考循环队列中插入元素和删除元素操作中，队头和队尾下标的处理方式，B 为正确选项。

156、已知输入序列为 abcd 经过输出受限的双向队列后能得到的输出序列有（ ）。

A、 dacb

B、 cadb

C、 dbca

D、

以上答案都不对

答案： B

解析：输出受限的双向队列，可以从两头进行插入操作，但智能从一头进行出队操作。根据选项，B 的结果都可以得到。

157、若以 1234 作为双端队列的输入序列，则既不能由输入受限的双端队列得到，也不能由输出受限的双端队列得到的输出序列是( )。

A、 1234

B、 4132

C、 4231

D、 4213

答案： C

解析：参考输入受限和输出受限的双端队列，根据输入顺序逐个选项进行尝试，只有 C 选项得不到。

158、最大容量为  $n$  的循环队列，队尾指针是 **rear**，队头是 **front**，则队空的条件是 ( )。

A、  $(\text{rear}+1) \text{ MOD } n = \text{front}$

B、  $\text{rear} = \text{front}$

C、  $\text{rear}+1 = \text{front}$

D、  $(\text{rear}-1) \text{ MOD } n = \text{front}$

答案： B

解析：循环队列中判断队列为空的方法就是  $\text{rear}=\text{front}$ 。

159、栈和队列的共同点是（ ）。

A、 都是先进先出

B、 都是先进后出

C、 只允许在端点处插入和删除元素

D、 没有共同点

答案： C

解析：参考栈和队列的定义，C 选项正确。

160、栈和队列都是（ ）。

A、 顺序存储的线性结构

B、 链式存储的线性结构

C、 限制存取点的线性结构

D、 限制存取点的非线性结构

答案： C

解析：

栈和队列都是限制存取点的特殊的线性结构。

161、进栈序列为 1, 2, 3, 4 则 ( ) 不可能是一个出栈序列 (不一定全部进栈后再出栈)

A、 3, 2, 1, 4

B、 3, 2, 4, 1

C、 4, 2, 3, 1

D、 4, 3, 2, 1

答案： C

解析： 参考栈和队列的定义及操作特点。

162、设栈 S 和队列 Q 的初始状态为空，元素 e1，e2，e3，e4,e5 和 e6 依次通过栈 S，一个元素出栈后即进队列 Q，若 6 个元素出队的序列是 e2，e4，e3,e6,e5,e1 则栈 S 的容量至少应该是( )。

- A、 6
- B、 4
- C、 3
- D、 2

答案： C

解析： 队列中的出队序列就是栈的出栈顺序，根据栈的出栈顺序，其内最多存储元素的时候应该是在出栈 e4 时，栈内应该存有 e2, e3, e4 三个元素。

163、用单链表表示的链式队列的队头在链表的（ ）位置。

A、 链头

B、 链尾

C、 链中

D、 以上都可以

答案： A

解析： 队列都是在对头进行删除的，而在单链表中为了便于删除操作，都是以链表的表头作为队列的队头的。

164、依次读入数据元素序列 {a, b, c, d, e, f, g} 进栈,每进一个元素, 机器可要求下一个元素进栈或弹栈, 如此进行, 则栈空时弹出的元素构成的序列是以下哪些序列?

A、 {d , e, c, f, b, g, a}

B、 {f, e, g, d, a, c, b}

C、 {e, f, d, g, b, c, a}

D、 以上都不对

答案: A

解析: 根据栈的入栈和出栈的操作顺序的不同, 逐个尝试各个选项, 最终可以实现的是 A 选项。

165、表达式  $a*(b+c)-d$  的后缀表达式是 ( )

A、  $abcd*+-$

B、  $abc+*d-$

C、  $abc*+d-$

D、  $--*abcd$

答案： B

解析： 参考后缀表达式的转换方法。

166、若用大小为 6 的数组来实现循环队列，且当前 **front** 和 **rear** 的值分别为 0 和 4。当从队列中删除两个元素，再加入两个元素后，**front** 和 **rear** 的值分别为多少（ ）

A、 2 和 6

B、 2 和 0

C、 2 和 6

D、 2 和 2

答案： B

解析： 参考循环链表中删除元素的操作。

167、设循环队列中数组的下标范围是  $1 \sim n$ ，其头尾指针分别为 **f** 和 **r**，则其元素个数为（ ）

A、  $r-f$

B、  $r-f+1$

C、  $(r-f)\% \ n+1$

D、  $(r-f+n)\% \ n$

答案： D

解析： 参考循环链表中求元素个数的操作。

168、 设数组  $data[m]$  作为循环队列 SQ 的存储空间，  $front$  为队头指针，  $rear$  为队尾指针， 则执行入队操作后其尾指针  $rear$  值为（ ）

A、  $rear=rear+1$

B、  $rear=(rear-1)\%m$

C、  $rear=(rear+1)\%(m-1)$

D、  $rear=(rear+1)\%m$



答案： D

解析： 参考循环链表中插入元素的操作。

169、消除递归不一定需要使用栈，此说法( )

答案： 正确

解析： 消除递归，也可以采用其他的算法来实现，有时候用循环也可以实现。

170、栈是实现过程和函数等子程序所必需的结构。( )

答案： 正确

解析： 栈是一种函数和子程序的调用过程中的基本结构。

171、两个栈共用静态存储空间，对头使用也存在空间溢出问题。( )

答案： 正确

解析： 只要是静态存储空间，栈都存在栈满和空间溢出问题。

172、两个栈共享一片连续内存空间时，为提高内存利用率，减少溢出机会，应把两个栈的栈底分别设在这片内存空间的两端。( )

答案： 正确

解析：栈底分别设为内存空间的两端，然后两个栈顶分别往中间移动，可以最大化的利用静态内存空间。

173、即使对不含相同元素的同一输入序列进行两组不同的合法的入栈和出栈组合操作，所得的输出序列也一定相同。（        ）

答案： 错误

解析：输入序列相同，但随着进出栈的次序的不同，可以得到不同的输出序列。

174、有  $n$  个数顺序（依次）进栈，出栈序列有  $C_n$  种， $C_n = \frac{1}{n+1} \cdot \frac{(2n)!}{n! \cdot n!}$ 。（        ）

答案： 正确

解析：排列组合的计算，将各种出栈状况考虑在内，采用插入法进行讨论，可以得到以上答案。

175、栈与队列是一种特殊操作的线性表。（        ）

答案： 正确

解析：栈和队列都是控制输入和输出方向的特殊的线性表。

176、若输入序列为 1, 2, 3, 4, 5, 6, 则通过一个栈可以输出序列 3,2,5,6,4,1。（    ）

答案： 正确

解析：按照后进先出的原则，然后在输入顺序中穿插不同的出栈操作，可以得到题中的输出序列。

177、栈和队列都是限制存取点的线性结构。（        ）

答案： 正确

解析：栈和队列都是特殊的线性结构，限制存取点。

178、若输入序列为 1， 2， 3， 4， 5， 6， 则通过一个栈可以输出序列 1， 5， 4， 6， 2， 3。（        ）

答案： 错误

解析：在 5 提前输出的前提下，2， 3， 4 必然是依次进在栈中，因此，不可能在后续的输出中 2 比 3 先输出。

179、任何一个递归过程都可以转换成非递归过程。（    ）

答案： 正确

解析：任何递归过程都可以转换成非递归的方式来实现。

180、只有那种使用了局部变量的递归过程在转换成非递归过程时才必须使用栈。（    ）

答案： 错误

解析：递归转非递归中使用栈的帮助，跟使用没使用局部变量没有关系。

181、队列是一种插入与删除操作分别在表的两端进行的线性表，是一种先进后出型结构。（        ）

答案： 错误

解析：队列是一种先进先出的结构。

182、通常使用队列来处理函数或过程的调用。（        ）

答案： 错误

解析：函数或过程的调用通常使用的是栈为工具。

183、队列逻辑上是一个下端和上端既能增加又能减少的线性表。（        ）

答案： 错误

解析：参考队列的定义。

184、循环队列通常用指针来实现队列的头尾相接。（        ）

答案： 错误

解析：循环队列如果是用数组来实现的话，通常采用的是数组下标的方式来实现。

185、循环队列也存在空间溢出问题。（        ）

答案： 正确

解析：如果循环队列采用数组的方式来实现，就存在空间溢出问题。

186、队列和栈都是运算受限的线性表，只允许在表的两端进行运算。（    ）

答案： 错误

解析：栈只允许在一端进行操作。

187、栈和队列都是线性表，只是在插入和删除时受到了一些限制。  
（        ）

答案： 正确

解析：栈和队列都是特殊的线性表。

188、栈和队列的存储方式，既可以是顺序方式，又可以是链式方式。  
（        ）

答案： 正确

解析： 参考栈和队列的实现方法。

189、 栈是一种对所有插入、删除操作限于在表的一端进行的线性表，是一种后进先出型结构。（        ）

答案： 正确

解析： 参考栈的定义。

190、对于不同的使用者，一个表结构既可以是栈，也可以是队列，也可以是线性表。（        ）

答案： 正确

解析： 正确，都是线性逻辑结构，栈和队列其实是特殊的线性表，对运算的定义略有不同而已。

191、栈和队列是一种非线性数据结构。（        ）

答案： 错误

解析： 错，他们都是线性逻辑结构，栈和队列其实是特殊的线性表，对运算的定义略有不同而已。

192、队是一种插入与删除操作分别在表的两端进行的线性表，是一种先进后出型结构。（        ）

答案： 错误

解析：错，后半句不对。

193、一个栈的输入序列是 12345，则栈的输出序列不可能是 12345。

(        )

答案： 错误

解析：错，有可能。

194、队列逻辑上是一个下端和上端既能增加又能减少的线性表。(        )

答案： 错误

解析：非循环队列中的队头无法增加，队尾无法减少。

195、任何一个递归过程都可以转换成非递归过程。(        )

答案： 正确

解析：递归可通过栈的方式展开。

196、在用单链表表示的链式队列中，队头在链表的链尾位置。(        )

答案： 错误

解析：如果采用头插法，队头在链表的头部位置。

197、若让元素 1, 2, 3 依次进栈，则出栈次序 1, 3, 2 是不可能出现的情况。（ ）

答案： 错误

解析：1 先进栈然后出栈，然后 2, 3 依次进栈，再出栈，就得到 3, 2。

198、在循环队列中，进队时队尾指针进一，出队时队头指针进一。（ ）

答案： 正确

解析：参考循环队列的进栈和出栈的标准操作。

199、设有两个栈 S1, S2 都采用顺序栈方式，并且共享一个存储区[0..maxsize-1],为了尽量利用空间，减少溢出的可能，可采用栈顶相向，迎面增长的存储方式。试设计 S1,S2 有关入栈和出栈的操作算法。

答案：

```
#define maxsize      两栈共享顺序存储空间所能达到的最多元素数
```

```
#define elemtp int      //假设元素类型为整型
```

```
typedef struct
```

```
{elemtp stack[maxsize];    //栈空间
```



```

int top[2];                                //top 为两个栈顶指针

}stk;

stk s;                                    //s 是如上定义的结构类型变量，为全局变量。

```

(1)入栈操作:

```

int push(int i,int x)

//入栈操作。i为栈号，i=0 表示左边的栈 s1，i=1 表示右边的栈 s2，x 是入栈元素。入栈成功返回 1，否则返回 0。

{if(i<0 || i>1) {printf(“栈号输入不对”);exit(0);}

if(s.top[1]-s.top[0]==1) {printf(“栈已满\n”);return(0);}

switch(i)

    {case 0: s.stack[++s.top[0]]=x; return(1); break;

     case 1: s.stack[--s.top[1]]=x; return(1);

    }

}

} //push

```

(2) 退栈操作

```

elemtp pop(int i)

//退栈算法。i代表栈号，i=0 时为 s1 栈，i=1 时为 s2 栈。退栈成功返回退栈元素，否则返回-1。

{if(i<0 || i>1) {printf(“栈号输入错误\n”); exit(0);}

switch(i)

    {case 0: if(s.top[0]==-1) {printf(“栈空\n”); return
(-1); }

```

```

else return(s.stack[s.top[0]--
]);

case 1: if(s.top[1]==maxsize {printf(“栈空\n”);
return(-1);}

else return(s.stack[s.top[1]++]);

}

} //算法结束

```

解析：两栈共享向量空间，将两栈栈底设在向量两端，初始时，s1 栈顶指针为-1，s2 栈顶为 maxsize。两栈顶指针相邻时为栈满。两栈顶相向，迎面增长，栈顶指针指向栈顶元素。

200、设从键盘输入一整数的序列：a<sub>1</sub>, a<sub>2</sub>, a<sub>3</sub>, …, a<sub>n</sub>,试编写算法实现：用栈结构存储输入的整数，当 a<sub>i</sub>≠-1 时，将 a<sub>i</sub> 进栈；当 a<sub>i</sub>=-1 时，输出栈顶整数并出栈。算法应对异常情况（入栈满等）给出相应的信息。

答案：

```

#define maxsize 栈空间容量

void InOutS(int s[maxsize])

//s 是元素为整数的栈，本算法进行入栈和退栈操作。

{int top=0; //top 为栈顶指针，定义
top=0 时为栈空。

for(i=1; i<=n; i++) //n 个整数序列作处理。

{scanf(“%d”, &x); //从键盘读入整数序列。

```

```

        if(x!=-1)                                // 读入的整数不等于-1
时入栈。

        if(top==maxsize-1){printf(“栈满
\n”);exit(0);}else s[++top]=x; //x 入栈。

        else //读入的整数等于-1 时退栈。

        {if(top==0){printf(“栈空\n”);exit(0);}
else printf(“出栈元素是%d\n”,s[top--]); }}

    }//算法结束。

```

解析：标准的入栈和出栈的基本代码，参考栈的基本实现的标准代码。

201、设表达式以字符形式已存入数组 E[n]中，‘#’为表达式的结束符，试写出判断表达式中括号（‘（’和‘）’）是否配对的 C 语言描述算法：  
**EXYX(E);** (注：算法中可调用栈操作的基本算法。)

答案：

```
int EXYX(char E[],int n)
```

//E[]是有 n 字符的字符数组，存放字符串表达式，以‘#’结束。本算法判断表达式中圆括号是否匹配。

```
{char s[30]; //s 是一维数组，容量足够大，用作存放括号的栈。
```

```
int top=0; //top 用作栈顶指针。
```

```
s[top]= ‘#’ ; // ‘#’ 先入栈，用于和表达式结束符号 ‘#’ 匹配。
```

```

int i=0;                                //字符数组 E 的工作指针。

while(E[i]!= '#' )    //逐字符处理字符表达式的数组。

    switch(E[i])

        {case '(' :    s[++top]= '(' ; i++ ;    break  ;

        case ')' :    if(s[top]== '(' {top--; i++; break;}

                                else{printf(“括号不配对”);exit(0);}

        case '#' :    if(s[top]== '#' ){printf(“括号配对
\n”);return  (1);}

                                else {printf(“ 括号不配对\n”);return  (0);} //括号
不配对

        default  :        i++;        //读入其它字符，不作处理。

    }

}

} //算法结束。

```

解析：判断表达式中括号是否匹配，可通过栈，简单说是左括号时进栈，右括号时退栈。退栈时，若栈顶元素是左括号，则新读入的右括号与栈顶左括号就可消去。如此下去，输入表达式结束时，栈为空则正确，否则括号不匹配。

本题是用栈判断括号匹配的特例：只检查圆括号的配对。一般情况是检查花括号（‘{’，‘}’）、方括号（‘[’，‘]’）和圆括号（‘（’，‘）’）的配对问题。编写算法中如遇左括号（‘{’，‘[’，或‘（’）就压入栈中，如遇右括号（‘}’，‘]’，或‘）’），则与栈顶元素比较，如是与其配对的括号（左花括号，左方括号或左圆括号），则弹出栈顶元素；否则，就结论括号不配对。在读入表达式结束符‘#’时，栈中若应只剩‘#’，表示括号全部配对成功；否则表示括号不匹配。

另外，由于本题只是检查括号是否匹配，故对从表达式中读入的不是括号的那些字符，一律未作处理。再有，假设栈容量足够大，因此入栈时未判断溢出。

202、从键盘上输入一个逆波兰表达式，用伪码写出其求值程序。规定：逆波兰表达式的长度不超过一行，以\$符作为输入结束，操作数之间用空格分隔,操作符只可能有+、-、\*、/四种运算。例如：234 34+2\*\$

答案：

```
float  expr( )
```

    //从键盘输入逆波兰表达式，以‘\$’表示输入结束，本算法求逆波兰式表达式的值。

```
    {float  OPND[30];        // OPND 是操作数栈。

    init(OPND);              //两栈初始化。

    float  num=0.0;          //数字初始化。

    scanf ( "%c" ,&x); //x 是字符型变量。

    while(x!=' $' )

        {switch

            {case '0' <=x<=' 9' :while((x>=' 0' && x<=' 9' ) || x=='.' ) //拼数

                if(x!=' . '

                    ) //处理整数

                    {num=num*10+ (ord(x)-ord( '0' )) ;

                    scanf( "%c" ,&x);}

                else

                    //处理小数部分。

                    {scale=10

                    .0; scanf( "%c" ,&x);
```

```

                                while (x>
=' 0' &&x<=' 9' )

                                {nu
m=num+(ord(x)-ord( '0' )/scale;

                                s
cale=scale*10;   scanf( "%c" ,&x); }

                                }//else

                                push(OP
ND, num); num=0.0;//数压入栈，下个数字初始化

                                case  x= ' ' :break;   //遇空格，继续读下一个字符。

                                case  x= '+' :push(OPND, pop(OPND)+pop(OPND));break;

                                case  x= '-'
' :x1=pop(OPND);x2=pop(OPND);push(OPND, x2-x1);break;

                                case  x= '*' :push(OPND, pop(OPND)*pop(OPND));break;

                                case  x= '/' :x1=pop(OPND);x2=pop(OPND);push(OPND, x2
/x1);break;

                                default:                //其它符号不作处理。

                                }//结束 switch

                                scanf( "%c" ,&x);//读入表达式中下一个字符。

                                }//结束 while (x!= '$' )

                                printf( "后缀表达式的值为%f" , pop(OPND));

                                }//算法结束。

```

解析：逆波兰表达式(即后缀表达式)求值规则如下：设立运算数栈 OPND, 对表

达式从左到右扫描(读入)，当表达式中扫描到数时，压入 OPND 栈。当扫描到运算符时，从 OPND 退出两个数，进行相应运算，结果再压入 OPND 栈。这个过程一直进行到读出表达式结束符\$，这时 OPND 栈中只有一个数，就是结果。

假设输入的后缀表达式是正确的，未作错误检查。算法中拼数部分是核心。若遇到大于等于‘0’且小于等于‘9’的字符，认为是数。这种字符的序号减去字符‘0’的序号得出数。对于整数，每读入一个数字字符，前面得到的部分数要乘以 10 再加新读入的数得到新的部分数。当读到小数点，认为数的整数部分已完，要接着处理小数部分。小数部分的数要除以 10（或 10 的幂数）变成十分位，百分位，千分位数等等，与前面部分数相加。在拼数过程中，若遇非数字字符，表示数已拼完，将数压入栈中，并且将变量 num 恢复为 0，准备下一个数。这时对新读入的字符进入‘+’、‘-’、‘\*’、‘/’及空格的判断，因此在结束处理数字字符的 case 后，不能加入 break 语句。

203、假设以 I 和 O 分别表示入栈和出栈操作。栈的初态和终态均为空，入栈和出栈的操作序列可表示为仅由 I 和 O 组成的序列，称可以操作的序列为合法序列，否则称为非法序列。

(1) 下面所示的序列中哪些是合法的？

- A. IOIOIOIO      B. IOOIIOIO      C. IIIIOIOIO  
D. IIIIOOIO

(2) 通过对 (1) 的分析，写出一个算法，判定所给的操作序列是否合法。若合法，返回 true，否则返回 false（假定被判定的操作序列已存入一维数组中）。

答案：

(1) A 和 D 是合法序列，B 和 C 是非法序列。

(2) 设被判定的操作序列已存入一维数组 A 中。

```
int Judge(char A[])
```

```

//判断字符数组 A 中的输入输出序列是否是合法
序列。如是，返回 true，否则返回 false。

{i=0;                                     //i 为下
标。

j=k=0;                                     //j 和 k
分别为 I 和字母 O 的的个数。

while(A[i]!='\0') //当未到字符数组尾就
作。

{switch(A[i])

{case 'I' : j++; break; //入栈次
数增 1。

case 'O' : k++;
if(k>j) {printf(“序列非法\n”); exit(0);}

}

i++; //不论 A[i]是 'I' 或 'O'，指针 i 均后移。}

if(j!=k) {printf(“序列非法\n”);
return(false);}

else {printf(“序列合法\n”);
return(true);}

} //算法结束。

```

解析：在入栈出栈序列（即由 'I' 和 'O' 组成的字符串）的任一位置，入栈次数（'I' 的个数）都必须大于等于出栈次数（即 'O' 的个数），否则视作非法序列，立即给出信息，退出算法。整个序列（即读到字符数组中字符串的结束标记 '\0'），入栈次数必须等于出栈次数（题目中要求栈的初态和终态都为空），否则视为非法序列。



204、设计一个算法，判断一个算术表达式中的括号是否配对。算术表达式保存在带头结点的单循环链表中，每个结点有两个域：**ch** 和 **link**，其中 **ch** 域为字符类型。

答案：

```
int Match(LinkedList la)

//算术表达式存储在以 la 为头结点的单循环链表中，本算法判断括号是否
正确配对

{char   s[];                                //s 为字符栈，容量足够大

p=la->link;                                //p 为工作指针，指向待处理结点

StackInit(s);                              //初始化栈 s

while (p!=la)                               //循环到头结点为止

{switch (p->ch)

{case   '(' :push(s, p->ch); break;

        case   ')' :if(StackEmpty(s) || StackGetTop(s) != '(' )

                    {printf( "括号不配对\n" ); return(0);}

else pop(s);break;

        case   '[' :push(s, p->ch); break;

        case   ']' : if(StackEmpty(s) || StackGetTop(s) != '[' )

                    {printf( "括号不配对\n" ); return(0);}

else pop(s);break;

        case   '{' :push(s, p->ch); break;
```

```

        case '}' : if (StackEmpty(s) || StackGetTop(s) != '{' )
            {printf(“括号不配对\n” ); return(0);}
else pop(s);break;

    } p=p->link; 后移指针

} //while

if (StackEmpty(s)) {printf(“括号配对\n” ); return(1);}

else {printf(“括号不配对\n” ); return(0);}

} //算法 match 结束

```

解析：表达式中的括号有以下三对：‘（’、‘）’、‘[’、‘]’、‘{’、‘}’，使用栈，当为左括号时入栈，右括号时，若栈顶是其对应的左括号，则退栈，若不是其对应的左括号，则结论为括号不配对。当表达式结束，若栈为空，则结论表达式括号配对，否则，结论表达式括号不配对。

算法中对非括号的字符未加讨论。遇到右括号时，若栈空或栈顶元素不是其对应的左圆（方、花）括号，则结论括号不配对，退出运行。最后，若栈不空，仍结论括号不配对。

205、请利用两个栈 S1 和 S2 来模拟一个队列。已知栈的三个运算定义如下：  
**PUSH(ST,x)**:元素 x 入 ST 栈；**POP(ST,x)**: ST 栈顶元素出栈，赋给变量 x；  
**Sempty(ST)**: 判 ST 栈是否为空。那么如何利用栈的运算来实现该队列的三个运算：**enqueue**:插入一个元素入队列； **dequeue**:删除一个元素出队列；  
**queue\_empty**: 判队列为空。（请写明算法的思想及必要的注释）

答案：

```

(1) int enqueue(stack s1,elemtp x)

```

//s1 是容量为 n 的栈，栈中元素类型是 elemtp。本算法将 x 入栈，若入栈成功返回 1，否则返回 0。

```
{if(top1==n && !Empty(s2))           //top1 是栈 s1 的栈顶指针，是全局变量。
```

```
{printf(“栈满”);return(0);} //s1 满 s2 非空，这时 s1 不能再入栈。
```

```
if(top1==n && Empty(s2))           //若 s2 为空，先将 s1 退栈，元素再压栈到 s2。
```

```
{while(!Empty(s1)) {POP(s1,x);PUSH(s2,x);}
```

```
PUSH(s1,x); return(1); //x 入栈，实现了队列元素的入队。
```

```
}
```

(2) void dequeue(stack s2,s1)

//s2 是输出栈，本算法将 s2 栈顶元素退栈，实现队列元素的出队。

```
{if(!Empty(s2))           //栈 s2 不空，则直接出队。
```

```
{POP(s2,x); printf(“出队元素为”,x); }
```

```
else           //处理 s2 空栈。
```

```
if(Empty(s1)) {printf(“队列空”);exit(0);} //若输入栈也为空，则判定队空。
```

```
else           //先将栈 s1 倒入 s2 中，再作出队操作。
```

```
{while(!Empty(s1)) {POP(s1,x);PUSH(s2,x);}
```

```
POP(s2,x);           //s2 退栈相当队列出队。
```

```
printf(“出队元素”，x);
```

```
}
```

```
    } //结束算法 dequeue。
```

```
(3) int queue_empty()
```

```
    //本算法判用栈 s1 和 s2 模拟的队列是否为空。
```

```
    {if (Sempty(s1)&&Sempty(s2))    return(1); //队列空。
```

```
        else return(0);
```

```
        //队列不空。
```

```
    }
```

解析：栈的特点是后进先出，队列的特点是先进先出。所以，用两个栈 s1 和 s2 模拟一个队列时，s1 作输入栈，逐个元素压栈，以此模拟队列元素的入队。当需要出队时，将栈 s1 退栈并逐个压入栈 s2 中，s1 中最先入栈的元素，在 s2 中处于栈顶。s2 退栈，相当于队列的出队，实现了先进先出。显然，只有栈 s2 为空且 s1 也为空，才算是队列空。

算法中假定栈 s1 和栈 s2 容量相同。出队从栈 s2 出，当 s2 为空时，若 s1 不空，则将 s1 倒入 s2 再出栈。入队在 s1，当 s1 满后，若 s2 空，则将 s1 倒入 s2，之后再入队。因此队列的容量为两栈容量之和。元素从栈 s1 倒入 s2，必须在 s2 空的情况下才能进行，即在要求出队操作时，若 s2 空，则不论 s1 元素多少（只要不空），就要全部倒入 s2 中。

206、如果允许在循环队列的两端都可以进行插入和删除操作。要求：

(1) 写出循环队列的类型定义；

(2) 写出“从队尾删除”和“从队头插入”的算法。

答案：

(1) #define M 队列可能达到的最大长度

```

typedef struct

{
    elemtp data[M];

    int front, rear;

} cycqueue;

```

(2) elemtp delqueue ( cycqueue Q)

//Q 是如上定义的循环队列，本算法实现从队尾删除，若删除成功，返回被删除元素，否则给出出错信息。

```

{ if (Q.front==Q.rear) {printf(“队列空”); exit(0);}

```

```

    Q.rear=(Q.rear-1+M)%M; //修改队尾指针。

```

```

    return(Q.data[(Q.rear+1+M)%M]); //返回出队元素。

```

```

} //从队尾删除算法结束

```

```

void enqueue (cycqueue Q, elemtp x)

```

// Q 是顺序存储的循环队列，本算法实现“从队头插入”元素 x。

```

{if (Q.rear==(Q.front-1+M)%M) {printf(“队满”); exit(0);}

```

```

    Q.data[Q.front]=x; //x 入队列

```

```

    Q.front=(Q.front-1+M)%M; //修改队头指针。

```

```

} // 结束从队头插入算法。

```

解析：用一维数组  $v[0..M-1]$  实现循环队列，其中  $M$  是队列长度。设队头指针  $front$  和队尾指针  $rear$ ，约定  $front$  指向队头元素的前一位置， $rear$  指向队尾元素。定义  $front=rear$  时为队空， $(rear+1)\%m=front$  为队满。约定队头端入队向下标小的方向发展，队尾端入队向下标大的方向发展。

207、在一个循环链队中只有尾指针（记为 rear，结点结构为数据域 data，指针域 next），请给出这种队列的入队和出队操作的实现过程。

答案：

(1) void EnQueue (LinkedList rear, ElemType x)

// rear 是带头结点的循环链队列的尾指针，本算法将元素 x 插入到队尾。

```
{ s= (LinkedList) malloc (sizeof(LNode)); //申请结点空间
```

```
    s->data=x;    s->next=rear->next;           //将 s 结点  
链入队尾
```

```
    rear->  
>next=s;    rear=s;           //rear 指向新队  
尾
```

```
}
```

(2) void DeQueue (LinkedList rear)

// rear 是带头结点的循环链队列的尾指针，本算法执行出队操作，操作成功输出队头元素；否则给出出错信息。

```
{ if (rear->next==rear) { printf(“队空\n”);  
exit(0);} 
```

```
    s=rear->next->next;           //s  
指向队头元素，
```

```
    rear->next->next=s->next;       //队头元素  
出队。
```

```
    printf ( “出队元素是” , s->data);
```

```
    if (s==rear) rear=rear->next;    //空队列
```

```
    free(s);
```

```
}
```

解析：在循环链队中，一个尾指针就能够找到队列的头指针，其他的实现同链队的基本操作实现一致。

208、要求循环队列不损失一个空间全部都能得到利用，设置一个标志 tag, 以 tag 为 0 或 1 来区分头尾指针相同时的队列状态的空与满，请编写与此相应的入队与出队算法。

答案：

入队算法：

```
int EnterQueue(SeqQueue *Q, QueueElementType x) { /*
将元素 x 入队*/

    if(Q->front==Q->front && tag==1) /*队满*/
return(FALSE);

    if(Q->front==Q->front && tag==0) /*x 入队前队空，x 入
队后重新设置标志*/

        tag=1;

    Q->elememt[Q->rear]=x;

    Q->rear=(Q->rear+1)%MAXSIZE; /*设置队尾指针*/
    Return(TRUE);

}
```

出队算法：

```

int DeleteQueue( SeqQueue *Q , QueueElementType *x)
{ /*删除队头元素，用 x 返回其值*/

if(Q->front==Q->rear && tag==0) /*队空*/
return(FALSE);

*x=Q->element[Q->front];

Q->front=(Q->front+1)%MAXSIZE; /*重新设置队头指针*/

if(Q->front==Q->rear) tag=0; /*队头元素出队后队列为空，重新设置标志域*/

Return(TUUE); }

```

解析： 参考队列的入队和出队的标准代码，在其中判断队列为空和队列满的条件成立的时候给 tag 变量进行相应的赋值。

209、 下面说法不正确的是（ ）。

A、

广义表的表头总是一个广义表

B、

广义表的表尾总是一个广义表

C、

广义表难以用顺序存储结构



D、

广义表可以是一个多层次的结构

答案： A

解析：广义表是线性表的推广，在它的定义中，表头既可以是单个元素，也可以是广义表，而表尾总是一个广义表，所以 A 错，B 对。由于广义表中数据元素可以有不同的结构（或原子，或列表），因此难以用顺序存储结构存储。而且广义表中也可以有列表，是一种典型的层次结构。

210、下面哪一项不可以作为广义表的存储结构（ ）。

A、 头尾链表的存储结构

B、

扩展线性链表的存储结构

C、 顺序存储结构

D、 都不可以

答案： C

解析：

由于广义表中数据元素可以有不同的结构（或原子，或列表），因此难以用顺序存储结构存储。通常可以用链式存储结构存储广义表，常见的链式存储结构有两种，一种是头尾链表的存储结构，另一种是扩展线性链表的存储结构。

211、广义表  $A=(a, b, (c, d), (e, (f, g)))$ ，则  $\text{Head}(\text{Tail}(\text{Head}(\text{Tail}(\text{Tail}(A))))$  的值为（ ）。

A、 (g)

B、 (d)

C、 c

D、 d

答案： D

解析：

$\text{Tail}(A) = b, (c, d), (e, (f, g))$ ;

$\text{Tail}(\text{Tail}(A)) = (c, d), (e, (f, g))$ ;

$\text{Head}(\text{Tail}(\text{Tail}(A))) = (c, d)$ ;

$\text{Tail}(\text{Head}(\text{Tail}(\text{Tail}(A)))) = (d)$ ;

$\text{Head}(\text{Tail}(\text{Head}(\text{Tail}(\text{Tail}(A)))) = d$ ;

212、广义表  $((a, b, c, d))$  的表头和表尾分别是（ ）。

A、

表头为 a，表尾为 d

B、

表头为 ()，表尾为 d

C、

表头为 a，表尾为()

D、

表头为 (a, b, c, d)，表尾为()

答案： D

解析：表头为非空广义表的第一个元素，可以是一个单原子，也可以是一个子表，所以根据题意可以选择 D。

213、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

答案： C

解析： 广义表的长度是指广义表中所含元素的个数， 广义表的深度是指广义表中展开后所含括号的层数， 因此选 C。

214、 广义表是线性表的推广， 是一类线性数据结构。

答案： 错误

解析： 广义表是线性表的推广， 但它是非线性的数据结构。

215、 广义表(((a, b, (), c), d), e, ((f), g))的长度是 1， 深度是 5。

答案： 错误

解析： 广义表长度是数第一层括号内的元素个数， 可见有一个元素 ((a, b, (), c), d), e, ((f), g)， 广义表的深度是括号的数目， 深度为 4。

216、 对任意一个非空的广义表， 其表头可能是单元素， 也可能是广义表。

答案： 正确

解析： 如 (a, b) 的表头为 a， ((a), b) 的表头为 (a)， 而 a 是元素， (a) 是广义表。

217、已知广义表  $L=(a, (b, (c, (d))), e), f)$ ， $L1=Tail(L)=((b,(c,(d))), e), f)$ 。

答案： 正确

解析：Tail(L) 为对广义表取表尾的操作，正确。

218、表尾一定是广义表。

答案： 正确

解析：表尾是由除了表头以外的其余元素组成的广义表，所以，需要在表尾的直接元素外面再加一层括号，所谓表尾必定是广义表

219、树中的所有节点都有前驱结点和后继结点。

答案： 错误

解析：树中根节点无前驱结点，叶节点无后继结点。

220、对于一棵具有  $n$  个结点、度为 4 的树来说，至少在某一层上正好有四个结点。

答案： 错误

解析：度为 4 只能说明某个节点最多有四个子结点

221、高度为  $h$  的  $m$  叉树至多有  $(m^h-1)/(m-1)$  个结点。

答案： 错误

解析：当树为全满时，高度为  $h$  的  $m$  叉树至多有  $(m^h - 1)/(m - 1)$  个结点(等比数列求和即可推导出最多的结点个数)

222、在一棵度为 4 的树中，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 81。

答案： 错误

解析：由结点总数=分支数+1 可推导出： $N_0 + N_1 + N_2 + N_3 + N_4 = 1 * N_1 + 2 * N_2 + 3 * N_3 + 4 * N_4 + 1$ ，依题意知： $1 * N_1 + 2 * N_2 + 3 * N_3 + 4 * N_4 = 122$ ， $N_0 + N_1 + N_2 + N_3 + N_4 = 41$ ，即可得  $N_0 = 82$ ，故错误。

223、具有  $n$  个结点的  $m$  叉树的最小高度为  $\lceil \log_m(n(m-1)+1) \rceil$ 。

答案： 错误

解析：具有  $n$  个结点的  $m$  叉树的最小高度应该为  $\lceil \log_m(n(m-1)+1) \rceil$

224、度为 2 的有序树就是二叉树。

答案： 错误

解析：二叉树是有序树，而有序树并不一定是二叉树，若有序树只有一个孩子结点，则这个孩子结点就无需区分其左右次序。

225、非空二叉树上叶子结点数等于度为 2 的结点数加 1，即  $N=N_2+1$ 。

答案： 正确

解析：结点总数  $N = N_0 + N_1 + N_2 = \text{分支总数} + 1 = N_1 + 2 * N_2 + 1$ ，则  $N = N_2 + 1$

226、假设一棵二叉树的结点个数为 50，则它的最小高度是 6。

答案： 正确

解析：第一层最多 1 个结点，第二层则最多 2 个结点，第三层最多  $2^2$  个结点，以此类推可以得到  $h$  最少为 6。

227、已知一棵完全二叉树的第 6 层（设根为第 1 层）有 8 个叶结点，则该完全二叉树的结点个数最少是 39。

答案： 正确

解析：要使完全二叉树结点个数最少，即前面 5 层全满而第 6 层只有 8 个结点，故结点最少为  $(2^5 - 1) + 8 = 39$ 。

228、已知一棵完全二叉树的第 6 层（设根为第 1 层）有 8 个叶结点，则该完全二叉树的结点个数最多是 110。

答案： 错误

解析：要使结点的个数最多，则必有第 6 层结点全满，又因为此树为二叉树且

第 6 层有 8 个叶结点，根据二叉树的性质可知：其结点的个数可能为 111、110 以及 39，111 的推导： $2^7-1-(8*2)=111$ ；而 110 的情况为第 6 层 8 个叶结点的相邻的结点其子结点只有一个，即： $2^7-1-(8*2+1)=110$ ；而 39 为结点最少的个数。

229、设二叉树有  $2n$  个结点，且  $m < n$ ，不可能存在  $2m$  个度为 1 的结点

答案： 正确

解析：由二叉树的性质可知  $n_0 = n_2 + 1$ ，结点总数  $= 2n = n_0 + n_1 + n_2 = n_1 + 2 * n_2 + 1$ ，于是有  $n_1 = 2 * (n - n_2) - 1$ ，所以  $n_1$  为奇数，说明该二叉树中不可能有  $2m$  个度为 1 的结点。

230、若一棵深度为 6 的完全二叉树的第 6 层有 3 个叶结点，则该二叉树共有 16 个叶子结点。

答案： 错误

解析：因为该树是完全二叉树，所以第 5 层全满有  $2^4=16$  个结点，其中 2 个结点有子结点，所以共计有  $16-2+3=17$  个叶子结点。

231、完全二叉树不适合顺序存储结构，只有满二叉树适合顺序存储结构。

答案： 错误

解析：完全二叉树和满二叉树均可以采用顺序存储结构。

232、在含有  $n$  个结点的二叉链表中含有  $n-1$  个空链域。



答案： 错误

解析：非空指针数=总分支数= $n-1$ ，空指针数= $2 \times \text{结点数} - \text{非空指针数} = 2n - (n-1) = n+1$ 。

233、在二叉树中有两个结点  $m$ ， $n$ ，如果  $m$  是  $n$  的祖先，使用后序遍历可以找到从  $m$  到  $n$  的路径。

答案： 正确

解析：在后序遍历退回时访问根结点，就可以把从下到上把从  $n$  到  $m$  的路径上的结点输出。

234、一棵非空的二叉树的先序遍历序列与后序遍历序列正好相反，则该二叉树一定满足只有一个子结点。

答案： 正确

解析：非空树的先序序列与后序序列相反，因此树只有根结点，或者根节点只有左子树或右子树，依次类推，其子树有同样的性质。可知树中所有的非叶结点的度均为 1，即二叉树仅有一个叶结点。

235、线索二叉树是一种逻辑和存储结构。

答案： 错误

解析：二叉树是一种逻辑结构，但线索二叉树是加上线索后的链表结构，也就是说它是二叉树在计算机内部的一种存储结构，所以是一种物理结构

236、中序线索化二叉树的遍历不需要借助栈。

答案： 正确

解析：中序线索化二叉树主要是为访问运算服务的，这种遍历不再需要栈，因为它的结点中隐含了线索二叉树的前驱和后继信息。

237、若森林 F 有 15 条边，25 个结点，则 F 包含树的个数是 10。

答案： 正确

解析：树有一个很重要的性质：在  $n$  个结点的树中有  $n-1$  条边，那么对于每棵树其结点数比边数多 1。题中的森林中的结点数比边数多 10，显然共有 10 棵树。

238、设霍夫曼编码的长度不超过 4，若已对两个字符编码为 1 和 01，则还最多可对 4 个字符编码。

答案： 正确

解析：在霍夫曼编码中，一个编码不能是任何其他编码的前缀。3 位编码可能是 001，对应的 4 位编码只能是 0000 和 0001。3 位编码也可能是 000，对应的 4 位编码只能是 0010 和 0011。若全采用 4 位编码，可以为 0000、0001、0010 和 0011，所以还最多可对 4 个字符编码。

239、树最适合用来表示（ ）的数据。

A、

有序

B、

无序

C、

任意元素之间具有多种联系

D、

元素之间具有分支层次关系

答案： D

解析：树是一种分层结构，它特别适合组织那些具有分支层次关系的数据。

240、 以下哪种方式不属于树的逻辑表示方法（ ）。

A、

树形表示法

B、

文氏图表示法

C、

链式表示法

D、

凹入表示法

答案： C

解析：树的逻辑表示法常见有四种，分别为树形表示法、文氏图表示法、括号表示法、凹入表示法。

241、树的路径长度是从树根到每一结点的路径长度的（ ）。

A、

总和

B、

最小值

C、

最大值

D、

平均值

答案： A

解析：

树的路径长度是所有路径长度的总和，树根到每一结点的路径的最大值应该是树的高度减 1（注意与哈夫曼树的带权路径长度的区别）。

242、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

答案： B

解析：设树中度为  $i$  ( $i=0, 1, 2, 3, 4$ ) 的结点分别为  $N_i$ ，树中结点总数为  $N$ ，则  $N = \text{分支数} + 1$ ，而分支数又等于树中各结点的度数之和，即  $N = 1 + N_1 + 2N_2 + 3N_3 + 4N_4 = N_0 + N_1 + N_2 + N_3 + N_4$ 。代入数据得  $N_0 = 82$ 。即树 T 的叶结点个数为 82。

243、 一般树的三种基本遍历方法不包括（ ）。

A、

先根遍历

B、

中根遍历

C、

后根遍历

D、

层次遍历

答案： B

解析：一般树的三种基本遍历方法包括先根遍历，后根遍历和层次遍历。

244、如果在树的孩子兄弟链存储结构中有 6 个空的左指针域，7 个空的右指针域，5 个结点左、右指针域都为空，则该树叶子结点有（ ）个。

- A、 7
- B、 6
- C、 5
- D、 不能确定

答案： B

解析：在树的孩子兄弟链存储结构中，左指针域指向第一个孩子结点，右指针域指向右兄弟结点，该树有 6 个空的左指针域，说明有 6 个结点没有任何孩子，则为叶子结点。故选 B。

245、 下列关于二叉树的说法，正确的是（ ）。

A、

度为 2 的有序树就是二叉树

B、

含有 N 个结点的二叉树其高度为  $\lceil \log_2 N \rceil + 1$

C、

在完全二叉树中，若一个结点没有左孩子，则它必是叶结点

D、

在任意一棵非空二叉排序树中，删除某结点后又将其插入，则所得二叉排序树与删除前原二叉排序树相同

答案： C

解析：二叉树是有序树，在二叉树中，如果某一个结点只有一个孩子结点，这个孩子结点的左右次序是确定的；而在度为 2 的有序树中，如果某个结点只有一个孩子结点，这个孩子结点就无需区分其左右次序，因此度为 2 的有序树不是二叉树，A 错。选项 B 仅当是完全二叉树才有意义。在二叉排序树中插入结点时一定是插入在叶结点的位置，故如果先删除分支结点再插入，会导致二叉排序树重构，则 D 错。根据完全二叉树的定义，在完全二叉树中，如果有度为 1 的结点，只可能有一个，且该结点只有左孩子而没有右孩子。

246、 以下说法中，正确的是（ ）。

A、

在完全二叉树中，叶子结点的双亲的左兄弟（如果存在）一定不是叶子结点。

B、

任何一棵二叉树，叶子结点个数为度为 2 的结点数减 1，即  $N_0 = N_2 - 1$

C、

完全二叉树不适合顺序存储结构，只有满二叉树适合顺序存储结构

D、

结点按完全二叉树层序编号的二叉树中，第  $i$  个结点的左孩子的编号为  $2i$

答案： A

解析：在完全二叉树中，叶子结点的双亲左兄弟的孩子一定在其前面（且一定存在），故双亲的左兄弟（如果存在）一定不是叶结点，A 正确。 $N_0$  应该等于  $N_2 + 1$ ，B 错误。完全二叉树和满二叉树均可以采用顺序存储结构，C 错误。第  $i$  个结点的左孩子不一定存在，D 错误。

247、已知一棵完全二叉树的第 6 层（设根为第 1 层）有 8 个叶结点，则该完全二叉树的结点个数最多是（ ）。

A、 39

B、 52

C、 111

D、 119



答案： C

解析：第 6 层有叶结点则完全二叉树的高度可能为 6 或 7，显然是树高为 7 时结点最多。完全二叉树相比满二叉树只是在最下一层的右边缺少部分叶结点，而最后一层之上是个满二叉树，并且只有最后两层有叶结点。如果第 6 层上有 8 个叶结点，则前 6 层为满二叉树，而第 7 层缺失了  $8 \times 2 = 16$  个叶结点，故完全二叉树的结点的个数最多为  $2^7 - 1 - 16 = 111$  个结点。

248、若一棵完全二叉树有 768 个结点，则该二叉树的叶结点的个数是（ ）。

- A、 257
- B、 258
- C、 384
- D、 385

答案： C

解析：显然  $N = N_0 + N_1 + N_2 = N_0 + N_1 + (N_0 - 1) = 2N_0 + N_1 - 1$ ，其中  $N = 768$ ，而在完全二叉树中， $N_1$  只能取 0 或者 1，当  $N_1 = 0$  时，不满足题意，所以  $N_1 = 1$ ，故  $N_0 = 384$ 。

249、 二叉树若用顺序方法存储，则下列 4 种运算中的（ ）最容易实现。

A、

先序遍历二叉树

B、

判断两个指定结点是不是在同一层上

C、 层次遍历二叉树

D、 根据结点的值查找其存储位置

答案： C

解析：

直接顺序扫描存储二叉树的数组即得到层次遍历二叉树序列。

250、对于一棵满二叉树，共有  $n$  个结点和  $m$  个叶子结点，高度为  $h$ ，则（ ）。

A、  $n=h+m$

B、  $h+m=2h$

C、  $m=h-1$

D、  $n=2^h-1$

答案： D

解析：对于高度为  $h$  的满二叉树， $n=2^0+2^1+\cdots+2^{h-1}=2^h-1$ ， $m=2^{h-1}$ ，故选 D。

251、若一棵二叉树的前序遍历序列和后序遍历序列分别为 1、2、3、4 和 4、3、2、1，则该二叉树的中序遍历序列不会是（ ）。

A、 1、2、3、4

B、 2、3、4、1

C、 3、2、4、1

D、 4、3、2、1

答案： C

解析：前序序列为 LRN，后序序列为 NLR，由于前序序列和后序序列正好相反，故不可能存在一个结点同时有左右孩子，即二叉树的高度为 4。1 为根节点，由于根节点只能有左孩子（或右孩子），因此，在中序序列中，1 或在序列首或在序列尾，ABCD 均满足要求。仅考虑以 1 的孩子结点 2 为根节点的子树，它也只能有左孩子（或右孩子），因此，在中序序列中，2 或在序列首或在序列尾，则 ABD 满足要求，故选 C。

252、 若一棵二叉树的前序遍历序列为 a、e、b、d、c，后序遍历序列为 b、c、d、e、a，则根结点的孩子结点（ ）。

A、

只有 e

B、

有 e、b

C、

有 e、c

D、 无法确定

答案： A

解析：

前序序列和后序序列不能唯一确定一棵二叉树，但是可以确定二叉树中结点的祖先关系。可以用排除法，显然 a 为根结点，且确定 e 为 a 的孩子结点，排除 D。各种遍历算法中左右子树的遍历次序是固定的，若 b 也为 a 的孩子结点，则在前序序列和后序序列中 e、b 的相对次序是不变的，故排除 B，同理可以排除 C。

253、线索二叉树是一种（ ）结构。

A、

逻辑

B、

逻辑和存储

C、

物理

D、

线性

答案： C

解析： 二叉树是一种逻辑结构，但是线索二叉树是一种加上线索后的链表结构，也就是一种物理结构。

254、在线索二叉树中，下列说法不正确的是（ ）。

A、

在中序线索树中，若某结点有右孩子，则其后继结点是它的右子树的最左下结点

B、

在中序线索树中，若某结点有左孩子，则其前驱结点是它的左子树的最右下结点

C、

线索二叉树是利用二叉树的  $n+1$  个空指针来存放结点的前驱和后继信息的

D、

每个结点通过线索都可以直接找到它们的前驱和后继

答案： D

解析：不是每个结点都通过线索可以直接找到它们的前驱和后继。在先序线索二叉树中查找一个结点的先序后继很简单，而查找先序前驱必须知道该结点的双亲结点。同样，在后序线索二叉树中查找一个结点的后序前驱也很简单，而查找后序后继也必须知道该结点的双亲结点，二叉链表中没有存放双亲的指针。

255、将森林 F 转换为对应的二叉树 T，F 中叶结点的个数等于（ ）。

A、 T 中叶结点的个数

B、 T 中 degree 为 1 的结点个数

C、 T 中左孩子指针为空的结点个数

D、 T 中右孩子指针为空的结点个数

答案： C

解析：将森林转换为二叉树即相当于用孩子兄弟表示法表示森林。在变化过程中，原森林某结点的第一个孩子结点作为它的左子树，它的兄弟作为它的右子树。那么森林中的叶结点由于没有孩子结点，转换为二叉树时，该结点就没有左结点，所以 F 中叶结点的个数就等于 T 中左孩子指针为空的结点个数，选 C。此题还可以通过一些特例来排除 ABD。

256、对  $n$  ( $n \geq 2$ ) 个权值不相同的字符构成的哈夫曼树，关于该树的叙述中，错误的是（ ）。

- A、 该树一定是一个完全二叉树
- B、 树中一定没有度为 1 的结点
- C、 树中两个权值最小的结点一定是兄弟结点
- D、 树中任一非叶结点的权值一定不小于下一层任一结点的权值

答案： A

解析：哈夫曼树为带权路径长度最小的 二叉树，不一定是完全二叉树。哈夫曼树没有度为 1 的结点，B 正确。构造哈夫曼树时，最先选取两个权值为最小的结点作为左、右子树构造一棵新的二叉树，C 正确。哈夫曼树任一非叶结点 P 的权值为其左、右子树结点权值之和，其权值不小于其左、右子树根节点的权值，可知哈夫曼树任一非叶结点的权值一定不小于下一层任一结点的权值，D 正确。

257、 下列编码中，（ ）不是前缀码。

- A、 {00, 01, 10, 11}
- B、 {0, 1, 00, 11}

C、 {0, 10, 110, 111}

D、 {10, 110, 1110, 1111}

答案： B



解析： 如果没有一个编码是另外一个编码的前缀，称这样的编码为前缀编码。B 选项中，0 是 00 的前缀，1 是 11 的前缀。

258、含有  $n$  个结点的 3 叉树的最小高度是多少？

答案：



解析： 要求含有  $n$  个结点的 3 叉树的最小高度，那么满足条件的一定是一颗完全 3 叉树，设含有  $n$  个结点的完全 3 叉树的高度为  $h$ ，第  $h$  层至少有一个结点，

至多有  个结点。则有： 



由于  $h$  只能为正整数， ，故这样的 3 叉树最小高度是 



259、已知一颗度为 4 的树中，其度为 0、1、2、3 的结点数分别为 14、4、3、2，求该树的结点总数  $n$  和度为 4 的结点个数。

答案：

该树结点总数为 25，度为 4 的结点个数为 2。

解析：设树中度为  $i$  ( $i=0、1、2、3、4$ ) 的结点数为  $n_i$ ，则结点总数为  $N = \sum_{i=0}^4 n_i$ ，即  $N = n_0 + n_1 + n_2 + n_3 + n_4$ ，根据“树中所有结点的度数加 1 等于结点数”的结论，有  $N = n_0 + 2n_1 + 3n_2 + 4n_3 + 5n_4$ 。综合两式得： $N = 25$ 。所以，该树结点总数为 25，度为 4 的结点个数为 2。

260、已知一颗度为  $m$  的树中，有  $n_1$  个度为 1 的结点，有  $n_2$  个度为 2 的结点，...，有  $n_m$  个度为  $m$  的结点，问该树有多少个叶子结点？

答案：




解析：根据“树中所有结点的度数加 1 等于结点数”的结论，有  $N = n_0 + 2n_1 + 3n_2 + \dots + (m+1)n_m$ ，


又有  $N = n_0 + n_1 + n_2 + \dots + n_m$ ，



所以  $n_0 = m n_m$ 。




261、在一棵完全二叉树中含有  $N_0$  个叶子结点，当度为 1 的结点数为 1 时，该树的高度是多少？当度为 1 的结点数为 0 时，该树的高度是多少？




答案：

当度为 1 的结点数为 1 时，该树的高度是 ;

当度为 1 的结点数为 0 时，该树的高度是 .

解析：在非空的二叉树中，由度为 0 和度为 2 的结点之间的关系 ，可知总结点数 .

当  时，，,

当  时，，








262、一棵有  $n$  个结点的满二叉树有多少个分支结点和多少个叶子结点？该满二叉树的高度是多少？

答案：

$(n-1)/2$  个分支结点

$(n+1)/2$  个叶子结点





高度为 

解析：满二叉树中 ，由二叉树性质可知，，，则 。分支结点个数 。高度为 h 的满二叉树的结点个数 ，即高度 。

263、已知完全二叉树的第 9 层有 240 个结点，则整个完全二叉树有多少个结点？有多少个叶子结点？

答案：




整个完全二叉树有 495 个结点，248 个叶子结点

解析：在完全二叉树中，若第 9 层是满的，则结点数为 =256 个，而现在第 9 层只有 240 个结点，说明第 9 层未滿，是最后一层。其 1~8 层是满的，所以总的结点数为  个。因为第 9 层是最后一层，所以第 9 层的结点都是叶子结点。且第 9 层的 240 个结点的的双亲在第 8 层中，其双亲个数为 120，即第 8 层有 120 个分支结点，其余为叶子结点，所以第 8 层叶子结点数为 。因此，总的叶子结点个数为  个。


264、已知二叉树有 50 个叶子结点，则该二叉树的总结点数至少应有多少个？

答案：

至少应有 99 个

解析：设度为 0、1、2 的结点数及总结点数分别为 、、 和 n，则有：





由以上三式可得：.

故, 所以当时,  $n$  最少为 99。

265、具有  $n$  个结点的满二叉树的叶子结点的个数是多少？

答案：

$$(n+1)/2$$

解析：设该满二叉树高度为  $h$ ，则总结点数为：, 叶子结点数为.

266、一棵有  $n$  个结点的满二叉树有多少个度为 1 的结点，有多少个分支结点（非叶结点）和多少个叶子结点，该满二叉树的深度为多少？

答案：

一棵有  $n$  个结点的满二叉树有 0 个度为 1 的结点，有  $(n-1)/2$  个分支结点和  $(n+1)/2$  个叶子结点，该满二叉树的深度为  $\log_2(n+1)$ 。

解析：根据满二叉树的性质可知，一棵有  $n$  个结点的满二叉树有 0 个度为 1 的

结点，有 $(n-1)/2$ 个分支结点和 $(n+1)/2$ 个叶子结点，该满二叉树的深度为



267、10 已知一棵完全二叉树共有 892 个结点，试求：

(1) 树的高度； (2) 叶子结点数； (3) 单支结点数； (4) 最后一个非叶结点的序号。


答案：










(1) 10

(2) 446


(3) 1

(4) 446

解析：(1) 已知深度为  $k$  的二叉树至多有  个结点，由于 ，所以树的高度为 10。

(2) 对完全二叉树来说，度为 1 的结点个数只能是 0 或 1。由  和  得：  
① 设 ，则 ，因  不为整数故舍去。② 设 ，则 ，带入  得 ，故叶子结点数为 446。

(3) 由 (2) 知单支结点数为 1。

(4) 对有  $n$  个结点的完全二叉树，最后一个节点序号为  $n$ ，其双亲节点即为最后一个非叶结点，序号为 .

268、已知一棵完全二叉树的第 8 层有 8 个结点，则其叶子结点数是多少？

答案：

68

解析：完全二叉树除最后一层外，其他各层结点是满的。显然这里第 8 层是最后一层，那么第 7 层节点数为 64 个，其中 4 个结点有 8 个叶子结点，余下的 60 个为叶子结点。所以该完全二叉树的叶子结点数为  $60+8=68$  个。

269、用一维数组存放一棵完全二叉树：ABCDEFGH IJ KL。写出该二叉树的后序序列。

答案：

后序序列为 H I D J K E B L F G C A。

解析：

270、写出该二叉树的先序序列、中序序列和后序序列。



答案：

先序序列：A B D C E F，中序序列：D B A E C F，后序序列：D B E F C A

解析：

先序序列：ABDCEF，中序序列：DBAECF，后序序列：DBEFCA

271、若某非空二叉树的先序遍历序列和后序遍历序列正好相同，则该二叉树的形态是什么？若先序遍历序列和后序遍历序列正好相反，则该二叉树的形态是什么？

答案：

若相同，则二叉树只有一个根结点；

相反，则二叉树每层只有一个结点，即高度等于结点个数

解析：二叉树的先序序列使 NLR，后序序列使 LRN。要使  $NLR=LRN$  成立，则 L 和 R 均为空，所以满足条件的二叉树只有一个根结点。要使  $NLR=NRL$ （后序序列倒序）成立，则 L 或 R 为空，这样的二叉树每层只有一个结点，即高度等于结点个数。

272、已知某二叉树的先序序列为 ABDEHCFIMGJKL，中序序列为 DBHEAIMFCGKLJ，请画出这棵二叉树。并画出二叉树对应的森林。

答案：



解析：把二叉树转换到树和森林的方式是，若结点 x 是双亲 y 的左孩子，则把 x 的右孩子，右孩子的右孩子……都与 y 用线连起来，最后去掉所有双亲到右孩子的连线。最后得到的二叉树及对应的森林如下图所示。



273、一棵二叉树的先序、中序和后序序列分别如下，其中一部分未显示出来。试求出空格处的内容，并画出该二叉树。

先序序列：\_\_B\_\_F\_\_ICEH\_\_G

中序序列：D\_\_KFIA\_\_EJC\_\_

后序序列：K\_\_FBHJ\_\_G\_\_A

答案：

该二叉树如下图所示。先序序列为：ABDFKICEHJG；中序序列为：DBKFIAHEJCG；后序序列为：DKIFBHJEGCA。



解析：该二叉树如下图所示。先序序列为：ABDFKICEHJG；中序序列为：DBKFIAHEJCG；后序序列为：DKIFBHJEGCA。



274、将下面一个由 3 棵树组成的森林转换为二叉树。



答案：



解析：根据“左孩子右兄弟”的转换规则，将森林转换为二叉树的过程：①将每棵树的根结点看成兄弟关系，在兄弟之间连一条线。②对每个阶段，只保留它与第一个子结点的连线，其他子节的连线全部抹掉。③以根为中心旋转 45°。







275、如果一棵哈弗曼树有  $110$  个叶子结点，那么该哈弗曼树有多少个结点？

答案：



解析：哈弗曼树中只有度为 2 和 0 的结点，由非空二叉树的性质可知 ，则总结点数 。

276、给定权集  $w = \{5, 7, 2, 3, 6, 8, 9\}$ ，试构造关于  $w$  的一颗哈弗曼树，并求其加权路径长度 WPL。

答案：



$$WPL = (2+3) \times 4 + (5+6+7) \times 3 + (8+9) \times 2 = 108$$

解析：根据哈弗曼树的构造方法，每次从森林中选取两个根结点值最小的树合并成一棵树，将原先的两棵树作为左、右子树，且新根结点的值为左、右孩子关键字之和。构造出的哈弗曼树如下：



$$WPL = (2+3) \times 4 + (5+6+7) \times 3 + (8+9) \times 2 = 108。$$

注意：哈弗曼树并不唯一，但带权路径长度一定是相同的。

277、下表给出了一篇有 19 710 个词的英文文章最普遍的 15 个单词的出现频度。假定一篇正文仅由此表中的词组成，那么他们的最佳编码是什么？平均长度是多少？

|                            |      |     |     |     |     |     |      |     |     |     |     |     |     |     |     |
|----------------------------|------|-----|-----|-----|-----|-----|------|-----|-----|-----|-----|-----|-----|-----|-----|
| 单<br>词<br>出<br>现<br>频<br>度 | The  | of  | a   | to  | and | in  | that | he  | is  | at  | on  | for | His | are | be  |
|                            | 1192 | 677 | 541 | 518 | 462 | 450 | 242  | 195 | 190 | 181 | 174 | 157 | 138 | 124 | 123 |

答案：

哈夫曼编码如下：

The: 01          of: 101          a: 001          to: 100          and: 1110  
in: 1101    that: 11110    he: 11001    is: 11000    at: 10011  
on: 10010    for: 10001    His: 10000    are: 111111    be: 111110

平均长度=(1192\*2 + 677\*3 + 541\*3 + 518\*3 + 462\*4 + 450\*4 + 242\*5 +  
195\*5 + 190\*5 + 181\*5 + 174\*5 + 157\*5 + 138\*5 + 124\*6 +  
123\*6)/5364=3.56

解析：哈夫曼树如下：



哈夫曼编码如下：

The: 01          of: 101          a: 001          to: 100          and: 1110  
in: 1101    that: 11110    he: 11001    is: 11000    at: 10011  
on: 10010    for: 10001    His: 10000    are: 111111    be: 111110

平均长度=(1192\*2 + 677\*3 + 541\*3 + 518\*3 + 462\*4 + 450\*4 + 242\*5 + 195\*5 + 190\*5 + 181\*5 + 174\*5 + 157\*5 + 138\*5 + 124\*6 + 123\*6)/5364=3.56

278、假设二叉树采取二叉链存储结构，设计一个算法，计算一颗给定二叉树的所有节点数。

答案：

```
int n = 0;

void count(BTNode* p)

{

    if (p != NULL) {

        ++n;

        count(p->lchild);

        count(p->rchild);

    }

}
```

279、假设二叉树采取二叉链存储结构，设计一个算法，计算一颗给定二叉树的所有叶子节点数。

答案：

```

void count(BTNode* p)
{
    int n1,n2;

    if (p == NULL)

        return 0;

    else if(p->lchild == NULL && p->rchild == NULL)

        return 1;

    else

    {

        n1 = count(p->lchild);

        n2 = count(p->rchild);

        return n1 + n2;

    }

}

```

280、假设二叉树采取二叉链存储结构，设计一个算法，利用节点的右孩子指针 rchild 将一颗二叉树的叶子节点按照从左往右的顺序串成一个单链表。

答案：

```

void link(BTNode *p,BTNode *head,BTNode *&tail)

{

```

```

if (p != NULL)
{
    if(p->lchild == NULL && p->rchild == NULL)
    {
        if(head == NULL)
        {
            head = p;
            tail = p;
        }
        else
        {
            tail->rchild = p;
            tail = p;
        }
    }

    link(p->lchild, head, tail);

    link(p->rchild, head, tail);
}
}

```

281、求一颗二叉树的深度

答案:

```
int depth(BTNode *T) {  
  
    if(!T)  
  
        return 0;  
  
    int ldepth = depth(T->lchild);  
  
    int rdepth = depth(T->rchild);  
  
    return ldepth>rchild ? (ldepth+1) : (rdepth+1);  
  
}
```

282、二叉树的双序遍历是指：对于一个二叉树的每一个节点来说，先访问这个节点，再按双序遍历它的左子树，然后再一次访问这个节点，接下来再双序遍历它的右子树。试写出执行这种双序遍历的算法。

答案:

```
void Double_order(BTNode * t)  
  
{  
  
    If(t != NULL)  
  
    {  
  
        Visit(t);  
  
        Double_order(t->lchild);  
  
        Visit(t);  
  
    }  
  
}
```

```

        Double_order(t->rchild);
    }
}

```

283、试给出二叉树自下而上、从右到左的层次遍历算法。

答案：

```

void InvertLevel(BiTree bt) {
    Stack s; Queue Q;

    if(bt!=NULL) {
        InitStack(s);          //栈初始化,栈中存放二叉树结点的指针
        InitQueue(Q);          //队列初始化,队列中存放二叉树结点的指针
        EnQueue(Q, bt);

        while(IsEmpty(Q)!=false) {          //从上而下层次遍历
            DeQueue(Q, p);

            Push(s, p);          //出队,入栈

            if(p->lchild)
                EnQueue(Q, p->lchild);      //若左子女不空,则入队列

            if(p->rchild)
                EnQueue(Q, P->rchild);      //若右子女不空,则入队列
        }
    }
}

```

```

while(IsEmpty(s)==false){

    Pop(s, p);

    visit(p->data);

}          //自下而上、 从右到左的层次遍历

}          //if 结束

```

解析：一般的二叉树层次遍历是自上而下、从左到右的遍历, 这里的遍历顺序恰好相反。算法思想:利用原有的层次遍历算法, 出队的同时将各结点指针入栈, 在所有结点入栈后再从栈顶开始依次访问即是所求的算法。具体实现为

- (1) 把根结点入队列
- (2) 把一个元素出队列, 遍历这个元素
- (3) 依次把这个元素的右孩子,左孩子入队列
- (4) 若队列不空,则跳到 (2),否则结束.

算法实现如下:

```

void InvertLevel(BiTree bt){

    Stack s; Queue Q;

    if(bt!=NULL){

        InitStack(s);          //栈初始化,栈中存放二叉树结点的指针

        InitQueue(Q);          //队列初始化, 队列中存放二叉树结点的指针

        EnQueue(Q, bt);

        while(IsEmpty(Q)!=false){          //从上而下层次遍历

            DeQueue(Q, p);

```



```

        Push(s, p);          //出队, 入栈

        if(p->lchild)

            EnQueue(Q, p->lchild);      //若左子女不空,则入队列

        if(p->rchild)

            EnQueue(Q, P->rchild);      //若右子女不空, 则入队列

    }

    while(IsEmpty(s)==false) {

        Pop(s, p);

        visit(p->data);

    }      //自下而上、 从右到左的层次遍历

}      //if 结束

```

**284、编写后序遍历二叉树的非递归算法。**

答案:

```

void PostOrder(BiTree T) {

    InitStack(S);

    p=T;

    r=NULL;

    while(p || !IsEmpty(S)) {

        if(p) {      //走到最左边

```

```

        push(S, p);

        p=p->lchild;

    }else{    //向右

        GetTop(S, p);    //取栈顶结点

        if(p->rchild&& p->rchild!=r){    //如果右子树存在,且未被
访问过

            p=p->rchild;    //转向右

            push(S, p);    //压入栈

            p=p->lchild;    //再走到最左

        else{    //否则,弹出结点并访 问

            pop(S, p);    //将结点弹出

            visit(p->data);    //访问该结点

            r=p;    //记录最近访问过的结点

            p=NULL;    //结点访问完后,重置 p 指针

        }//else

    }//while

}

```

解析：算法的思想：因为后序非递归遍历二叉树的顺序是先访问左子树，再访问右子树，最后访问根结点。当用堆栈来存储结点，必须分清返回根结点时，是从左子树返回的，还是从右子树返回的。所以，使用辅助指针 *r*，其指向最近访问过的结点。也可以在结点中增加一个标志域，记录是否已被访问。

```

void PostOrder(BiTree T){

```

```

InitStack(S);

p=T;

r=NULL;

while(p||!IsEmpty(S)) {

    if(p) {    //走到最左边

        push(S, p);

        p=p->lchild;

    }else{    //向右

        GetTop(S, p);    //取栈顶结点

        if(p->rchild&& p->rchild!=r) {    //如果右子树存在,且未被
访问过

            p=p->rchild;    //转向右

            push(S, p);    //压入栈

            p=p->lchild;    //再走到最左

        }else{    //否则,弹出结点并访 问

            pop(S, p);    //将结点弹出

            visit(p->data);    //访问该结点

            r=p;    //记录最近访问过的结点

            p=NULL;    //结点访问完后,重置 p 指针

        }//else

    }//while

}

```

285、用二叉链表存储二叉树，写出中序打印二叉树中结点元素值的递归算法。

答案：

二叉链表的数据结构：

```
typedef struct BiTNode {                                //结点
    int data;                                           //数据域
    struct BiTNode *lchild, *rchild;                  //左右孩子指针
} BiTNode, *BiTree;
```

函数首部：void MidOrderPrint (BiTree T)

解答：

```
void MidOrderPrint (BiTree T)
{
    if(T)
    {
        MidOrderPrint(T->lchild);                    //3分
        printf("%d\t", T->data);                      //2分
        MidOrderPrint(T->rchild);                     //2分
    }
}
```

286、二叉树采用链式存储结构，结点的存储结构如下图所示：



结点的结构类型定义如下：struct NODE { double data; // 数据域 struct NODE \*lch, \*rch; // 指针域 }; 请用递归方法编写算法求二叉树的深度。 int Depth(NODE \*root); 函数的参数是根指针，函数值是二叉树的深度。如：printf("%d\n", Depth(root)); 要求：用文字描述算法思想，并估算时间复杂度，然后用 C/C++语言编码。

答案：

参考代码如下，时间复杂度为  $O(n)$ ：

```
int Depth(NODE *root)
{
    int n = 0, d1, d2;

    if (root != NULL)
    {
        d1 = Depth(root->lch);
        d2 = Depth(root->rch);
        n = d1 >= d2 ? d1 : d2;
    }

    return n;
}
```

287、设计判断两个二叉树是否相同的算法。

答案：

```
typedef struct BNode
```

```
{
```

```
    datatype data;
```

```
    struct BNode *lchild, *rchild;
```

```
}    BNode, *BTree;
```

```
int JudgeBTree(BTree bt1, BTree bt2)
```

```
{
```

```
    if (bt1 == NULL && bt2 == NULL)
```

```
        return (1);
```

```
    else if (bt1 == NULL || bt2 == NULL || bt1->data != bt2->data)
```

```
        return (0);
```

```
    else
```

```
        return (JudgeBTree (bt1->lchild, bt2->lchild) * JudgeBTree  
        (bt1->rchild, bt2->rchild));
```

```
}
```

288、用二叉链表存储二叉树，写出中序打印二叉树中结点元素值的递归算法。

答案：

二叉链表的数据结构：

```
Typedef struct BiTNode  
{  
  
    //结点结构  
  
    int data;  
  
    //数据域  
  
    Struct BiTNode *lchild, *rchild;  
  
    // 左右孩子指针  
  
} BiTNode, *BiTree;
```

函数首部：void MidOrderPrint (BiTreeT)

解答：

```
void MidOrderPrint (BiTreeT)  
{  
  
    if(T) //3 分  
  
    {
```

```
MidOrderPrint(T->lchild); //3 分

printf("%d\t", T->data); //2 分

MidOrderPrint(T->rchild); //2 分

}

}
```

289、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

解析：握手定理。

290、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

解析：出边=入边。



291、对于一个具有  $n$  个顶点的有向图的边数最多有（ ）。

A、  $n$

B、  $n(n-1)$

C、  $n(n-1)/2$

D、  $2n$

答案： B

解析： 完全有向图边数最多。

292、在一个具有  $n$  个顶点的无向图中，要连通全部顶点至少需要（ ）条边。

A、  $n$

B、  $n+1$

C、  $n-1$

D、  $n/2$

答案： C

解析： 生成树  $n-1$  条边。

293、有 8 个结点的有向完全图有( )条边。

A、 14

B、 28

C、 56

D、 112

答案： C

解析：  $n$  个顶点的完全有向图的边数为  $n(n-1)$  。

294、图的深度优先遍历类似于二叉树的( )。

A、 前序遍历

B、 中序遍历

C、 后序遍历

D、 层次遍历

答案： A

解析： 图的深度优先遍历类似于二叉树的前序遍历。

295、图的广度优先遍历类似于二叉树的( )。

A、 前序遍历

B、 中序遍历

C、 后序遍历

D、 层次遍历

答案： D

解析：图的广度优先遍历类似于二叉树的层次遍历。

296、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

答案： A

解析：  $n$  个结点的最小生成树有  $n-1$  条边。

297、无向图顶点  $v$  的度是关联于该顶点（ ）的数目。

A、 顶点

B、 边

C、 序号

D、 下标

答案： B

解析： 无向图顶点  $v$  的度是指关联于该顶点边的数目。

298、有  $n$  个顶点的无向图的邻接矩阵是用（ ）数组存储。

A、 一维

B、  $n$  行  $n$  列

C、 任意行  $n$  列

D、  $n$  行任意列

答案： B

解析： 邻接矩阵是一个二维数组。

299、 对于一个具有  $n$  个顶点和  $e$  条边的无向图，采用邻接表表示，则表头向量大小为（ ）。

A、  $n-1$

B、  $n+1$

C、  $n$

D、  $n+e$

答案： C

解析：  $n$  个顶点的邻接表是一个含  $n$  个元素的一维数组。

300、在图的表示法中，表示形式唯一的是（ ）。

A、 邻接矩阵表示法

B、 邻接表表示法

C、 逆邻接表表示法

D、 邻接表和逆邻接表表示法

答案： A

解析： 在图的表示法中，表示形式唯一的是邻接矩阵表示法。

301、下列图中，度为 3 的结点是（ ）。



A、  $V_1$

B、  $V_2$

C、  $V_3$

D、  $V_4$

答案： B

解析： 无向图顶点  $v$  的度是指关联于该顶点边的数目。

302、下列图是（ ）。



A、 连通图

B、 强连通图

C、 生成树

D、 无环图

答案： A

解析： 若无向图中任意两个顶点都连通，则称为连通图；有向图中的任意两个顶点  $i$  和  $j$  都连通，即从顶点  $i$  到  $j$  和从顶点  $j$  到  $i$  都存在路径，则称图是强连通图。

303、如下图所示，从顶点  $a$  出发，按深度优先进行遍历，则可能得到的一种顶点序列为（ ）。



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

答案： D

解析： 深度优先搜索遍历的过程是：

(1) 从图中某个初始顶点  $v$  出发，首先访问初始顶点  $v$ 。

(2) 选择一个与顶点  $v$  相邻且没被访问过的顶点  $w$  为初始顶点，再从  $w$  出发进行深度优先搜索，直到图中与当前顶点  $v$  邻接的所有顶点都被访问过为止。

304、 如下图所示，从顶点  $a$  出发，按广度优先进行遍历，则可能得到的一种顶点序列为 ( )。



A、

a, b, e, c, d, f

B、

a, b, e, c, f, d



C、

a, e, b, c, f, d

D、

a, e, d, f, c, b

答案： A

解析： 广度优先搜索遍历的过程是：

- (1) 访问初始点  $v$ ，接着访问  $v$  的所有未被访问过的邻接点  $v_1, v_2, \dots, v_t$ 。
- (2) 按照  $v_1, v_2, \dots, v_t$  的次序，访问每一个顶点的所有未被访问过的邻接点。
- (3) 依次类推，直到图中所有和初始点  $v$  有路径相通的顶点都被访问过为止。

305、最小生成树的构造可使用（      ）算法。

A、 prim 算法

B、 卡尔算法

C、 哈夫曼算法

D、 迪杰斯特拉算法

答案： A

解析： 最小生成树的构造可使用 prim 算法和克鲁斯卡尔 (Kruskal) 算法。

306、下面关于图的存储结构的叙述中正确的是（ ）。

A、

用邻接矩阵存储图，占用空间大小只与图中顶点数有关，而与边数无关

B、 用邻接矩阵存储图，占用空间大小只与图中边数有关，而与顶点数无关

C、 用邻接表存储图，占用空间大小只与图中顶点数有关，而与边数无关

D、

用邻接表存储图，占用空间大小只与图中边数有关，而与顶点数无关

答案： A

解析： 常量时间。

307、连通分量是（ ）的极大连通子图。

A、 树

B、 图

C、 无向图

D、 有向图

答案： C

解析： 无向图  $G$  中的极大连通子图称为  $G$  的连通分量。

308、强连通分量是（ ）的极大强连通子图。

A、 树

B、 图

C、 无向图

D、 有向图

答案： D

解析： 有向图  $G$  中的极大强连通子图称为  $G$  的强连通分量。

309、图可以没有边，但不能没有顶点。（）

答案： 正确

解析：图可以没有边，但不能没有顶点。

310、在无向图中， $(V_1, V_2)$  与  $(V_2, V_1)$  是两条不同的边。（）

答案： 错误

解析：在无向图中， $(V_1, V_2)$  与  $(V_2, V_1)$  是同一条边。

311、稀疏图适合邻接矩阵存储比较节省空间，稠密图采用邻接表存储比较节省空间。（）

答案： 错误

解析：正好相反。

312、用邻接矩阵法存储一个图时，所占用的存储空间大小与图中顶点个数无关，而只与图的边数有关。（）

答案： 错误

解析：用邻接矩阵法存储一个图时，所占用的存储空间大小与图中顶点个数有关，而与图的边数无关。

313、有向图不能进行广度优先遍历。（ ）

答案： 错误

解析：有向图可以进行广度优先遍历。

314、有向图 G 用邻接矩阵存储，其第 i 行的所有元素之和等于顶点 i 的出度。  
( )

答案： 正确

解析：在有向图中，以顶点 i 为终点的入边的数目，称为该顶点的入度。以顶点 i 为始点的出边的数目，称为该顶点的出度。一个顶点的入度与出度的和为该顶点的度。

315、存储无向图的邻接矩阵是对称的，因此只要存储邻接矩阵的上三角（或下三角）部分就可以了。（ ）

答案： 正确

解析：对称矩阵可以压缩存储。

316、用邻接表法存储图时，占用的存储空间大小只与图中的边数有关，而与结点的个数无关。（ ）

答案： 错误

解析：都有关。

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

答案： 正确

解析： $n*n$  的二维数组。

318、 $n$  个顶点  $e$  条边的图若采用邻接表存储，则空间复杂度为： $O(n+e)$ 。（ ）

答案： 正确

解析：一维数组  $n$ ，边表结点  $e$ 。

319、若一个无向图的以顶点  $V_1$  为起点进行深度优先遍历，所得的遍历序列唯一，则可以唯一确定该图。（ ）

答案： 正确

解析：若一个无向图的以顶点  $V_1$  为起点进行深度优先遍历，所得的遍历序列唯一，则可以唯一确定该图。

320、若一个无向图中任一顶点出发，进行一次深度优先遍历，就可以访问图中所有的顶点，则该图一定是连通的。（ ）

答案： 正确

解析：连通图的特性。

321、已知一个图如图所示，若从顶点 a 出发按深度搜索法进行遍历，则可得到顶点序列为 **abcefd**。( )



答案： 错误

解析：是广度搜索法进行遍历的结果。

322、已知一个图如图所示，若从顶点 a 出发按广度搜索法进行遍历，则可得到顶点序列为 **aedfcb**。( )



答案： 错误

解析：是深度搜索法进行遍历的结果。

323、已知一有向图的邻接表存储结构如图所示，根据有向图的深度优先遍历算法，从 **v1** 顶点出发，所得到的顶点序列是 **v1,v3,v4,v5,v2**。( )



答案： 正确

解析：深度优先搜索遍历的过程是：

(1) 从图中某个初始顶点 **v** 出发，首先访问初始顶点 **v**。

(2) 选择一个与顶点 **v** 相邻且没被访问过的顶点 **w** 为初始顶点，再从 **w** 出发进行深度优先搜索，直到图中与当前顶点 **v** 邻接的所有顶点都被访问过为止。

324、已知一有向图的邻接表存储结构如图所示，根据有向图的广度优先遍历算法，从  $v_1$  顶点出发，所得到的顶点序列是  $v_1, v_3, v_2, v_4, v_5$ 。( )



答案： 正确

解析： 广度优先搜索遍历的过程是：

- (1) 访问初始点  $v$ ，接着访问  $v$  的所有未被访问过的邻接点  $v_1, v_2, \dots, v_t$ 。
- (2) 按照  $v_1, v_2, \dots, v_t$  的次序，访问每一个顶点的所有未被访问过的邻接点。
- (3) 依次类推，直到图中所有和初始点  $v$  有路径相通的顶点都被访问过为止。

325、已知图  $G$  的邻接表如图所示，其从  $v_1$  顶点出发的深度优先搜索序列为 1、2、5、4、3、6。( )



答案： 错误

解析： 是广度优先搜索序列。

326、已知图  $G$  的邻接表如图所示，其从  $v_1$  顶点出发的深度优先搜索序列为 1、2、3、6、5、4。( )



答案： 正确

解析： 深度优先搜索遍历的过程是：

- (1) 从图中某个初始顶点  $v$  出发，首先访问初始顶点  $v$ 。
- (2) 选择一个与顶点  $v$  相邻且没被访问过的顶点  $w$  为初始顶点，再从  $w$  出发进行深度优先搜索，直到图中与当前顶点  $v$  邻接的所有顶点都被访问过为止。



327、若要求一个稠密图 G 的最小生成树，最好用 Prim 算法来求解。( )

答案： 正确  
解析： Prim 算法适合稠密图。

328、判定一个有向图是否存在回路除了可以利用拓扑排序方法外，还可以利用深度优先遍历算法。( )

答案： 正确  
解析：判定一个有向图是否存在回路除了可以利用拓扑排序方法外，还可以利用深度优先遍历算法。

329、有向图如下图所示，画出邻接矩阵和邻接表



答案：

邻接矩阵

|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 0 | 0 |

邻接表

| 顶点 | 邻接点        |
|----|------------|
| 1  | 2, 3, 4, 5 |
| 2  | 3          |
| 3  | 4          |
| 4  | 1          |
| 5  | 2, 3       |

330、已知某图 G 的邻接矩阵如图，

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

(1) 画出相应的图； (2) 要使此图为完全图需要增加几条边。

答案：

(1)

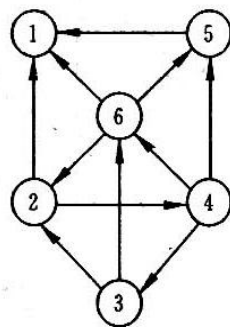


(2)

完全无向图应具有边数为： $n*(n-1)/2=4*(4-1)/2=6$ ，所以还要增加 2 条边（如图）



331、已知如图所示的有向图，请给出该图的：



(1) 每个顶点的入/出度；

(2) 邻接表；

(3) 邻接矩阵。

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)



(3)

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 1 | 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 |

332、 已知如图所示的有向图，请给出该图的：



(1) 每个顶点的入度和出度；

(2) 逆邻接表。

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)



333、请画出如下图所示的邻接矩阵和邻接表。



答案：

邻接矩阵

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

邻接表如下图所示



334、已知一个无向图的顶点集为：{a, b, c, d, e}，其邻接矩阵如下，画出草图，写出顶点 a 出发按深度优先搜索进行遍历的结点序列。

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

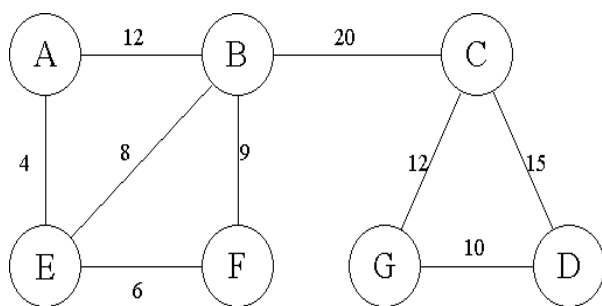
答案：



深度优先搜索：a b d c e （答案不唯一）

广度优先搜索：a b e d c （答案不唯一）

335、给定下列网 G:



- (1) 写出网 G 以 B 为顶点的广度优先遍历的序列；
- (2) 画出网 G 的最小生成树。

答案：

- (1) 以 B 为顶点的广度优先遍历的序列：

B A E F C G D

- (2) 最小生成树



336、无向图 G 如图所示，（1）试画出邻接矩阵；（2）写出从 A 出发的深度优先遍历的序列。



答案：

- (1) 邻接矩阵

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 0 | 0 | 0 | 0 |
| B | 1 | 0 | 0 | 1 | 0 | 0 |
| C | 0 | 0 | 1 | 0 | 1 | 0 |
| D | 0 | 1 | 0 | 1 | 0 | 0 |
| E | 0 | 0 | 1 | 0 | 1 | 1 |
| F | 0 | 0 | 0 | 1 | 1 | 0 |
| G | 0 | 0 | 0 | 0 | 1 | 1 |

- (2) 从 A 出发的深度优先遍历的序列：

A B D C E G F (不唯一)

337、已知图 G 的邻接表如下，以顶点 1 为出发点，完成下列问题：



- (1) 写出以顶点 1 为出发点的广度优先遍历序列；
- (2) 画出以顶点 1 为出发点的深度优先搜索得到的一棵二叉树。

答案：

(1) 广度优先遍历序列：1，2，5，4，3

(2) 深度优先搜索得到的一棵二叉树：



338、网 G 的邻接矩阵如下，试画出该图，并画出它的一棵最小生成树。

|    |    |    |    |    |
|----|----|----|----|----|
| 9  | 0  | 36 | 13 | 6  |
| 0  | 0  | 5  | 2  | 13 |
| 36 | 5  | 0  | 4  | 6  |
| 13 | 2  | 4  | 0  | 7  |
| 6  | 13 | 6  | 7  | 0  |

答案：



339、已知一个无向图有 6 个结点，9 条边，这 9 条边依次为  $(0, 1)$ ， $(0, 2)$ ， $(0, 4)$ ， $(0, 5)$ ， $(1, 2)$ ， $(2, 3)$ ， $(2, 4)$ ， $(3, 4)$ ， $(4, 5)$ 。试画出该无向图，并从顶点 0 出发，分别写出按深度优先搜索和按广度优先搜索进行遍历的结点序列。

答案：



从顶点 0 出发的深度优先搜索遍历的结点序列：0 1 2 3 4 5（答案不唯一）

从顶点 0 出发的广度优先搜索遍历的结点序列：0 1 2 4 5 3（答案不唯一）

340、对于如下图所示的图 G，邻接点按从小到大的次序。



- (1) 图 G 有几个连通分量？
- (2) 按深度优先搜索所得的树是什么？
- (3) 按深度优先搜索所得的顶点序列是什么？

答案：

- (1) 图 G 有 2 个连通分量。
- (2) 按深度优先搜索所得的树如下图所示：



- (3) 按深度优先搜索所得的顶点序列：ABHFGCDE

341、如下所示的有向图，回答下面问题：



- (1) 该图是强连通的吗？若不是，给出强连通分量。
- (2) 请给出图的邻接矩阵和邻接表表示。

答案：

(1) 是强连通图

(2) 邻接矩阵和邻接表为：

邻接矩阵：  

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

342、设无向图 G（如图所示），给出该图的最小生成树上边的集合并计算最小生成树各边上的权值之和。



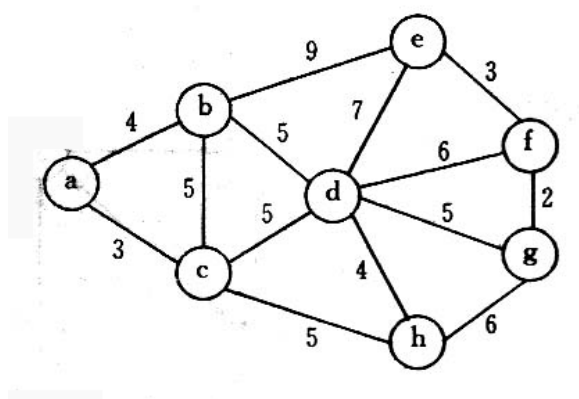
答案：

$E = \{(1,5)2, (5,2)1, (5,3)1, (3,4)6\}, W = 10$

343、如图，请完成以下操作：

- (1) 写出无向带权图的邻接矩阵；
- (2) 设起点为 a，求其最小生成树。





答案:

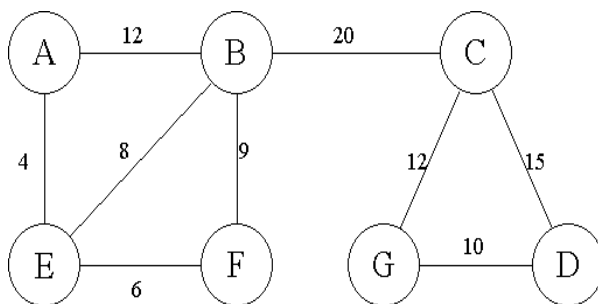
(1) 邻接矩阵为:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 4 | 3 | 5 | 7 | 6 | 5 | 4 |
| 4 | 0 | 5 | 9 | 5 | 0 | 0 | 0 |
| 3 | 5 | 0 | 5 | 0 | 0 | 0 | 5 |
| 5 | 9 | 5 | 0 | 4 | 6 | 5 | 0 |
| 7 | 5 | 0 | 4 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 6 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 5 | 0 | 0 | 0 | 0 |
| 4 | 0 | 5 | 0 | 0 | 0 | 0 | 0 |

(2) 起点为 a, 可以直接由原始图画出生成树



344、给定下列网 G:



(1) 画出网 G 的邻接矩阵;

(2) 画出网 G 的最小生成树。

答案:

(1) 邻接矩阵

|    |    |    |   |    |    |    |
|----|----|----|---|----|----|----|
| 1  | 2  | 3  | 4 | 5  | 6  | 7  |
| 12 | 2  | 2  | 1 | 1  | 1  | 1  |
| 7  | 10 | 10 | 5 | 10 | 10 | 10 |
| 4  | 10 | 10 | 1 | 1  | 1  | 1  |
| 1  | 1  | 1  | 1 | 1  | 1  | 1  |
| 4  | 1  | 1  | 1 | 1  | 1  | 1  |
| 4  | 1  | 1  | 1 | 1  | 1  | 1  |

(2) 最小生成树



345、 已知一个图的顶点集  $V$  和边集  $E$  分别为：

$V = \{1, 2, 3, 4, 5, 6, 7\}$  ;

$E = \{(1, 2) 3, (1, 3) 5, (1, 4) 8, (2, 5) 10, (2, 3) 6, (3, 4) 15, (3, 5) 12, (3, 6) 9, (4, 6) 4, (4, 7) 20, (5, 6) 18, (6, 7) 25\}$  ;

用克鲁斯卡尔算法得到最小生成树，试写出在最小生成树中依次得到的各条边并计算最小生成树各边上的权值之和。

答案：

用克鲁斯卡尔算法得到的最小生成树为：

$E = \{(1, 2) 3, (4, 6) 4, (1, 3) 5, (1, 4) 8, (2, 5) 10, (4, 7) 20\}$ ,  $W = 50$

346、对于如下图所示的  $G$ ，用 **Kruskal** 算法构造最小生成树，要求图示出每一步的变化情况并计算最小生成树各边上的权值之和。



答案：

用 **Kruskal** 算法构造最小生成树的过程如下图所示：



$$W=23$$

347、已知一个图的顶点集  $V$  为：  $V=\{1,2,3,4,5,6,7\}$ ，弧如下表所示。

图的弧集

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 起点 | 1 | 2 | 2 | 5 | 5 | 2 | 2 | 6 | 1 | 3 |
| 终点 | 6 | 4 | 5 | 4 | 7 | 6 | 7 | 7 | 7 | 5 |
| 权  | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 4 | 5 | 7 |

试用克鲁斯卡尔算法依次求出该图的最小生成树中所得到的各条边并计算最小生成树各边上的权值之和。

答案：

用克鲁斯卡尔算法得到的最小生成树为：

$$E=\{(1, 6) 1, \quad (2, 4) 1, \quad (2, 5) 2, \quad (5, 7) 2, \quad (2, 6) 3, \quad (3, 5) 7\}, \quad W=16$$

348、设有无向图  $G$ ，要求给出用普里姆算法构造最小生成树所走过的边的集合（从顶点 1 开始）并计算最小生成树各边上的权值之和。



答案：

$$E=\{(1, 3) 2, \quad (1, 2) 3, \quad (3, 5) 4, \quad (5, 6) 1, \quad (6, 4) 1\}, \quad W=11$$

349、已知一个带权图的顶点集  $V$  和边集  $G$  分别为：

$$V=\{0, 1, 2, 3, 4, 5, 6\};$$

$$E=\{(0, 1) 19, (0, 2) 10, (0, 3) 14, (1, 2) 6, (1, 5) 5, (2, 3) 26, (2, 4) 15, (3, 4) 18, (4, 5) 6, (4, 6) 6, (5, 6) 12\};$$

试根据迪克斯特拉(Dijkstra)算法求出从顶点 0 到其余各顶点的最短路径，在下面表中填写对应的路径长度。



答案：



350、已知一个带权图如下，试根据迪克斯特拉(Dijkstra)算法求出从顶点 0 到其余各顶点的最短路径，写出 dist 和 path 数组变化情况。



答案：

| dist | path  |
|------|-------|
| 0    | 0     |
| 10   | 0-2   |
| 14   | 0-3   |
| 19   | 0-1   |
| 24   | 0-2-5 |
| 26   | 0-2-3 |
| 31   | 0-2-4 |

351、请用图示说明图从顶点 a 到其余各顶点之间的最短路径。



答案：



352、试列出如下图中全部可能的拓扑排序序列。



答案：

全部可能的拓扑排序序列为：1523634、152634、156234、561234、516234、512634、512364

353、已知一个图的顶点集  $V$  和边集  $E$  分别为：

$V = \{1, 2, 3, 4, 5, 6, 7\}$ ；

$E = \{\langle 2, 1 \rangle, \langle 3, 2 \rangle, \langle 3, 6 \rangle, \langle 4, 3 \rangle, \langle 4, 5 \rangle, \langle 4, 6 \rangle, \langle 5, 1 \rangle, \langle 5, 7 \rangle, \langle 6, 1 \rangle, \langle 6, 2 \rangle, \langle 6, 5 \rangle\}$ ；

若存储它采用邻接表，并且每个顶点邻接表中的边结点都是按照终点序号从小到大的次序链接的，试给出得到的拓扑排序的序列。

答案：

拓扑排序为： 4    3    6    5    7    2    1

354、已知有向图如下所示，请写出该图所有的拓扑序列。



答案：

拓扑排序如下：

v1, v2, v4, v6, v5, v3, v7, v8                      v1, v2, v4, v6, v5,  
v7, v3, v8

v1, v2, v6, v4, v5, v3, v7, v8                      v1, v2, v6, v4, v5,  
v7, v3, v8

v1, v6, v2, v4, v5, v3, v7, v8                      v1, v6, v2, v4, v5,  
v7, v3, v8

355、 设有如下图所示的 AOE 网（其中  $v_i$  ( $i=1, 2, \dots, 6$ ) 表示事件，弧上表示活动的天数）。



- (1) 找出所有的关键路径。
- (2)  $v_3$  事件的最早开始时间是多少？

答案：

- (1) 找出所有的关键路径有：  $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_5 \rightarrow v_6$ ，以及  $v_1 \rightarrow v_4 \rightarrow v_6$ 。
- (2)  $v_3$  事件的最早开始时间是 13。

356、对给定的有 7 个顶点的有向图的邻接矩阵如下：

|          |   |   |   |   |   |   |
|----------|---|---|---|---|---|---|
| $a_{ij}$ | 1 | 2 | 3 | 4 | 5 | 6 |
| $a_{1j}$ | 0 | 0 | 1 | 0 | 0 | 0 |
| $a_{2j}$ | 0 | 0 | 0 | 1 | 0 | 0 |
| $a_{3j}$ | 0 | 0 | 0 | 0 | 1 | 0 |
| $a_{4j}$ | 0 | 0 | 0 | 0 | 0 | 1 |
| $a_{5j}$ | 0 | 0 | 0 | 0 | 0 | 0 |
| $a_{6j}$ | 0 | 0 | 0 | 0 | 0 | 0 |

- (1) 画出该有向图；
- (2) 若将图看成是 AOE-网，画出关键路径。

答案：

- (1) 由邻接矩阵所画的有向图如下图所示：



- (2) 关键路径如下图所示：



357、已知 AOE 网有 9 个结点：V1, V2, V3, V4, V5, V6, V7, V8, V9，其邻接矩阵如下：

- (1) 请画出该 AOE 图。
- (2) 计算完成整个计划需要的时间。
- (3) 求出该 AOE 网的关键路径。

|          |          |          |          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| $\infty$ | 6        | 4        | 5        | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| $\infty$ | $\infty$ | $\infty$ | $\infty$ | 1        | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| $\infty$ | $\infty$ | $\infty$ | $\infty$ | 1        | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 2        | $\infty$ | $\infty$ | $\infty$ |
| $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 9        | 7        | $\infty$ |
| $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 4        | $\infty$ |
| $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 2        |
| $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 4        |
| $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |

答案：

(1)该 AOE 图为:



(2)完成整个计划需要 18 天。

(3)关键路径为: (V1, V2, V5, V7, V9) 和 (V1, V2, V5, V8, V9, )


358、顺序查找指的是在顺序存储结构上进行查找。

答案: 错误

解析: 顺序查找也可以在链表中进行。

359、在顺序表中按值查找运算的复杂性为  $O(1)$ 。

答案: 错误

解析: 等概率前提下, 在顺序表中按值查找的平均比较次数为  次。

360、下列选项中, 既能在顺序存储结构上, 也能在链式存储结构上进行查找的方法是\_\_\_\_\_。

A、散列查找



B、 顺序查找

C、 折半查找

D、 索引查找

答案： B

解析：折半查找因为需要按位号随机访问元素，因此只能在顺序存储结构上进行；

散列查找和索引查找的存储结构一般不是简单地直接构建在单独的顺序存储或者链式存储上；

只有顺序查找对存储结构没有特殊要求。

361、在顺序存储结构中进行顺序查找：

(1) 设监视哨放在高位，完成有监视哨的顺序查找算法，查找成功返回下标，失败返回-1，相关数据类型和查找函数的原型如下：

```
#define MAXSIZE 100
```

```
typedef int KeyType;
```

```
typedef struct
```

```
{
```

```
    KeyType key;
```

```
} RecType;
```

```
typedef RecType SeqList[MAXSIZE];
```

`int Sentinel(SeqList data, KeyType keyword, int n); // 带监视哨查找算法的原型`

`// data 是存放关键字的数组, keyword 为待查找关键字, n 为元素个数`

(2) 增加监视哨后，等概率前提下，查找成功和查找失败的平均查找长度分别是多少？

(3) 监视哨的作用是什么？


答案：

```
int Sentinel(SeqList data, KeyType keyword, int n)
{
    int i;

    data[n].key = keyword;

    for (i = 0; data[i].key != keyword; i ++);

    return i == n ? -1 : i;
}
```

(2) 有监视哨时，查找成功的平均查找长度还是 ，但是查找失败的平均查找长度是  $n + 1$ 。

(3) 监视哨的作用是免去在查找过程中，每次比较关键字前都要比较检测整个表是否查找完毕（也就是越界），虽然增加监视哨后，并没有降低算法时间复杂度的数量级，但是减少了绝对的比较时间，同样也提高了查找效率。

362、用数组和单链表表示的有序表均可使用折半查找方法来提高查找速度。

答案： 错误

解析：折半查找的按位号访问只能在顺序表上进行。

363、折半查找法的查找速度一定比顺序查找法快。

答案： 错误

解析：折半查找法的查找速度不一定比顺序查找法快。

364、设有序表中关键字序列为(9, 12, 21, 32, 41, 45, 52)，当折半查找值为 52 的结点时，元素之间的比较次数是\_\_\_\_\_。

- A、 1
- B、 2
- C、 3
- D、 4

答案： C

解析：表长为 7 时，折半查找判定树的形态正好是一棵高度为 3 的满二叉树，并且关键字的最大值 52 位于最右边结点，查找成功需要依次比较 32, 45, 52 这 3 个关键字。

365、对线性表 L 进行折半查找时，要求 L 必须满足\_\_\_\_\_。

A、 以顺序方式存储

B、 以顺序方式存储，且数据元素有序

C、 以链接方式存储

D、 以链接方式存储，且数据元素有序

答案： B

解析：折半查找过程中不仅要能够按位号随机访问各个元素，而且一旦和中点的关键字相比较不等后，需要舍弃序列中一半的关键字，只有元素有序时才能满足。

366、下列线性表中，能使用折半查找的是\_\_\_\_\_。

A、 顺序存储(2, 12, 5, 6, 9, 3, 89, 34, 25)

B、 链式存储(2, 12, 5, 6, 9, 3, 89, 34, 25)

C、 顺序存储(2, 3, 5, 6, 9, 12, 25, 34, 89)

D、 链式存储(2, 3, 5, 6, 9, 12, 25, 34, 89)

答案： C

解析：折半查找对关键字的要求就是有序的顺序表。

367、对表长为  $n$  的有序表进行折半查找，其判定树的高度为\_\_\_\_\_。

A、 不小于  $\log_2(n+1)$  的最小整数

B、 不大于  $\log_2(n+1)$  的最大整数

C、 不小于  $\log_2 n$  的最小整数

D、 不大于  $\log_2 n$  的最大整数

答案： A

解析：长度为  $n$  的折半查找判定树，高度  $h$  与相同结点个数的完全二叉树相等，根据完全二叉树的性质可知： $h = \lceil \log_2(n+1) \rceil$ 。

368、长度为 11 的有序顺序表 (4, 9, 11, 16, 23, 28, 37, 46, 69, 71, 88)，用折半查找对该表进行查找。

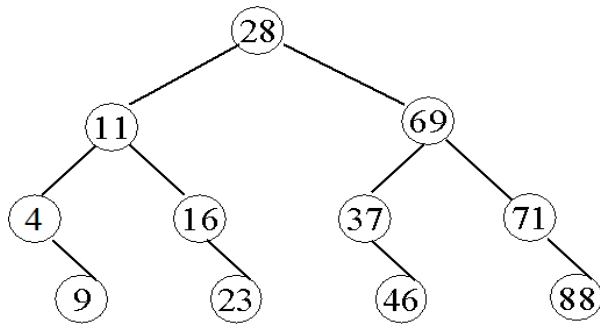
(1) 画出折半查找所对应的判定树；

(2) 查找元素 46，需要依次与哪些元素比较？

(3) 查找元素 72，需要进行的元素比较次数？

答案：

(1) 该有序表折半查找的判定树：



(2) 从判定树可知，查找关键字 46 依次比较的是 28, 69, 37, 46，查找成功；

(3) 从判定树可知，查找关键字 72 依次比较的是 28, 69, 71, 88，查找失败。

369、假定对有序表：(2, 11, 15, 27, 34, 40, 42, 54, 63, 72, 87, 95) 进行折半查找：

(1) 查找关键字 90，需依次与哪些元素相比较？

(2) 查找不成功时，最多的比较次数是多少？

(3) 设各元素的查找概率相等，求查找成功时的平均查找长度；

(4) 设各元素的查找概率相等，求查找失败时的平均查找长度。

答案：

(1) 查找关键字 90 依次比较的是 40, 63, 87, 95，查找失败；

(2) 查找不成功最多比较 4 次；

(3)  $ASL_{succ} = (1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) / 12 = 37 / 12$ ；

(4)  $ASL_{unsucc} = (3 \times 3 + 10 \times 4) / 13 = 49 / 13$ 。

解析：该长度为 12 的有序表折半查找的判定树，高度为 4：



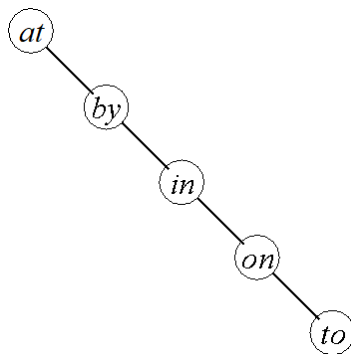
370、设有序表中依次存放有 5 个数据  $at$ ,  $by$ ,  $in$ ,  $on$ ,  $to$ , 其查找概率分别为  $p_1 = 0.2$ ,  $p_2 = 0.15$ ,  $p_3 = 0.1$ ,  $p_4 = 0.03$ ,  $p_5 = 0.01$ , 而查找它们之间不存在数据的概率分别为  $q_0 = 0.2$ ,  $q_1 = 0.15$ ,  $q_2 = 0.1$ ,  $q_3 = 0.03$ ,  $q_4 = 0.02$ ,  $q_5 = 0.01$ :

|       |       |       |       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
|       | $at$  |       | $by$  |       | $in$  |       | $on$  |       | $to$  |       |
| $q_0$ | $p_1$ | $q_1$ | $p_2$ | $q_2$ | $p_3$ | $q_3$ | $p_4$ | $q_4$ | $p_5$ | $q_5$ |

- (1) 画出对该有序表进行顺序查找时的判定树，并分别计算顺序查找时的查找成功和查找失败的平均查找长度；
- (2) 画出对该有序表进行折半查找时的判定树，并分别计算折半查找时的查找成功和查找失败的平均查找长度；
- (3) 就该问题而言，顺序查找还是折半查找的性能好？

答案：

- (1) 顺序查找的判定树：



顺序查找的性能：

$$ASL_{succ} = 1 \cdot p_1 + 2 \cdot p_2 + 3 \cdot p_3 + 4 \cdot p_4 + 5 \cdot p_5 = 0.97;$$

$$ASL_{unsucc} = 1 \cdot q_0 + 2 \cdot q_1 + 3 \cdot q_2 + 4 \cdot q_3 + 5 \cdot q_4 + 5 \cdot q_5 = 1.07。$$

(2) 折半查找的判断树:



折半查找的性能:

$$ASL_{succ} = 1 \cdot p_3 + 2(p_1 + p_4) + 3(p_2 + p_5) = 1.04;$$

$$ASL_{unsucc} = 2 \cdot q_0 + 3 \cdot q_1 + 3 \cdot q_2 + 2 \cdot q_3 + 3 \cdot q_4 + 3 \cdot q_5 = 1.30。$$

(3) 就该问题而言, 顺序查找的性能优于折半查找。

371、顺序表的数据类型定义如下:

```
#define MAXSIZE 100
```

```
typedef int KeyType;
```

```
typedef struct
```

```
{
```

```
    KeyType key;
```

```
} RecType;
```

```
typedef RecType SeqList[MAXSIZE];
```

(1) 写出折半查找的递归算法, 函数原型如下:

```
int Binary(SeqList data, KeyType keyword, int low, int high);
```

```
// 查找成功返回下标, 失败返回-1
```

```
// data 是存放关键字的有序数组, keyword 为待查找关键字
```

```
// low 和 high 分别代表序列的最低和最高下标
```



(2) 分析此递归算法的时间复杂度。

答案：

(1) 折半查找的递归算法为：

```
int Binary(SeqList data, KeyType keyword, int low, int high)
{
    int mid;

    if (low <= high)
    {
        mid = (low + high) / 2;

        if (data[mid].key == keyword)

            return mid;

        else if (data[mid].key < keyword)

            return Binary(data, keyword, mid + 1, high);

        else

            return Binary(data, keyword, low, mid - 1);
    }

    else

        return -1;
}
```

(2) 该算法中的两个递归调用语句分别处于同一个条件判断语句的不同分支中，因此最多只有一个能执行，考虑到最坏情况下需要再次判断的序列也只是原长度的一半，依照递归算法可以归纳出递归方程如下：

$$\begin{cases} T(n) = 1 + T\left(\frac{n}{2}\right) & n > 1 \\ T(1) = 1 \end{cases}$$

其中递归出口和递归体中的 1 均代表一次基本操作，改为其他常量并不影响最后的时间复杂度结论，下面求解递归方程。

设  $n = 2^k$ ， $k \geq 1$ ，代入递归方程右边逐步展开可得：

$$f(n) = 1 + f(2^{k-1}) = 1 + 1 + f(2^{k-2}) = 1 + 1 + 1 + f(2^{k-3})$$

= .....

$$= 1 + 1 + \cdots + 1 + f(2^{k-k}) = k + 1 = \log_2 n + 1$$

因此算法时间复杂度为  $O(\log_2 n)$ 。

372、在索引顺序表中实现分块查找，在等概率查找情况下，其平均查找长度不仅与表中元素个数有关，而且与每块中元素个数有关。

答案： 正确

解析：因为索引顺序表中的查找过程分两步：首先在索引表中确定分块，然后在该分块中查找元素，分块的元素个数直接决定了索引表中的索引项数。

373、就平均查找长度而言，分块查找最小，折半查找次之，顺序查找最大。

答案： 错误

解析：就平均查找长度而言，折半查找 < 分块查找 < 顺序查找。

374、查找比较快，并且插入和删除操作也比较方便的查找方法是\_\_\_\_\_。

A、 分块查找

B、 折半查找

C、 顺序查找

D、 二分查找

答案： A

解析：折半查找虽然查找效率高，但是在有序顺序表中插入和删除的效率并不高；

顺序查找虽然插入删除比较方便，但是查找的效率较低；

二分查找是折半查找的同义词。

375、当采用分块查找时，数据的组织方式为\_\_\_\_\_。

A、 数据分成若干块，每块内数据有序

B、 数据分成若干块，每块中数据个数必须相同

C、 数据分成若干块，每块内数据有序，块间是否有序均可

D、数据分成若干块，每块内数据不必有序，但块间必须有序

答案： D

解析：分块查找只需要保证块间有序，就能在索引表中按分块查找，分块内数据是否有序并无强制要求。

376、长度为 255 的有序表采用分块查找，块的大小应取多少？

答案：

设分块查找的记录有  $n$  项，在索引表和分块中均采用顺序查找：

当分块的大小  $s = \boxed{\times}$  时，平均查找长度有最小值  $\boxed{\times} + 1$ ，

因此当  $n = 255$  时，每块长度为  $\boxed{\times} = 15$ ，平均查找长度为  $15 + 1 = 16$ 。

377、有 4000 条记录的表用分块查找法：

(1) 分成多少块最理想？ 每块的理想长度是多少？

(2) 如果每块长度为 80，平均查找长度是多少？

答案：

(1) 表长 4000， $\boxed{\times} = 64$ ， $63 \times 64 = 4032$ ；

因此分成 63 块，每块的理想长度为 64(最后一块长 32)。

(2) 如果每块长 80， $4000 / 80 = 50$ ，即有 50 个索引表项：

当索引表中用顺序查找确定块时， $ASL = (50 + 1) / 2 + (80 + 1) / 2 = 66$ ；

当索引表中用折半查找确定块时， $ASL \approx \log_2(50 + 1) - 1 + (80 + 1) / 2 = 45.2$ 。

378、一棵二叉排序树中，关键字  $n$  所在结点是关键字  $m$  所在结点的子孙，则\_\_\_\_\_。

A、  $n$  一定大于  $m$

B、  $n$  一定小于  $m$

C、  $n$  一定等于  $m$

D、  $n$  与  $m$  的大小关系不确定

答案： D

解析：显然  $n$  在  $m$  的左子树上就会小于  $m$ ， $n$  在  $m$  的右子树上就会大于  $m$ ，所以两者的大小并不确定，除非准确知道  $n$  在  $m$  的左子树或者右子树上。

379、要输出二叉排序树中结点的有序序列，则采用的遍历方法是\_\_\_\_\_。

A、 按层遍历

B、 先序遍历

C、 中序遍历

D、 后序遍历

答案： C

解析： 二叉排序树的关键字排列为：左子树的所有关键字都小于根的关键字，右子树的所有关键字都大于根的关键字，所以使用中序遍历就能得到有序序列。

380、 分别用以下序列生成二叉排序树，其中三个序列生成的二叉排序树是相同的，不同的序列是\_\_\_\_\_。

A、 (4, 1, 2, 3, 5)

B、 (4, 2, 5, 3, 1)

C、 (4, 5, 2, 1, 3)

D、 (4, 2, 1, 5, 3)

答案： A

解析：从空树开始，将各个关键字序列逐个插入二叉排序树，即可知 (4, 1, 2, 3, 5) 与其他三者不同。

381、关键字序列为(12, 7, 17, 11, 16, 2, 13, 9, 21, 4)：

- (1) 按元素的顺序依次插入一棵初始为空的二叉排序树，画出插入完成之后的二叉排序树；
- (2) 求等概率情况下，二叉排序树中查找成功和查找失败的平均查找长度；
- (3) 将该关键字序列排序构成有序表，求等概率情况下，对该有序表进行折半查找查找成功和查找失败时的平均查找长度。

答案：

- (1) 以下是该关键字序列构成的二叉排序树：



- (2) 该二叉排序树的查找性能：

$$ASL_{succ} = (1 \times 1 + 2 \times 2 + 4 \times 3 + 3 \times 4) / 10 = 29 / 10;$$

$$ASL_{unsucc} = (5 \times 3 + 6 \times 4) / 13 = 39 / 11。$$

- (3) 以下是该关键字排序后，进行折半查找的判定树：



$$ASL_{succ} = (1 \times 1 + 2 \times 2 + 4 \times 3 + 3 \times 4) / 10 = 29 / 10;$$

$$ASL_{unsucc} = (5 \times 3 + 6 \times 4) / 13 = 39 / 11;$$

两者凑巧一致。

382、输入一个正整数序列 (53, 17, 12, 66, 58, 70, 87, 25, 56, 60)：

- (1) 用此序列构造一棵二叉排序树，然后删除“70”，再插入“21”，画出最后的二叉排序树。
- (2) 依此二叉排序树，如何得到一个从大到小的有序序列？
- (3) 求等概率情况下，最后删除插入完成后，二叉排序树中查找成功和查找失败的平均查找长度。

答案：

- (1) 删除 70，再插入 21 后的二叉排序树为：



- (2) 使用逆中序遍历，即：“右子树，根，左子树”的次序遍历即可得到关键字的递减有序序列。

- (3) 该二叉排序树的查找性能：

$$ASL_{succ} = (1 \times 1 + 2 \times 2 + 4 \times 3 + 3 \times 4) / 10 = 29 / 10;$$

$$ASL_{unsucc} = (5 \times 3 + 6 \times 4) / 13 = 39 / 11。$$

383、设二叉排序树中关键字由 1 到 1000 的整数组成，如果要查找关键字为 363 的结点，下述各关键字序列中，哪些不可能是在二叉排序树中查找到的关键字序列？为什么？

- (1) 51, 250, 501, 390, 320, 340, 382, 363
- (2) 24, 877, 125, 342, 501, 623, 421, 363
- (3) 925, 202, 911, 240, 912, 45, 363



(4) 924, 220, 911, 244, 898, 258, 362, 363

答案:

序列(2)、(3) 不可能。

序列(2): 在查找到关键字 623 后, 接下来查找到的关键字只可能位于区间(501, 623) 或者区间(623, 877), 但是序列中接着出现的关键字是 421, 故不可能;

序列(3): 在查找到关键字 240 后, 接下来查找到的关键字只可能位于区间(202, 240) 或者区间(240, 911), 但是序列中接着出现的关键字是 912, 故不可能。

384、设有二叉排序树如图所示:



(1) 假定该二叉排序树初始为空, 能得到此二叉排序树的不同输入数据序列有多少个? 写出所有的数据输入序列, 按此序列插入时能得到该二叉排序树。

(2) 现在需要在该二叉排序中删除结点 *e*, 画出所有可行的方案删除后的二叉排序树, 并说明该删除方案中, 指针域更新的次数是多少?

答案:

(1) 不同的输入序列一共 4 个:

*a, g, e, f, b, d, c*

*a, g, e, b, f, d, c*

*a, g, e, b, d, f, c*

*a, g, e, b, d, c, f*

(2) 删除  $e$  结点的方案有如下 4 种，但实际运用中出于查找的性能，一般只用到前面 2 种的中序前驱或者后继替换：

1) 中序前驱  $d$  的值替换  $e$ ,  $c$  结点成为  $b$  的右孩子，指针域更新 1 次：



2) 中序后继  $f$  的值替换  $e$ ,  $e$  结点的右指针置空，指针域更新 1 次：



3)  $b$  结点替换  $e$  结点,  $f$  结点成为  $d$  结点的右子树，指针域更新 2 次：



4)  $f$  结点替换  $e$  结点,  $b$  结点成为  $f$  结点的左子树，指针域更新 2 次：

因为  $f$  结点是叶子，所以最终形态恰好和 2) 一致。

解析：(1) 显然除了结点  $f$  以外，其他所有结点都在不同层次，因此输入序列前 3 个数据次序一定是  $a, g, e$ ，后面除了  $f$  以外， $b, d, e$  一样是有序排列， $f$  可以安排在后面 3 个数据的前中后任意的 4 个位置。

385、二叉排序树的存储结构类型定义如下：

```
typedef int DataType;
```

```
typedef struct node
```

```
{
```

```
    DataType data;        // data 是数据域
```

```
struct node *lchild, *rchild; // 分别指向左右孩子
```

```
} BinNode;
```

```
typedef BinNode *BinTree;
```

阅读下列算法，并回答问题。

```
void f(BinTree root, int left, int right)
```

```
{
```

```
if (root)
```

```
{
```

```
f(root->lchild, left, right);
```

```
if (root->data >= left && root->data < right)
```

```
printf("%d ", root->data);
```

```
f(root->rchild, left, right);
```

```
}
```

```
}
```

(1) 设二叉排序树结构如下图所示， $bt$  是指向该二叉排序树根结点的指针，写出执行  $f(bt, 14, 30)$  的输出结果。



(2) 说明该算法的功能。

答案：

(1) 15 19 23 28 29 ；

(2) 从小到大顺序输出二叉排序树  $bt$  中所有  $data$  域在  $[left, right)$  的数据。

386、写出在二叉排序树中删除一个结点的算法，使删除后保持二叉排序树。  
如果待删除结点的度为 2，使用中序前驱替代。设删除结点由指针  $p$  所指，其  
双亲结点由指针  $parent$  所指。二叉排序树的数据类型和删除函数的原型定义  
如下：

```
typedef int DataType;
```

```
typedef struct node
```

```
{
```

```
    DataType data;        // data 是数据域
```

```
    struct node *lchild, *rchild; // 分别指向左右孩子
```

```
} BinNode;
```

```
typedef BinNode *BinTree;
```

```
void Delete(node *parent, node *p); // 函数原型
```

答案：

```
void Delete(node *parent, node *p)
```

```
{
```

```
    node *q;
```

```
    if (p->lchild && p->rchild)
```

```
    {                // p 的度为 2
```

```
        q = parent = p;
```

```
        for (p = p->lchild; p->rchild; parent = p, p = p->rchild);
```

```

    q->data = p->data;

    if (parent != q)

        parent->rchild = p->lchild;

    else

        parent->lchild = p->lchild;

    }

else if (!p->lchild && !p->rchild)

{
    // p 是叶子

    if (p == parent->lchild)

        parent->lchild = 0;

    else

        parent->rchild = 0;

    }

else if (!p->rchild)

{
    // p 只有左子树

    if (p == parent->lchild)

        parent->lchild = p->lchild;

    else

        parent->rchild = p->lchild;

    }

else

{
    // p 只有右子树

```

```

    if (p == parent->lchild)

        parent->lchild = p->rchild;

    else

        parent->rchild = p->rchild;

    }

    free(p);

}

```

387、使用平衡二叉树的目的是为了节省存储空间。

答案： 错误

解析：在  $n$  个关键字序列随机分布的前提下，建立的二叉排序树差不多一半场合会导致查找性能近似于顺序查找的  $O(n)$  查找长度，使用平衡二叉树的原因就是可以将平均查找长度限制在  $1.5\log_2 n$  以内。

另外，平衡二叉树的结点中需要增加数据域“平衡因子”，因此就结点的存储密度而言，使用平衡二叉树需要更多的存储空间。

388、对于平衡二叉树的任意结点，其左子树的高度不得超过其右子树高度加 1。

答案： 错误

解析：说法不全面，精确地说是高度差的绝对值不超过 1，而不是仅仅小于等于 1。

389、折半查找所对应的判定树，是一棵平衡的二叉排序树。

答案： 正确

解析：对于折半查找判定树而言，类似于完全二叉树，任意一个结点的左右子树的高度要么相等，要么只是右边比左边多 1，并且叶子结点最多只是在最下面 2 层。

390、用关键字序列(27, 16, 75, 38, 51, 18, 69, 2) 从空树开始构造平衡二叉树，则首次出现的最小不平衡子树的根（即离插入结点最近且平衡因子的绝对值为 2 的结点）为\_\_\_\_\_。

A、 27

B、 38

C、 51

D、 75

答案： D

解析：当插入关键字 51 时，该平衡二叉树首次出现不平衡，如下图，其中最小不平衡子树的根为 75。



391、用关键字序列(46, 88, 45, 39, 70, 58, 93, 10, 66, 34) 建立一棵平衡二叉树：

(1) 画出该树；

(2) 求在等概率情况下查找成功和查找失败的平均查找长度。

答案：

(1) 以下是该关键字序列构建的平衡二叉树：



其中的旋转类型为：插入 58 向右单旋转、插入 10 向右单旋转

(2) 该二叉排序树的查找性能：

$$ASL_{succ} = (1 \times 1 + 2 \times 2 + 4 \times 3 + 3 \times 4) / 10 = 29 / 10;$$

$$ASL_{unsucc} = (5 \times 3 + 6 \times 4) / 13 = 39 / 13。$$

392、依次输入表(30, 15, 28, 20, 24, 10, 12, 68, 35, 50, 46, 55) 中的元素，构建一棵平衡二叉树：

(1) 画出构建的平衡二叉树；

(2) 假定每个元素的查找概率相等，试计算该平衡二叉树的平均查找长度；

(3) 画出依次删除关键字 35, 46, 20, 50 后的平衡二叉树，约定如果删除度为 2 的结点，用其中序后继替代。

答案：

(1) 构建的平衡二叉树：



其中的旋转类型为：28 双旋转、24 单旋转、10 单旋转、12 双旋转、50 单旋转、55 单旋转。

(2) 该平衡二叉树的平均查找长度：

$$ASL_{succ} = (1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) / 12 = 37 / 12;$$

$$ASL_{unsucc} = (3 \times 3 + 10 \times 4) / 13 = 49 / 13。$$

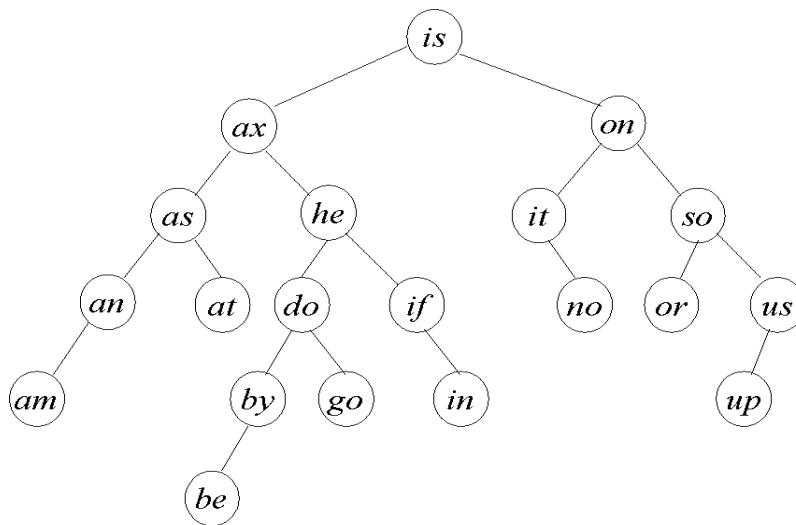
(3) 删除关键字 35, 46, 20, 50 后的平衡二叉树：





其中删除 35 发生了双旋转、删除 50 发生了单旋转。

393、已知某关键字为字符串型的平衡二叉排序树如下图：



- (1) 求在等概率的情况下，查找成功和查找失败的平均查找长度；
- (2) 如果约定删除度为 2 的结点后，用中序前驱替代，则画出删除关键字 *on* 后的平衡二叉树。

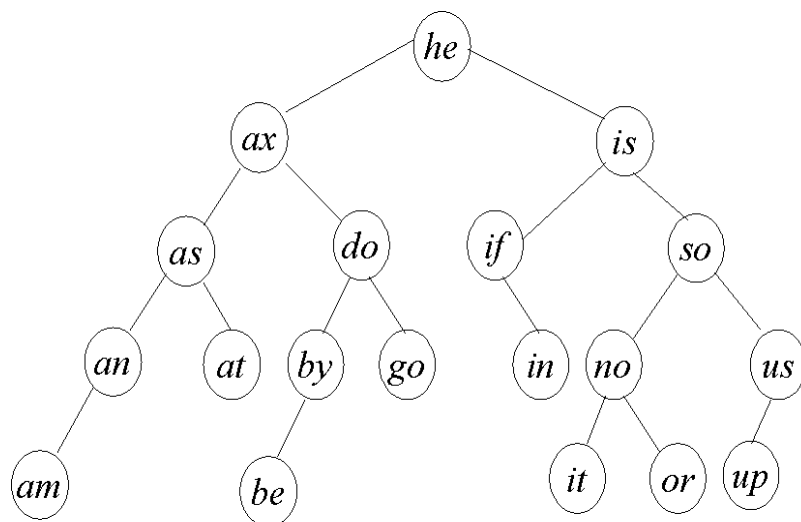
答案：

- (1) 该平衡二叉树的平均查找长度：

$$ASL_{succ} = (1 \times 1 + 2 \times 2 + 4 \times 3 + 7 \times 4 + 5 \times 5 + 1 \times 6) / 20 = 76 / 20;$$

$$ASL_{unsucc} = (1 \times 3 + 9 \times 4 + 9 \times 5 + 2 \times 6) / 21 = 96 / 21。$$

- (2) 删除关键字 *on* 后的平衡二叉树：



其中的旋转为：

*no* 替换 *on* 后，引发结点 *no* 和结点 *so* 的向左单旋转，结果使根结点 *is* 的右子树高度比其左子树高度小 2；

继续引发结点 *is*、结点 *ax* 和结点 *he* 的先左后右双旋转。

394、一棵具有  $m$  层 ( $m \geq 0$ ) 的平衡二叉树最多有多少个结点，至少有多少个结点？

答案：

最多情况即为满二叉树时，也就是有  $2^m - 1$  个结点；


最少结点个数则为  $F(m + 2) - 1$ ，其中的  $F$  函数为 *Fibonacci* 序列，从  $F(1)$  开始依次是 1, 1, 2, 3, 5, 8, 13, ...。

395、给定四个关键字 1, 2, 3, 4：

(1) 详细说明能构造出几种不同的二叉排序树？

(2) 详细说明能构造出几种不同的平衡二叉树?

答案:

(1) 等价于已知结点个数的不同形态二叉树计数: 设二叉树中结点数为  $n$ , 则不同形态二叉树的总数为 。当  $n = 4$  时, 有 14 种不同形态。

(2) 按照平衡二叉树的性质, 4 个结点的平衡二叉树的高度一定为 3, 考虑到平衡二叉树平衡因子的要求, 因此根结点一定同时存在非空的左右子树 (或者说, 最上两层有 3 个结点), 于是第 4 个结点可行的位置只有 4 个, 也就是, 一共能构造出 4 种。

解析:

396、假设一棵平衡二叉树的每个结点都标明了平衡因子  $bf$ , 设计算法, 用尽量少的时间求平衡二叉树的高度, 约定结点的平衡因子为左子树的高度减去右子树的高度, 平衡二叉树的存储结构类型和函数的原型定义如下:

```
typedef int DataType;
```

```
typedef struct node
```

```
{
```

```
    DataType key;        // key 是数据域
```

```
    int bf;              // bf 是平衡因子
```

```
    struct node *lchild, *rchild; // 分别指向左右孩子
```

```
} BinNode;
```

```
typedef BinNode *BinTree;
```

```
int Depth(BinTree root); // 求平衡二叉树深度函数原型, root 指向根
```

答案：

由于每个分支结点内均有平衡因子，因此只要一直沿着该平衡二叉树的左边到底即可得知整个平衡二叉树的高度，以下是递归算法：

```
int Depth(BinTree root)

{

    int depth = 0, left;

    if (root)

        {

            left = Depth(root->left);

            depth = ((root->bf == -1 ? 1 : 0) + left) + 1;

        }

    return depth;

}
```

397、在 15 阶 B- 树中，各结点的关键字最少为 7 个，最多为 14 个。

答案： 错误

解析：按照 B- 树的规定，除根结点以外，15 阶 B- 树的其他结点的关键字必须满足此要求，但是根结点最少可以只有 1 个关键字。

398、在下面 3 阶 B- 树中插入关键字 65 后，其根结点内的关键字是\_\_\_\_\_。



A、 53    90

B、 53

C、 90

D、 65

答案： D

解析：在关键字 61 和 70 之间插入 65 后，引起该结点分裂，关键字 65 上升，插入到上层结点（也就是当前的根结点）的 53 和 90 之间，引发当前根结点继续分裂，65 继续上升到上层结点（也就是新生成的根结点）中。

399、在一棵 5 阶 B- 树中，每个非根结点中所含关键字的个数最少是\_\_\_\_\_。

A、 1

B、 2

C、 3

D、 4

答案： B

解析：按照 B- 树的规定，5 阶 B- 树中非根结点中关键字个数最少为  $\lceil \frac{5}{2} \rceil - 1 = 2$ 。

400、下列关于  $m$  ( $m \geq 3$ ) 阶 B- 树的叙述中，错误的是\_\_\_\_\_。

A、 每个结点至多有  $m$  个关键字

- B、 根结点至少有 1 个关键字
- C、 所有的叶子结点均在同一层上
- D、 根结点至少有 2 棵子树

答案： A

解析：  $m$  阶 B- 树中每个结点的关键字个数最多为  $m - 1$  个。

401、 下列关于  $m$  ( $m \geq 3$ ) 阶 B- 树的叙述中， 错误的是\_\_\_\_\_。

- A、 每个结点至多有  $m$  棵子树
- B、 结点内部的关键字可以无序
- C、 插入关键字时， 如果有结点分裂， 则增加了树的高度
- D、 删除关键字时， 如果有结点合并， 则降低了树的高度

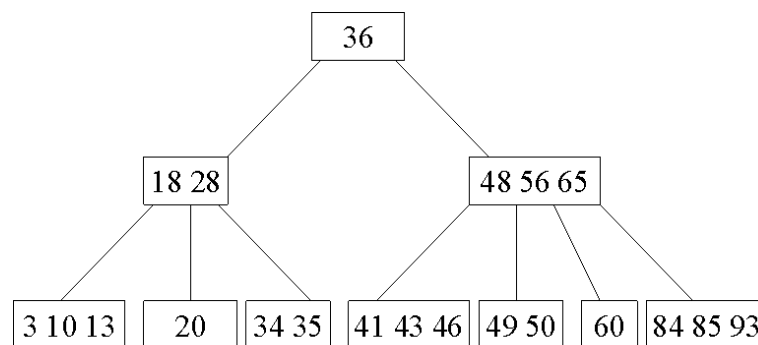
答案： B

解析： B 树结点中关键字只有有序才能保证多路分支查找的正确进行。

402、从 4 阶 B- 树的空树开始，依次插入如下关键字：48, 43, 36, 18, 56, 28, 50, 60, 65, 46, 93, 34, 10, 85, 20, 41, 3, 35, 84, 49, 13, 画出最后的 B- 树。

答案：

构建的 4 阶 B- 树：



其中在插入关键字 18、50、65、10、85、35 时有结点分裂。

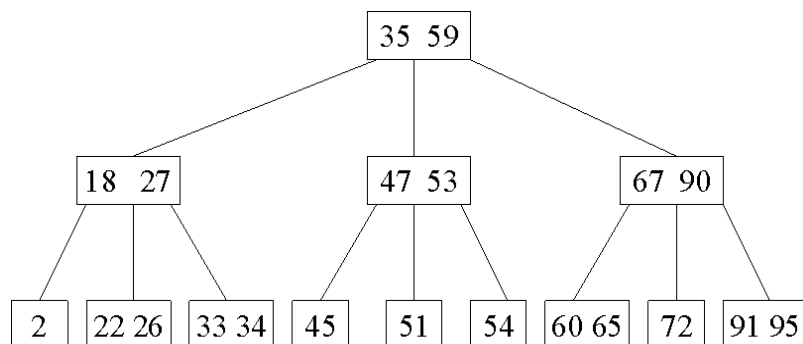
403、设有一棵空的 3 阶 B-树，依次插入关键字  
59, 27, 90, 2, 35, 53, 67, 34, 65, 54, 72, 47, 33, 95,  
91, 51, 18, 22, 26, 60, 45:

(1) 画出该 B- 树；

(2) 从中依次删除关键字 2, 27, 35 后，画出 B- 树。

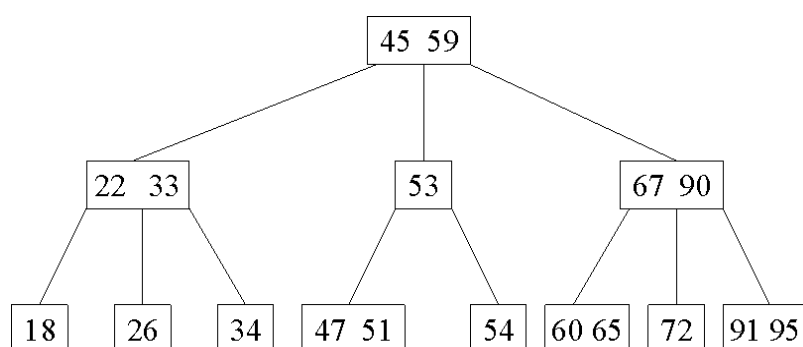
答案：

(1) 构建的 3 阶 B- 树：



其中在插入关键字 90、35、34、65、47、95、22、45 时有结点分裂；

(2) 删除关键字 2, 27, 35 后的 B- 树：



其中在删除关键字 45 时有结点合并。

解析：

404、对于 3 阶 B- 树：

- (1) 含 9 个叶子结点的 3 阶 B-树中至少有多少个非叶子结点？
- (2) 含 10 个叶子结点的 3 阶 B-树中至多有多少个非叶子结点？

答案：

(1) 在 3 阶 B- 树中，每个结点最多有 3 个孩子结点，因此 9 个叶子的双亲结点数为  $9 / 3 = 3$ ，3 个双亲结点有  $3 / 3 = 1$  个双亲，因此至少有  $3 + 1 = 4$  个非叶子结点。



(2) 在 3 阶 B- 树中，每个结点最少有 2 个孩子结点，因此 10 个叶子的双亲结点数为  $10 / 2 = 5$ ，5 个双亲结点的双亲个数为  $\lceil 5 / 2 \rceil = 3$ ，3 个结点有 1 个双亲结点，因此至多有  $5 + 2 + 1 = 8$  个非叶子结点。

405、设 3 阶 B- 树的高度为 6:

(1) 至多有多少个结点? 其中有多少个关键字?

(2) 至少有多少个结点? 其中有多少个关键字?

答案:

(1) 在 3 阶 B- 树结点中，最多有 2 个关键字、有 3 个孩子结点，因此最多结点个数为： $1 + 3 + 3^2 + \dots + 3^5 = 364$  个，关键字个数为： $2 \times 364 = 728$  个。

(2) 在 3 阶 B- 树中，除根结点以外，每个结点中最少有 1 个关键字，最少有 2 个孩子结点，根结点最少有 1 个关键字，最少有 2 个孩子结点，因此最少结点数为： $1 + 2 + 2 \times 2^1 + 2 \times 2^2 + 2 \times 2^3 + 2 \times 2^4 = 63$  个；

最少关键字个数为： $1 + 2 \times 1 + 2 \times 2^1 + 2 \times 2^2 + 2 \times 2^3 + 2 \times 2^4 = 63$  个。

因为是 3 阶 B- 树，一个结点中最少只有 1 个关键字，此时结构上实质退化为和满二叉树一样。

406、在下列查找方法中，平均查找长度与关键字数量无直接关系的是\_\_\_\_\_。

A、 顺序查找

B、 分块查找

C、 散列查找

D、 基于 B 树的查找

答案： C

解析： 散列查找的性能与装填因子直接相关，并不直接依赖于关键字的数量。

407、选择好的散列函数就可以避免冲突的发生。

答案： 错误

解析： 无论什么散列函数，只要待插入的关键字序列完全随机，无规律可循，并没有一个万能的散列函数可以保证永远绝对无冲突。

408、散列函数的选取平方取中法最好。

答案： 错误

解析： 如果关键字序列完全随机，常见散列函数中，平方取中法的理论性能最好，但是对于某任意关键字集合，并不能保证完全随机，因此平方取中法就不一定适合了。

409、哈希函数越复杂越好，因为这样随机性好，冲突概率小。

答案： 错误

解析： 只要能将待处理的关键字集合简单均匀分散开就是好的散列函数，并不一定是复杂的函数。

410、下列各关于散列函数的说法中，正确的是\_\_\_\_\_。

A、 散列函数越复杂越好

B、 散列函数越简单越好

C、 用除留余数法构造的散列函数是最好的

D、 在冲突尽可能少的情况下，散列函数越简单越好

答案： D

解析： 散列函数设计并无通则，只要能将关键字集合尽量分散，少冲突甚至不冲突就是好的散列函数，为提高算法效率，运算当然是越简单越好。

411、散列表的平均查找长度与处理冲突的方法无关。

答案： 错误

解析： 处理冲突的方法直接影响到散列表的查找性能，一般而言，使用开放地址法解决冲突散列表的性能低于开散列表。

412、开散列表和闭散列表的装填因子都可大于、等于或者小于 1。

答案： 错误

解析：只有开散列表的装填因子可以大于 1，闭散列表由于开放所有散列地址，装填因子最多只是 1。

413、用线性探查法解决突出时，同义词在散列表中是相邻的。

答案： 错误

解析：用线性探查法解决冲突时，只有相邻位置空闲时，同义词才会存放在相邻位置。

414、在开散列表中不会出现堆积现象。

答案： 正确

解析：所谓堆积指的是因为开放地址解决冲突，使得非同义词占用了其散列位置，导致同义词和非同义词连续堆积，冲突的概率大幅度上升；

开散列表中同义词存放在同一个链表，并不会占用其他位置，避免了堆积产生。

415、散列表的查找效率主要取决于所选择的散列函数与处理冲突的方法。

答案： 正确

解析：散列查找的过程就是由散列函数与处理冲突两者相结合。

416、采用线性探查法处理散列时的冲突，当从哈希表删除一个记录时，不应将这个记录从所在位置物理删除，因为这会影响以后的查找。

答案： 正确

解析：因为一旦物理删除，会造成查找过程中认为已经探查到空位而终止，冲突发生时存放后面探查位置的元素无法找到。

417、在散列查找中，“比较”操作一般也是不可避免的。

答案： 正确

解析：在散列函数求得位置后，接下来的探查过程实质上就是关键字的比较操作。

418、设一组记录的关键字为(12, 22, 10, 20, 88, 27, 54, 11)，散列函数为  $H(key) = key \% 11$ ，用链地址法解决冲突，则散列地址为 0 的链中结点数是\_\_\_\_\_。

- A、 1
- B、 2
- C、 3
- D、 4

答案： C

解析：链地址法解决冲突时，同义词插入在同一个链表中，关键字 22, 88, 11 的散列函数值均为 0。

419、设散列表长  $m = 14$ ，散列函数  $H(key) = key \% 11$ 。表中已保存 4 个关键字： $addr(15) = 4$ ， $addr(38) = 5$ ， $addr(61) = 6$ ， $addr(84) = 7$ ，其余地址均为空。保存关键字 49 时存在冲突，采用线性探查法来处理。则查找关键字 49 时的探查次数是\_\_\_\_\_。

- A、 2
- B、 3
- C、 4
- D、 5

答案： C

解析：线性探查法解决冲突是从散列位置开始，顺序往后挨个探查空位， $H(49) = 5$ ，发生冲突， $d_1 = 6$ ， $d_2 = 7$  也发生冲突， $d_3 = 8$  才是空位，当查找该关键字时，同样依照冲突解决的方法从散列位置往后探查，因此需要顺序比较下标 5 ~ 8 的关键字才能查找成功。

420、在散列查找中处理冲突时，可以采用开放定址法。下列方法中，不属于开放定址法的是\_\_\_\_\_。

- A、 线性探查法
- B、 平方探查法
- C、 双重散列法
- D、 链地址法

答案： D

解析： 开放地址法解决冲突是将所有的散列位置全部开放，链地址法则是将同义词放在同一个链表中。

421、设散列函数为  $H(key) = key \% 11$ ，散列地址空间为  $0 \dots 10$ ，对关键字序列(27, 13, 55, 32, 18, 49, 24, 38, 43) 用线性探查法解决冲突，构建散列表。现已有前 4 个关键字构建的散列表如下，将剩余 5 个关键字填入表中合适的位置。

|      |    |   |    |   |   |    |   |   |   |   |    |
|------|----|---|----|---|---|----|---|---|---|---|----|
| 散列地址 | 0  | 1 | 2  | 3 | 4 | 5  | 6 | 7 | 8 | 9 | 10 |
| 关键字  | 55 |   | 15 |   |   | 27 |   |   |   |   | 32 |

答案：

$18 \% 11 = 7$ ，无冲突；

$49 \% 11 = 5$ ，有冲突

$d_1: (5 + 1) \% 11 = 6$ ，无冲突；

$24 \% 11 = 2$ ，有冲突

$d_1: (2 + 1) \% 11 = 3$ ，无冲突；

$38 \% 11 = 5$ ，有冲突

$d_1: (5 + 1) \% 11 = 6$ ，有冲突

$d_2: (5 + 2) \% 11 = 7$ ，有冲突

$d_3: (5 + 3) \% 11 = 8$ ，无冲突；

$43 \% 11 = 10$ ，有冲突

$d_1: (10 + 1) \% 11 = 0$ ，有冲突

$d_2: (10 + 2) \% 11 = 1$ ，无冲突。

构造的散列表示意如下：

|      |    |    |    |    |   |    |    |    |    |   |    |
|------|----|----|----|----|---|----|----|----|----|---|----|
| 散列地址 | 0  | 1  | 2  | 3  | 4 | 5  | 6  | 7  | 8  | 9 | 10 |
| 关键字  | 55 | 43 | 15 | 24 |   | 27 | 49 | 18 | 38 |   | 32 |

422、设散列表的地址范围为  $0 \sim 17$ ，散列函数为  $H(key) = key \% 16$ ，其中  $key$  为关键字值，用线性探查再散列法解决冲突，关键字序列为 (10, 24, 32, 17, 31, 30, 46, 47, 40, 63, 49)：

- (1) 画出散列表的示意图；
- (2) 查找关键字 63，需要依次与哪些关键字进行比较？
- (3) 查找关键字 60，需要依次与哪些关键字比较？

答案：

$10 \% 16 = 10$ ，无冲突；

$24 \% 16 = 8$ ，无冲突；

$32 \% 16 = 0$ ，无冲突；

$17 \% 16 = 1$ ，无冲突；

$31 \% 16 = 15$ ，无冲突；

$30 \% 16 = 14$ ，无冲突；

$46 \% 16 = 14$ ，有冲突

$d_1: (14 + 1) \% 18 = 15$ ，有冲突

$d_2: (14 + 2) \% 18 = 16$ ，无冲突；



$47 \% 16 = 15$ ，有冲突

$d_1: (15 + 1) \% 18 = 16$ ，有冲突

$d_2: (15 + 2) \% 18 = 17$ ，无冲突；

$40 \% 16 = 8$ ，有冲突

$d_1: (8 + 1) \% 18 = 9$ ，无冲突；

$63 \% 16 = 15$ ，有冲突

$d_1: (15 + 1) \% 18 = 16$ ，有冲突

$d_2: (15 + 2) \% 18 = 17$ ，有冲突

$d_3: (15 + 3) \% 18 = 0$ ，有冲突

$d_4: (15 + 4) \% 18 = 1$ ，有冲突

$d_5: (15 + 5) \% 18 = 2$ ，无冲突；

$49 \% 16 = 1$ ，有冲突

$d_1: (1 + 1) \% 18 = 2$ ，有冲突

$d_2: (1 + 2) \% 18 = 3$ ，无冲突。

(1) 构造的散列表示意如下：

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

(2) 查找关键字 63，首先用散列函数计算出散列下标为 15，但是该位置元素并不是 63，也不是空位，继续用线性探查法求得下一个下标 16，同样也不是，...，直到探查到下标 2 才查找成功，依次与 31, 46, 47, 32, 17, 63 相比较。

(3) 查找关键字 60，首先用散列函数计算出散列下标为 12，该位置为空位，表明查找失败，没有和关键字比较。

423、一组关键字(55, 76, 44, 32, 64, 82, 20, 16, 43)，用散列函数  $H(key) = key \% 11$  将记录散列到散列表 HT[0..12] 中去，用线性探查法解决冲突。

(1) 画出存入所有记录后的散列表。

(2) 求等概率情况下，查找成功时的平均查找长度。

答案：

55 % 11 = 0，无冲突；

76 % 11 = 10，无冲突；

44 % 11 = 0，有冲突

$d_1: (0 + 1) \% 13 = 1$ ，无冲突；

32 % 11 = 10，有冲突

$d_1: (10 + 1) \% 13 = 11$ ，无冲突；

64 % 11 = 9，无冲突；

82 % 11 = 5，无冲突；

20 % 11 = 9，有冲突

$d_1: (9 + 1) \% 13 = 10$ ，无冲突

$d_2: (9 + 2) \% 13 = 11$ ，无冲突

$d_3: (9 + 3) \% 13 = 12$ ，无冲突；

16 % 11 = 5，有冲突

$d_1: (5 + 1) \% 13 = 6$ ，无冲突；

43 % 11 = 10，有冲突

$d_1: (10 + 1) \% 13 = 11$ ，无冲突

$d_2: (10 + 2) \% 13 = 12$ ，无冲突

$d_3: (10 + 3) \% 13 = 0$ ，无冲突

$d_4: (10 + 4) \% 13 = 1$ ，无冲突

$d_5: (10 + 5) \% 13 = 2$ ，无冲突。

(1) 构造的散列表示意如下：

|      |    |    |    |   |   |    |    |   |   |    |    |    |    |
|------|----|----|----|---|---|----|----|---|---|----|----|----|----|
| 散列地址 | 0  | 1  | 2  | 3 | 4 | 5  | 6  | 7 | 8 | 9  | 10 | 11 | 12 |
| 关键字  | 55 | 44 | 43 |   |   | 82 | 16 |   |   | 64 | 76 | 32 | 20 |
| 比较次数 | 1  | 2  | 6  |   |   | 1  | 2  |   |   | 1  | 1  | 2  | 4  |

(2)  $ASL_{succ} = (1 + 2 + 6 + 1 + 2 + 1 + 1 + 2 + 4) / 9 = 20 / 9$ 。

424、对关键字序列(22, 41, 53, 46, 30, 13, 1, 67, 51)，采用散列函数  $H(key) = (3 \times key) \% 13$ ，在散列地址空间  $[0 \dots 12]$  中利用平方探查法处理冲突：

- (1) 画出散列表的示意图；
- (2) 求装填因子；
- (3) 求等概率下查找成功的平均查找长度；
- (4) 求等概率下查找不成功的平均查找长度。

答案：

$(3 \times 22) \% 13 = 1$ ，无冲突；

$(3 \times 41) \% 13 = 6$ ，无冲突；

$(3 \times 53) \% 13 = 3$ ，无冲突；

$(3 \times 46) \% 13 = 8$ ，无冲突；

$(3 \times 30) \% 13 = 12$ ，无冲突；

$(3 \times 13) \% 13 = 0$ ，无冲突；

$(3 \times 1) \% 13 = 3$ ，有冲突

$d_1: (3 + 1^2) \% 13 = 4$ ，无冲突；

$(3 \times 67) \% 13 = 6$ ，有冲突

$d_1: (6 + 1^2) \% 13 = 7$ ，无冲突；

$(3 \times 51) \% 13 = 10$ ，无冲突。

(1) 构造的散列表示意如下：

|        |    |    |   |    |   |   |    |    |    |   |    |    |    |
|--------|----|----|---|----|---|---|----|----|----|---|----|----|----|
| 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 |
| 成功比较次数 | 1  | 1  |   | 1  | 2 |   | 1  | 2  | 1  |   | 1  |    | 1  |
| 失败探查次数 | 5  | 2  | 1 | 3  | 2 | 1 | 3  | 4  | 2  | 1 | 2  | 1  | 3  |

(2) 装填因子  $\alpha = 9 / 13 \approx 0.69$ ;

(3)  $ASL_{succ} = (1 + 1 + 1 + 2 + 1 + 2 + 1 + 1 + 1) / 9 = 11 / 9$ ;

(4)  $ASL_{unsucc} = (5 + 2 + 1 + 3 + 2 + 1 + 3 + 4 + 2 + 1 + 2 + 1 + 3) / 13 = 30 / 13$ 。

425、已知关键字序列为(19, 14, 23, 01, 68, 20, 84, 27, 55, 11, 10, 79)，散列函数为  $H(key) = key \% 13$ ，散列表的地址空间为  $0 \sim 15$ ，用线性探查再散列处理冲突：

(1) 假设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；

(2) 假设每个关键字的查找概率相等，求查找失败时的平均查找长度。

答案:

$19 \% 13 = 6$ , 无冲突;

$14 \% 13 = 1$ , 无冲突;

$23 \% 13 = 10$ , 无冲突;

$01 \% 13 = 1$ , 有冲突

$d_1: (1 + 1) \% 16 = 2$ , 无冲突;

$68 \% 13 = 3$ , 无冲突;

$20 \% 13 = 7$ , 无冲突;

$84 \% 13 = 6$ , 有冲突

$d_1: (6 + 1) \% 16 = 7$ , 有冲突

$d_2: (6 + 2) \% 16 = 8$ , 无冲突;

$27 \% 13 = 1$ , 有冲突

$d_1: (1 + 1) \% 16 = 2$ , 有冲突

$d_2: (1 + 2) \% 16 = 3$ , 无冲突

$d_3: (1 + 3) \% 16 = 4$ , 无冲突;

$55 \% 13 = 3$ , 有冲突

$d_1: (3 + 1) \% 16 = 4$ , 有冲突

$d_2: (3 + 2) \% 16 = 5$ , 无冲突;

$11 \% 13 = 11$ , 无冲突;

$10 \% 13 = 10$ , 有冲突

$d_1: (10 + 1) \% 16 = 11$ , 有冲突

$d_2: (10 + 2) \% 16 = 12$ , 无冲突;

$79 \% 13 = 1$ , 有冲突

$d_1: (1 + 1) \% 16 = 2$ , 有冲突

$d_2: (1 + 2) \% 16 = 3$ , 有冲突

.....

$d_8: (1 + 8) \% 16 = 9$ , 无冲突。

构造的散列表示意如下:

|        |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|--------|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 散列地址   | 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| 关键字    |   | 14 | 01 | 68 | 27 | 55 | 19 | 20 | 84 | 79 | 23 | 11 | 10 |    |    |    |
| 成功比较次数 |   | 1  | 2  | 1  | 4  | 3  | 1  | 1  | 3  | 9  | 1  | 1  | 3  |    |    |    |
| 失败探查次数 | 1 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  |    |    |    |

(1)  $ASL_{succ} = (1 + 2 + 1 + 4 + 3 + 1 + 1 + 3 + 9 + 1 + 1 + 3) / 12 = 30 / 12 = 2.5$ ;

(2)  $ASL_{unsucc} = (1 + 2 + \dots + 13) / 13 = 7$ 。

426、设有一组关键字(09, 01, 23, 14, 55, 20, 84, 27), 散列函数为  $H(key) = key \% 7$ , 表长为 10, 用没有负数的平方探查再散列方法解决冲突,  $H_i = (H(key) + d_i) \% 10$ , 其中  $d_i = 1^2, 2^2, 3^2, \dots$ :

(1) 画出散列表的示意图;

(2) 假设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。

答案:

$09 \% 7 = 2$ , 无冲突;

$01 \% 7 = 1$ , 无冲突;

$23 \% 7 = 2$ ，有冲突

$$d_1: (2 + 1^2) \% 10 = 3, \text{无冲突};$$

$14 \% 7 = 0$ ，无冲突；

$55 \% 7 = 6$ ，无冲突；

$20 \% 7 = 6$ ，有冲突

$$d_1: (6 + 1^2) \% 10 = 7, \text{无冲突};$$

$84 \% 7 = 0$ ，有冲突

$$d_1: (0 + 1^2) \% 10 = 1, \text{有冲突}$$

$$d_2: (0 + 2^2) \% 10 = 4, \text{无冲突};$$

$27 \% 7 = 6$ ，有冲突

$$d_1: (6 + 1^2) \% 10 = 7, \text{有冲突}$$

$$d_2: (6 + 2^2) \% 10 = 0, \text{有冲突}$$

$$d_3: (6 + 3^2) \% 10 = 5, \text{无冲突}。$$

(1) 构造的散列表示意如下：

|        |    |    |    |    |    |    |    |    |   |   |
|--------|----|----|----|----|----|----|----|----|---|---|
| 散列地址   | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8 | 9 |
| 关键字    | 14 | 01 | 09 | 23 | 84 | 27 | 55 | 20 |   |   |
| 成功比较次数 | 1  | 1  | 1  | 2  | 3  | 4  | 1  | 2  |   |   |

(2) 查找成功时的平均查找长度：

$$ASL_{succ} = (1 + 1 + 1 + 2 + 3 + 4 + 1 + 2) / 8 = 15 / 8。$$

427、已知散列表的长度为 11，散列函数为  $H(key) = key \% 11$ ，散列表的当前状态如下：

|     |   |   |   |   |   |    |    |    |   |   |    |
|-----|---|---|---|---|---|----|----|----|---|---|----|
| 下标  | 0 | 1 | 2 | 3 | 4 | 5  | 6  | 7  | 8 | 9 | 10 |
| 关键字 |   |   |   |   |   | 60 | 17 | 29 |   |   |    |

现准备插入关键字 38：

- (1) 如果用线性探查法解决冲突，则 38 所在位置的下标是什么？
- (2) 如果用平方探查法解决冲突，则 38 所在位置的下标是什么？
- (3) 以上两种方法中，各需要多少次探查次数？

答案：

$38 \% 11 = 5$ ，该位置有冲突。

- (1) 线性探查法解决冲突：

$$d_1: (5 + 1) \% 11 = 6, \text{ 有冲突}$$

$$d_2: (5 + 2) \% 11 = 7, \text{ 有冲突}$$

$$d_3: (5 + 3) \% 11 = 8, \text{ 无冲突,}$$

存放的下标为 8，探查次数为 4 次；

- (2) 平方探查法解决冲突：

$$d_1: (5 + 1^2) \% 11 = 6, \text{ 有冲突}$$

$$d_2: (5 - 1^2) \% 11 = 4, \text{ 无冲突,}$$

存放的下标为 4，探查次数为 3 次。



428、设一组数据为(1, 10, 55, 14, 68, 11, 27, 23, 29)，散列函数是  $H(key) = key \% 13$ ，用链地址法解决冲突，设散列表的大小为 13(0..12)，试画出插入上述数据后的散列表。

答案：

$1 \% 13 = 1$ ，无冲突；

$10 \% 13 = 10$ ，无冲突；

$55 \% 13 = 3$ ，无冲突；

$14 \% 13 = 1$ ，有冲突；

$68 \% 13 = 3$ ，有冲突；

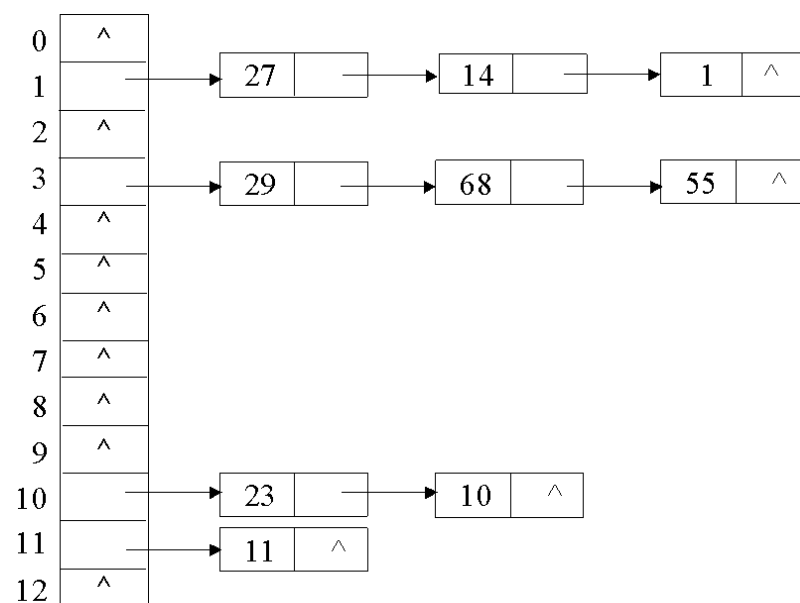
$11 \% 13 = 11$ ，无冲突；

$27 \% 13 = 1$ ，有冲突；

$23 \% 13 = 10$ ，有冲突；

$29 \% 13 = 3$ ，有冲突。

构造的散列表如下图：



429、设散列函数  $H(key) = (3 \times key) \% 11$ ，散列地址空间为  $0 \sim 10$ ，对关键字序列 (32, 13, 49, 24, 38, 21, 4, 12)，按下述两种解决冲突的方法构造散列表

(1) 线性探查再散列，求等概率下查找成功时的平均查找长度和查找失败时的平均查找长度；

(2) 链地址法，求等概率下查找成功时的平均查找长度和查找失败时的平均查找长度。

答案：

$(3 \times 32) \% 11 = 8$ ，无冲突；

$(3 \times 13) \% 11 = 6$ ，无冲突；

$(3 \times 49) \% 11 = 4$ ，无冲突；

$(3 \times 24) \% 11 = 6$ ，有冲突；

$(3 \times 38) \% 11 = 4$ ，有冲突；

$(3 \times 21) \% 11 = 8$ ，有冲突；

$(3 \times 4) \% 11 = 1$ ，无冲突；

$(3 \times 12) \% 11 = 3$ ，无冲突。

(1) 线性探查再散列

24 的散列位置有冲突，探查下一个位置 7 成功

38 的散列位置有冲突，探查下一个位置 5 成功

21 的散列位置有冲突，探查下一个位置 9 成功

构造的散列表示意如下：

|        |   |   |   |    |    |    |    |    |    |    |    |
|--------|---|---|---|----|----|----|----|----|----|----|----|
| 散列地址   | 0 | 1 | 2 | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| 关键字    |   | 4 |   | 12 | 49 | 38 | 13 | 24 | 32 | 21 |    |
| 成功比较次数 |   | 1 |   | 1  | 1  | 2  | 1  | 2  | 1  | 2  |    |
| 失败探查次数 | 1 | 2 | 1 | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  |

$$ASL_{succ} = (1 + 1 + 1 + 2 + 1 + 2 + 1 + 2) / 8 = 11 / 8;$$

$$ASL_{unsucc} = (1 + 2 + 1 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1) / 11 = 40 / 11。$$

## (2) 链地址法

构造的散列表如下图：



$$ASL_{succ} = (1 + 1 + 3 \times (1 + 2)) / 8 = 11 / 8;$$

$$ASL_{unsucc} = (1 + 1 + 2 + 2 + 2) / 11 = 8 / 11。$$

430、对关键字序列 (*Sun, Mon, Tue, Wed, Thu, Fri, Sat*) 建立散列表，散列函数为  $H(key) = (\text{关键字第一个字母在字母表中序号}) \% 7$ ，用线性探查法处理冲突：

(1) 构造一个装填因子为 0.7 的散列表；

(2) 分别计算出在等概率情况下查找成功与不成功的平均查找长度。

答案：

(1)  $\alpha = 0.7$ ，所以表长取  $m = 7 / 0.7 = 10$ 。

*Sun*:  $19 \% 7 = 5$ ，无冲突；

*Mon*:  $13 \% 7 = 6$ ，无冲突；

*Tue*:  $20 \% 7 = 6$ ，有冲突

$d_1: (6 + 1) \% 10 = 7$ ，无冲突；

*Wed*:  $23 \% 7 = 2$ ，无冲突；

*Thu*:  $20 \% 7 = 6$ ，有冲突

$d_1: (6 + 1) \% 10 = 7$ ，有冲突

$d_2: (6 + 2) \% 10 = 8$ ，无冲突；

*Fri*:  $6 \% 7 = 6$ ，有冲突，

$d_1: (6 + 1) \% 10 = 7$ ，有冲突

$d_2: (6 + 2) \% 10 = 8$ ，有冲突

$d_3: (6 + 3) \% 10 = 9$ ，无冲突；

*Sat*:  $19 \% 7 = 5$ ，有冲突，

$d_1: (5 + 1) \% 10 = 6$ ，有冲突

$d_2: (5 + 2) \% 10 = 7$ ，有冲突

$d_3: (5 + 3) \% 10 = 8$ ，有冲突

$d_4: (5 + 4) \% 10 = 9$ ，有冲突

$d_5: (5 + 5) \% 10 = 0$ ，无冲突。

构造的散列表示意如下：

|        |            |   |            |   |   |            |            |            |            |            |
|--------|------------|---|------------|---|---|------------|------------|------------|------------|------------|
| 散列地址   | 0          | 1 | 2          | 3 | 4 | 5          | 6          | 7          | 8          | 9          |
| 关键字    | <i>Sat</i> |   | <i>Wed</i> |   |   | <i>Sun</i> | <i>Mon</i> | <i>Tue</i> | <i>Thu</i> | <i>Fri</i> |
| 成功比较次数 | 6          |   | 1          |   |   | 1          | 1          | 2          | 3          | 4          |
| 失败探查次数 | 2          | 1 | 2          | 1 | 1 | 7          | 6          |            |            |            |

(2) 查找的性能指标为：

$ASL_{succ} = (6 + 1 + 1 + 1 + 2 + 3 + 4) / 7 = 18 / 7;$

$$ASL_{unsucc} = (2 + 1 + 2 + 1 + 1 + 7 + 6) = 20 / 7。$$

431、已知一组关键字为(26, 36, 41, 38, 44, 15, 68, 12, 06, 51, 25)，用链地址法解决冲突。要求装填因子 $\alpha \leq 0.75$ ，散列函数的形式为  $H(key) = key \% p$ ：

(1) 设计出散列函数；

(2) 分别计算等概率情况下查找成功和查找失败的平均查找长度；

答案：

由装填因子 $\alpha \leq 0.75$  和关键字个数 11 个，得表长  $m = \lceil 11 / 0.75 \rceil = 15$ 。

(1) 散列函数  $H(key) = key \% 13$  ( $p$  取小于等于表长的最大素数)

因为  $p = 13$ ，散列地址为 0 ~ 12，用链地址法解决冲突，表长就等于 13。

26 % 13 = 0，无冲突；

36 % 13 = 10，无冲突；

41 % 13 = 2，无冲突；

38 % 13 = 12，无冲突；

44 % 13 = 5，无冲突；

15 % 13 = 2，有冲突；

68 % 13 = 3，无冲突；

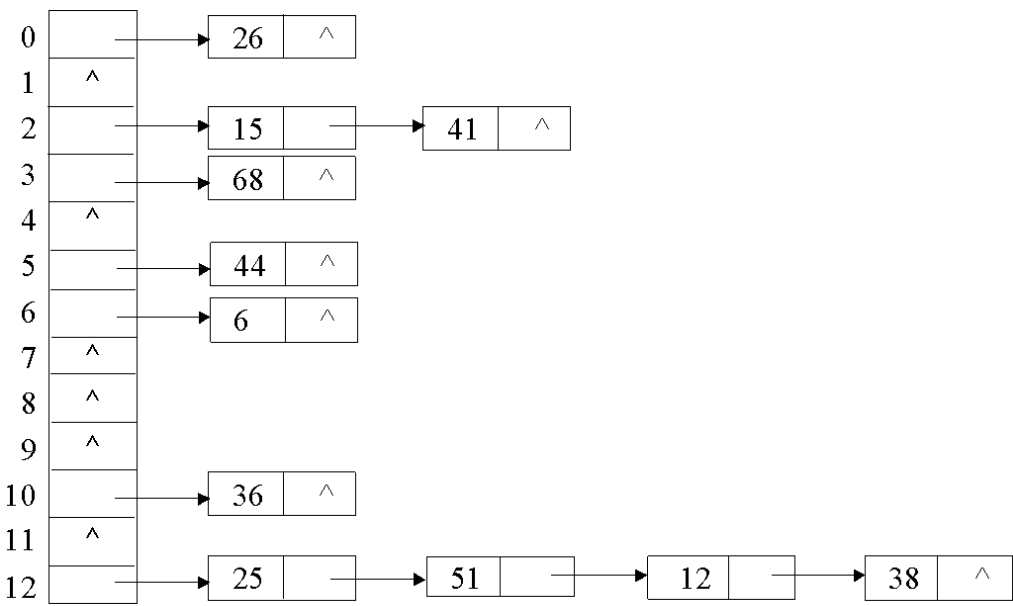
12 % 13 = 12，有冲突；

6 % 13 = 6，无冲突；

51 % 13 = 12，有冲突；

25 % 13 = 12，有冲突。

构造的散列表如下图：



(2) 查找的性能指标为：

$$ASL_{succ} = (1 + 1 + 2 + 1 + 1 + 1 + 1 + 1 + 2 + 3 + 4) / 11 = 18 / 11;$$

$$ASL_{unsucc} = (1 + 2 + 1 + 1 + 1 + 1 + 4) / 13 = 11 / 13.$$

432、试为关键字序列(*dog, rat, cat, hen, ass, pig, ram, cow, fox, ant, boa, eel, cod, yak, hog*) 设计散列表，要求所设计的表在查找成功时的平均查找长度不超过 2.0，并验证散列表的实际平均查找长度是否满足要求。

答案：

为简化起见，选择使用线性探查再散列解决冲突，查找成功时的平均查找长度

理论值  $S_{nl} \approx \boxed{\text{red X}}$ ，代入  $ASL_{succ} \leq 2.0$  可以求出装填因子  $\alpha \leq 2/3$ ;

根据关键字个数和装填因子，可知表长  $m = \lceil 15 / (2/3) \rceil = 23$ 。

考虑到关键字由英文字母构成，设计散列函数为  $H(key) = (\text{关键字首尾字母在字母表中序号之和}) \% 23$ ，各关键字的散列情况如下：

*dog*:  $(4 + 7) \% 23 = 11$ ，无冲突；

*rat*:  $(18 + 20) \% 23 = 15$ ，无冲突；

*cat*:  $(3 + 20) \% 23 = 0$ ，无冲突；

*hen*:  $(8 + 14) \% 23 = 22$ ，无冲突；

*ass*:  $(1 + 19) \% 23 = 20$ ，无冲突；

*pig*:  $(16 + 7) \% 23 = 0$ ，有冲突

$d_1$ :  $(0 + 1) \% 23 = 1$ ，无冲突；

*ram*:  $(18 + 13) \% 23 = 8$ ，无冲突；

*cow*:  $(3 + 23) \% 23 = 3$ ，无冲突；

*fox*:  $(6 + 24) \% 23 = 7$ ，无冲突；

*ant*:  $(1 + 20) \% 23 = 21$ ；

*boa*:  $(2 + 1) \% 23 = 3$ ，有冲突

$d_1$ :  $(3 + 1) \% 23 = 4$ ，无冲突；

*eel*:  $(5 + 12) \% 23 = 17$ ，无冲突；

*cod*:  $(3 + 4) \% 23 = 7$ ，有冲突

$d_1$ :  $(7 + 1) \% 23 = 8$ ，有冲突

$d_2$ :  $(7 + 2) \% 23 = 9$ ，无冲突；

*yak*:  $(25 + 11) \% 23 = 13$ ，无冲突；

*hog*:  $(8 + 7) \% 23 = 15$ ，有冲突

$d_1$ :  $(15 + 1) \% 23 = 16$ ，无冲突。

实际查找成功时的平均查找长度为：

$$ASL_{succ} = (11 \times 1 + 3 \times 2 + 1 \times 3) / 15 = 20 / 15 < 2.0, \text{ 满足要求。}$$

433、已知某散列表的装填因子小于 1，关键字为英文小写字母串，散列函数  $H(key)$  = 关键字的第一个字母在字母表中的序号，处理冲突的方法为链地址法。

编写一个计算在等概率情况下查找不成功的平均查找长度的算法，规定不能用理论公式直接求解计算，散列表的存储结构类型和算法的函数原型定义如下：

```
#define MAXSIZE 26

typedef string KeyType;

typedef struct node
{
    KeyType key;    // 关键字

    struct node *next; // 后继结点指针
} NodeType;

typedef NodeType *Head[MAXSIZE + 1]; // 头指针数组类型定义

double Unsuccessful(Head hash); // 查找不成功长度原型

// hash 为头指针数组
```

答案：

计算链地址法查找不成功的次数，只需要顺着链表计算结点个数即可。

```
double Unsuccessful(Head hash)
```



```

{

    int count = 0, i;

    NodeType *p;

    for (i = 1; i <= MAXSIZE; i++)
    {

        for (p = hash[i]; p; p = p->next, count++);

    }

    return (double)count / MAXSIZE;

}

```

434、排序的目的是为了方便以后的查找。

答案： 正确

解析：有序数据的查找性能远远高于乱序数据。

435、内排序要求数据一定要以顺序方式存储。

答案： 错误

解析：没有这个要求，某些内排序算法例如归并排序和 LSD 基数排序特别适合用链式存储而不是顺序存储。

436、当待排序的元素很大时，为了交换元素的位置，移动元素要占用较多的时间，这是影响时间复杂度的主要因素。

答案： 正确

解析：排序算法的时间复杂度性能中包括了关键字的比较和记录的移动两个部分，重点在于哪一个部分占用较多时间，如果记录占用空间大，移动记录的时间相对于关键字的比较便处于主导地位。

437、如果某种排序算法是不稳定的，则该方法没有实际的应用价值。

答案： 错误

解析：如果没有稳定性要求，自然可以使用不稳定的排序算法。

438、在执行某个排序算法过程中，出现了待排序关键字朝着最终排序序列位置相反方向移动，则该算法是不稳定的。

答案： 错误


解析：排序性能的稳定性并不是单独由关键字的移动方向来确定，而是依照不同的相等关键字相互之间的相对移动方向来决定。

439、用关键字比较的排序方法，在最坏的情况下，能达到的最好时间复杂性是什么？给出详细证明。

答案：

基于关键字比较的排序方法，最坏的情况下，可以达到的最好时间复杂性是  $O(n \log_2 n)$ 。

考虑到关键字比较一次只能区分出两种不同的排列状态，例如比较关键字  $a$  和  $b$  能够区分出  $a > b$  和  $a < b$ ，其他的关键字比较类似，因此这个决策过程可以用二叉树来描述，叶子结点就是其最终的排列状态，根到叶子的边数(也就是二叉树的高度 - 1) 就是区分出该排列状态需要进行比较的次数。

$n$  个关键字的不同排列有  $n!$  种，区分其排列情况用二叉树来描述，也就是有  $n!$  个叶子，其最小可能高度等价于相同叶子数量的完全二叉树，因此最坏情况下的最少比较次数不会低于  $\lfloor \log_2(n!) \rfloor$ ，按照 *Stirling* 公式：，代入即可知  $\lim_{n \rightarrow \infty} \lfloor \log_2(n!) \rfloor = O(n \log_2 n)$ 。

440、折半插入排序所需比较次数与待排序记录的初始排列状态相关。

答案： 错误

解析：因为需要在有序序列中折半查找插入位置，因此比较次数与待排序初始状态关系不大。

441、对数据序列(25, 15, 7, 18, 10, 0, 4) 采用直接插入排序进行升序排序，两趟排序后，得到的排序结果为\_\_\_\_\_。

A、 0, 4, 7, 18, 10, 25, 15

B、 0, 4, 25, 15, 7, 18, 10

C、 7, 15, 10, 0, 4, 18, 25

D、 7, 15, 25, 18, 10, 0, 4

答案： D

解析：按照直接插入排序的过程，两趟排序后，只有前 3 个关键字“25, 15, 7”有序，与后面所有关键字无关。

442、下列排序方法中，稳定的是\_\_\_\_\_。

A、 直接插入排序

B、 直接选择排序

C、 堆排序

D、 快速排序

答案： A

解析：答案的 4 个选项中，只有直接插入排序可以保证排序的稳定。

443、设顺序存储的待排序 7 条记录中关键字分别为(8, 3, 2, 5, 9, 1, 6)：

(1) 用直接插入排序进行递增排序，写出各趟排序中所有关键字的排列；

(2) 说明该排序算法的稳定性。

答案：

(1) 直接插入排序全过程：

第 1 趟：3, 8, 2, 5, 9, 1, 6

第 2 趟：2, 3, 8, 5, 9, 1, 6

第 3 趟：2, 3, 5, 8, 9, 1, 6

第 4 趟：2, 3, 5, 8, 9, 1, 6

第 5 趟：1, 2, 3, 5, 8, 9, 6

第 6 趟：1, 2, 3, 5, 6, 8, 9

(2) 直接插入排序是稳定的。

444、现有 1000 条待排序记录存放在内存，其中只有少量记录次序不对，且它们距离正确位置不远；如果以比较和移动次数作为度量，将其排序最好采用什么方法？为什么？

答案：

采用直接插入排序，该排序算法最适合于关键字个数很少或者关键字排列接近有序，并且能保持稳定。

因为只有少量记录次序不对，且距离正确位置并不远：

从比较次数而言，简单排序方法中的冒泡排序和直接插入排序较为合适，因为两个排序算法的主要比较操作都是和附近的记录相比较

考虑到记录的移动次数，显然直接插入排序更为合适，毕竟冒泡排序属于交换排序，两条记录换位需要移动 3 次，而直接插入排序可以做到一次到位，没有冗余的换位操作。

445、待排序记录的数据类型定义如下：

```
#define MAXSIZE 100
```

```
typedef int KeyType;
```

```
typedef struct
```

```
{
```

```
    KeyType key;
```

```
} RecType;
```

```
typedef RecType SeqList[MAXSIZE];
```

下列函数实现顺序表的直接插入排序，请在空白处填上适当内容是算法完整。

```
void f(SeqList R, int n)
```

```
{
```

```
    int i, j;
```

```
    RecType temp;
```

```
    for (i = 1; i <= _____; i++) // (1)
```

```
    {
```

```
        temp = R[i];
```

```
        j = i;
```

```
        while (j > 0 && temp.key < R[j - 1].key)
```

```
        {
```

```
            R[j] = R[j - 1];
```

```

        _____;    //    (2)

    }

    _____;    //    (3)

}

}

```

答案:

(1)  $n - 1$

(2)  $j --$

(3)  $R[j] = temp$

446、Shell 排序的时间性能与增量序列的选取有关，但关系不大。

答案： 错误

解析：Shell 排序的增量序列直接影响排序性能。

447、对序列(15, 9, 7, 8, 20, -1, 4) 排序，第一趟排序后的序列变为(4, 9, -1, 8, 20, 7, 15)，则采用的排序方法是\_\_\_\_\_排序。

A、 直接选择

B、 快速

C、 希尔

D、 冒泡

答案： C

解析：对比给定的两个序列，经过一趟排序后，除了位置没有变动的 8 以外，其他关键字都没有位于最终位置，按照排序的特点：经过一趟排序后，选择排序和交换排序将至少有一个关键字位于排序最终的位置，因此可以排除归类于选择排序和交换排序的直接选择排序、快速排序和冒泡排序，并且可以推断出，希尔排序第一趟的增量为 2。

448、下列选项中，不稳定的排序方法是\_\_\_\_\_。

A、 希尔排序

B、 归并排序

C、 直接插入排序

D、 冒泡排序

答案： A

解析：希尔排序的分组插入过程使得排序无法保证稳定性。



449、数据序列(19, 14, 23, 01, 68, 20, 84, 27, 55, 11, 10, 79, 12), 使用希尔排序方法将其排成升序序列:

- (1) 写出增量为 4 时对上述数据序列进行一趟希尔排序的结果;
- (2) 给出一个可行的希尔排序增量序列。

答案:

- (1) 增量为 4 的一趟希尔排序的结果为:

12, 11, 10, 01, 19, 14, 23, 27, 55, 20, 84, 79, 68。

- (2) 考虑到待排序记录个数为 13, 是一个素数, 因此增量序列为(6, 3, 1) 或者(4, 2, 1) 皆可。

450、用自底向上的冒泡排序方法对序列(8, 13, 26, 55, 29, 44)从大到小排序, 第一趟排序需进行交换的次数为\_\_\_\_\_。

- A、 2
- B、 3
- C、 4
- D、 5

答案: C

解析: 交换的关键字对为: (29, 44)、(26, 55)、(13, 55)、(8, 55)。

451、对序列(15, 9, 7, 8, 20, -1, 4) 进行排序, 如果一趟排序后的结果为(-1, 15, 9, 7, 8, 20, 4), 则采用的排序方法是\_\_\_\_\_。

A、 归并排序

B、 快速排序

C、 直接选择排序

D、 冒泡排序

答案： D

解析： 根据排序结果和排序算法的特性可知为小数上浮的冒泡排序。

452、对一组数据(2, 12, 16, 88, 5, 10) 进行排序，如果前 3 趟排序结果如下：

第一趟：2, 12, 16, 5, 10, 88

第二趟：2, 12, 5, 10, 16, 88

第三趟：2, 5, 10, 12, 16, 88

则采用的排序方法是\_\_\_\_\_。

A、 冒泡排序

B、 希尔排序

C、 归并排序

D、 基数排序

答案： A

解析： 根据排序结果和排序算法的特性可知序列为大数下沉的冒泡排序的结果。

453、将关键字序列(11, 25, 6, 26, 55, 77, 43, 23, 19, 45, 42, 75, 16, 33) 用冒泡排序递增排序：

(1) 写出对该序列进行大数下沉一趟排序后关键字的排列，并说明关键字的交换次数；

(2) 在排序过程中，某些关键字可能会朝着最终位置的反方向移动，是否说明冒泡排序是不稳定排序？为什么？

(3) 写出对该序列进行小数上浮一趟排序后关键字的排列，并说明关键字的交换次数。

答案：

(1) 大数下沉冒泡排序第一趟排序的结果：

(11, 6, 25, 26, 55, 43, 23, 19, 45, 42, 75, 16, 33, | 77 |)

关键字的交换次数为 9 次：(25, 6)、(77, 43)、(77, 23)、(77, 19)、(77, 45)、(77, 42)、(77, 75)、(77, 16)、(77, 33)；

(2) 虽然可能有部分关键字在排序中朝着最终位置的反方向移动，但是并不能说明排序算法一定是不稳定的，只要排序算法能够保持多个等值关键字前后的相对位置永远不变即可保证排序稳定。

(3) 小数上浮冒泡排序第一趟排序的结果：

(6, 11, 25, 16, 26, 55, 77, 43, 23, 19, 45, 42, 75, 33)

关键字的交换次数为 11 次：(75, 16)、(42, 16)、(45, 16)、(19, 16)、(23, 16)、(43, 16)、(77, 16)、(55, 16)、(26, 16)、(25, 6)、(11, 6)。

454、初始序列  $a[1], a[2], \dots, a[n]$  的奇偶交换递增排序过程如下：

第一趟对所有奇数  $i(1 \leq i < n)$ ，将  $a[i]$  和  $a[i + 1]$  比较，如果  $a[i] > a[i + 1]$ ，则将两者交换；

第二趟对所有偶数  $i(2 \leq i < n)$ ，将  $a[i]$  和  $a[i + 1]$  比较，如果  $a[i] > a[i + 1]$ ，则将两者交换；

第三趟对所有奇数  $i(1 \leq i < n)$ ；

第四趟对所有偶数  $i(2 \leq i < n)$ ；

……；

依次类推直至到整个序列有序为止。

(1) 分析这种排序方法的结束条件；

(2) 写出用这种排序方法对关键字序列(35, 70, 33\*, 65, 24, 21, 33) 进行排序时，每一趟的结果。

答案：

(1) 连续的 2 趟中均没有交换即可终止排序；

(2) 奇偶交换递增排序的过程：

第 1 趟：(35, 70, 33\*, 65, 21, 24, 33)

第 2 趟：(35, 33\*, 70, 21, 65, 24, 33)

第 3 趟：(33\*, 35, 21, 70, 24, 65, 33)

第 4 趟: (33\*, 21, 35, 24, 70, 33, 65)

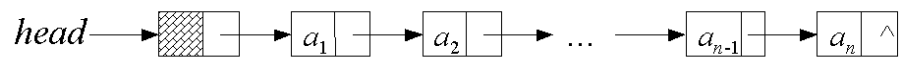
第 5 趟: (21, 33\*, 24, 35, 33, 70, 65)

第 6 趟: (21, 24, 33\*, 33, 35, 65, 70)

第 7 趟: (21, 24, 33\*, 33, 35, 65, 70)

第 8 趟: (21, 24, 33\*, 33, 35, 65, 70)。

455、带头结点单链表如下图所示:



数据类型定义如下:

```
typedef int KeyType;
```

```
typedef struct node
```

```
{
```

```
    KeyType key;
```

```
    struct node *next;
```

```
} RecNode;
```

有多条记录存储在单链表中, 用大数下沉的冒泡排序法对其进行原地递增排序, 写出排序算法, 函数原型如下:

```
void Bubble_Sort(RecNode *head);
```

答案:

```
void Bubble_Sort(RecNode *head)
```

```

{

    RecNode *p, *q, *end = 0, *last = 0;

    KeyType temp;

    int swap = 1;

    if (!head->next || !head->next->next)

        return;

    while (swap)

    {

        p = head->next;

        q = p->next;

        swap = 0;

        for (p != end && q != end)

        {

            if (q->key < p->key)

            {

                temp = q->key;

                q->key = p->key;

                p->key = temp;

                swap = 1;

                last = q;

            }

            p = q;

```

```
        q = q->next;

    }

    end = last;

}

}
```

456、有时冒泡排序的速度会快过快速排序。

答案： 正确

解析： 当待排序关键字序列接近或者完全有序时。

457、顾名思义，快速排序法是在所有情况下，速度最快的排序方法。

答案： 错误

解析： 快速排序是平均性能最好，但是特殊情况不一定。

458、快速排序总比简单排序快。

答案： 错误

解析： 关键字接近有序时，即使使用优化的快速排序方法，算法也只能达到  $O(n \log n)$ ，比一些简单排序的  $O(n)$  慢很多。

459、快速排序的辅助存储为可能为  $O(1)$ 。

答案： 错误

解析：快速排序的辅助存储空间最低为  $O(\log_2 n)$ 。

460、快速排序的速度在所有排序方法中为最快，而且所需附加空间也最少。

答案： 错误

解析：快速排序的平均性能最好，但并不是原地排序。

461、下列排序算法中，时间复杂度为  $O(n\log_2 n)$  的算法是\_\_\_\_\_。

A、 快速排序

B、 冒泡排序

C、 直接选择排序

D、 直接插入排序



答案： A

解析：快速排序的平均性能为  $O(n\log_2 n)$ ，其他排序算法为  $O(n^2)$ 。

462、下列排序算法中，初始数据有序时，花费时间反而更多的算法是\_\_\_\_\_。

A、 插入排序

B、 冒泡排序

C、 快速排序

D、 希尔排序

答案： C

解析：快速排序的特征就是初始数据越有序，则性能反而越差，其他答案中的排序算法则是越有序越快。

463、对关键字序列(47, 15, 16, 65, 41, 19, 5, 50, 37, 45, 90, 56, 42)，写出以序列第一个元素为基准值的快速排序的第一趟排序结果。

答案：

(42, 15, 16, 45, 41, 19, 5, 37, 47, 50, 90, 56, 65)。

464、对于  $n$  个元素的线性表快速排序时，所需的比较次数与  $n$  个元素的初始排序有关，如果以序列的最左元素做为划分的基准值，并且设  $n = 7$  时：

(1) 最好情况下关键字需比较多少次？说明理由，并给出一个最好情况时初始排序的实例；

(2) 最坏情况下关键字需比较多少次？说明理由，并给出一个最坏情况时初始排序的实例。

答案：

(1) 最好情况是每次划分时，正好基准值将无序序列分割成左右等长的两个序列，并且各个序列都是一直进行到 1 个元素为止，用基准值构成的递归树形态正好是满二叉树。

例如 (4, 1, 3, 2, 6, 5, 7)：

第 1 趟排序的划分，比较次数为 6 次；

第 2 趟排序的划分，左右子树合计比较 4 次；

第 3 趟没有比较，比较的总次数为  $6 + 4 + 0 = 10$  次。

(2) 最坏情况是每次划分时，只有基准值到了最终位置，序列中其他元素依然在一个序列中，用基准值构成的递归树形态正好是一棵单支树。

例如完全有序的 (1, 2, 3, 4, 5, 6, 7)：

第 1 趟划分仅仅 1 就位，比较次数为 6 次；

第 2 趟划分仅仅 2 就位，比较次数为 5 次；

.....；

第 6 趟划分 6 就位，比较 1 次；

第 7 趟划分没有比较，比较的总次数为  $6 + 5 + \dots + 1 + 0 = 21$  次。

465、如果只要找出一个具有  $n$  个元素的集合的第  $k$  ( $1 \leq k \leq n$ ) 个最小元素，哪种排序方法最适合？以实例说明实现的思想。

答案：

通过改写快速排序算法，使用快速排序的一趟排序划分出基准位置  $pivot$ ：

1)  $pivot == k$ ，则  $pivot$  位置数据就是；

2)  $pivot > k$ ，则在左半继续寻找；

3)  $pivot < k$ ，则在右半继续寻找。

继续划分直到找到为止。

例：求序列 (35, 5, 63, 47, 7, 65, 31, 60, 21, 17) 第 3 个最小元素的过程。

对序列 (35, 5, 63, 47, 7, 65, 31, 60, 21, 17) 用最左元素做基准值：

1) 以 35 做基准值划分后的结果为：

(17, 5, 21, 31, 7, |35|, 65, 60, 47, 63)

基准值 35 为第 6 小元素，继续在其左半划分。

2) 以 17 做左半部分的基准值划分左半后的结果为：

(7, 5, |17|, 31, 21)，恰好 17 就是待求的第 3 个最小元素。

466、在有  $n$  个不同元素的数组中，找出第  $k$  小的元素 ( $k \leq n$ )，要求算法的时间复杂度为  $O(n)$ ，函数原型如下：

```
int kth_element(int a[ ], int n, int k);
```

答案：

利用快速排序一趟划分的过程完成，函数 Partition 用于一趟划分。

```
int Partition(int a[ ], int begin, int end)
```

```
{           // 以第一个元素为基准，对 a[begin .. end] 进行划分，返回分割点的下标
```

```
    int i = begin, j = end, pivot = a[i];
```

```
    while (i < j)
```

```
    {
```

```
        while (i < j && a[j] >= pivot)
```

```
            j--;
```

```
        if (i < j)
```

```
            a[i++] = a[j];
```

```
        while (i < j && a[i] <= pivot)
```

```
            i++;
```

```
        if (i < j)
```

```
            a[j--] = a[i];
```

```
    }
```

```
    a[i] = pivot;
```

```
    return i;
```

```
}
```

```
int kth_element(int a[], int n, int k)
```

```
{
```

```

int loc = 0, begin = 0, end = n - 1;

if (k < 1)

    return a[0];

if (k > n)

    return a[n - 1];

while (loc != k - 1)

{

    loc = Partition(a, begin, end);

    if (loc > k - 1)

        end = loc - 1;

    else if (loc < k - 1)

        begin = loc + 1;

}


return a[loc];

}

```

467、直接选择排序算法在最好情况下的时间复杂度为  $O(n)$ 。

答案： 错误

解析：直接选择排序的比较次数永远是  次。

468、无论待排序列是否有序，排序算法时间复杂度都是  $O(n^2)$  的排序方法是\_\_\_\_\_。


A、 快速排序

B、 归并排序

C、 冒泡排序

D、 直接选择排序

答案： D

解析： 只有直接选择排序的关键字比较次数永远是  次。

469、比较次数与待排序列初始状态无关的排序方法是\_\_\_\_\_。


A、 快速排序

B、 冒泡排序

C、 直接插入排序

D、 直接选择排序

答案： D

解析：直接选择排序无论初始关键字是什么排列，比较次数永远是  次，其他答案中的排序算法依照数据的初始排列在某些数量级间变动。

470、下列排序算法中，每一趟都能选出一个元素放到其最终位置上的是\_\_\_\_\_。

A、 直接插入排序

B、 希尔排序

C、 二路归并排序

D、 直接选择排序

答案： D

解析：选择排序和交换排序可以保证每一趟都能至少选出一个元素放到其最终位置上。

471、设顺序存储的待排序 7 条记录中关键字分别为(8, 3, 2, 5, 9, 1, 6)：

- (1) 用直接选择排序进行递减排序，写出各趟排序中所有关键字的排列；
- (2) 说明该排序算法的稳定性。

答案：

- (1) 直接选择排序的全部过程：

第 1 趟：9, 3, 2, 5, 8, 1, 6

第 2 趟：9, 8, 2, 5, 3, 1, 6

第 3 趟：9, 8, 6, 5, 3, 1, 2

第 4 趟：9, 8, 6, 5, 3, 1, 2

第 5 趟：9, 8, 6, 5, 3, 1, 2

第 6 趟：9, 8, 6, 5, 3, 2, 1

- (2) 直接选择排序是不稳定的。

472、由  $n$  个关键字不同的记录构成的序列，能否用少于  $2n - 3$  次关键字比较选出该序列中关键字最大和最小的记录？说明如何实现？在最坏的情况下至少进行多少次比较？

答案：

一种比较简单的方法是将  $n$  个元素对称两两配对比较：

第一个元素与最后一个元素比较，第二个元素与倒数第二个元素比较， $\dots$ ，相比较的小者放前半部，大者放后半部，用了  $\lceil n/2 \rceil$  次比较；

再在前后两部分中分别直接比较选择最小和最大元素，各用  $\lceil n/2 \rceil - 1$  次比较；



总共用了  $3 \cdot \frac{n}{2} - 2$  次比较。

473、某排序算法的代码以下：

```
void f(int a[], int n)
{
    int i, j, k, t;

    for (i = 0; i < n; i++) // 语句 1
    {
        j = i;

        for (k = j + 1; k <= n; k++)
        {
            if (a[k] < a[j]) // 语句 2
                j = k;
        }

        t = a[i];
        a[i] = a[j];
        a[j] = t;
    }
}
```

- (1) 说明语句 1 和语句 2 的频度以及此算法的时间复杂度；
- (2) 该排序算法是属于哪一种排序方法？

答案:

(1) 各语句的频度如下:

语句 1:  $n$  次

语句 2:  $\frac{n(n-1)}{2}$  次;

(2) 排序算法为在顺序存储上的直接选择排序。

474、有一种简单的排序算法叫做计数排序(*count sorting*), 该排序算法对一个待排序的表进行排序, 并将排序结果存放到另一个新的表中, 表中所有待排序的关键字互不相同。

计数排序算法针对表中的每个记录, 扫描待排序的表一趟, 统计表中有多少个记录的关键码比该记录的关键码小。假设某记录统计出的计数值为  $c$ , 于是该记录在新的有序表中的存放位置即为  $c$ 。

数据类型的定义如下:

```
#define MAXSIZE 100
```

```
typedef int KeyType;
```

```
typedef struct {
```

```
    KeyType key;
```

```
} RecType;
```

```
typedef RecType SeqList[MAXSIZE];
```

(1) 编写实现计数排序的算法, 函数原型如下:

```
void CountSorting(SeqList data, SeqList order, int n);
```

// *data* 为存放原始数据的数组, *order* 为有序数组, *n* 为元素个数

(2) 对于有  $n$  条记录的表, 关键码比较次数是多少?

(3) 与直接选择排序相比较, 这种方法是否更好?为什么?

答案:

```
void CountSorting(SeqList data, SeqList order, int n)
{
    int i, j, count;

    for (i = 0; i < n; i++)
    {
        for (count = j = 0; j < n; j++)
        {
            if (data[j].key < data[i].key)

                count++;
        }

        order[count] = data[i];
    }
}
```

(2) 关键字永远比较  $n^2$  次。

(3) 与直接选择排序相比, 不仅关键字的比较次数更多, 而且需要  $O(n)$  的辅助空间; 记录的移动次数永远为  $n$  次, 比直接选择排序最坏时的性能略好, 直接选择排序重要缺点就是不稳定, 但是因为待排序关键字互不相等, 并没有稳定性的问题, 因此该排序算法并不比直接选择排序好。

475、以中序方式遍历一个堆，则得到一个有序序列。

答案： 错误

解析：按照堆的定义，分支结点的关键字值只是永远大于等于或者小于等于孩子结点的关键字值，但是并没有限定左右孩子之间的大小关系，因此一般情况下并不能在堆中利用二叉树的先序、中序、后序和层次序遍历得到一个有序序列，某些很特殊情况可以利用层次序遍历得到有序序列。

476、堆排序是一种巧妙的树型选择排序。

答案： 正确

解析：堆排序相对树型选择排序而言，很巧妙地在原地进行，大幅度降低了树型选择排序  $O(n)$  的空间复杂度。

477、堆是满二叉树。

答案： 错误

解析：堆的逻辑结构是完全二叉树，满二叉树只是完全二叉树的特例。

478、下列序列中，\_\_\_\_\_不是堆。

A、 75, 45, 65, 30, 15, 25

B、 75, 65, 45, 30, 25, 15

C、 75, 65, 30, 15, 25, 45

D、 75, 45, 65, 25, 30, 15

答案： C

解析：按照堆的定义，堆中每个分支结点的关键字值都是不小于（不大于）孩子结点的关键字值，其中的关键字序列 75, 65, 30, 15, 25, 45 不满足条件。

479、一组记录的关键字为(35, 48, 47, 23, 44, 88)，利用堆排序算法进行降序排序，建立的初始堆为\_\_\_\_\_。

A、 23, 35, 48, 47, 44, 88

B、 23, 35, 47, 48, 44, 88

C、 35, 23, 47, 48, 44, 88

D、 35, 23, 47, 44, 48, 88

答案： B

解析：利用堆进行降序排序，首先需要建立最小堆，从堆中最后一个元素的双亲开始，逐步自底向上调整直到堆顶。

480、如果希望在 1000 个无序元素中尽快求得前 10 个最大元素，应借用\_\_\_\_\_。

A、 堆排序

B、 快速排序

C、 冒泡排序

D、 归并排序

答案： A

解析：在先建立好最大堆后，取得前 10 个最大元素的比较次数不超过  $20 \log_2 1000 \approx 200$  次

附：

设初始待排序元素有  $n$  个 ( $n > 100$ )，并且是最坏情况下的排序，下面讨论主要排序方法的关键字比较次数。

(1) 归并排序只能在完全排序后才能保证得到前 10 个最大元素，其关键字比较次数共计  $n \log_2 n$  次，而且尚未考虑记录的移动次数；

(2) 冒泡排序则关键字需要比较  $(n - 1) + (n - 2) + \cdots + (n - 10) = 10n - 55$  次；

(3) 堆排序包括建立初始堆在内的关键字比较总次数，在最坏时不超过  $4n + 2 \cdot 10 \log_2 n$  次；

(4) 使用快速排序也需要完全排序后才能得到前 10 个最大元素，比较次数平均情况也是  $n \log_2 n$ 。

(5) 如果利用快速排序的一趟分割算法，其中基准值的选择使用随机化策略，最多  $2n - 2$  次比较即可得到前 10 个最大元素，但是这个方法并不是快速排序，而且得到的前 10 个元素还需要重新排序，最小比较次数约为  $10 \log_2 10$ 。

(6) 还有一个选项是树形选择排序，也叫做锦标赛排序，得到前 10 个最大元素的比较次数最多为  $(n - 1) + 9 \log_2 n$ ，这个是所有方法里面的最小值。

因此所有答案中可行的就是堆排序。

481、判断下列各关键字序列是否是堆，如果不是，用最少的比较次数调整为堆。

(1) 100, 85, 40, 77, 80, 60, 66, 98, 82, 10, 20

(2) 10, 80, 60, 40, 82, 77, 20, 66, 85, 98, 100

答案：

(1) 最大值 100 处于堆顶、最小值 10 处于叶子，但是关键字 40 和 77 的孩子并不满足最大堆的要求，由此调整为最大堆比较次数较少：(100, 98, 66, 85, 80, 60, 40, 77, 82, 10, 20)；

(2) 最小值 10 处于堆顶、最大值 100 处于叶子，但是关键字 80 和 60 的孩子并不满足最小堆的要求，由此调整为最小堆比较次数较少：(10, 40, 20, 66, 82, 77, 60, 80, 85, 98, 100)。

482、对关键字序列(26, 18, 60, 14, 7, 45, 13, 32) 进行降序的堆排序, 写出构建的初始堆及前两趟排序重建堆之后序列状态。

答案:

构建的初始最小堆: (7, 14, 13, 26, 18, 45, 60, 32);

第一趟排序后结果: (13, 14, 32, 26, 18, 45, 60, 7);

第二趟排序后结果: (14, 18, 32, 26, 60, 45, 13, 7)。

483、在任何情况下, 归并排序都比直接插入排序快。

答案: 错误

解析: 在关键字完全乱序时, 虽然时间复杂度一致, 但是就关键字的移动次数而言, 归并排序远远多于直接插入排序。

484、下列排序算法中, 空间复杂度最差的是\_\_\_\_\_。

A、 归并排序

B、 希尔排序

C、 冒泡排序



D、 堆排序

答案： A

解析：归并排序需要与关键字个数成正比的辅助空间，其他答案中的排序算法都是原地进行。

485、要以  $O(n \log n)$  时间复杂度进行稳定的排序，可用的排序方法是\_\_\_\_\_。

A、 归并排序

B、 快速排序

C、 堆排序

D、 冒泡排序

答案： A

解析：既要排序稳定，又要时间复杂度  $O(n \log n)$ ，只有归并排序满足要求。

486、对数据序列(26, 14, 17, 12, 7, 4, 3) 采用二路归并排序进行升序排序，两趟排序后，得到的排序结果为\_\_\_\_\_。

A、 14, 26, 17, 12, 4, 7, 3

B、 12, 14, 17, 26, 3, 4, 7

C、 14, 26, 12, 17, 3, 4, 7

D、 14, 26, 12, 17, 3, 7, 4

答案： B

解析：依照二路归并排序的过程，经过两趟排序后，原始序列按每  $2^2 = 4$  个数据顺序分段有序。

487、用二路归并排序法对序列 (98, 36, -9, 0, 47, 23, 1, 8) 进行递增排序：

(1) 一共需要多少趟归并即可完成排序；

(2) 写出第 1 趟归并后数据的排列次序。

答案：

(1) 3 趟；

(2)  $|\underline{36}, \underline{98}|, |\underline{-9}, \underline{0}|, |\underline{23}, \underline{47}|, |\underline{1}, \underline{8}|$ 。

解析：(1) 待排序关键字个数为 8，因此二路归并的趟数为： $\lceil \log_2 8 \rceil = 3$  趟；

(2) 第 1 趟归并后，从前到后依次两两递增有序。

488、在堆排序、快速排序和归并排序中：

- (1). 如果只从存储空间考虑，则应首先选取哪种排序方法，其次选取哪种排序方法，最后选取哪种排序方法？
- (2). 如果只从排序结果的稳定性考虑，则应选取哪种排序方法？
- (3). 如果只从平均情况下排序最快考虑，则应选取哪种排序方法？
- (4). 如果只从最坏情况下排序最快并且要节省内存考虑，则应选取哪种排序方法？

答案：

- (1) 堆排序第一、快速排序第二、归并排序第三；
- (2) 归并排序；
- (3) 快速排序；
- (4) 堆排序。

解析：堆排序的性能：时间复杂度最好最坏平均都是  $O(n\log_2 n)$ ，空间复杂度  $O(1)$ ，排序不稳定；

快速排序的性能：普通算法的时间复杂度最好和平均都是  $O(n\log_2 n)$ ，此时空间复杂度为  $O(\boxed{\times})$ ，最坏时间复杂度  $O(n^2)$ ，空间复杂度  $O(n)$ ；

归并排序性能：时间复杂度最好最坏平均都是  $O(n\log_2 n)$ ，空间复杂度  $O(n)$ ，排序不稳定。

这三个算法的  $O(n\log_2 n)$  中，相对而言，快速排序的最快，归并排序最慢。

489、基数排序不需进行关键字之间的比较，因此执行时间比所有基于比较的排序方法要快。

答案： 错误

解析：基数排序只是不直接进行关键字之间的相互比较，最坏时的最好排序性能并不会优于基于比较的排序方法。

490、在分配排序时，最高位优先分配法比最低位优先分配法简单。

答案： 错误

解析：最高位优先分配按关键字值的各个分组中元素个数并不能事先确定，不像最低位优先分配的规则简单。

491、对序列(8, 13, 26, 55, 29, 44)从小到大进行基数排序，第一趟排序的结果是\_\_\_\_\_。

A、 (13, 44, 55, 26, 8, 29)

B、 (13, 26, 55, 44, 8, 29)

C、 (8, 13, 26, 29, 44, 55)

D、 (29, 26, 8, 44, 55, 13)

答案： A

解析：基数排序第一趟完成后，关键字的最低位有序，并且相同最低位的关键字保持稳定性。

492、写出关键字序列(12, 2, 16, 30, 8, 28, 4, 10, 20, 6, 18) 用链式基数排序(基数为 10) 从大到小排序时，第一趟结束时的序列；

答案：

8, 28, 18, 16, 6, 4, 12, 2, 30, 10, 20。

解析：第一趟结束后按个位递减且保持稳定性。

493、外排序是指\_\_\_\_\_。

A、 用机器指令直接对硬盘中需排序数据排序

B、 把需排序数据，用其他大容量机器排序

C、 把外存中需排序数据一次性调入内存，排好序后再存储到外存

D、 对外存中大于内存允许空间的待排序的数据，通过多次内外间的交换实现排序

答案： D

解析： 外排序用于当内存不足以将所有待排序记录全部存放到内存时。

494、 设磁盘文件中共有 12 条记录，其关键字依次分别为(28, 26, 88, 16, 69, 25, 50, 31, 64, 42, 29, 63)，如果内存工作区可以容纳 4 条记录，用置换选择法可以产生几个初始归并段？各初始有序归并段的记录是？

答案：

采用置换选择法，生成了 2 个初始归并段：

Run<sub>0</sub>: {16, 26, 28, 50, 69, 88}, Run<sub>1</sub>: {25, 29, 31, 42, 63, 64}。

解析： 置换选择的详细过程如下表：

| 输入记录   | 内存工作区       | 输出归并段                     |
|--|-------------|---------------------------|
| 28, 26, 88, 16, 69, 25, 50, 31, 64, 42, 29, 63 |             |                           |
| 69, 25, 50, 31, 64, 42, 29, 63                 | 28 26 88 16 |                           |
| 25, 50, 31, 64, 42, 29, 63                     | 28 26 88 69 | 16                        |
| 50, 31, 64, 42, 29, 63                         | 28 25 88 69 | 16, 26                    |
| 31, 64, 42, 29, 63                             | 50 25 88 69 | 16, 26, 28                |
| 64, 42, 29, 63                                 | 31 25 88 69 | 16, 26, 28, 50            |
| 42, 29, 63                                     | 31 25 88 64 | 16, 26, 28, 50, 69        |
| 29, 63   | 31 25 42 64 | 16, 26, 28, 50, 69, 88    |
| 29, 63   | 31 25 42 64 | 16, 26, 28, 50, 69, 88, ∞ |
| 29, 63   | 31 25 42 64 |                           |

|    |             |                           |
|----|-------------|---------------------------|
| 63 | 31 29 42 64 | 25                        |
|    | 31 63 42 64 | 25, 29                    |
|    | — 63 42 64  | 25, 29, 31                |
|    | — 63 — 64   | 25, 29, 31, 42            |
|    | — — — 64    | 25, 29, 31, 42, 63        |
|    | — — — —     | 25, 29, 31, 42, 63, 64    |
|    | — — — —     | 25, 29, 31, 42, 63, 64, ∞ |

495、可以利用归并排序来实现外排序。

答案： 正确

解析：外排序文件很多时候是顺序文件，采用多路归并可以占用较少的内存。

496、设某文件经内排序后得到 100 个初始归并段(初始顺串)，如果使用多路归并排序算法，并要求三趟归并完成排序，问归并路数最少为多少？

答案：

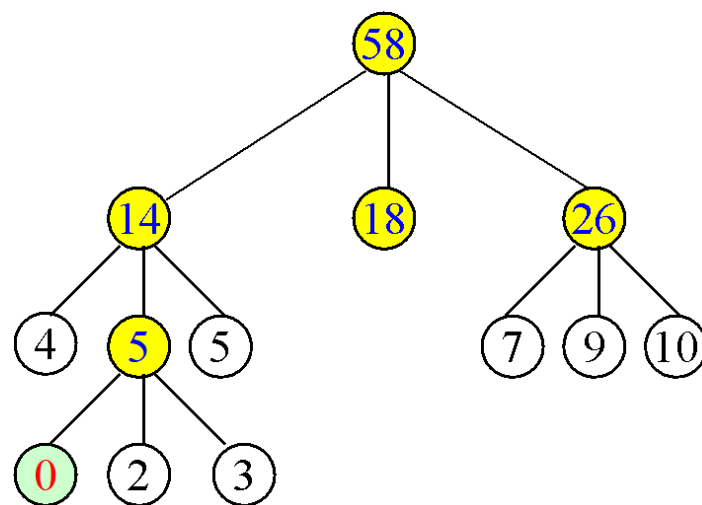
设多路归并的路数为  $k$ ，于是归并趟数  $= \lceil \log_k 100 \rceil = 3$ 。

可以解得：  $5 \leq k \leq 9$ ，即归并路数最少为 5 路，最多为 9 路。

497、给定 8 个权值集合 (2, 5, 3, 10, 4, 7, 9, 18)，画出含有 8 个叶子结点的最佳三叉归并树，并计算 WPL。

答案：

最佳三叉归并树：



$$WPL = (0 + 2 + 3) \times 3 + (4 + 5 + 7 + 9 + 10) \times 2 + 18 \times 1 = 103。$$

解析： $(8 - 1) \% (3 - 1) = 1$ ，因此需要增加 1 个内结点，还需要另加  $3 - 1 - 1 = 1$  个空的归并段。

因此初始归并段为 (0, 2, 5, 3, 10, 4, 7, 9, 18)，递增排序为 (0, 2, 3, 4, 5, 7, 9, 10, 18)

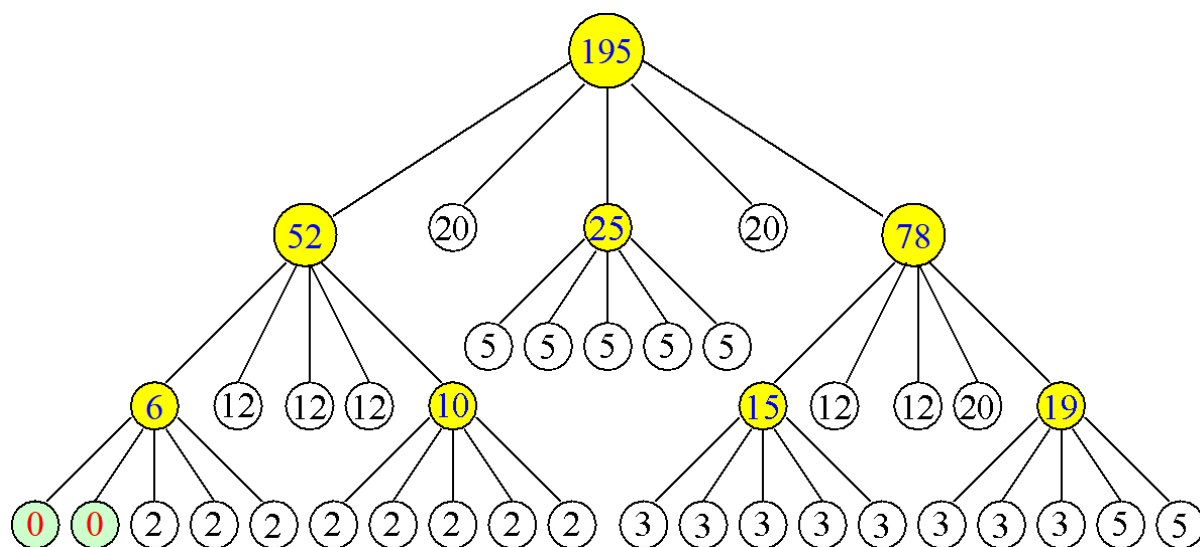
归并过程为：(0, 2, 3, 4, 5, 7, 9, 10, 18)  $\rightarrow$  (4, 5, 5, 7, 9, 10, 18)  $\rightarrow$  (7, 9, 10, 14, 18)  $\rightarrow$  (14, 18, 26)  $\rightarrow$  (58)。

498、已知有 31 个长度不等的初始归并段，其中 8 段长度为 2；8 段长度为 3；7 段长度为 5；5 段长度为 12；3 段长度为 20(单位均为物理块)，请为此归并段设计一个最佳 5-路归并方案，并计算总的（归并所需）读/写外存的次数。

答案：



最佳 5-路归并树：



$$\begin{aligned} \text{WPL} &= (8'2)'3 + (8'3)'3 + (2'5)'3 + (5'12)'2 \\ &+ (1'20)'2 + (5'5)'2 + (2'20)'1 = 48 + 72 + 30 + \\ &120 + 40 + 50 + 40 = 400. \end{aligned}$$

因此读写外存次数为  $2' \text{WPL} = 800$  次。

解析： $(31 - 1) \% (5 - 1) = 2$ ，因此需要增加 1 个内结点，还需要另加  $5 - 2 - 1 = 2$  个空的归并段。

构造最佳 5 路归并树的过程为：

$(2'0, 8'2, 8'3, 7'5, 5'12, 3'20) \text{ P } (5'2, 8'3, 7'5, 6, 5'12, 3'20)$

$\text{P } (8'3, 7'5, 6, 10, 5'12, 3'20) \text{ P } (3'3, 7'5, 6, 10, 5'12, 15, 3'20)$

$\text{P } (5'5, 6, 10, 5'12, 15, 19, 3'20) \text{ P } (6, 10, 5'12, 15, 19, 3'20, 25)$

$\text{P } (2'12, 15, 19, 3'20, 25, 52) \text{ P } (2'20, 25, 52, 78)$   
 $\text{P } (195)。$