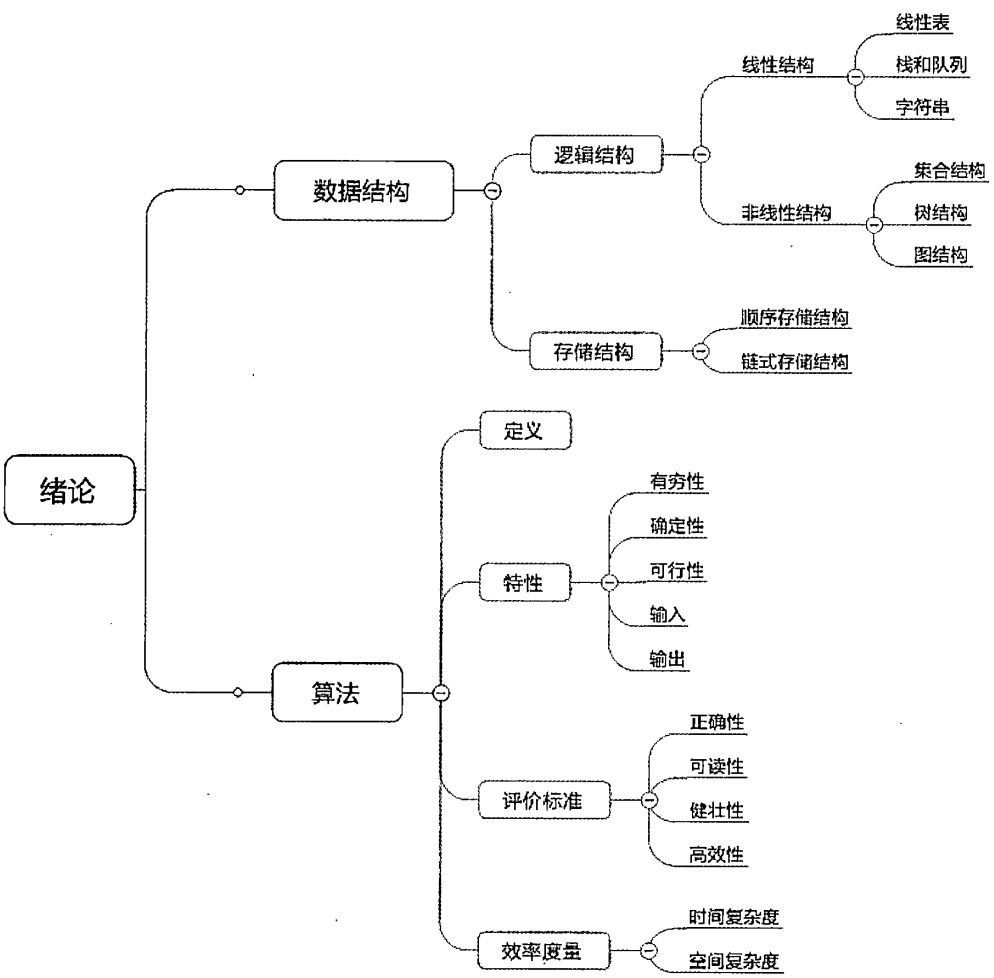
目 录

第一篇　习题

第1章 绪论

【知识导图】



【学习目标】

1. 掌握数据结构相关的基本概念，包括数据、数据元素、数据项、数据对象、数据结构等，明确数据元素和数据项的关系。

2. 掌握数据结构所含两个层次（逻辑结构和存储结构）的具体含义及其相互关系。逻辑结构是从具体问题抽象出来的数学模型，它与数据的存储无关。存储结构是逻辑结构在计算机中的存储表示，其中，顺序存储结构借助元素在存储器中的相对位置来表示数据元素之间的逻辑关系，通常借助程序设计语言的数组类型来描述；链式存储结构无需占用一整块存储空间，通常借助程序设计语言的指针类型来描述，用指针来存放后继元素的存储地址。

3. 算法是为了解决某类问题而规定的一个有限长的操作序列。理解算法五个特性的含义和明确算法优劣的四个评价标准。

4. 算法分析的两个主要方面是分析算法的时间复杂度和空间复杂度，以考察算法的时间和空间效率。掌握算法时间复杂度和空间复杂度的分析方法。一般情况下，鉴于运算空间较为充足，故将算法的时间复杂度作为分析的重点。

1.1　习题

一、单项选择题

1. 在数据结构中，从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

2. 与数据元素本身的形式、内容、相对位置、个数无关的是数据的（　）。

A. 存储结构　　B. 存储实现　　C. 逻辑结构　　D. 运算实现

3. 通常要求同一逻辑结构中的所有数据元素具有相同的特性，这意味着（　）。

A. 数据具有同一特点

B. 不仅数据元素所包含的数据项的个数要相同，而且对应数据项的类型要一致

C. 每个数据元素都一样

D. 数据元素所包含的数据项的个数要相等

4. 以下说法正确的是（　）。

A. 数据元素是数据的最小单位

B. 数据项是数据的基本单位

C. 数据结构是带有结构的各数据项的集合

D. 一些表面上很不相同的数据可以有相同的逻辑结构

5. 算法的时间复杂度取决于（　）。

A. 问题的规模　　B. 待处理数据的初态　　C. 计算机的配置　　D. A 和 B

6. 以下数据结构中，（　）是非线性数据结构。

A. 树　　B. 字符串　　C. 队列　　D. 栈

7.【2011 年第 1 题】设 n 是描述问题规模的非负整数，下面程序段的时间复杂度是（　　）。

```
x=2;
while(x<n/2)
   x=2*x;
```

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

8.【2014年第1题】下面程序段的时间复杂度是（　　）。

```
count=0;
for(k=1;k<=n;k*=2)
   for(j=1;j<=n;j++)
      count++;
```

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

9. 某算法的语句执行频度为（$3n+n\log_2 n+n^2+8$），其时间复杂度表示（　　）。

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

10. 以下程序段中语句“x++;”的语句频度为（　　）。

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

A. $\frac{n(n+1)(2n+1)}{2}$　　B. $\frac{n(n+1)(n+1)}{2}$　　C. $\frac{n(n+1)(2n+1)}{6}$　　D. $\frac{n(n+1)(n+2)}{6}$

11. 以下程序段中语句“m++;”的语句频度为（　　）。

```
int m=0,i,j;
for(i=1;i<=n;i++)
   for(j=1;j<=2*i;j++)
      m++;
```

A. $n(n+1)$　　B. n　　C. $n+1$　　D. n^2

12. 设一维数组中有 n 个数组元素，则读取第 i 个数组元素的平均时间复杂度为（　　）。

A. $O(n)$　　B. $O(n\log_2 n)$　　C. $O(1)$　　D. $O(n^2)$

13. 下面说法错误的是（　　）。

（1）算法原地工作的含义是指不需要任何额外的辅助空间

（2）在相同的规模 n 下，复杂度 $O(n)$的算法在时间上总是优于复杂度 $O(2^n)$的算法

（3）所谓时间复杂度是指最坏情况下，估算算法执行时间的一个上界

（4）某算法的时间复杂度为 $O(n^2)$，表明该算法的执行时间与 n^2 成正比

A.（1）　　B.（1），（2）　　C.（1），（4）　　D.（3）

14. 下面算法将一维数组 a 中的 n 个数逆序存放到原数组中，空间复杂度为（　　）。

```
for(i=0;i<n;i++)
   b[i]=a[n-i-1];
for(i=0;i<n;i++)
   a[i]=b[i];
```

A. $O(1)$　　B. $O(n)$　　C. $O(\log_2 n)$　　D. $O(n^2)$

15. 下面算法将一维数组 a 中的 n 个数逆序存放到原数组中，空间复杂度为（　　）。

```
for(i=0;i<n/2;i++)
{  t=a[i];
   a[i]=a[n-i-1];
   a[n-i-1]=t;
}
```

A. $O(1)$　　B. $O(n)$　　C. $O(\log_2 n)$　　D. $O(n^2)$

16. 下列叙述中正确的是（　　）。

A. 一个算法的空间复杂度大，则其时间复杂度也必定大

B．一个算法的空间复杂度大，则其时间复杂度必定小

C．一个算法的时间复杂度大，则其空间复杂度必定小

D．上述三种说法都不对

17．数据结构是指（　　）。

A．数据元素的组织形式　　B．数据类型

C．数据存储结构　　D．数据定义

18．树形结构是数据元素之间存在的一种（　　）。

A．一对一关系　　B．多对多关系　　C．多对一关系　　D．一对多关系

19．设数据结构 $A=(D, R)$，其中 $D=\{1, 2, 3, 4\}$，$R=\{r\}$，$r=\{<1, 2>, <2, 3>, <3, 4>, <4, 1>\}$，则数据结构 A 是（　　）。

A．线性结构　　B．树结构　　C．图　　D．集合

20．数据在计算机存储器内表示时，物理地址与逻辑地址不相同的，称之为（　　）。

A．存储结构　　B．逻辑结构　　C．链式存储结构　　D．顺序存储结构

21．数据在计算机内有链式和顺序两种存储方式，在存储空间使用的灵活性上，链式存储比顺序存储要（　　）。

A．低　　B．高　　C．相同　　D．不确定

22．顺序存储结构中数据元素之间的逻辑关系是由（　　）表示的。

A．线性结构　　B．非线性结构　　C．存储位置　　D．指针

23．链式存储结构中的数据元素之间的逻辑关系是由（　　）表示的。

A．线性结构　　B．非线性结构　　C．存储位置　　D．指针

24．以下与数据的存储结构有关的术语是（　　）。

A．有序表　　B．链表　　C．有向图　　D．树

25．抽象数据类型的三个组成部分分别为（　　）。

A．数据对象、数据关系和基本操作　　B．数据元素、逻辑结构和存储结构

C．数据项、数据元素和数据类型　　D．数据元素、数据结构和数据类型

26．算法是（　　）。

A．计算机程序　　B．解决问题的计算方法

C．排序算法　　D．解决问题的有限运算序列

27．下面关于算法说法正确的是（　　）。

A．算法最终必须由计算机程序实现

B．为解决某问题的算法同为该问题编写的程序含义是相同的

C．算法的可行性是指指令不能有二义性

D．以上几个都是错误的

28．算法分析的两个主要方面是（　　）。

A．空间复杂度和时间复杂度　　B．正确性和简单性

C．可读性和文档性　　D．数据复杂性和程序复杂性

29．对一个算法的评价，不包括如下（　　）方面的内容。

A．健壮性和可读性　B．并行性　　C．正确性　　D．时空复杂度

30. 通常从正确性、易读性、健壮性、高效性等 4 个方面评价算法的质量，以下解释错误的是（　　）。

A. 正确性算法应能正确地实现预定的功能

B. 易读性算法应易于阅读和理解，以便调试、修改和扩充

C. 健壮性指当环境发生变化时，算法能适当地做出反应或进行处理，不会产生不需要的运行结果

D. 高效性即达到所需要的时间性能

二、应用题

1. 简述下列概念：数据、数据元素、数据项、数据对象、数据结构、逻辑结构、存储结构、抽象数据类型。

2. 试举一个数据结构的例子，叙述其逻辑结构和存储结构两方面的含义和相互关系。

3. 简述逻辑结构的四种基本关系并画出它们的关系图。

4. 存储结构由哪两种基本的存储方法实现？

5. 试分析下列各算法的时间复杂度。

(1)

```
x=90; y=100;
  while(y>0)
    if(x>100)
      {x=x-10;y--;}
    else x++;
```

(2)

```
for(i=0; i<n; i++)
  for(j=0; j<m; j++)
    a[i][j]=0;
```

(3)

```
s=0;
for(i=0; i<n; i++)
  for(j=0; j<n; j++)
    s+=B[i][j];
sum=s;
```

(4)

```
i=1;
while(i<=n)
  i=i*3;
```

(5)

```
x=0;
for(i=1; i<n; i++)
  for(j=1; j<=n-i; j++)
    x++;
```

(6)

```
x=n; //n>1
y=0;
while(x>=(y+1)*(y+1))
  y++;
```

1.2　答案及解析

一. 单项选择题

1. C	2. C	3. B	4. D	5. D	6. A	7. A	8. C	9. C	10. D
11. A	12. C	13. A	14. B	15. A	16. D	17. A	18. D	19. C	20. C
21. B	22. C	23. D	24. B	25. A	26. D	27. D	28. A	29. B	30. D

1.【答案】C

【考点】逻辑结构

【解析】

数据的逻辑结构是从逻辑关系上描述数据，可以看作从具体问题抽象出来的数学模型。根据数据元素之间关系的不同特性，逻辑结构通常划分为集合结构、线性结构、树结构和图结构。其中集合结构、树结构和图结构都属于非线性结构。因此，逻辑结构又可以分为线性结构与非线性结构两大类。

2.【答案】C

【考点】逻辑结构

【解析】

逻辑结构是从具体问题抽象出来的数学模型，从逻辑关系上描述数据，它与数据的存储无关，也就是说与数据本身的具体形式、内容、相对位置、个数都无关。

3.【答案】B

【考点】数据元素与数据项的定义

【解析】

数据项是组成数据元素的、有独立含义的、不可分割的最小单位，同一逻辑结构中的数据元素所包括数据项的个数要相同，且要求对应数据项的类型也要一致。例如，对于学生基本信息表这种线性表结构，其中数据元素为所有的学生，数据项为学号、姓名、性别等，这里学号、姓名、性别的数据类型必须一致。因此，选项 B 是正确的。

4.【答案】D

【考点】数据、数据元素与数据项的定义

【解析】

数据元素是数据的基本单位，数据项是数据的最小单位，数据结构是带有结构的各数据元素的集合。因此，选项 A、B 和 C 都是错误的。选项 D 是正确的，因为逻辑结构是从逻辑关系上描述数据，它与数据本身的具体形式无关。例如，学生表和图书表都可以看作线性结构，而学生数据和图书数据表面上是完全不同的数据。所以答案选择 D。

5.【答案】D

【考点】时间复杂度

【解析】

算法的时间复杂度不仅与问题的规模有关，还与问题的其他因素有关。如某些排序的算法，其执行时间与待排序记录的初始状态有关。因此，有时会对算法有最好、最坏以及平均时间复杂度的评价。

6.【答案】A

【考点】逻辑结构

【解析】

非线性结构包括树结构、图结构和集合结构，而字符串、队列和栈都属于线性结构。

7.【答案】A

【考点】时间复杂度

【解析】

设程序中基本语句“x=2*x;”执行的次数为 k，执行 k 次时：$x=2^{k+1}$。由循环结束条件 $x<n/2$ 可得，$2^{k+1}<n/2$，即 $k<\log_2 n-2$，$k=\log_2 n+C$（C 为常数），因此 $T(n)=O(\log_2 n)$。

8.【答案】C

【考点】时间复杂度

【解析】

内层循环条件 $j\leqslant n$ 与外层循环的变量无关，每次循环 j 自增 1，每次内层循环都执行 n 次。所以内层循环的时间复杂度是 $O(n)$。外层循环条件为 $k\leqslant n$，增量定义为 $k*=2$，可知循环次数为 $2^k\leqslant n$，即 $k\leqslant\log_2 n$。所以外层循环的时间复杂度是 $O(\log_2 n)$。对于嵌套循环，根据乘法规则可知，该段程序的时间复杂度 $T(n)=T1(n)*T2(n)=O(n)*O(\log_2 n)=O(n\log_2 n)$。

9.【答案】C

【考点】时间复杂度

【解析】

时间复杂度具体计算数量级时，可以遵循以下定理：

若 $f(n)=a_m n^m+a_{m-1}n^{m-1}+\cdots+a_1 n+a_0$ 是一个 m 次多项式，则 $T(n)=O(n^m)$。

上述定理说明，在计算算法时间复杂度时，可以忽略所有低次幂项和最高次幂项的系数。

10.【答案】D

【考点】时间复杂度

【解析】

“x++;”的语句频度为:

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\sum_{i=1}^{n}\sum_{j=1}^{i}j=\sum_{i=1}^{n}\frac{i(i+1)}{2}$$

$$=\frac{1}{2}\left(\sum_{i=1}^{n}i^2+\sum_{i=1}^{n}i\right)$$

$$=\frac{1}{2}\left(\frac{n(n+1)(2n+1)}{6}+\frac{n(n+1)}{2}\right)$$

$$=\frac{n(n+1)(n+2)}{6}$$

11.【答案】A

【考点】时间复杂度

【解析】

“m++;”的语句频度为:

$$\sum_{i=1}^{n}\sum_{j=1}^{2i}1=\sum_{i=1}^{n}2i=2\sum_{i=1}^{n}i=n(n+1)$$

12.【答案】C

【考点】时间复杂度

【解析】

读取第 i 个数组元素可以直接通过数组的下标定位，与 n 无关，因此平均时间复杂度为 O(1)。

13.【答案】A

【考点】时间复杂度和空间复杂度

【解析】

原地工作指算法执行时所需要的辅助空间，其相对于输入量而言是个常数，语句(1)错误，其他语句的描述均正确，因此答案选择 A。

14.【答案】B

【考点】空间复杂度

【解析】

算法的空间复杂度只需分析该算法在实现时所需要的辅助空间与问题规模 n 的函数关系。该算法需要另外借助一个大小为 n 的辅助数组 b，所以其空间复杂度为 O(n)。

15.【答案】A

【考点】空间复杂度

【解析】

该算法仅需要另外借助一个变量 t，与问题规模 n 大小无关，所以其空间复杂度为 O(1)。

16.【答案】D

【考点】时间复杂度和空间复杂度

【解析】

算法的时间复杂度和空间复杂度没有直接关系。

17.【答案】A

【考点】数据结构

【解析】

数据结构是相互之间存在一种或多种特定关系的数据元素的集合。换句话说，数据结构是带“结构”的数据元素的集合，“结构”就是指数据元素之间存在的关系。因此，数据结构是指数据元素的组织形式。

18.【答案】D

【考点】逻辑结构

【解析】

树结构中数据元素之间存在一对多的关系。

19.【答案】C

【考点】逻辑结构

【解析】数据元素之间存在多对多的关系，因此是图结构。

20.【答案】C

【考点】存储结构

【解析】

数据对象在计算机中的存储表示称为数据的存储结构，由于链式存储结构不要求逻辑上相邻的元素在物理位置上也相邻，因此无需占用一整块存储空间。为了表示结点之间的关系，需要给每个结点附加指针字段，用于存放后继元素的存储地址。

21.【答案】B

【考点】存储结构

【解析】

顺序存储结构要求所有的元素依次存放在一段连续的存储空间中，必须预先分配；而链式存储结构用一组任意的存储单元存储数据元素，这组存储单元可以是连续的，也可以是不连续的，不需要预先分配空间。因此，在存储空间使用的灵活性上，链式存储更高。

22.【答案】C

【考点】存储结构

【解析】

顺序存储结构借助元素在存储器中的相对位置来表示数据元素之间的逻辑关系，其逻辑关系由存储位置（即元素在数组中的下标）表示的。

23.【答案】D

【考点】存储结构

【解析】

链式存储结构无需占用一整块连续的存储空间。但为了表示结点之间的关系，需要给每个结点附加指针字段，用于存放后继元素的存储地址。所以链式存储结构通常借助程序设计语言的指针类型来描述。

24.【答案】B

【考点】存储结构

【解析】

有序表、有向图、树是与数据的逻辑结构有关的术语，不涉及数据的存储情况，只有 B 选项链表与数据的存储结构有关。

25.【答案】A

【考点】抽象数据类型的定义

【解析】

抽象数据类型一般指由用户定义的、表示应用问题的数学模型，以及定义在这个模型上的一组操作的总称，具体包括三部分：数据对象、数据对象上关系的集合以及对数据对象的基本操作的集合。

26.【答案】D

【考点】算法的定义及特性

【解析】

算法是为了解决某类问题而规定的一个有限长的操作序列。

27.【答案】D

【考点】算法的定义及特性

【解析】

算法是为了解决某类问题而规定的一个有限长的操作序列，不是必须由计算机实现的，因此选项 A 是错误的。为解决某问题的算法与为该问题编写的程序含义是不同的，因为程序的编写与

具体的语言有关，而算法与语言无关，所以选项 B 也是错误的。算法的可行性是指算法中的所有操作都可以通过已经实现的基本操作运算执行有限次来实现，显然，选项 C 错误。因此答案选择 D。

28.【答案】A

【考点】评价算法优劣的基本标准

【解析】

时间高效是指算法设计合理，执行效率高，可以用时间复杂度来度量；空间高效是指算法占用存储容量合理，可以用空间复杂度来度量。时间复杂度和空间复杂度是衡量算法的两个主要指标。

29.【答案】B

【考点】评价算法优劣的基本标准

【解析】

一个算法优劣应该从以下几方面来评价：正确性、可读性、健壮性和高效性。选项 D 中的“时空复杂度”属于高效性评价的指标，因此答案选择 B。

30.【答案】D

【考点】评价算法优劣的基本标准

【解析】

高效性包括时间和空间两个方面，而不应该仅指时间性能高效。

二、应用题

1．解答

数据：是客观事物的符号表示，指所有能输入到计算机中并被计算机程序处理的符号的总称。如数学计算中用到的整数和实数，文本编辑所用到的字符串，多媒体程序处理的图形、图像、声音、动画等通过特殊编码定义后的数据。

数据元素：是数据的基本单位，在计算机中通常作为一个整体进行考虑和处理。在有些情况下，数据元素也称为元素、结点、记录等。数据元素用于完整地描述一个对象，如一个学生记录、树中棋盘的一个格局（状态）、图中的一个顶点等。

数据项：是组成数据元素的、有独立含义的、不可分割的最小单位。例如，学生基本信息表中的学号、姓名、性别等都是数据项。

数据对象：是性质相同的数据元素的集合，是数据的一个子集。例如，整数数据对象是集合 $N=\{0, \pm1, \pm2, \cdots\}$，字母字符数据对象是集合 $C=\{'A', 'B', \cdots, 'Z', 'a', 'b', \cdots, 'z'\}$，学生基本信息表也可是一个数据对象。

数据结构：是相互之间存在一种或多种特定关系的数据元素的集合。换句话说，数据结构是带“结构”的数据元素的集合，“结构”就是指数据元素之间存在的关系。

逻辑结构：从逻辑关系上描述数据，它与数据的存储无关，是独立于计算机的。因此，数据的逻辑结构可以看作从具体问题抽象出来的数学模型。

存储结构：是数据对象在计算机中的存储表示，也称为物理结构。

抽象数据类型：是由用户定义的，表示应用问题的数学模型，以及定义在这个模型上的一组操作的总称。具体包括三部分：数据对象、数据对象上关系的集合和对数据对象的基本操作的集合。

2．解答

例如，有一张学生基本信息表，包括学生的学号、姓名、性别、籍贯、专业等。每个学生的基本信息记录对应一个数据元素，学生记录按顺序号排列，形成了学生基本信息记录的线性序列。对于整个表来说，只有一个开始结点（它的前面无记录）和一个终端结点（它的后面无记录），其

他的结点则各有一个也只有一个直接前驱和直接后继。学生记录之间的这种关系就确定了学生表的逻辑结构，即线性结构。

这些学生记录在计算机中的存储表示就是存储结构。如果用连续的存储单元（如用数组表示）来存放这些记录，则称为顺序存储结构；如果存储单元不连续，而是随机存放各个记录，然后用指针进行链接，则称为链式存储结构。

即相同的逻辑结构，可以对应不同的存储结构。

3．解答

数据的逻辑结构有两个要素：一是数据元素；二是关系。其中，关系是指数据元素间的逻辑关系。根据数据元素之间关系的不同特性，通常有四类基本结构关系，如图 1.1 所示。它们的复杂程度依次递进。

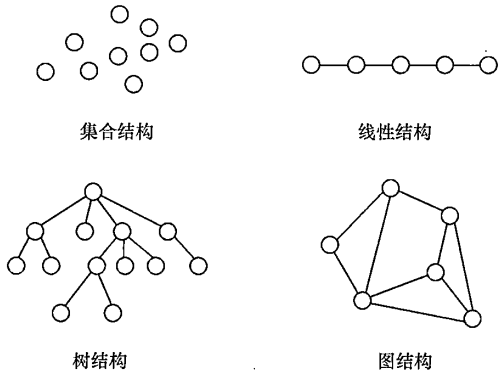


图 1.1　四类基本逻辑结构关系

（1）集合结构

数据元素之间除了“属于同一集合”的关系外，别无其他关系。例如，确定一名学生是否为班级成员，只需将班级看作一个集合结构。

（2）线性结构

数据元素之间存在一对一的关系。例如，将学生信息数据按照其入学报到的时间先后顺序进行排列，将组成一个线性结构。

（3）树结构

数据元素之间存在一对多的关系。例如，在班级的管理体系中，班长管理多个组长，每位组长管理多名组员，从而构成树形结构。

（4）图结构或网状结构

数据元素之间存在多对多的关系。例如，多位同学之间的朋友关系，任何两位同学都可以是朋友，从而构成图形结构或网状结构。

其中，集合结构、树结构和图结构都属于非线性结构。

4．解答

（1）顺序存储结构

顺序存储结构是借助元素在存储器中的相对位置来表示数据元素之间的逻辑关系，通常借助程序设计语言的数组类型来描述。

（2）链式存储结构

顺序存储结构要求所有的元素依次存放在一段连续的存储空间中，而链式存储结构无需占用

一整块存储空间，但为了表示结点之间的关系，需要给每个结点附加指针字段，用于存放后继元素的存储地址。所以链式存储结构通常借助程序设计语言的指针类型来描述。

5．解答

（1）O(1)

程序中基本语句“y--;”或“x++;”执行的次数是由 x 和 y 决定的，而 x 和 y 都是一个常数。所以，$T(n)=O(1)$。

（2）$O(n\times m)$

由于程序为嵌套循环，外层循环的执行次数为 n，内层循环的执行次数为 m。所以，$T(n)=O(n\times m)$。

（3）$O(n^2)$

由于程序为嵌套循环，外层循环的执行次数为 n，内层循环的执行次数也为 n。所以，$T(n)=O(n^2)$。

（4）$O(\log_3 n)$

设基本语句“i=i*3;”的执行次数为 $f(n)$，则有：$3^{f(n)}\leq n$。因此，$T(n)=O(\log_3 n)$。

（5）$O(n^2)$

基本语句“x++;”的执行次数为：$n-1+n-2+\cdots+1=n(n-1)/2$。因此，$T(n)=O(n^2)$。

（6）$O(\sqrt{n})$

设基本语句“y++;”的执行次数为 $f(n)$，则 $x\geqslant(f(n)+1)^2$，又由于 $x=n$，所以 $f(n)\leqslant\sqrt{n}-1$，即 $T(n)=O(\sqrt{n})$。

$n\geq(f(n)+1)^2$

$i*3\leq n$　$3^{f(n)}\leq n$，

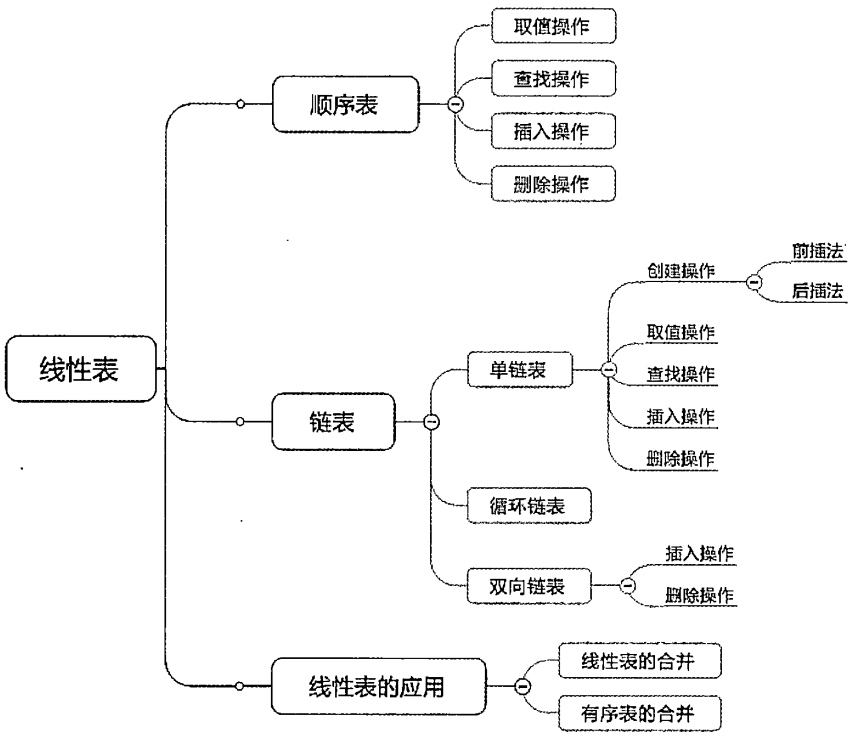
$f(n)$　$i=1*3=3$　$f(n)\log_3 n$.

$i=3*3=9$

$3=9*3=$　$T(n)=O(\sqrt{n})$

第2章 线性表

【知识导图】



【学习目标】

1. 线性表的顺序存储结构即顺序表是一种随机存取结构，可借助数组来表示。给定数组的下标，便可以存取相应的元素。而线性表的链式存储结构即链表是一种顺序存取结构，链表结点的存取都要从头指针开始，顺链而行。要求能够从时间和空间复杂度的角度比较两种存储结构的不同特点及其适用场合，明确它们各自的优缺点。

2. 掌握顺序表和链表的查找、插入和删除以及链表的创建（前插和后插）等基本操作，并能够设计出线性表应用的常用算法，比如线性表的合并、有序表的合并等算法。

3. 除了单链表外，掌握不同形式链表（循环链表和双向链表）的特点、插入和删除等基本操作的实现及其应用场合。

2.1 习题

一、单项选择题

1. 顺序表中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

2. 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（$1 \leqslant i \leqslant n$）和求第 i 个结点的直接前驱（$2 \leqslant i \leqslant n$）
B. 在第 i 个结点后插入一个新结点（$1 \leqslant i \leqslant n$）
C. 删除第 i 个结点（$1 \leqslant i \leqslant n$）
D. 将 n 个结点从小到大排序

3. 向一个有127个元素的顺序表中插入一个新元素并保持原来顺序不变，平均要移动的元素个数为（　　）。
A. 8　　B. 63.5　　C. 63　　D. 7

4. 链接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，一部分存放结点值，另一部分存放结点所占单元数

5. 线性表若采用链式存储结构时，要求内存中可用存储单元的地址（　　）。
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. 连续或不连续都可以

6. 线性表 L 在（　　）情况下适用于使用链式结构实现。
A. 需经常修改 L 中的结点值　　B. 需不断对 L 进行删除插入
C. L 中含有大量的结点　　D. L 中结点结构复杂

7. 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

8. 将两个各有 n 个元素的有序表归并成一个有序表，其最少的比较次数是（　　）。
A. n　　B. $2n-1$　　C. $2n$　　D. $n-1$

9. 在一个长度为 n 的顺序表中，在第 i 个元素（$1 \leqslant i \leqslant n+1$）之前插入一个新元素时须向后移动（　　）个元素。
A. $n-i$　　B. $n-i+1$　　C. $n-i-1$　　D. i

10. 线性表 $L=(a_1,a_2,\cdots,a_n)$，下列说法正确的是（　　）。
A. 每个元素都有一个直接前驱和一个直接后继
B. 线性表中至少有一个元素
C. 表中诸元素的排列必须是由小到大或由大到小
D. 除第一个和最后一个元素外，其余每个元素都有一个且仅有一个直接前驱和直接后继

11. 创建一个包括 n 个结点的有序单链表的时间复杂度是（　　）。
A. O(1)　　B. $O(n)$　　C. $O(n^2)$　　D. $O(n\log_2 n)$

12. 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

13. 在单链表中，要将 *s* 所指结点插入到 *p* 所指结点之后，其语句应为（　　）。

A. s->next=p+1;　p->next=s;

B. (*p).next=s;　(*s).next=(*p).next;

C. s->next=p->next;　p->next=s->next;

D. s->next=p->next;　p->next=s;

14. 在双向链表存储结构中，删除 *p* 所指的结点时须修改指针（　　）。

A. p->prior->next=p->next;　p->next->prior=p->prior;

B. p->next=p->next->next;　p->next->prior=p;

C. p->prior->next=p;　p->prior=p->prior->prior;

D. p->prior=p->next->next;　p->next=p->prior->prior;

15. 在双向循环链表中，在 *p* 指针所指的结点后插入 *q* 所指向的新结点，其修改指针的操作是（　　）。

A. p->next=q;　q->prior=p;　p->next->prior=q;　q->next=q;

B. p->next=q;　p->next->prior=q;　q->prior=p;　q->next=p->next;

C. q->prior=p;　q->next=p->next;　p->next->prior=q;　p->next=q;

D. q->prior=p;　q->next=p->next;　p->next=q;　p->next->prior=q;

16.【2013 年第 1 题】已知两个长度分别为 *m* 和 *n* 的升序链表，若将它们合并为一个长度为 *m*+*n* 的降序链表，则最坏情况下的时间复杂度是（　　）。

A. O(*n*)　　B. O(*m*×*n*)　　C. O(min(*m*,*n*))　　D. O(max(*m*,*n*))

17.【2016 年第 1 题】已知表头元素为 *c* 的单链表在内存中的存储状态如图 2.1 所示。现将 *f* 存放于 1014H 处并插入到单链表中，若 *f* 在逻辑上位于 *a* 和 *e* 之间，则 *a*，*e*，*f* 的“链接地址”依次是（　　）。

地址	元素	链接地址
1000H	a	1010H
1004H	b	100CH
1008H	c	1000H
100CH	d	NULL
1010H	e	1004H
1014H		

图 2.1　单链表的存储状态

A. 1010H，1014H，1004H　　B. 1010H，1004H，1014H

C. 1014H，1010H，1004H　　D. 1014H，1004H，1010H

18.【2016 年第 2 题】已知一个带有表头结点的双向循环链表 L，结点结构为 prev data next，其中，prev 和 next 分别是指向其直接前驱和直接后继结点的指针。现要删除指针 p 所指的结点，

正确的语句序列是（　　）。

A．p->next->prev=p->prev; p->prev->next=p->prev; free (p);

B．p->next->prev=p->next; p->prev-> next=p->next; free (p);

C．p->next->prev=p->next; p->prev->next=p->prev; free (p);

D．p-> next-> prev=p->prev; p->prev->next=p->next; free (p);

19．将两个长度为 n 的递增有序表归并成一个长度为 $2n$ 的递增有序表，最少需要进行关键字比较（　　）次。

A．2　　B．$n-1$　　C．n　　D．$2n$

20．将长度为 n 的单链表链接在长度为 m 的单链表之后的算法的时间复杂度为（　　）。

A．O(1)　　B．O(n)　　C．O(m)　　D．O($m+n$)

21．若线性表最常用的操作是存取第 i 个元素及其前驱的值，则采用（　　）存储方式节省时间。

A．单链表　　B．双链表　　C．单循环链表　　D．顺序表

22．在线性表的下列运算中，不改变数据元素之间结构关系的运算是（　　）。

A．插入　　B．删除　　C．排序　　D．定位

23．对于一个头指针为 head 的带头结点的单链表，判定该表为空表的条件是（　　）。

A．head==NULL　　B．head->next==NULL　　C．head->next==head　　D．head!=NULL

24．对于单链表表示法，以下说法错误的是（　　）。

A．数据域用于存储线性表的一个数据元素

B．指针域或链域用于存放一个指向本结点的直接后继结点的指针

C．所有数据通过指针的链接而组织成单链表

D．NULL 称为空指针，它不指向任何结点，只起标志作用

25．线性表($a_1,a_2,\cdots,a_n$)以链接方式存储时，访问第 i 个位置上元素的时间复杂度为（　　）。

A．O(i)　　B．O(1)　　C．O(n)　　D．O($i-1$)

26．访问单链表中当前结点的后继和前驱的时间复杂度分别是（　　）。

A．O(n)和 O(1)　　B．O(1)和 O(1)　　C．O(1)和 O(n)　　D．O(n)和 O(n)

27．在具有 n 个结点的有序单链表中插入一个新结点并使链表仍然有序的时间复杂度是（　　）。

A．O(1)　　B．O(n)　　C．O($n\log_2 n$)　　D.O(n^2)

28．设一个链表最常用的操作是在末尾插入结点和删除尾结点，则选用（　　）最节省时间。

A．单链表　　B．单循环链表

C．带尾指针的单循环链表　　D．带头结点的双循环链表

29．在一个以 L 为头指针的单循环链表中，p 指针指向链尾的条件是（　　）。

A．p->next==L　　B．p->next==NULL　　C．p->next->next==L　　D．p->data=-1

30．在循环链表中，将头指针改设为尾指针（rear）后，其首元结点和尾结点的存储位置分别是（　　）。

A．rear 和 rear->next->next　　B．rear->next 和 rear

C．rear->next->next 和 rear　　D．rear 和 rear->next

二、应用题

1．线性表有两种存储结构：一是顺序表，二是链表。试问：

（1）如果有 n 个线性表同时并存，并且在处理过程中各表的长度会动态变化，线性表的总数也会自动地改变。在此情况下，应选用哪种存储结构？为什么？

（2）若线性表的元素总数基本稳定，且很少进行插入和删除，但要求以最快的速度存取线性表中的元素，那么应采用哪种存储结构？为什么？

2．在单链表和双向链表中，能否从当前结点出发访问到任一结点？

3．说明在线性表的链式存储结构中，头指针与头结点之间的根本区别，头结点与首元结点的关系。

4．线性表$(a_1, a_2,\cdots,a_n)$，采用顺序存储结构。试问：

（1）在等概率的前提下，平均每插入一个元素需要移动的元素个数为多少？

（2）若元素插在 a_i 与 a_{i+1} 之间（$0\leqslant i\leqslant n-1$）的概率为 $\dfrac{n-i}{n(n+1)/2}$，则平均每插入一个元素所要移动的元素个数又是多少？

5．线性表$(a_1, a_2,\cdots,a_n)$用顺序映射表示时，a_i 与 a_{i+1}（$1\leqslant i<n$）的物理位置相邻吗？链表表示时呢？

三、算法设计题

1．将两个递增的有序链表合并为一个递增的有序链表。要求结果链表仍使用原来两个链表的存储空间，不另外占用其他的存储空间。表中不允许有重复的数据。

2．将两个非递减的有序链表合并为一个非递增的有序链表。要求结果链表仍使用原来两个链表的存储空间，不另外占用其他的存储空间。表中允许有重复的数据。

3．已知两个链表 A 和 B 分别表示两个集合，其元素递增排列。请设计一个算法，用于求出 A 与 B 的交集，并存放在 A 链表中。

4．已知两个链表 A 和 B 分别表示两个集合，其元素递增排列。请设计算法求出两个集合 A 和 B 的差集（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。

5．设计算法将一个带头结点的单链表 A 分解为两个具有相同结构的链表 B 和 C，其中 B 表的结点为 A 表中值小于零的结点，而 C 表的结点为 A 表中值大于零的结点（链表 A 中的元素为非零整数，要求 B、C 表利用 A 表的结点）。

6．设计一个算法，通过一趟遍历确定长度为 n 的单链表中值最大的结点，返回该结点的数据域。

7．设计一个算法，将链表中所有结点的链接方向“原地”逆转，即要求仅利用原表的存储空间，换句话说，要求算法的空间复杂度为 O(1)。

8．设计一个算法，删除递增有序链表中值大于 mink 且小于 maxk（mink 和 maxk 是给定的两个参数，其值可以和表中的元素相同，也可以不同）的所有元素。

9．已知 p 指向双向循环链表中的一个结点，其结点结构为 data、prior、next 三个域，写出算法 Exchange(p)，交换 p 所指向的结点及其前驱结点的顺序。

10．已知长度为 n 的线性表 A 采用顺序存储结构，请写一个时间复杂度为 O(n)、空间复杂度为 O(1)的算法，该算法可删除线性表中所有值为 item 的数据元素。

11．【2009 年第 42 题】已知一个带有表头结点的单链表，结点结构为（data，link），假设该链表只给出了头指针 list。在不改变链表的前提下，请设计一个尽可能高效的算法，查找链表中倒数第 k 个位置上的结点（k 为正整数）。若查找成功，算法输出该结点的 data 域的值，并返回 1；否则，只返回 0。要求：

（1）描述算法的基本设计思想；

（2）描述算法的详细实现步骤；

（3）根据设计思想和实现步骤，采用程序设计语言描述算法（使用 C、C++或 Java 语言实现），关键之处请给出简要注释。

12.【2010 年第 42 题】设将 n（n>1）个整数存放到一维数组 R 中。试设计一个在时间和空间两方面都尽可能高效的算法，将 R 中保存的序列循环左移 p（0<p<n）个位置，即将 R 中的数据由(x_0，x_1，…，x_{n-1})变换为(x_p，x_{p+1}，…，x_{n-1}，x_0，x_1，…，x_{p-1})。要求：

（1）给出算法的基本设计思想；

（2）根据设计思想，采用 C 或 C++或 Java 语言描述算法，关键之处给出注释；

（3）说明你所设计算法的时间复杂度和空间复杂度。

13.【2011 年第 42 题】一个长度为 L（L≥1）的升序序列 S，处在第⌈L/2⌉个位置的数称为 S 的中位数。例如，若序列 $S1$=(11，13，15，17，19)，则 $S1$ 的中位数是 15。两个序列的中位数是含它们所有元素的升序序列的中位数。例如，若 $S2$=(2，4，6，8，20)，则 $S1$ 和 $S2$ 的中位数是 11。现有两个等长升序序列 A 和 B，试设计一个在时间和空间两方面都尽可能高效的算法，找出两个序列 A 和 B 的中位数。

要求：

（1）给出算法的基本设计思想；

（2）根据设计思想，采用 C 或 C++或 Java 语言描述算法，关键之处给出注释；

（3）说明你所设计算法的时间复杂度和空间复杂度。

14.【2012 年第 42 题】假定采用带头结点的单链表保存单词，当两个单词有相同的后缀时，则可共享相同的后缀存储空间。例如，“loading”和“being”的存储映像如图 2.2 所示。

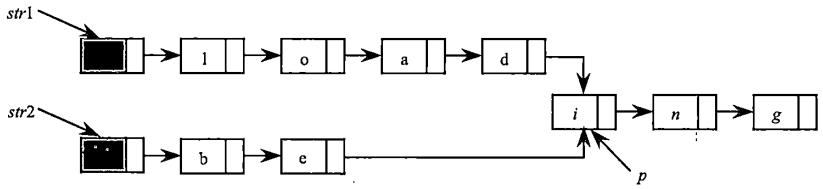


图 2.2　共享相同后缀存储空间两个单词链表

设 *str*1 和 *str*2 分别指向两个单词所在单链表的头结点，链表结点结构为（data，next），请设计一个时间上尽可能高效的算法，找出由 *str*1 和 *str*2 所指的两个链表共同后缀的起始位置（如图中字符 i 所在结点的位置 p）。要求：

（1）给出算法的基本设计思想；

（2）根据设计思想，采用 C 或 C++或 Java 语言描述算法，关键之处给出注释；

（3）说明你所设计算法的时间复杂度。

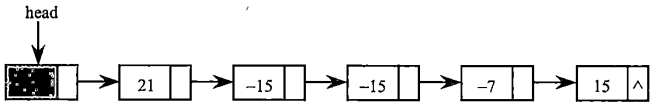
15.【2013 年第 41 题】已知一个整数序列 A=(a_0，a_1，…，a_{n-1})，其中 0≤a_i<n(0≤i<n)。若存在 a_{p1}=a_{p2}=…=a_{pm}=x 且 m>n/2(0≤p_k<n，1≤k≤m)，则称 x 为 A 的主元素。例如 A=(0，5，5，3，5，7，5，5)，则 5 为主元素；又如 A=(0，5，5，3，5，1，5，7)，则 A 中没有主元素。假设 A 中的 n 个元素保存在一个一维数组中，请设计一个尽可能高效的算法，找出 A 的主元素。若存在主元素，则输出该元素；否则输出-1。要求：

（1）给出算法的基本设计思想；

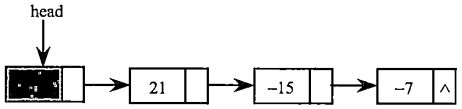
（2）根据设计思想，采用 C 或 C++或 Java 语言描述算法，关键之处给出注释；

（3）说明你所设计算法的时间复杂度和空间复杂度。

16.【2015年第41题】用单链表保存 m 个整数，结点的结构为（data，link），且|data|≤n（n为正整数）。现要求设计一个时间复杂度尽可能高效的算法，对于链表中 data 的绝对值相等的结点，仅保留第一次出现的结点而删除其余绝对值相等的结点。例如，若给定的单链表 head 如图2.3（a）所示，则删除结点后的 head 如图2.3（b）所示。



（a）删除前



（b）删除后

图2.3　整数链表的存储结构

要求：

（1）给出算法的基本设计思想；

（2）使用C或C++语言，给出单链表结点的数据类型定义；

（3）根据设计思想，采用C或C++语言描述算法，关键之处给出注释；

（4）说明你所设计算法的时间复杂度和空间复杂度。

17.【2016年第43题】已知由 $n(n\geq 2)$ 个正整数构成的集合 $A=\{a_k\}$（$0\leq k<n$），将其划分为两个不相交的子集 A_1 和 A_2，元素个数分别是 n_1 和 n_2，A_1 和 A_2 中元素之和分别为 S_1 和 S_2。设计一个尽可能高效的划分算法，满足 $|n_1-n_2|$ 最小且 $|S_1-S_2|$ 最大。要求：

（1）给出算法的基本设计思想；

（2）根据设计思想，采用C或C++语言描述算法，关键之处给出注释；

（3）说明你所设计算法的平均时间复杂度和空间复杂度。

2.2 答案及解析

一、单项选择题

1. B	2. A	3. B	4. A	5. D	6. B	7. C	8. A	9. B	10. D
11. C	12. D	13. D	14. A	15. C	16. D	17. D	18. D	19. C	20. C
21. D	22. D	23. B	24. C	25. C	26. C	27. B	28. D	29. A	30. C

1.【答案】B

【考点】线性表的顺序存储表示

【解析】

顺序表的特点是，逻辑上相邻的数据元素，其物理次序也是相邻的。假设线性表的每个元素需占用 l 个存储单元，并以所占的第一个单元的存储地址作为数据元素的存储起始位置，则线性表的第 i 个数据元素 a_i 的存储位置为：

$$\mathrm{LOC}(a_i) = \mathrm{LOC}(a_1) + (\mathrm{i} - 1) \times l$$

因此，第 5 个元素的地址为：100+(5–1)×2=108。

2.【答案】A

【考点】顺序表基本操作的实现

【解析】

由于顺序表是由数组实现的，它是一种随机存取结构，因此访问第 i 个结点和求第 i 个结点的直接前驱都可以直接通过数组的下标直接定位，时间复杂度是 O(1)，因此答案选择 A。而选项 B 和 C 对应的插入和删除操作都可能需要移动元素，移动元素的个数取决于插入和删除元素的位置，其时间复杂度均为 $O(n)$。选项 D 为排序操作，排序算法需要进行数据的比较和移动，时间复杂度通常为 $O(n^2)$或 $O(n\log_2 n)$。

3.【答案】B

【考点】顺序表的插入

【解析】

假设 p_i 是在第 i 个元素之前插入一个元素的概率，E_{ins} 为在长度为 n 的线性表中插入一个元素时所需移动元素次数的期望值（平均次数），则有：

$$E_{\mathrm{ins}} = \sum_{i=1}^{n+1} p_i(n-i+1)$$

为了不失一般性，可以假定在线性表的任何位置上插入元素都是等概率的，即：

$$p_i = \frac{1}{n+1}$$

则上式可简化为：

$$E_{\mathrm{ins}} = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{n}{2}$$

根据题目可知 n=127，此时 E_{ins}=127/2=63.5。

4.【答案】A

【考点】单链表的定义和表示

【解析】

链接存储的存储结构包括两部分：存储结点的数据域和存储表示结点间关系的指针，因此答案选择 A。

5.【答案】D

【考点】单链表的定义和表示

【解析】

线性表链式存储结构是用一组任意的存储单元存储线性表的数据元素，这组存储单元可以是连续的，也可以是不连续的。

6.【答案】B

【考点】顺序表和链表时间性能的比较

【解析】

链表最大的优点在于进行插入和删除操作时不需要移动数据，只需要修改指针。

7.【答案】C

【考点】顺序表和链表空间性能的比较

【解析】

链表的每个结点除了设置数据域用来存储数据元素外，还要额外设置指针域，用来存储指示元素之间逻辑关系的指针。存储密度是指数据元素本身所占的存储量和整个结点结构所占的存储量之比，假设单链表数据元素本身所占的存储量为 D，指针域所占的存储量为 N，则存储密度为：$D/(D+N)$，一定小于 1。

存储密度越大，存储空间的利用率就越高。显然，顺序表的存储密度为 1。如果单纯从存储密度上来讲，链表的这种存储方式是不经济的。基于此，当线性表的长度变化不大，易于事先确定其大小时，为了节约存储空间，宜采用顺序表作为存储结构。

8.【答案】A

【考点】有序表的合并

【解析】

当一个有序表的最小元素大于另一个表的最大元素时，比较次数最少，总计比较次数为 n 次。

9.【答案】B

【考点】顺序表的插入

【解析】

在顺序表的第 i 个元素（$1 \leqslant i \leqslant n+1$）之前插入一个新元素时，需从最后一个元素即第 n 个元素开始，依次向后移动一个位置，直至第 i 个元素，总计共 $n-i+1$ 个元素，因此答案选择 B。

10.【答案】D

【考点】线性表的定义和特点

【解析】

线性表的结构特点是除第一个元素没有直接前驱，最后一个元素没有直接后继之外，其余每个元素都有一个且仅有一个直接前驱和直接后继，因此选项 D 是正确的，同时可以排除选项 A。线性表中元素的个数 n（$n \geqslant 0$）定义为线性表的长度，$n=0$ 时称为空表，因此选项 B 是错误的。

线性表中的元素可以是无序的，选项 C 显然也是错误的。

11.【答案】C

【考点】单链表的创建

【解析】

创建单链表的时间复杂度是 O(n)，而要创建有序的单链表，则每生成一个新结点都要与已有的结点进行比较，确定恰当的插入位置，因此时间复杂度是 O(n^2)。

12.【答案】D

【考点】顺序表和链表的比较

【解析】

顺序表在定义时将表长 length 定义为顺序表的一个“属性”，所以可以通过返回 length 的值实现求表长的操作，求表长运算的效率高。另外，因为顺序表是由数组实现的，它是一种随机存取结构，指定任意一个位置序号 i，都可以在 O(1)时间内直接存取该位置上的元素，即定位运算的效率高。对于顺序表，求表长、定位这两种运算的时间复杂度均是 O(1)，而对于链表，这两种运算的时间复杂度均是 O(n)。因此选项 A 和 B 的说法都是正确的。

顺序表的存储空间必须预先分配，元素个数扩充受一定限制，易造成存储空间浪费或空间溢出现象；而链表不需要为其预先分配空间，只要内存空间允许，链表中的元素个数就没有限制。因此选项 C 的说法也是正确的。

链式存储结构和顺序存储结构各有优缺点，适用场合各有不同，不能单纯地说谁优于谁。选项 D 的说法是错误的，因此答案选择 D。

13.【答案】D

【考点】单链表的插入

【解析】

如图 2.4 所示，将 s 所指结点 x 插入到 p 所指结点 a 之后需要修改两个指针：

① s->next=p->next;　　//将结点 x 的后继指针指向结点 b

② p->next=s;　　//将结点 a 的后继指针指向结点 x

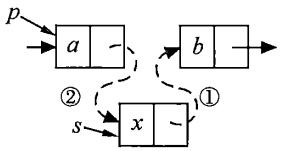


图 2.4　单链表的插入

因此答案选择 D。

注意：选项 D 中的两个语句顺序不能颠倒，即按照这种写法，①和②对应两个指针的修改顺序不能颠倒。如果想先修改左侧指针，再修改右侧指针，只能写成以下三条语句：

① q=p->next;　　//引入指针 q 指向结点 b

② p->next=s;　　//将结点 a 的后继指针指向结点 x

③ s->next=q;　　//将结点 x 的后继指针指向结点 b

14.【答案】A

【考点】双向链表的删除

【解析】

如图 2.5 所示，删除 p 所指的结点 b 时需要修改两个指针：

① p->prior->next=p->next;　　//将结点 a 的后继指针指向结点 c

② p->next->prior=p->prior;　　//将结点 c 的前驱指针指向结点 a

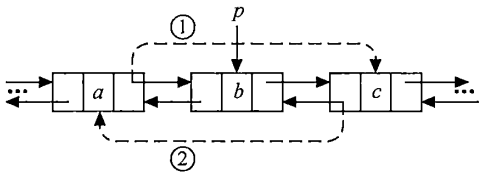


图 2.5　双向链表的删除

因此答案选择 A。

注意：选项 A 中的两个语句顺序是可以颠倒的。

15.【答案】C

【考点】双向链表的插入

【解析】

如图 2.6 所示，在 p 指针所指的结点 a 后插入 q 所指向的新结点 x 时需要修改四个指针：

① q->prior=p;　　//将结点 x 的前驱指针指向结点 p

② q->next=p->next;　　//结点 x 的后继指针指向结点 b

③ p->next->prior=q;　　//结点 b 的前驱指针指向结点 x

④ p->next=q;　　//结点 a 的后继指针指向结点 x

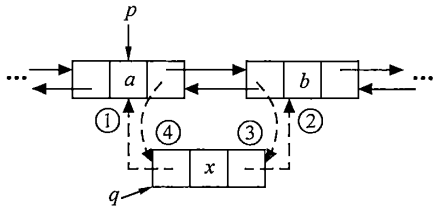


图 2.6　双向链表的插入

因此答案选择 C。

注意：题目描述中为双向循环链表，而在解答此题时，是否为循环链表对答案并无影响。

16.【答案】D

【考点】有序表的合并

【解析】

对两个长度分别为 m 和 n 的有序表的合并问题，如果要求合并后仍然按照原序排列，则最好的情况是其中一个有序表中的元素全部小于另一个有序表中的元素，此时只需比较 $\min(m,n)$次就能完成，时间复杂度为 $O(\min(m,n))$。而最坏的情况是两个表中的元素交替增加，时间复杂度为 $O(m+n)$。如果要求合并后仍然按照原序的逆序排列，则无论最好情况还是最坏情况，时间复杂度都是 $O(m+n)$，但选项中没有提供此答案，我们可以通过分析选择一个与此答案相符的选项。在具体分析时我们假设 m 大于 n，然后可以分以下两种情况进行讨论。

（1）如果 m 非常接近于 n，则 $O(m+n)$接近于 $O(m)$或 $O(n)$，而 $O(m*n)$接近于 $O(n^2)$，因此有 $O(m*n)>>O(m+n)$，选项 B 不正确。

（2）如果 $m>>n$，则 $O(m+n)>>O(n)$，而 $O(\min(m,n))$接近于 $O(n)$，因此有 $O(m+n)>>O(\min(m,n))$，选项 A、C 均不正确。

对于以上两情况，选项 D 都成立。当 m 非常接近于 n 时，$O(m+n)$接近于 $O(m)$或 $O(n)$，也接近于 $O(\max(m,n))$；当 m>>n 时，$O(m+n)$也接近于 $O(\max(m,n))$。

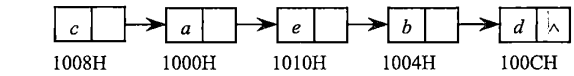
因此答案选择 D。

17.【答案】D

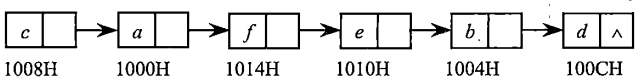
【考点】单链表的插入

【解析】

根据单链表的存储状态，可以得到如图 2.7（a）所示的单链表的逻辑状态。当将 f 存放于 1014H 处并插入到 a 和 e 之间时，只需将 a 的链接地址修改为 f 的存储地址 1014H，同时将 f 的链接地址修改为 e 的存储地址 1010H，得到如图 2.7（b）所示的单链表的逻辑状态。因此，a、e、f 的“链接地址”依次为 1014H、1004H、1010H，答案选择 D。



（a）f 插入前



（b）f 插入后

图 2.7　单链表的存储状态

18.【答案】D

【考点】双向链表的插入

【解析】

如图 2.8 所示，在删除 p 指针所指的结点 b 时需要修改两个指针：

① p->next->prev=p->prev　　//将结点 b 的后继结点的前驱指针指向结点 a

② p->prev->next=p->next　　//将结点 b 的前驱结点的后继指针指向结点 c

因此答案选择 D。

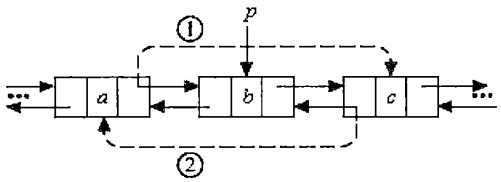


图 2.8　双向链表的插入

注意：题目描述中为双向循环链表，而在解答此题时，是否为循环链表对答案并无影响。

19.【答案】C

【考点】有序表的合并

【解析】

两个递增的有序表在归并为一个递增的有序表时，最好的情况是其中一个有序表中全部元素

都小于另一个有序表，这样只需比较 n 次就能完成归并。

20.【答案】C

【考点】链表的合并

【解析】

如果将一个单链表 A 链接到长度为 m 的单链表 B 之后，首先需要对链表 B 进行遍历，找到链表 B 的表尾（假设用指针 r 指向），然后直接将链表 A 链接到链表 B 的表尾（执行 r->next=A; 操作即可）。由于遍历的时间复杂度为 O(m)，链接的时间复杂度为 O(1)，因此，总的时间复杂度为 O(m)。

21.【答案】D

【考点】顺序表和链表的比较

【解析】

顺序表是一种随机存取结构，指定任意一个位置序号 i，都可以在 O(1)时间内直接存取该位置上的元素。因此答案选择 D。

22.【答案】D

【考点】线性表基本操作的实现

【解析】

插入和删除时因为有数据变更，会改变数据元素之间的结构关系，而排序则因为顺序的改变会改变数据元素之间的结构关系，定位操作不会改变数据元素之间的结构关系，所以答案 D 正确。

23.【答案】B

【考点】单链表的定义和表示

【解析】

若一个头指针为 head 的带头结点的单链表为空表，那么头结点的后继结点为空，所以答案选择 B。

24.【答案】C

【考点】单链表的定义和表示

【解析】

在链表中，为了表示每个数据元素 a_i 与其直接后继数据元素 a_{i+1} 之间的逻辑关系，对数据元素 a_i 来说，除了存储其本身的信息之外，还需存储一个指示其直接后继的信息（即直接后继的存储位置）。这两部分信息组成数据元素 a_i 的存储映像，称为结点。结点包括两个域：其中存储数据元素信息的域称为数据域；存储直接后继存储位置的域称为指针域。指针域中存储的信息称作指针或链，n 个结点（a_i（$1 \leqslant i \leqslant n$）的存储映像）链结成一个链表。因此，选项 C 的说法是错误的，应该为所有结点（包括数据和指针两项）通过指针的链接而组织成单链表。本题答案选择 C。

25.【答案】C

【考点】单链表的取值

【解析】

链表是一种顺序存取结构，按位置访问链表中第 i 个元素时，只能从表头开始依次向后遍历链表，直到找到第 i 个位置上的元素，时间复杂度为 O(n)。

26.【答案】C

【考点】单链表的遍历

【解析】

在单链表中，访问当前结点*p 的后继结点只是进行一次间接寻址的操作，通过 p=p->next 即可完成，时间复杂度是 O(1)。而访问前驱结点则需要从头指针出发顺链进行寻找，直到找到满足条件 q->next==p 的结点*q 为止，时间复杂度是 O(n)。

27.【答案】B

【考点】有序单链表的插入

【解析】

此题插入操作的关键是查找插入位置，时间主要就是花在插入位置的查找上。在有 n 个结点的单链表中，有 n+1 个可能插入的位置，即第一个结点之前和第 i（1≤i≤n）个结点之后。在第一个结点之前插入，需比较一次；在第 i 个结点之后插入需比较 i+1 次。假设在线性表的任何位置上插入元素都是等概率的，则查找插入位置所需的平均比较次数为：

$$\frac{1}{n+1}\sum_{i=1}^{n+1}(i+1)=\frac{n+2}{2}$$

由此可见，在具有 n 个结点的有序单链表中插入一个新结点后，仍然要保持它的有序性，时间复杂度是为 O(n)。

28.【答案】D

【考点】循环链表

【解析】

在链表的末尾插入结点和删除尾结点时，需要修改其相邻结点的指针域，因此需要寻找尾结点或尾结点的前驱结点。对于单链表或单循环链表，寻找尾结点的时间复杂度均为 O(n)。对于带尾指针的单循环链表，可以直接通过尾指针定位到尾结点进行插入，时间复杂度为 O(1)，但在删除尾结点时，则需要通过时间复杂度 O(n)的操作定位到尾结点的前驱结点。而对带头结点的双循环链表而言，可以通过时间复杂度 O(1)的操作直接定位到尾结点或尾结点的前驱结点。所以答案选择 D。

29.【答案】A

【考点】循环链表

【解析】

如图 2.9 所示，循环链表的特点是表中最后一个结点的指针域指向头结点，整个链表形成一个环。由此，p 指针指向链尾的条件是 p 的指针域指向 L，即 p->next==L。

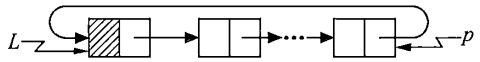


图 2.9　带头指针 L 的循环链表

30.【答案】C

【考点】循环链表

【解析】

如图 2.10 所示，循环链表中如果无头指针而只设尾指针 rear，其头结点的存储位置表示为 rear->next，首元结点的存储位置则为 rear->next->next，而尾结点的存储位置直接通过尾指针 rear 指向。

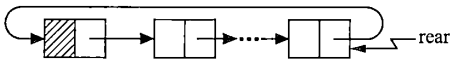


图 2.10　带尾指针 rear 的循环链表

二、应用题

1．解答

（1）选用链式存储结构。链式存储结构可动态申请内存空间，只要内存空间允许，链表中的元素个数就没有限制；此外这种存储结构对元素进行插入和删除操作时都无须移动元素，而仅仅修改指针即可，所以很适用于线性表容量变化的情况。

（2）选用顺序存储结构。顺序表是由数组实现的，它是一种随机存取结构，指定任意一个位置序号 i，都可以在 O(1)时间内直接存取该位置上的元素，即存取速度较高；另外由于题目中描述的线性表的元素总数基本确定，且很少进行插入和删除，故这一特点恰好避开了顺序存储结构的缺点。因此，应选用顺序存储结构。

2．解答

因为单链表没有指向前驱的指针，所以只能由当前结点访问其后的任一结点，而不能从当前结点出发访问到此结点的前驱结点，即单链表不能从当前结点出发访问到任一结点；在双向链表中，由于当前结点既有指向后继结点的指针，又有指向前驱结点的指针，所以双向链表中可以由当前结点出发访问表中的任一结点。

3．解答

在线性表的链式存储结构中，头指针是指向链表中第一个结点的指针。若链表设有头结点，则头指针所指结点为线性表的头结点；若链表不设头结点，则头指针所指结点为该线性表的首元结点。头指针具有标识作用，故常用头指针冠以链表的名字。如图 2.11 所示 L 为头指针，可以称此链表为链表 L。

首元结点是指链表中存储第一个数据元素 a_1 的结点，图 2.11 所示的结点“ZHAO”即是。

头结点是为了操作的统一、方便而设立的。头结点是在首元结点之前附设的一个结点，其指针域指向首元结点。头结点的数据域可以不存储任何信息，也可存储与数据元素类型相同的其他附加信息。例如，当数据元素为整数型时，头结点的数据域中可存放该线性表的长度。增加了头结点后，便于首元结点的处理。因为首元结点的地址保存在头结点(即其“前驱”结点)的指针域中，则对链表的第一个数据元素的操作与其他数据元素相同，无须进行特殊处理。有头结点后，也便于空表和非空表的统一处理，因为无论链表是否为空，头指针都是指向头结点的非空指针。

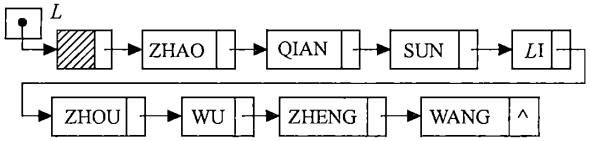


图 2.11　增加头结点的单链表的逻辑状态

4．解答

（1）假设 p_i 是在第 i 个元素之前插入一个元素的概率，E_{ins} 为在长度为 n 的线性表中插入一个元素时所需移动元素次数的期望值（平均次数），则有：

$$E_{\text{ins}}=\sum_{i=1}^{n+1}p_i(n-i+1)$$

若在线性表的任何位置上插入元素都是等概率的，即：

$$p_i=\frac{1}{n+1}$$

因此：

$$E_{\text{ins}}=\frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1)=\frac{n}{2}$$

（2）若元素插在 a_i 与 a_{i+1} 之间（$0\leqslant i\leqslant n-1$）的概率为 $\dfrac{n-i}{n(n+1)/2}$，则有：

$$E_{\text{ins}}=\sum_{i=0}^{n-1}p_i(n-i)$$

若元素插在 a_i 与 a_{i+1} 之间（$0\leqslant i\leqslant n-1$）的概率为：

$$p_i=\frac{n-i}{n(n+1)/2}$$

因此：

$$E_{\text{ins}}=\sum_{i=0}^{n-1}\frac{(n-i)^2}{n(n+1)/2}$$

$$=\frac{n(n+1)(2n+1)/6}{n(n+1)/2}$$

$$=\frac{2n+1}{3}$$

5．解答

顺序映射是用一组地址连续的存储单元依次存储线性表的数据元素，此时 a_i 与 a_{i+1} 的物理位置相邻；链表表示是用一组任意的存储单元存储线性表的数据元素（这组存储单元可以是连续的，也可以是不连续的），此时 a_i 与 a_{i+1} 的物理位置不一定相邻。

三、算法设计题

1．解答

【算法思想】

要求利用现有的表中的结点空间建立新表，可通过更改结点的指针域来重新建立新的元素之间的线性关系。此题的关键点在于：为保证新表和原来一样递增有序，可以利用后插法建立单链表。

算法思想是：如图 2.12 所示，假设待合并的链表为 *La* 和 *Lb*，合并后的新表使用头指针 *Lc*（*Lc* 的表头结点设为 *La* 的表头结点）指向。*pa* 和 *pb* 分别是链表 *La* 和 *Lb* 的工作指针，初始化为相应链表的首元结点。从首元结点开始进行比较，当两个链表 *La* 和 *Lb* 均未到达表尾结点时，依

次摘取其中较小者重新链接在 *Lc* 表的最后。如果两个表中的元素相等，只摘取 *La* 表中的元素，删除 *Lb* 表中的元素，这样确保合并后的表中无重复的元素。当一个表到达表尾结点为空时，将非空表的剩余元素直接链接在 *Lc* 表的最后。最后释放链表 *Lb* 的头结点。

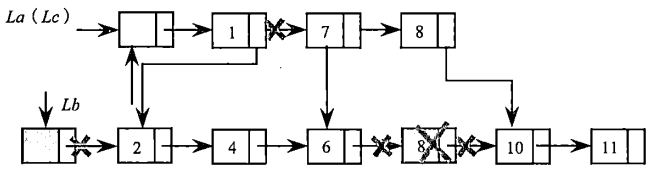


图 2.12　两个递增的有序链表的合并

【算法描述】

```
void MergeList(LinkList &La,LinkList &Lb,LinkList &Lc)
{//将两个递增的有序链表 La 和 Lb 合并为一个递增的有序链表 Lc
  pa=La->next;                          //pa 是链表 La 的工作指针,初始化为首元结点
  pb=Lb->next;                          //pb 是链表 Lb 的工作指针,初始化为首元结点
  //pa 和 pb 分别是链表 La 和 Lb 的工作指针,初始化为相应链表的首元结点
  Lc=pc=La;                             //用 La 的头结点作为 Lc 的头结点
  while(pa&&pb)                         //两个链表 La 和 Lb 均未到达表尾结点
  {
    if(pa->data<pb->data)
    {//取较小者 La 中的元素，将 pa 链接在 pc 的后面，pa 指针后移
      pc->next=pa;
      pc=pa;
      pa=pa->next;
    }
    else if(pa->data>pb->data)
    {//取较小者 Lb 中的元素，将 pb 链接在 pc 的后面，pb 指针后移
      pc->next=pb;
      pc=pb;
      pb=pb->next;
    }
    else
    {//相等时取 La 中的元素，删除 Lb 中的元素
      pc->next=pa;
      pc=pa;
      pa=pa->next;
      q=pb->next;
      delete pb;
      pb=q;
    }
  }
  pc->next=pa?pa:pb;                    //将非空表的剩余元素直接链接在 Lc 表的最后
  delete Lb;                            //释放 Lb 的头结点
}
```

2．解答

【算法思想】

与第 1 题类似的思路，从原有两个链表中依次摘取结点，通过更改结点的指针域来重新建立

新的元素之间的线性关系，得到一个新链表。但与第1题不同的关键点有两个：（1）为保证新表与原表顺序相反，需要利用前插法建立单链表，而不能利用后插法；（2）当一个表到达表尾结点为空时，另一个非空表的剩余元素应该依次摘取，依次链接在 *Lc* 表的表头结点之后，而不能全部直接链接在 *Lc* 表的最后。

算法思想是：假设待合并的链表为 *La* 和 *Lb*，合并后的新表使用头指针 *Lc*（*Lc* 的表头结点设为 *La* 的表头结点）指向。*pa* 和 *pb* 分别是链表 *La* 和 *Lb* 的工作指针，初始化为相应链表的首元结点。从首元结点开始进行比较，当两个链表 *La* 和 *Lb* 均为到达表尾结点时，依次摘取其中较小者重新链接在 *Lc* 表的表头结点之后，如果两个表中的元素相等，只摘取 *La* 表中的元素，保留 *Lb* 表中的元素。当一个表到达表尾结点为空时，将非空表的剩余元素依次摘取，链接在 *Lc* 表的表头结点之后。最后释放链表 *Lb* 的头结点。

【算法描述】

```
void MergeList(LinkList &La,LinkList &Lb,LinkList &Lc)
{//将两个非递减的有序链表 La 和 Lb 合并为一个非递增的有序链表 Lc
  pa=La->next;                          //pa 是链表 La 的工作指针,初始化为首元结点
  pb=Lb->next;                          //pb 是链表 Lb 的工作指针,初始化为首元结点
  Lc=pc=La;                             //用 La 的头结点作为 Lc 的头结点
  Lc->next=NULL;
  while(pa||pb)                         //只要有一个表未到达表尾结点，用 q 指向待摘取的元素
  {
    if(!pa)                             //La 表为空，用 q 指向 pb，pb 指针后移
    {
      q=pb;
      pb=pb->next;
    }
    else if(!pb)                        //Lb 表为空，用 q 指向 pa，pa 指针后移
    {
      q=pa;
      pa=pa->next;
    }
    else if(pa->data<=pb->data)         //取较小者 La 中的元素，用 q 指向 pa，pa 指针后移
    {
      q=pa;
      pa=pa->next;
    }
    else                                //取较小者 Lb 中的元素，用 q 指向 pb，pb 指针后移
    {
      q=pb;
      pb=pb->next;
    }
    q->next=Lc->next; Lc->next=q;       //将 q 指向的结点插在 Lc 的表头结点之后
  }
  delete Lb;                            //释放 Lb 的头结点
}
```

3. 解答

【算法思想】

A 与 B 的交集是指同时出现在两个集合中的元素，因此，此题的关键点在于：依次摘取两个表中相等的元素重新进行链接，删除其他不等的元素。

算法思想是：假设待合并的链表为 *La* 和 *Lb*，合并后的新表使用头指针 *Lc*（*Lc* 的表头结点设为 *La* 的表头结点）指向。*pa* 和 *pb* 分别是链表 *La* 和 *Lb* 的工作指针，初始化为相应链表的首元结点。从首元结点开始进行比较，当两个链表 *La* 和 *Lb* 均为到达表尾结点时，如果两个表中的元素相等，摘取 *La* 表中的元素，删除 *Lb* 表中的元素；如果其中一个表中的元素较小，删除此表中较小的元素，此表的工作指针后移。当链表 *La* 和 *Lb* 有一个到达表尾结点为空时，依次删除另一个非空表中的所有元素。最后释放链表 *Lb* 的头结点。

【算法描述】

```
void Intersection(LinkList &La,LinkList &Lb,LinkList &Lc)
{                                    //求两个递增的有序链表 La 和 Lb 的交集，使用头指针 Lc 指向
  pa=La->next;                       //pa 是链表 La 的工作指针,初始化为首元结点
  pb=Lb->next;                       //pb 是链表 Lb 的工作指针,初始化为首元结点
  Lc=pc=La;                          //用 La 的头结点作为 Lc 的头结点
  while(pa&&pb)                      //两个链表 La 和 Lb 均未到达表尾结点
  {
    if(pa->data==pb->data)           //相等，交集并入结果表中
    {
      pc->next=pa; pc=pa; pa=pa->next;
      //取 La 中的元素，将 pa 链接在 pc 的后面，pa 指针后移
      u=pb; pb=pb->next; delete u;
      //删除 Lb 中的对应的相等元素
    }
    else if(pa->data<pb->data)       //删除较小者 La 中的元素
    {
      u=pa;
      pa=pa->next;
      delete u;
    }
    else                             //删除较小者 Lb 中的元素
    {
      u=pb;
      pb=pb->next;
      delete u;
    }
  }
  while(pa)                          //Lb 为空，删除非空表 La 中的所有元素
  {
    u=pa;
    pa=pa->next;
    delete u;
  }
  while(pb)                          //La 为空，删除非空表 Lb 中的所有元素
  {
    u=pb;
    pb=pb->next;
    delete u;
  }
  pc->next=NULL;                     //置链表 Lc 尾标记
  delete Lb;                         //释放 Lb 的头结点
}
```

4．解答

【算法思想】

求两个集合 A 和 B 的差集是指在 A 中删除 A 和 B 中共有的元素，即删除链表中的数据域相等的结点。由于要删除结点，此题的关键点在于：在遍历链表时，需要保存待删除结点的前驱。

算法思想是：假设待求的链表为 *La* 和 *Lb*，*pa* 和 *pb* 分别是链表 *La* 和 *Lb* 的工作指针，初始化为相应链表的首元结点。pre 为 *La* 中 *pa* 所指结点的前驱结点的指针，初始化为 *La*。从首元结点开始进行比较，当两个链表 *La* 和 *Lb* 均未到达表尾结点时，如果 *La* 表中的元素小于 *Lb* 表中的元素，*La* 表中的元素即为待求差集中的元素，差集元素个数增 1，pre 置为 *La* 表的工作指针 *pa*，*pa* 后移；如果 *Lb* 表中的元素小于 *La* 表中的元素，*pb* 后移；如果 *La* 表中的元素等于 *Lb* 表中的元素，则在 *La* 表删除该元素值对应的结点。

【算法描述】

```
void Difference(LinkList &La,LinkList &Lb,int &n)
{//La 和 Lb 差集的结果存储于 La 中，n 是结果集合中元素个数，调用时为 0
   pa=La->next;                          //pa 是链表 La 的工作指针,初始化为首元结点
   pb=Lb->next;                          //pb 是链表 Lb 的工作指针,初始化为首元结点
   pre=La;                               //pre 为 La 中 pa 所指结点的前驱结点的指针
   while(pa&&pb)                         //两个链表 La 和 Lb 均未到达表尾结点
   {
      if(pa->data<pb->data)              //A 链表中当前结点指针后移
      {
         n++;
         pre=pa;
         pa=pa->next;
      }
      else if(pa->data>pb->data)
         pb=pb->next;                    //B 链表中当前结点指针后移
      else                               //在 La 表删除 La 和 Lb 中元素值相同的结点
      {
         pre->next=pa->next;
         u=pa;
         pa=pa->next;
         delete u;                       //删除结点
      }
   }
   while(pa)
   {
     n++
     pa=pa->next
   }
}
```

5．解答

【算法思想】

题目要求将一个单链表 *A* 分解为两个具有相同结构的链表 *B*、*C*，因此，此题的关键点在于：（1）*B* 表的头结点可以使用原来 *A* 表的头结点，而需要为 *C* 表新申请一个头结点；（2）对 *A* 表进行遍历的同时进行分解，完成结点的重新链接，在此过程中需要记录遍历的后继结点以防止链接后丢失后继结点；（3）本题并未要求链表中结点的数据值有序，所以在摘取满足条件的结点进行链接时，可以采取前插法，也可以采取后插法。因为前插法的实现相对简单，下面的算法描述中采取了前插法。

算法思想是：首先将 *B* 表的头结点初始化 *A* 表的头结点，为 *C* 表新申请一个头结点，初始化

为空表。从 A 表的首元结点开始，依次对 A 表进行遍历。p 为工作指针，r 为 p 的后继指针。当 p->data<0 时，将 p 指向的结点使用前插法插入到 B 表；当 pa->data>0 时，将 p 指向的结点使用前插法插入到 C 表，然后 p 指向新的待处理的结点（$p=r$）。

【算法描述】

```
void Decompose(LinkList &A, LinkList &B, LinkList &C)
{//单链表 A 分解为两个具有相同结构的链表 B 和 C
  B=A;
  B->next=NULL;                    //B 表初始化
  C=new LNode;                     //为 C 申请结点空间
  C->next=NULL;                    //C 初始化为空表
  p=A->next;                       //p 为工作指针
  while(p!=NULL)                   //表 A 未到达表尾结点
  {
    r=p->next;                     //暂存 p 的后继
    if(p->data<0)                  //将小于 0 的结点链入 B 表,前插法
    {
      p->next=B->next;
      B->next=p;
    }
    else                           //将大于 0 的结点链入 C 表,前插法
    {
      p->next=C->next;
      C->next=p;
    }
    p=r;                           //p 指向新的待处理结点
  }
}
```

6. 解答

【算法思想】

此题的关键点在于：在遍历的时候利用指针 pmax 记录值最大的结点的位置。

算法思想是：初始时指针 pmax 指向首元结点，然后在遍历过程中，用 pmax 依次和后面的结点进行比较，发现大者则用 pmax 指向该结点。这样将链表从头到尾遍历一遍时，pmax 所指向的结点中的数据即为最大值。

【算法描述】

```
ElemType Max(LinkList L)
{//确定单链表中值最大的结点
  if(L->next==NULL) return NULL;
  pmax=L->next;                    //pmax 指向首元结点
  p=L->next->next;                 //p 指向第二个结点
  while(p!=NULL)                   //遍历链表，如果下一个结点存在
  {
    if(p->data>pmax->data)
      pmax=p;                      //pmax 指向数值大的结点
    p=p->next;                     //p 指向下一个结点，继续遍历
  }
  return pmax->data;
}
```

7. 解答

【算法思想】

此题的关键点在于：不能开辟新的空间，只能改变指针的指向。因此，可以考虑逐个摘取结点，利用前插法创建链表的思想，将结点依次插到头结点的后面。因为先插入的结点为表尾，后插入的结点为表头，即可实现链表的逆转。

算法思想是：利用原有的头结点 L，p 为工作指针，初始时 p 指向首元结点。因为摘取的结点依次向前插入，为确保链表尾部为空，初始时应将头结点的指针域置为空。然后从前向后遍历链表，依次摘取结点，在摘取结点前需要用指针 q 记录后继结点，以防止链接后丢失后继结点，之后将摘取的结点插入到头结点的后面，最后 p 指向新的待处理的结点（$p=q$）。

【算法描述】

```
void Inverse(LinkList &L)
{//逆置带头结点的单链表 L
   p=L->next;                          //p 指向首元结点
   L->next=NULL;                       //头结点的指针域置为空
   while(p!=NULL)                      //遍历链表，如果下一个结点存在
   {
      q=p->next;                       //q 指向*p 的后继
      p->next=L->next;
      L->next=p;                       //*p 插入在头结点之后
      p=q;
   }
}
```

8. 解答

【算法思想】

此题的关键点在于：通过遍历链表能够定位待删除元素的下边界和上边界，即可找到第一个值大于 mink 的结点和第一个值大于等于 maxk 的结点，分别如图 2.13 所示指针 q 和 p 指向的两个结点。

算法思想是：（1）查找第一个值大于 mink 的结点，用 q 指向该结点，pre 指向该结点的前驱结点；（2）继续向下遍历链表，查找第一个值大于等于 maxk 的结点，用 p 指向该结点；（3）修改下边界前驱结点的指针域，使其指向上边界（pre->next=p）；（4）依次释放待删除结点的空间（图中介于 pre 和 p 之间所有的结点）。

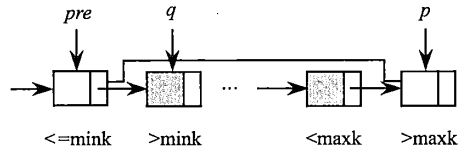


图 2.13 删除递增有序链表中值大于 mink 且小于 maxk 的所有元素

【算法描述】

```
void DeleteMinMax(LinkList &L, int mink, int maxk)
{//删除递增有序链表 L 中值大于 mink 且小于 maxk 的所有元素
   p=L->next;                          //p 指向首元结点
   while(p&&p->data<=mink)             //查找第一个值大于 mink 的结点
   {
```

```
        pre=p;                          //pre 指向前驱结点
        p=p->next;
    }
    while(p&&p->data<maxk)              //查找第一个值大于等于 maxk 的结点
        p=p->next;
    q=pre->next;
    pre->next=p;                        //修改待删除结点的指针
    while(q!=p)                         //依次释放待删除结点的空间
    {
        s=q->next;
        delete q;
        q=s;
    }
}
}
```

9. 解答

【算法思想】

此题的关键点在于：双向循环链表的一个结点与前驱交换将涉及四个结点（结点*p，前驱结点*q，*q 的前驱结点，*p 的后继结点），如图 2.14 所示，总计需要改变六条链（结点*p 的前驱指针和后继指针、结点*q 的前驱指针和后继指针、结点*q 的前驱结点的后继指针、结点*p 的后继结点的前驱指针）。由于在改变指针前，需要用 q 指向结点*p 的前驱结点，因此，在图 2.14 中，总计有七条指针的指向变化。

算法思想是：（1）用 q 指向结点*p 的前驱结点；（2）结点*q 的前驱结点的后继指针指向结点*p；（3）结点*p 的前驱指针指向结点*q 的前驱结点；（4）结点*q 的后继指针指向结点*p 的后继结点；（5）结点*q 的前驱指针指向结点*p；（6）结点*p 的后继结点的前驱指针指向 q；（7）结点*p 的后继指针指向结点*q。

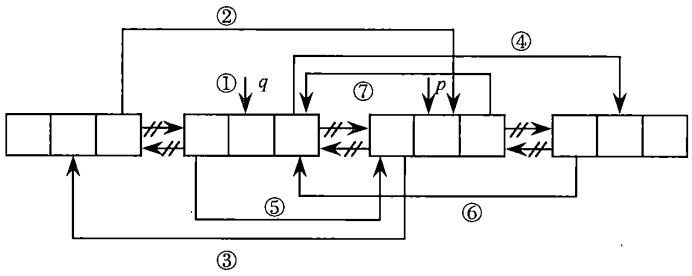


图 2.14　双向循环链表交换 p 所指向的结点及其前驱结点的指针改变示意图

【算法描述】

```
void Exchange(DuLinkList p)
{//在双向循环链表，交换 p 所指向的结点及其前驱结点的顺序
    q=p->prior;                     //对应图 2.14①
    q->prior->next=p;               //对应图 2.14②
    p->prior=q->prior;              //对应图 2.14③
    q->next=p->next;                //对应图 2.14④
    q->prior=p;                     //对应图 2.14⑤
```

```
    p->next->prior=q;                    //对应图 2.14⑥
    p->next=q;                           //对应图 2.14⑦
}
```

10. 解答

【算法思想】

此题的关键点在于：在时间复杂度为 O(n)、空间复杂度为 O(1)条件的限定下，不能另开辟空间，且只能通过遍历一趟顺序表来完成。

算法思想是：用 k 记录顺序表 A 中不等于 item 的元素个数，k 初始为 0。可以采用类似建立顺序表的思想，从前向后遍历顺序表，查找值不为 item 的元素，如果找到，则利用原表的空间记录值不为 item 的元素，同时使 k 增 1。遍历结束后，顺序表中前 k 个元素即为值不为 item 的元素，最后将顺序表的长度置为 k。

【算法描述】

```
void DeleteItem(SqList &A,ElemType item)
{//删除顺序表 A 中所有值为 item 的元素
   k=0;                                  //k 记录值不等于 item 的元素个数
   for(i=0;i<A.length;i++)               //从前向后扫描顺序表
      if(A.Elem[i]!=item)                //查找值不为 item 的元素
      {
         A.Elem[k]=A.Elem[i];            //利用原表的空间记录值不为 item 的元素
         k++;                            //不等于 item 的元素增 1
      }
   A.Length=k;                           //顺序表 A 的长度等于 k
}
```

11. 解答

(1)【算法思想】

定义指针 p 和 q，初始化时均指向单链表的的首元结点。首先将 p 沿链表移动到第 k 个结点，而 q 指针保持不动，这样当 p 移动到第 k+1 个结点时，p 和 q 所指结点的间隔距离为 k。然后 p 和 q 同时向下移动，当 p 为 NULL 时，q 所指向的结点就是该链表倒数第 k 个结点。

(2)【算法步骤】

① 计数器 i=0，用指针 p 和 q 指向首元结点。

② 从首元结点开始依次顺着链域 link 依次向下遍历链表，若 p 为 NULL，则转步骤⑤。

③ 若 i 小于 k，则 i 加 1；否则，q 指向下一个结点。

④ p 指向下一个结点，转步骤②。

⑤ 若 i 等于 k，则查找成功，输出该结点的 data 域的值，返回 1；否则，查找失败，返回 0。

(3)【算法描述】

```
typedef struct LNode
{
   int data;
   struct LNode *next;
   }LNode. *LinkList;
int Search_k(LinkList list, int k)
{//查找链表 list 中倒数第 k 个位置上的结点
   i=0;                                  //计数器赋初值
   p=q=list->next;                       //p 和 q 指向首元结点
```

```
    while(p!=NULL)                    //顺链域向后扫描，直到 p 为空
    {
      if(i<k) i++;                    //计数器加 1
      else q=q->next;                 //q 移到下一个结点
      p=p->next;                      //p 移到下一个结点
    }
    if(i==k)
    {
      cout<<q->data;                  //查找成功，输出该结点的 data 域的值
      return 1;
    }
    else
      return 0;                       //如果链表的长度小于 k，查找失败
}
```

12. 解答

（1）【算法思想】

先将 n 个数据 x_0，x_1，…，x_{n-2}，x_{n-1} 原地逆置，得到 x_{n-1}，x_{n-2}，…，x，x_0，然后再将前 n-p 个数据和后 p 个数据分别原地逆置，得到最终结果：x_p，x_{p+1}，…，x_{n-1}，x_0、x_1，…，x_{p-1}。

（2）【算法描述】

```
void Reverse(int R[],int left,int right)
{//将数组 R 中的数据原地逆置
  i=left; j=right;                    //i 等于左边界 left，j 等于右边界 right
  while(i<j)                          //交换 r[i]与 r[j]的值
  {
    temp=r[i];
    r[i]=r[j];
    r[j]=temp;
    i++;                              //i 右移一个位置
    j--;                              //j 左移一个位置
  }
}
void LeftShift(int R[],int n,int p)
{//将长度为 n 的数组 R 中的数据循环左移 p 个位置
  if(p>0&&p<n)
  {
    Reverse(r,0,n-1);                 //将全部数据逆置
    Reverse(r,0,n-p-1);               //将前 n-p 个数据逆置
    Reverse(r,n-p,n-1);               //将后 p 个数据逆置
  }
}
```

（3）【算法分析】

算法执行了三趟逆置，时间复杂度为 O(n)；用了一个辅助变量空间，空间复杂度为 O(1)。

13. 解答

（1）【算法思想】

分别求出序列 A 和 B 的中位数，设为 a 和 b，算法具体求解过程如下：

① 若 a 等于 b，则 a 或 b 即为所求的中位数，算法结束；

② 若 a 小于 b，则舍弃序列 A 中较小的一半，同时舍弃序列 B 中较大的一半，且要求两次舍

弃的元素个数相同；

③ 若 a 大于 b，则舍弃序列 A 中较大的一半，同时舍弃序列 B 中较小的一半，且要求两次舍弃的元素个数相同；

在保留的两个升序序列中，重复上述过程，直到两个序列中均只含一个元素时为止，较小者即为所求的中位数。

（2）【算法描述】

```
int Search_Mid(int A[], int B[], int n)
{//求两个长度均为 n 的序列 A 和 B 的中位数
  start1=0; end1=n-1;                    //序列 A 的头、尾指针初始化
  start2=0; end2=n-1;                    //序列 B 的头、尾指针初始化
  while(start1!=end1||start2!=end2)
  {
    m1=(start1+end1)/2;
    m2=(start2+end2)/2;
    if(A[m1]==B[m2])                     //满足条件①，中位数相等，直接返回值
      return A[m1];
    else if(A[m1]<B[m2])                 //满足条件②，a 为较小中位数时
    {//分别考虑奇数和偶数，保持两个子数组元素个数相等
      if((start1+end1)%2==0)             //若元素为奇数个
      {
        start1=m1;                       //舍弃 A 中间点以前的部分且保留中间点
        end2=m2;                         //舍弃 B 中间点以后的部分且保留中间点
      }
      else                               //若元素为偶数个
      {
        start1=m1+1;                     //舍弃 A 的前半部分
        end2=m2;                         //舍弃 B 的后半部分
      }
    }
    else                                 //满足条件③，a 为较大中位数
    {
      if((start1+end1)%2==0)             //若元素为奇数个
      {
        end1=m1;                         //舍弃 A 中间点以后的部分且保留中间点
        start2=m2;                       //舍弃 B 中间点以前的部分且保留中间点
      }
      else                               //若元素为偶数个
      {
        end1=m1;                         //舍弃 A 的后半部分
        start2=m2+1;                     //舍弃 B 的前半部分
      }
    }
  }
  return A[start1]<B[start2]?A[start1]:B[start2];
}
```

（3）【算法分析】

算法时间复杂度为 $O(\log_2 n)$，空间复杂度为 $O(1)$。

14. 解答

（1）【算法思想】

因为两个链表的长度不一定相同，所以当从头开始同时遍历两个链表到尾结点时，并不能保证两个链表同时到达尾结点。假设一个链表比另一个链表长 k 个结点，则可以先在较长的链表上遍历 k 个结点，之后同步从头遍历较短的链表，在遍历过程中，判断两个指针是否指向同一结点。若是，则该结点为共同后缀的起始位置。具体思路如下：

① 分别求出链表 *str*1 和 *str*2 的长度，记为 m 和 n。

② 比较 m 和 n，计算两个链表的长度之差 k，$k=|L1-L2|$。令指针 long 指向较长链表的首元结点，指针 short 指向较短链表的首元结点。

③ 在较长的链表上遍历 k 个结点，这样将两个链表以表尾对齐，保证指针 long 和 short 所指的结点到表尾的长度相等。

④ 同步遍历两个链表，反复将指针 long 和 short 同步向后移动，并判断它们是否指向同一结点。若 long 和 short 指向同一结点（通过比较结点的地址是否相等进行判断，而不是比较结点的值），则该结点即为所求的共同后缀的起始位置。

（2）【算法描述】

```
typedef struct LNode
{
   char data;
   struct LNode *next;
   }LNode. *LinkList;
LinkList FindSuffix(LinkList str1, LinkList str2)
{//求 str1 和 str2 所指的两个链表共同后缀的起始位置
   p=str1->next;                    //p 指向链表 str1 的首元结点
   q=str2->next;                    //q 指向链表 str2 的首元结点
   m=n=0;
   while(p!=NULL)                   //求链表 str1 的长度，记为 m
      {m++;p=p->next;}
   while(q!=NULL)                   //求链表 str2 的长度，记为 n
      {n++;q=q->next;}
   if(m>n)                          //链表 str1 较长
   {
      long=str1->next;              //long 指向较长链表的首元结点
      short=str2->next;             //short 指向较短链表的首元结点
      k=m-n;                        //表长之差
   }
   else
   {
      long=str2->next;
      short=str1->next;
      k=n-m;
   }
   while(k--)                       //在较长的链表上遍历 k 个结点
      long=long->next;
   while(long!=NULL)                //同步遍历两个链表，寻找共同后缀的起始位置
   {
      if(long==short)               //查找成功，返回共同后缀的起始位置
```

```
      return long;
    else                                    //未找到，两个指针同步向后移动
    {
      long=long->next;
      short=short->next;
    }
  }
  return NULL;                              //查找失败
```

（3）【算法分析】

算法时间复杂度为 O(m+n)或 O(max(m，n))，空间复杂度为 O (1)。

15．解答

（1）【算法思想】

主元素是数组中出现次数超过一半的元素。当数组中存在主元素时，所有非主元素的个数和必少于一半。如果让主元素与一个非主元素“配对”，则最后多出来的元素（没有元素与之配对）就是主元素。此题的关键点是：在主元素未知时，如何判断主元素并完成“配对”。具体思路如下。

① 选取候选主元素：从前向后依次扫描数组中的每个整数，假定第一个整数为主元素，将其保存到 Key 中，计数为 1；若遇到的下一个整数仍等于 Key，则计数加 1，否则计数减 1。当计数减到 0 时，将遇到的下一个整数保存到 Key 中，计数重新记为 1，开始新一轮计数，即从当前位置开始重复上述过程，直到将全部数组元素扫描完毕。

② 判断 Key 中的元素是否是真正的主元素：再次扫描该数组，统计 Key 中元素出现的次数，若大于 n/2，则为主元素，否则，序列中不存在主元素。

（2）【算法描述】

```
int MainElement(int A[], int n)
{//求整数序列 A 的主元素
  count=1;                                  //count 用来计数
  Key=A[0];                                 //Key 用来保存候选主元素,初始为 A[0]
  for(i=1;i<n;i++)                          //扫描数组，选取候选主元素
  {
    if(A[i]==Key) count++;                  //候选主元素计数加 1
    else
      if(count>0) count--;                  //非候选主元素计数减 1
      else                                  //更换候选主元素，重新计数
      {
        Key=A[i];
        count=1;
      }
  }
  if(count>0)
    for(i=count=0;i<n;i++)                  //统计候选主元素的实际出现次数
      if(A[i]==Key) count++;
  if(count>n/2) return Key;                 //确认主元素
  else return -1;                           //不存在主元素
}
```

（3）【算法分析】

算法时间复杂度为 O(n)，空间复杂度为 O(1)。

16. 解答

（1）【算法思想】

因为题目要求设计一个时间复杂度尽可能高效的算法，而已知|data|≤n，所以可以考虑用空间换时间的方法。申请一个空间大小为 n+1（0 号单元未用）的辅助数组，保存链表中已出现的数值，通过对链表进行一趟扫描来完成删除。具体思路如下：

① 申请大小为 n+1 的辅助数组 t 并赋初值 0；

② 从首元结点开始遍历链表，依次检查 t[|data|]的值，若 t[|data|]为 0，即结点首次出现，则保留该结点，并置 t[|data|]=1；若 t[|data|]不为 0，则将该结点从链表中删除。

（2）

单链表结点的数据类型定义如下：

```
typedef struct LNode
{
  int data;
  struct LNode *next;
}LNode,*LinkList;
```

（3）【算法描述】

```
void DeleteEqualNode(LinkList head, int n)
{//删除单链表中绝对值相等的结点
  t=new int[n+1];                    //辅助数组 t，0 号单元未用，故大小为 n+1
  for(i=0;i<n;i++)                   //数组元素初值置为 0
    *(t+i)=0;
  p=head->next;                      //p 指向首元结点
  while(p!=NULL)                     //顺链域向后扫描，直到 p 为空
  {
    if(t[abs(p->data)]==1)           //此绝对值已经在结点中出现过，删除该结点
    {
      r->next=p->next;
      delete p;
      p=r->next;
    }
    else                             //未出现，保留该结点
    {
      t[abs(p->data)]=1;             //将数组中对应位置的元素置为 1
      r=p;
      p=p->next;                     //向后遍历链表
    }
  }
}
```

（4）【算法分析】

对长度为 m 的链表进行一趟遍历，因此算法的时间复杂度为 O(m)；而申请空间大小为 n+1 的辅助数组，因此空间复杂度为 O (1)。

17. 解答

（1）【算法思想】

将最小的⌊n/2⌋个元素放在 A_1 中，其余的元素放在 A_2 中，划分结果即可满足要求。该算法并不需要对全部元素进行全排序，可仿照快速排序的思想，基于枢轴将 n 个整数划分为两个子集。根据划分后枢轴所处的位置 i 分以下三种情况处理：

① 若 $i=\lfloor n/2 \rfloor$，则划分成功，算法结束；

② 若 $i<\lfloor n/2 \rfloor$，则枢轴及之前的所有元素均属于 A_1，继续对 i 之后的元素进行划分；

③ 若 $i>\lfloor n/2 \rfloor$，则枢轴及之后的所有元素均属于 A_2，继续对 i 之前的元素进行划分。

（2）【算法描述】

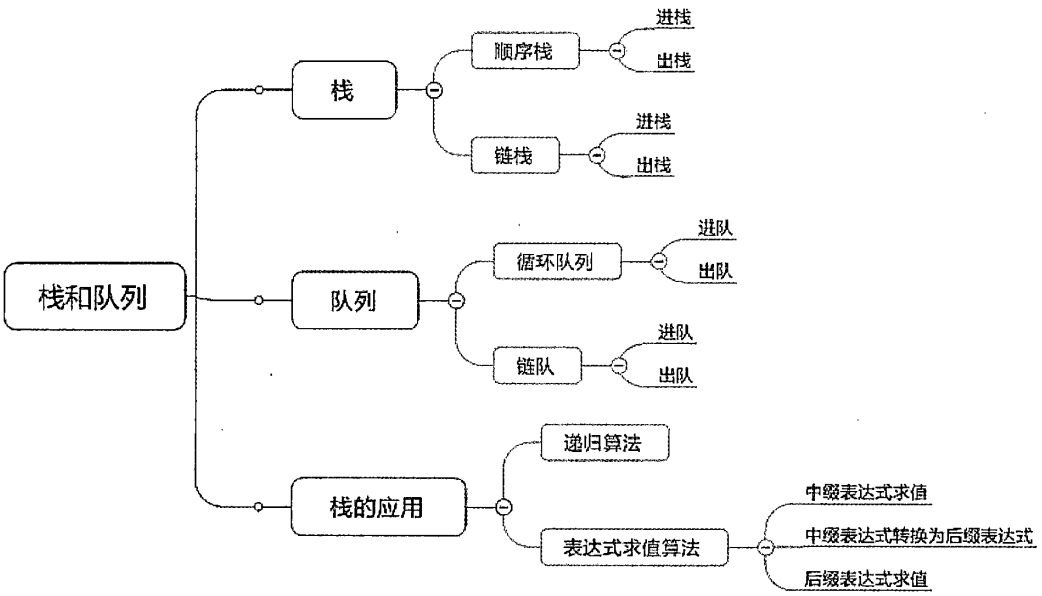
```
int Partition(int a[],int n)
{//将正整数构成的集合划分为两个不相交的子集 A1 和 A2
   low=0,high=n-1;                    //分别指向表的下界和上界
   low0=0,high0=n-1;                  //分别指向新的子表的下界和上界
   s1=0,s2=0;                         //分别记录 A1 和 A2 中元素的和
   flag=1;                            //标记划分是否成功
   k=n/2;                             //记录表的中间位置
   while(flag)                        //循环进行划分
   {
      pivotkey=a[low];                //选择枢轴
      while(low<high)                 //从两端交替地向中间扫描
      {
         while(low<high&&a[high]>=pivotkey)
            --high;                   //从最右侧位置依次向左搜索
         if(low!=high)
            a[low]=a[high];           //将比枢轴记录小的记录移到低端
         while(low<high&&a[low]<=pivotkey)
            ++low;                    //从最左侧位置依次向右搜索
         if(low!=high)
            a[high]=a[low];           //将比枢轴记录大的记录移到高端
      }//end of while(low<high)
      a[low]=pivotkey;                //枢轴记录到位
      if(low==k-1)                    //满足条件①,枢轴的位置是 n/2 的前一位置,
        flag=0;                       //划分成功,下次循环将退出划分
      else                            //继续划分
      {
         if(low<k-1)                  //满足条件②,枢轴 low 及之前的所有元素均属于 A1
         {
            low0=++low;               //继续对 low 之后的元素进行划分
            high=high0;
         }
         else                         //满足条件③,枢轴 low 及之后的所有元素均属于 A2
         {
            high0=--high;             //继续对 low 之前的元素进行划分
            low=low0;
         }
       }
   }
   for(i=0;i<k;i++) s1+=a[i];         //求解子集 A1 中元素的和
   for(i=k;i<n;i++) s2+=a[i];         //求解子集 A2 中元素的和
   return s2-s1;
}
```

（3）【算法分析】

平均时间复杂度是 $O(n)$，空间复杂度是 $O(1)$。

第3章 栈和队列

【知识导图】



【学习目标】

1. 掌握栈的顺序栈和链栈的进栈和出栈算法，明确栈空和顺序栈栈满的条件。
2. 掌握循环队列和链队列的进队和出队算法，明确队空和循环队列队满的条件。
3. 深刻理解递归算法执行过程中栈的状态变化过程，便于更好地使用递归算法。
4. 要求能够灵活运用栈和队列设计解决实际应用问题，掌握中缀表达式求值、中缀表达式转换后缀表达式以及后缀表达式的求值算法。

3.1 习题

一、单项选择题

1. 若让元素1，2，3，4，5依次进栈，则出栈次序不可能出现在（　　）种情况。

A. 5，4，3，2，1　　B. 2，1，5，4，3　　C. 4，3，1，2，5　　D. 2，3，5，4，1

2．若已知一个栈的入栈序列是 1，2，3，…，n，其输出序列为 p_1，p_2，p_3，…，p_n，若 $p_1=n$，则 p_i 为（　　）。

A．i　　B．$n-i$　　C．$n-i+1$　　D．不确定

3．数组 Q[n]用来表示一个循环队列，f 为当前队列头元素的前一位置，r 为队尾元素的位置，假定队列中元素的个数小于 n，计算队列中元素个数的公式为（　　）。

A．$r-f$　　B．$(n+f-r)\%n$　　C．$n+r-f$　　D．$(n+r-f)\%n$

4．链式栈结点为（data，link），top 指向栈顶，若想删除栈顶结点，并将删除结点的值保存到 x 中，则应执行操作（　　）。

A．x=top->data;top=top->link;　　B．top=top->link;x=top->link;

C．x=top;top=top->link;　　D．x=top->link;

5．设有一个递归算法如下：

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

则计算 fact(n)需要调用该函数的次数为（　　）。

A．$n+1$　　B．$n-1$　　C．n　　D．$n+2$

6．栈在（　　）中有所应用。

A．递归调用　　B．函数调用　　C．表达式求值　　D．前三个选项都有

7．【2009 年第 1 题】为解决计算机主机与打印机间速度不匹配问题，通常设一个打印数据缓冲区。主机将要输出的数据依次写入该缓冲区，而打印机则依次从该缓冲区中取出数据。该缓冲区的逻辑结构应该是（　　）。

A．队列　　B．栈　　C．线性表　　D．有序表

8．设栈 S 和队列 Q 的初始状态为空，元素 $e1$，$e2$，$e3$，$e4$，$e5$ 和 $e6$ 依次进入栈 S，一个元素出栈后即进入 Q，若 6 个元素出队的序列是 $e2$，$e4$，$e3$，$e6$，$e5$ 和 $e1$，则栈 S 的容量至少应该是（　　）。

A．2　　B．3　　C．4　　D．6

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

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

10．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

11．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

12．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)%(m-1)

C．rear=(rear+1)%m　　D．rear=(rear+1)%(m+1)

13．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．(rear+1)%n==front　　B．rear==front　　C．rear+1==front　　D．(rear-1)%n==front

14. 栈和队列的共同点是（C）。

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

15. 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

16.【2009 年第 2 题】设栈 S 和队列 Q 的初始状态均为空，元素 a，b，c，d，e，f，g 依次进入栈 S。若每个元素出栈后立即进入队列 Q，且 7 个元素出队的顺序是 b，d，c，f，e，a，g，则栈 S 的容量至少是（　　）。

A. 1　　B. 2　　C. 3　　D. 4

17.【2010 年第 1 题】若元素 a，b，c，d，e，f 依次进栈，允许进栈、退栈操作交替进行，但不允许连续三次进行退栈工作，则不可能得到的出栈序列是（　　）。

A. d，c，e，b，f，a　　B. c，b，d，a，e，f

C. b，c，a，e，f，d　　D. a，f，e，d，c，b

18.【2010 年第 2 题】某队列允许在其两端进行入队操作，但仅允许在一端进行出队操作，若元素 a，b，c，d，e 依次入此队列后再进行出队操作，则不可能得到的出队序列是（　　）。

A. b，a，c，d，e　　B. d，b，a，c，e

C. d，b，c，a，e　　D. e，c，b，a，d

19.【2011 年第 2 题】元素 a，b，c，d，e 依次进入初始为空的栈中，若元素进栈后可停留、可出栈，直到所有元素都出栈，则在所有可能的出栈序列中，以元素 d 开头的序列个数是（　　）。

A. 3　　B. 4　　C. 5　　D. 6

20.【2011 年第 3 题】已知循环队列存储在一维数组 $A[0...n-1]$ 中，且队列非空时 front 和 rear 分别指向队头元素和队尾元素。若初始时队列为空，且要求第 1 个进入队列的元素存储在 $A[0]$ 处，则初始时 front 和 rear 的值分别是（　　）。

A. 0，0　　B. 0，$n-1$　　C. $n-1$，0　　D. $n-1$，$n-1$

21.【2012 年第 1 题】求整数 $n(n\geq 0)$ 阶乘的算法如下，其时间复杂度是（　　）。

```
fact(int n)
{
  if(n<=1)  return;
  else  return(n*fact(n-1));
}
```

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

22.【2012 年第 2 题】操作符包括 '+'，'-'，'*'，'/' 和 '('，')'。将中缀表达式 a+b-a*((c+d)/e-f)+g 转换为等价的后缀表达式 ab+acd+e/f-*-g+时，用栈来存放暂时还不能确定运算次序的操作符，若栈初始时为空，则转换过程中同时保存在栈中的操作符的最大个数是（　　）。

A. 5　　B. 7　　C. 8　　D. 11

23.【2013 年第 2 题】一个栈的入栈序列为 1，2，3，…，n，其出栈序列是 p_1，p_2，p_3，…，p_n。若 $p_2=3$，则 p_3 可能取值的个数是（　　）。

A. $n-3$　　B. $n-2$　　C. $n-1$　　D. 无法确定

24.【2014 年第 2 题】假设栈初始为空，将中缀表达式 a/b+(c*d-e*f)/g 转换为等价的后缀表达式的过程中，当扫描到 f 时，栈中的元素依次是（　　）。

A. +(*-　　B. +(-*　　C. /+(*-*　　D. /+-*

25.【2014 年第 3 题】循环队列放在一维数组 $A[0...M-1]$中，end1 指向队头元素，end2 指向队尾元素的后一个位置。假设队列两端均可进行入队和出队操作，队列中最多能容纳 $M-1$ 个元素。初始时为空。下列判断队空和队满的条件中，正确的是（　　）。

A．队空：end1 == end2 ；队满：end1 ==(end2+1)mod M

B．队空：end1 == end2；队满：end2 == (end1+1)mod (M-1)

C．队空：end2 == (end1+1)mod M；队满：end1 == (end2+1)mod M

D．队空：end1 == (end2+1)mod M；队满：end2 == (end1+1)mod (M-1)

26.【2015 年第 1 题】已知程序如下：

```
int S(int n)
{ return (n<=0)?0:s(n-1)+n;}
void main( )
{ cout<< s(1); }
```

程序运行时使用栈来保存调用过程的信息，自栈底到栈顶保存的信息一次对应的是（　　）。

A．main( )->$S(1)$->$S(0)$　　B．$S(0)$->$S(1)$->main( )

C．main( )->$S(0)$->$S(1)$　　D．$S(1)$->$S(0)$->main( )

27.【2016 年第 3 题】设有如图 3.1 所示的火车车轨，入口到出口之间有 n 条轨道，列车的行进方向均从左至右，列车可驶入任意一条轨道，现有编号为 1~9 的 9 列列车，驶入的次序依次是 8，4，2，5，3，9，1，6，7。若期望驶出的次序依次为 1~9，则 n 至少是（　　）。

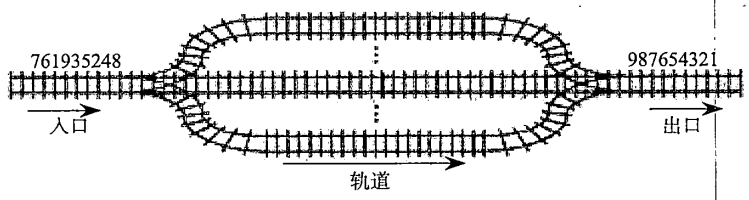


图 3.1　火车车轨

A．2　　B．3　　C．4　　D．5

28．若一个栈的输入序列为 1，2，3，…，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

29．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，再插入两个元素后，*rear* 和 *front* 的值分别为（　　）。

A．1 和 5　　B．2 和 4　　C．4 和 2　　D．5 和 1

30．设有一个递归算法如下：

```
int X(int n)
{ if(n<=3) return 1;
  else return X(n-2)+X(n-4)+1;
}
```

则计算 $X(X(8))$时需要计算 X 函数（　　）次。

A．8　　B．9　　C．16　　D．18

二、应用题

1．有 5 个元素，其入栈次序为：A，B，C，D，E，在各种可能的出栈次序中，以元素 C、D 最先出栈（即 C 第一个且 D 第二个出栈）的次序有哪几个？

2．对下面的递归函数，写出调用 *f*(3)的运行结果。

```
void f(int w)
{
   if(w>0)
   {
      f(w-1);
      cout<<w;
      f(w-1);
   }
}
```

3．有递归函数如下：

```
int s(int n)
{
   if(n==0) sum=0;
   else
   {cin>>x;  sum=s(n-1)+x;}
   return sum;
}
```

设初值 n=4，读入 x=4，9，6，2。问：

（1）若 x 为局部变量时，该函数递归结束后返回调用程序时 sum 的值为多少？并画出在递归过程中栈状态的变化过程；

（2）若 x 为全局变量时，该函数递归结束后返回调用程序时 sum 的值为多少？

4．对中缀算法表达式，可以借助运算符栈 OPTR 和运算数栈 OPND 进行求值，按照四则运算加、减、乘、除优先关系的惯例，以表格形式完成表达式 3*(6–5)的求值过程，给出 OPTR 栈和 OPND 栈的具体变化过程。

5．用栈实现将中缀表达式 8–(3+5)*(5–6/2)转换成后缀表达式，画出栈的变化过程图。

三、算法设计题

1．将编号为 0 和 1 的两个栈存放于一个数组空间 *V*[*m*]中，栈底分别处于数组的两端。当第 0 号栈的栈顶指针 top[0]等于–1 时该栈为空；当第 1 号栈的栈顶指针 top[1]等于 m 时，该栈为空。两个栈均从两端向中间增长（见图 3.2）。试编写双栈初始化，判断栈空、栈满、进栈和出栈等算法的函数。双栈数据结构的定义如下：

```
typedef struct
{
   int top[2], bot[2];          //栈顶和栈底指针
   SElemType *V;                //栈数组
   int m;                       //栈最大可容纳元素个数
}DblStack;
```

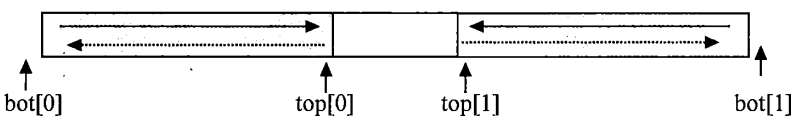


图 3.2　双栈结构的表示

2．回文是指正读反读均相同的字符序列，如“abba”和“abdba”均是回文，但“good”不是回文。试写一个算法判定给定的字符序列是否为回文。（提示：将一半字符入栈。）

3．设从键盘输入一整数的序列：a_1，a_2，a_3，…，a_n，试编写算法实现：用栈结构存储输入

的整数，当 $a_i \neq -1$ 时，将 a_i 进栈；当 $a_i=-1$ 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。

4. 从键盘上输入一个后缀表达式，试编写算法计算表达式的值。规定：后缀表达式的长度不超过一行，以“$”作为输入结束，操作数之间用空格分隔，操作符只可能有+、−、*、/四种运算。例如：234 34 +2×$。

5. 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. IOIIOIOO　　B. IOOIOIIO　　C. IIIOIOIO　　D. IIIOOIOO

（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）。

6. 假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾元素结点（注意：不设头指针），试编写相应的置空队列、判断队列是否为空、入队和出队等算法。

7. 假设以数组 $Q[m]$存放循环队列中的元素，同时设置一个标志 tag，以 tag == 0 和 tag == 1 来区别在队头指针（front）和队尾指针（rear）相等时，队列状态为“空”还是“满”。试编写与此结构相应的插入（EnQueue）和删除（DeQueue）算法。

8. 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。

9. 已知 Ackermann 函数定义如下：

$$Ack(m,n)=\begin{cases} n+1 & \text{当} m=0\text{时} \\ Ack(m-1,1) & \text{当} m\neq 0, n=0\text{时} \\ Ack(m-1,Ack(m,n-1)) & \text{当} m\neq 0, n\neq 0\text{时} \end{cases}$$

（1）写出计算 $Ack(m, n)$的递归算法，并根据此算法给出 $Ack(2, 1)$的计算过程。

（2）写出计算 $Ack(m, n)$的非递归算法。

10. 已知 f 为单链表的表头指针，链表中存储的都是整型数据，试写出实现下列运算的递归算法：

（1）求链表中的最大整数；

（2）求链表的结点个数；

（3）求所有整数的平均值。

3.2 答案及解析

一、单项选择题

1. C	2. C	3. D	4. A	5. A	6. D	7. A	8. B	9. C	10. D
11. D	12. D	13. B	14. C	15. B	16. C	17. D	18. C	19. B	20. B
21. B	22. A	23. C	24. B	25. A	26. A	27. C	28. D	29. B	30. D

1.【答案】C

【考点】栈的基本操作

【解析】

栈是后进先出的线性表，不难发现 C 选项中元素 1 比元素 2 先出栈，违背了栈的后进先出原则，所以答案选择 C。

值得注意的是，n 个数依次入栈，可以通过一个公式计算得到出栈序列的种数。

假设用 $f(n)$表示 n 个数的出栈序列的种数，假设第一个出栈的序数是 k，则 k 将 $1\sim n$ 的序列分成两个序列，其中一个是 $1\sim k-1$，序列个数为 $k-1$，另一个是 $k+1\sim n$，序列个数是 $n-k$。此时，若将 k 视为确定的一个序数，根据乘法原理，则 $f(n)$等于序列个数为 $k-1$ 的出栈序列种数乘以序列个数为 $n-k$ 的出栈序列的种数，即

$$f(n)=f(k-1)*f(n-k)$$

由于 k 可以从 1 取到 n，所以再根据加法原理，将 k 取不同值的序列的种数相加，得到的总的序列的种数为：

$$f(n)=f(0)f(n-1)+f(1)f(n-2)+\cdots+f(n-1)f(0)$$

而此式恰为一个著名数列——卡特兰数的递推式。

卡特兰数（Catalan number）是组合数学中一个常出现在各种计数问题中的数列，其前几项为：1，1，2，5，14，42，132…(n=0,1,2,…)。

设卡特兰数的第 n 项为 $h(n)$，令 $h(0)=1$，$h(1)=1$，则卡特兰数满足递推式：

$$h(n)=h(0)*h(n-1)+h(1)*h(n-2)+\cdots+h(n-1)*h(0)(n>=2)$$

递推关系的解为：

$$h(n)=C(2n,n)/(n+1)\quad (n=0,1,2,\cdots)$$

因此，当计算 n 个数的出栈序列的种数时可以利用公式 $C(2n,n)/(n+1)$来进行计算。例如，5 个数的出栈序列的种数为 42。

2.【答案】C

【考点】栈的基本操作

【解析】

一个栈的入栈序列是 1，2，3，…，n，而输出序列的第一个元素为 n，说明 1，2，3，…，n 一次性全部进栈，再进行输出，所以 $p_1=n$，$p_2=n-1$，…，$p_i=n-i+1$，所以答案选择 C。

3.【答案】D

【考点】循环队列的表示和实现

【解析】

对于非循环队列，尾指针和头指针的差值便是队列的长度，而对于循环队列，差值可能为负数，所以需要将差值加上 n，然后与 n 求余，即（$n+r-f$)%n，所以答案选择 D。

4.【答案】A

【考点】链栈的出栈操作

【解析】

由于 top 指向栈顶，因此在出栈时首先将栈顶结点的数据域保存下来（x=top->data），然后修改栈顶指针，将栈顶指针指向栈顶下一结点（top=top->link），即删除栈顶结点，所以答案选择 A。

5.【答案】A

【考点】递归执行过程

【解析】

递归调用一次 n 减 1，从 n 开始，直到 n 等于 0，易知总计调用 fact(n)函数的次数为 n+1，故答案选择选 A。

6.【答案】D

【考点】栈的应用

【解析】

编译器是借助栈完成递归调用和函数调用的，按照“后调用、先返回”的原则，函数之间的信息传递和控制转移必须通过“栈”来实现，即系统将整个程序运行时所需的数据空间安排在一个栈中，每当调用一个函数时，就为它在栈顶分配一个存储区；每当从一个函数退出时，就释放它的存储区，即当前正运行的函数的数据区必在栈顶。

在表达式计算中先出现的运算符不一定先运算，具体运算顺序是需要通过运算符优先关系的比较来确定合适的运算时机，而运算时机的确定是可以借助栈来完成的。将扫描到的不能进行运算的运算数和运算符先分别压入运算数栈和运算符栈中，在条件满足时再分别从栈中弹出进行运算。

因此，答案选择 D。

7.【答案】A

【考点】队列的应用

【解析】

打印机取出数据的顺序与数据被写入缓冲区的顺序相同，因此解决缓冲区问题应利用一种先进先出的线性表，而队列正是一种先进先出的线性表，所以答案选择 A。

8.【答案】B

【考点】栈与队列的基本操作

【解析】

由于一个元素出栈 S 后即进入 Q，因此由出队的序列 $e2$，$e4$，$e3$，$e6$，$e5$ 和 $e1$ 可知，出栈序列同为 $e2$，$e4$，$e3$，$e6$，$e5$ 和 $e1$。然后结合入栈顺序 $e1$，$e2$，$e3$，$e4$，$e5$ 和 $e6$ 可进行如下判断：$e2$ 出栈前，栈里的元素为 $e2$ 和 $e1$，共 2 个元素；$e4$ 出栈前，栈里的元素为 $e4$，$e3$，$e1$，共 3 个元素；$e3$ 出栈前，栈里的元素为 $e3$ 和 $e1$，共 2 个元素；$e6$ 出栈前，栈里的元素为 $e6$，$e5$，$e1$，共 3 个元素；$e5$ 出栈前，栈里的元素为 $e5$ 和 $e1$，共 2 个元素；综上所述，栈 S 的容量至少应该是 3，所以答案选择 B。

9.【答案】C

【考点】顺序栈的进栈操作

【解析】

初始栈顶指针 top 为 $n+1$，说明元素从数组向量的高端地址进栈，因为初始栈顶指针 top 设为 $n+1$，所以元素 x 进栈时 top 指针先下移变为 n（top--;），之后将元素 x 存储在 $V[n]$中。所以答案选择 C。

10.【答案】D

【考点】栈的应用

【解析】

检验括号是否匹配的方法可用“期待的急迫程度”这个概念来描述。

每读入一个括号，若是左括号，则作为一个新的更急迫的期待压入栈中，自然使原有的在栈中的所有未消解的期待的急迫性都降了一级；若是右括号，则或者使置于栈顶的最急迫的期待得以消解，或者是不合法的情况。可见，这个处理过程恰与栈的后进先出的特点相吻合。所以答案选择 D。

11.【答案】D

【考点】链队的出队操作

【解析】

一般情况下，队列在出队时只需修改头指针。但是，当删除的是队列中最后一个元素时，队列变空，队尾指针也丢失了，因此需对队尾指针重新赋值，使其指向队头结点。所以答案选择 D。

12.【答案】D

【考点】循环队列的入队操作

【解析】

在循环队列中，头、尾指针“依环状增 1”的操作可用“模”运算来实现。通过取模，循环队列的头指针和尾指针就可以在顺序表空间内以头尾衔接的方式“循环”移动。数组 $A[0, \cdots, m]$中共含有 $m+1$ 个元素，故在进队时，队尾指针增 1 后与 $m+1$ 进行取模运算，所以答案选择 D。

13.【答案】B

【考点】循环队列的表示和实现

【解析】

循环队列为了区别队满还是队空，通常采取少用一个元素空间的存储方案，即队列空间大小为 n 时，有 $n-1$ 个元素就认为是队满。这样判断队空的条件不变，即当头、尾指针的值相同时，则认为队空；而当尾指针在循环意义上加 1 后是等于头指针，则认为队满。因此，在循环队列中队空和队满的条件是：

队空的条件：front == rear。

队满的条件：(rear + 1)%n== front。

14.【答案】C

【考点】栈和队列的特点

【解析】

栈和队列是一种操作受限的特殊的线性表，其特殊性体现在：栈只允许在栈顶处进行插入和删除元素，队列只允许在队尾插入元素和在队头删除元素。所以答案选择 C。

15.【答案】B

【考点】递归算法的组成

【解析】

一个递归算法必须包括终止条件和递归部分。终止条件是递归算法的出口。因此本题答案选择 B。

16.【答案】C

【考点】栈与队列的基本操作

【解析】

元素 a，b，c，d，e，f，g 依次进入栈 S，元素出栈后立即进入队列 Q。队列是一种先进先出的线性表，因此 7 个元素出队顺序与入队序列相同，即与 7 个元素出栈顺序相同。要得到 b，d，c，f，e，a，g 这样的出栈顺序，对栈 S 可以进行如表 3.1 的操作。

表 3.1　　序列进栈和出栈详细过程

顺　序	操　作	栈　内	栈　外	顺　序	操　作	栈　内	栈　外
1	a 入栈	a		8	e 入栈	ae	bdc
2	b 入栈	ab		9	f 入栈	aef	bdc
3	b 出栈	a	b	10	f 出栈	ae	$bdcf$
4	c 入栈	ac	b	11	e 出栈	a	$bdcfe$
5	d 入栈	acd	b	12	a 出栈		$bdcfea$
6	d 出栈	ac	bd	13	g 入栈	g	$bdcfea$
7	c 出栈	a	bdc	14	g 出栈		$bdcfeag$

由上表可知栈内的最大深度为 3，因此栈 S 的容量至少是 3，所以答案选择 C。

17.【答案】D

【考点】栈的基本操作

【解析】

方法一：

（1）要得到 A 中的序列 d，c，e，b，f，a，可以进行以下操作：

Push(S,a)，Push(S,b)，Push(S,c)，Push(S,d)，Pop(S,d)，Pop（S,c)，

Push(S,e)，Pop（S,e)，Pop（S,b)，Push(S,f)，Pop（S,f)，Pop（S,a)；

（2）要得到 B 中的序列 c，b，d，a，e，f，可以进行以下操作：

Push(S,a)，Push(S,b)，Push(S,c)，Pop（S,c)，Pop（S,b)，Push(S,d)，

Pop(S,d)，Pop(S,a)，Push(S,e)，Pop(S,e)，Push(S,f)，Pop(S,f);

（3）要得到 C 中的序列 b，c，a，e，f，d，可以进行以下操作：

Push(S,a)，Push(S,b)，Pop(S,b)，Push(S,c)，Pop(S,c)，Pop(S,a)，

Push(S,d)，Push(S,e)，Pop(S,e)，Push(S,f)，Pop(S,f)，Pop(S,d);

（4）要得到 D 中的序列 a，f，e，d，c，b，可以进行以下操作：

Push(S,a)，Pop(S,a)，Push(S,b)，Push(S,c)，Push(S,d)，Push(S,e)，

Push(S,f)，Pop(S,f)，Pop(S,e)，Pop(S,d)，Pop(S,c)，Pop(S,b)。

根据题目要求不允许连续三次进行出栈操作，所以选项 D 是不可能得到的序列。

方法二：

由于题目要求不允许连续三次进行出栈操作，因此选项所给序列中出现长度大于等于 3 的连续逆序子序列，即为不符合要求的出栈序列，所以答案选择 D。

18.【答案】C

【考点】双端队列的基本操作

【解析】

方法一（L 代表左入，R 表右入）：

（1）要得到 A 中的序列 *b*，*a*，*c*，*d*，*e*，可以进行操作：*aL*(或 *aR*)，*bL*，*cR*，*dR*，*eR*。

（2）要得到 B 中的序列 *d*，*b*，*a*，*c*，*e*，可以进行以下操作得到：*aL*(或 *aR*)，*bL*，*cR*，*dL*，*eR*。

（3）要得到 C 中的序列 *d*，*b*，*c*，*a*，*e*，可以进行以下操作 *aL*(或 *aR*)，*bL*，因为 *d* 未出，此时只能进队，C 不可能出现在 *b* 和 *a* 之间。

（4）要得到 D 中的序列 *e*，*c*，*b*，*a*，*d*，可以进行以下操作得到：*aL*(或 *aR*)，*bL*，*cR*，*dR*，*eL*。

方法二：

队列是一种先进先出的线性表，无论先从左边入队还是先从右边入队，*a* 和 *b* 都应该相邻，这是出队序列合理的必要条件。只有选项 C 所给序列中 *a* 与 *b* 不相邻，所以答案选择 C。

19.【答案】B

【考点】栈的基本操作

【解析】

方法一：

出栈顺序必为 *d_c_b_a_*，*e* 的顺序不定，在任意一个“_”上都有可能。

方法二：

d 首先出栈，则 *abc* 留在栈中，若 *e* 进栈后直接出栈，可以得到 *decba*；若 *c* 先出栈，*e* 再进栈后出栈，出栈序列为 *dceba*；若 *cb* 先出栈，*e* 再进栈后出栈，得到序列 *dcbea*；若 *cba* 先出栈，*e* 再进栈后出栈，得到序列 *dcbae*。即以元素 *d* 开头的序列个数是 4，所以答案选择 B。

20.【答案】B

【考点】循环队列的表示和实现

【解析】

入队需要执行(rear+1)%*n* 操作，如果第 1 个进入队列的元素存储在 *A*[0]处，此时头指针 front 和尾指针 rear 都指向 0，则 rear 初始应该为 *n*−1，front 为 0，所以答案选择 B。

21.【答案】B

【考点】递归算法的时间复杂度

【解析】

设 fact(*n*)的运行时间函数是 *T*(*n*)。该函数中语句“return;”的运行时间是 O(1)，语句“return(n*fact(n-1));”的运行时间是 *T*(*n*−1)+O(1)，其中 O(1)为常量运行时间。

由此可得 fact(*n*)的时间复杂度为 O(*n*)。

22.【答案】A

【考点】中缀表达式转换后缀表达式

【解析】

为实现中缀表达式变后缀表达式，可以使用一个工作栈 OPTR 寄存运算符，初始化为空栈；使用一个字符串 Postfix 寄存转换得到的后缀表达式，初始化为空串。具体步骤如下：

① 初始化 OPTR 栈，将表达式起始符“#”压入 OPTR 栈。

② 扫描表达式，读入第一个字符 ch，如果表达式没有扫描完毕至“#”或 OPTR 的栈顶元素不为“#”时，则循环执行以下操作：

- 若 ch 不是运算符，则加入字符串 Postfix。
- 若 ch 是运算符，则根据 OPTR 的栈顶元素和 ch 的优先级比较结果，做不同的处理：
  - 若是小于，则 ch 压入 OPTR 栈，读入下一字符 ch；
  - 若是大于，则弹出 OPTR 栈顶的运算符，加入字符串 Postfix；
  - 若是等于，则 OPTR 的栈顶元素是“(”且 ch 是“)”，这时弹出 OPTR 栈顶的“(”，相

当于括号匹配成功，然后读入下一字符 ch。

③ 字符串 Postfix 中的元素即为后缀表达式，返回后缀表达式。

根据上述步骤，中缀表达式 a+b-a*((c+d)/e-f)+g 转换为等价的后缀表达式 ab+acd+e/f-*-g+的具体过程如表 3.2 所示。可见，转换过程中同时保存在栈中的操作符的最大个数是 5 个，所以答案选择 A。

表 3.2　　中缀表达式 a+b-a*((c+d)/e-f)+g 转换后缀表达式的具体过程

步　骤	OPTR 栈	字符串 Postfix	读入字符	主要操作
1	#		a+b-a*((c+d)/e-f)+g#	‘a’ 加入字符串 Postfix
2	#	a	+b-a*((c+d)/e-f)+g#	Push（OPTR，‘+’）
3	#+	a	b-a*((c+d)/e-f)+g#	‘b’ 加入字符串 Postfix
4	#+	ab	-a*((c+d)/e-f)+g#	Pop（OPTR，‘+’） ‘+’ 加入字符串 Postfix
5	#	ab+	-a*((c+d)/e-f)+g#	Push（OPTR，‘-’）
6	#-	ab+	a*((c+d)/e-f)+g#	‘a’ 加入字符串 Postfix
7	#-	ab+a	*((c+d)/e-f)+g#	Push（OPTR，‘*’）
8	#-*	ab+a	((c+d)/e-f)+g#	Push（OPTR，‘(’）
9	#-*(	ab+a	(c+d)/e-f)+g#	Push（OPTR，‘(’）
10	#-*((	ab+a	c+d)/e-f)+g#	‘c’ 加入字符串 Postfix
11	#-*((	ab+ac	+d)/e-f)+g#	Push（OPTR，‘+’）
12	#-*((+	ab+ac	d)/e-f)+g#	‘d’ 加入字符串 Postfix
13	#-*((+	ab+acd	)/e-f)+g#	Pop（OPTR，‘+’）； ‘+’ 加入字符串 Postfix
14	#-*((	ab+acd+	)/e-f)+g#	Pop(OPTR){消去一对括号}
15	#-*(	ab+acd+	/e-f)+g#	Push（OPTR，‘/’）
16	#-*(/	ab+acd+	e-f)+g#	Push（OPND，‘e’）
17	#-*(/	ab+acd+e	-f)+g#	‘-’ 加入字符串 Postfix
18	#-*(-	ab+acd+e/	f)+g#	‘f’ 加入字符串 Postfix
19	#-*(-	ab+acd+e/f	)+g#	Pop（OPTR，‘-’）； ‘-’ 加入字符串 Postfix
20	#-*(	ab+acd+e/f-	)+g#	Pop(OPTR){消去一对括号
21	#-*	ab+acd+e/f-	+g#	Pop（OPTR，‘*’）； ‘*’ 加入字符串 Postfix
22	#-	ab+acd+e/f-*	+g#	Pop（OPTR，‘-’）； ‘-’ 加入字符串 Postfix
23	#	ab+acd+e/f-*-	+g#	Push（OPTR，‘+’）
24	#+	ab+acd+e/f-*-	g#	‘g’ 加入字符串 Postfix
25	#+	ab+acd+e/f-*-g	#	Pop（OPTR，‘+’）； ‘+’ 加入字符串 Postfix
26	#	ab+acd+e/f-*-g+	#	return Postfix

23.【答案】C

【考点】栈的基本操作

【解析】

根据栈是一种先进后出的线性表可知，p_3 可以取 3 之后的 4，5，…，n（一直进栈直到该元素入栈后马上出栈）。当 p_1=1 时，1 进栈直接出栈，2 进栈，3 进栈，然后 3 出栈，接着 2 出栈，此时 p_3 可以取 2；当 p_1=2,1 进栈，2 进栈，2 出栈，3 进栈接着出栈，然后 1 出栈，此时 p_3 可以取 1。因此除了 3 本身以外，其他的值均可以取到，所以答案选择 C。

24.【答案】B

【考点】中缀表达式转换后缀表达式

【解析】

同 22 题的转换步骤，中缀表达式 a/b+(c*d-e*f)/g 转换后缀表达式的具体过程如表 3.3 所示。可见，当扫描到 f 时（第 14 步），栈中的元素依次是+(-*，所以答案选择 B。

表 3.3　　中缀表达式 a/b+(c*d-e*f)/g 转换后缀表达式的具体过程

步　骤	OPTR 栈	字符串 Postfix	读入字符	主要操作
1	#		a/b+(c*d-e*f)/g #	‘a’ 加入字符串 Postfix
2	#	a	/b+(c*d-e*f)/g #	Push（OPTR，‘/’）
3	#/	a	b+(c*d-e*f)/g #	‘b’ 加入字符串 Postfix
4	#/	ab	+(c*d-e*f)/g #	Pop（OPTR，‘/’）； ‘/’ 加入字符串 Postfix
5	#	ab/	+(c*d-e*f)/g #	Push（OPTR，‘+’）
6	#+	ab/	(c*d-e*f)/g #	Push（OPTR，‘(’）
7	#+(	ab/	c*d-e*f)/g #	‘c’ 加入字符串 Postfix
8	#+(	ab/c	*d-e*f)/g #	Push（OPTR，‘*’）
9	#+(*	ab/c	d-e*f)/g #	‘d’ 加入字符串 Postfix
10	#+(*	ab/cd	-e*f)/g #	Pop（OPTR，‘*’）； ‘*’ 加入字符串 Postfix
11	#+(	ab/cd*	-e*f)/g #	Push（OPTR，‘-’）
12	#+(-	ab/cd*	e*f)/g #	‘e’ 加入字符串 Postfix
13	#+(-	ab/cd*e	*f)/g #	Push（OPTR，‘*’）
14	#+(-*	ab/cd*e	f)/g #	‘f’ 加入字符串 Postfix
15	#+(-*	ab/cd*ef	)/g #	Pop（OPTR，‘*’） ‘*’ 加入字符串 Postfix
16	#+(-	ab/cd*ef*	)/g #	Pop（OPTR，‘-’） ‘-’ 加入字符串 Postfix
17	#+(	ab/cd*ef*-	)/g #	Pop(OPTR){消去一对括号
18	#+	ab/cd*ef*-	/g #	Push（OPTR，‘/’）
19	#+/	ab/cd*ef*-	g#	‘g’ 加入字符串 Postfix
20	#+/	ab/cd*ef*-g	#	Pop（OPTR，‘/’） ‘/’ 加入字符串 Postfix
21	#+	ab/cd*ef*-g/	#	Pop（OPTR，‘+’） ‘+’ 加入字符串 Postfix
22	#	ab/cd*ef*-g/+	#	return Postfix

25.【答案】A

【考点】循环队列的表示和实现

【解析】

当头、尾指针的值相同时，则认为队空；而当尾指针在循环意义上加 1 后是等于头指针，则认为队满。本题中，end1 指向队头元素，end2 指向队尾元素的后一个位置，因此，队空：end1 == end2；队满：end1 ==(end2+1)mod M，所以答案选择 A。

26.【答案】A

【考点】递归工作栈

【解析】

递归调用函数时，在系统栈里保存的函数信息需满足先进后出的特点，依次调用了 main( )、S(1)、S(0)，所以答案选择 A。

27.【答案】C

【考点】栈的基本操作

【解析】

此题会有一个典型的错误方向：认为一个轨道只能装一列火车，但实际不是。因为题目并未直接告知，所以需要推断。此题类似于多级任务调度队列。经分析得，n=4 即可满足。因此答案选择 C。

28.【答案】D

【考点】栈的基本操作

【解析】

n 个数据依次通过一个栈，并不能保证出栈数据的次序总是倒置，可以产生多种出栈序列，n 个数的出栈序列种数时可以利用卡特兰公式 $f(n)=C(2n,n)/(n+1)$ 来进行计算(详解见选择题第 1 题)。只有当所有数据全部进栈后再全部出栈才能使数据倒置。题目中输出序列的第一个元素是 i，则第 j 个输出元素是不确定的。

29.【答案】B

【考点】循环队列的表示和实现

【解析】

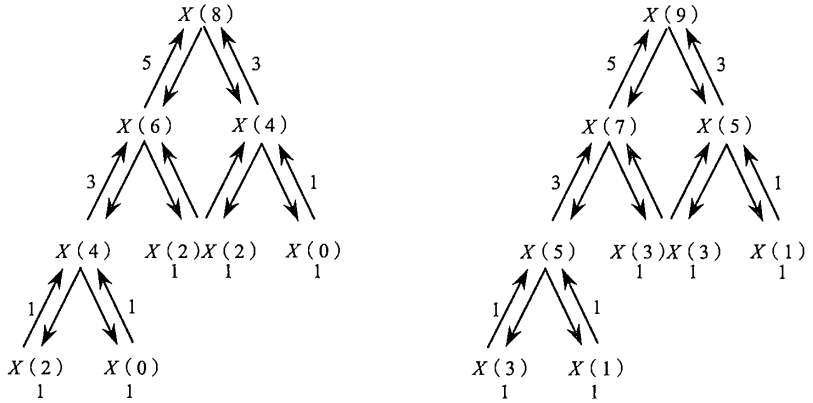
队头元素出队时，队头指针执行 Q.front=(Q.front+1)%M；队尾有新的元素要进队时，队尾打好针执行 Q.rear=(Q.rear+1)%M。本题中 M=6，当前 *rear* 和 *front* 的值分别为 0 和 3，当从队列中删除一个元素，队头指针 Q.front=(3+1)%6=4，再加入两个元素，则队尾指针先执行 Q.rear=(0+1)%6=1，再执行 Q.rear=(1+1)%6=2。所以答案选择 B。

30.【答案】D

【考点】递归算法的执行过程

【解析】

本题首先要得到求解 $X(8)$时 X 函数的计算次数，并求出 $X(8)$的返回值。借助图 3.3（a）所示的 $X(8)$的递归调用过程，可以直观地看出求解 $X(8)$时 X 函数的计算次数为 9 次，$X(8)$的返回值为 9。同理，借助图 3.3（b）所示的 $X(9)$的递归调用过程，可以看出求解 $X(9)$时 X 函数的计算次数也为 9 次，因此，计算 $X(X(8))$时需要计算 X 函数 18 次。



（a）$X(8)$的递归调用过程　　（b）$X(9)$的递归调用过程

图 3.3　计算 $X(X(8))$时的递归调用过程

二、应用题

1．解答

方法一：

出栈顺序必为 *CD_B_A_*，*E* 的顺序不定，在任意一个“_”上都有可能。

方法二：

C 首先出栈，则 *AB* 留在栈中，*D* 进栈直接出栈：若 *E* 进栈后直接出栈，可以得到 *CDEBA*；若 *B* 先出栈，*E* 再进栈后出栈，得到序列 *CDBEA*；若 *BA* 先出栈，*E* 再进栈后出栈，得到序列 *CDBAE*。

即以元素 *C*、*D* 最先出栈的次序有三个分别是：*CDEBA*、*CDBEA*、*CDBAE*。

2．解答

递归调用过程如图 3.4 所示，其中 p->n 表示输出 *n*，p->n 前面的数字代表具体的输出次序。因此，输出结果依次为：1213121。

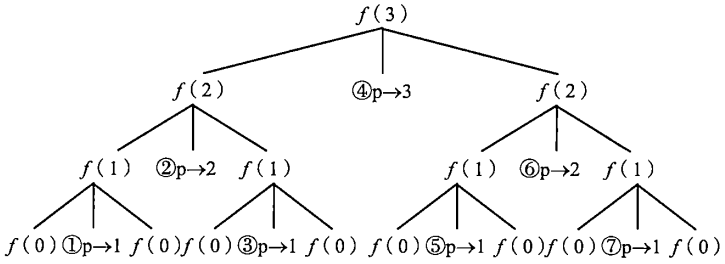


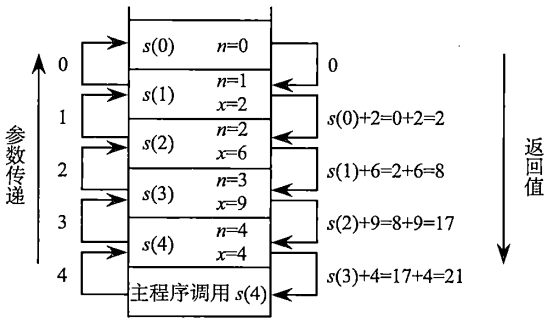
图 3.4　调用 $f(3)$时的递归调用过程

3．解答

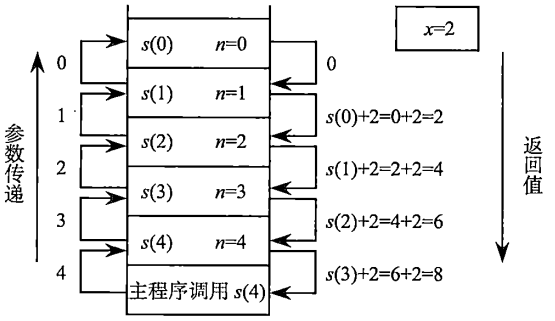
（1）当 *x* 为局部变量时，递归过程如图 3.5（a）所示。每次递归时，将实参 *n*、局部变量 *x*，以及上一层的返回地址作为一个新的工作记录压入栈顶。每退出一层递归，就从栈顶弹出一个工作记录。由于 *x* 为局部变量，因此每次递归返回进行计算时其值不同，递归结束后返回调用程序时 sum 的值为 21。

（2）当 *x* 为全局变量时，在整个程序执行期间，*x* 只占一个存储单元，这个单元是与递归工作栈所占用的空间无关的。虽然先后读入四个数 4、9、6、2，但只有最后一个数据 2 起作用。因

此该函数递归结束后逐层返回时，x 为 2，递归过程如图 3.5（b）所示。



（a）x 为局部变量时递归工作栈的变化过程



（b）x 为全局变量时递归工作栈的变化过程

图 3.5　计算 $X(X(8))$时的递归调用过程

4．解答

算术表达式 3*(6-5)的求值过程如表 3.4 所示。

表 3.4　　算术表达式 3*(6-5)的求值过程

步　骤	OPTR 栈	OPND 栈	读入字符	主要操作
1	#		3*(6-5)#	Push(OPND, '3')
2	#	3	*(6-5)#	Push(OPTR, '*')
3	#*	3	(6-5)#	Push(OPTR, '(')
4	#*(	3	6-5)#	Push(OPND, '6')
5	#*(	3 6	-5)#	Push(OPTR, '-')
6	#*(-	3 6	5)#	Push(OPND, '5')
7	#*(-	3 6 5	)#	Push(OPND, Operate('6', '-', '5'))
8	#*(	3 1	)#	Pop(OPTR){消去一对括号}
9	#*	3 1	#	Push(OPND, Operate('3', '*', '1'))
10	#	3	#	return(GetTop(OPND))

5．解答

中缀表达式转换成后缀表达式的具体步骤见 22 题，8-(3+5)*(5-6/2)转换成后缀表达式的具体过程如表 3.5 所示。

表 3.5　　中缀表达式 8-(3+5)*(5-6/2)转换成后缀表达式的具体过程

步　骤	OPTR 栈	字符串 Postfix	读入字符	主要操作
1	#		8-(3+5)*(5-6/2)#	‘8’ 加入字符串 Postfix
2	#	8	-(3+5)*(5-6/2)#	Push（OPTR，‘-’）
3	#-	8	(3+5)*(5-6/2)#	Push（OPTR，‘(’）
4	#-(	8	3+5)*(5-6/2)#	‘3’ 加入字符串 Postfix
5	#-(	83	+5)*(5-6/2)#	Push（OPTR，‘+’）
6	#-(+	83	5)*(5-6/2)#	‘5’ 加入字符串 Postfix
7	#-(+	835	)*(5-6/2)#	Pop（OPTR，‘+’） ‘+’ 加入字符串 Postfix
8	#-(	835+	)*(5-6/2)#	Pop(OPTR){消去一对括号
9	#-	835+	*(5-6/2)#	Push（OPTR，‘*’）
10	#-*	835+	(5-6/2)#	Push（OPTR，‘(’）
11	#-*(	835+	5-6/2)#	‘5’ 加入字符串 Postfix
12	#-*(	835+5	-6/2)#	Push（OPTR，‘-’）
13	#-*(-	835+5	6/2)#	‘6’ 加入字符串 Postfix
14	#-*(-	835+56	/2)#	Push（OPTR，‘/’）
15	#-*(-/	835+56	2)#	‘2’ 加入字符串 Postfix
16	#-*(-/	835+562	)#	Pop（OPTR，‘/’） ‘/’ 加入字符串 Postfix
17	#-*(-	835+562/	)#	Pop（OPTR，‘-’） ‘-’ 加入字符串 Postfix
18	#-*(	835+562/-	)#	Pop(OPTR){消去一对括号
19	#-*	835+562/-	#	Pop（OPTR，‘*’） ‘*’ 加入字符串 Postfix
20	#-	835+562/-*	#	Pop（OPTR，‘-’） ‘-’ 加入字符串 Postfix
21	#	835+562/-*-	#	return Postfix

三、算法设计题

1．解答

【算法思想】

两栈共享向量空间，将两栈栈底设在向量两端，初始时，左栈顶指针为-1，右栈顶为 m。两栈顶指针相邻时为栈满。两栈顶相向增长，栈顶指针指向栈顶元素。左栈是通常意义下的栈，在进栈操作时，其栈顶指针右移（加 1），出栈时，栈顶指针左移（减 1）。而右栈进栈操作时，其栈顶指针左移（减 1），出栈时，栈顶指针右移（加 1）。

【算法描述】

```
Status InitDblStack(DblStack &S,int m)
{//初始化一个大小为m的双向栈s
   S.V=new SElemType[m];                    //动态分配一个最大容量为m的数组空间
   S.bot[0]=-1;                             //左栈栈底指针
   S.bot[1]=m;                              //右栈栈底指针
   S.top[0]=-1;                             //左栈栈顶指针
   S.top[1]=m;                              //右栈栈顶指针
   return OK;
}
Status DblPush(DblStack S,int i,int x)
{//向指定的i号栈中插入元素x
   if(S.top[1]-S.top[0]==1)
      return ERROR;                         //栈满
   if(i==0)
      S.V[++S.top[0]]=x;                    //左栈：栈顶指针先加1,然后按此地址进栈
   else
      S.V[--S.top[1]]=x;                    //右栈：栈顶指针先减1,然后按此地址进栈
}
Status DblPop(DblStack &S,int i,SElemType &x)
{//删除指定的i号栈的栈顶元素，用x返回其值
   if(S.top[i]==S.bot[i])                   //栈空
      return ERROR;
   if(i==0)
      x=S.V[S.top[0]--];                    //左栈：栈顶指针减1
   else
      x=S.V[S.top[0]++];                    //右栈：栈顶指针加1
   return OK;
}
int IsEmpty(DblStack S,int i)
{//判断指定的i号栈是否为空，空返回1,否则返回0
   return S.top[i]==S.bot[i];
}
int IsFull(DblStack S)
{//判断栈是否满,满返回1,否则返回0
   if(S.top[0]+1==S.top[1])
      return 1;
   else
      return 0;
}
```

2. 解答

【算法思想】

将字符串前一半入栈，然后，栈中元素和字符串后一半进行比较。即将第一个出栈元素和后一半串中第一个字符比较，若相等，则再出栈一个元素与后一个字符比较，以此类推，直至栈空，在这种情况下可判定该字符序列是回文。如果出栈元素与串中的字符进行比较时出现不等的情况，则可判定该字符序列不是回文。

【算法描述】

```
int IsPalindrome(char *t)
```

```
{//判断 t 字符向量是否为回文，若是，返回 1，否则返回 0
  InitStack(S);
  len=strlen(t);                          //求向量长度
  for(i=0;i<len/2;i++)                    //将一半字符入栈
    Push(S,t[i]);
  If(len%2!=0)
    i++;                                  //长度为奇数，跳过中间字段
  while(!EmptyStack(S))                   //每弹出一个字符与相应字符比较
  {
    temp=Pop(S);
    if(temp!=t[i])
      return 0;                           //不等则返回 0
    else
      i++;
  }
  return 1;                               //比较完毕均相等则返回 1
}
```

3. 解答

【算法思想】

根据 a_i 的值判断进行入栈或者出栈操作，入栈前首先判断是否栈满，出栈前先判断是否栈空，随后再进行相关操作。

【算法描述】

```
void InOutS(int s[])
{//s 是元素为整数的栈，根据读入的数据完成入栈和出栈操作
  top=0;                                  //top 为栈顶指针，top=0 时为栈空
  for(i=1;i<=n;i++)                       //n 个整数序列作处理
  {
    cin>>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)

      {
        cout<<"栈满"<<endl;
        exit(0);
      }
     else
      s[++top]=x;                         //x 入栈
    }
    else                                  //读入的整数等于-1 时退栈
    {
     if(top==0)
     {
       cout<<"栈空"<<endl;
       exit(0);
       }
       else
       cout<<"出栈元素是"<<s[top--]<<endl;
    }
```

```
    }
}
```

4. 解答

【算法思想】

后缀表达式中运算符出现的先后顺序已经隐含了运算符的优先级，求值时不需要再考虑运算符的优先级，只需从左到右扫描一遍后缀表达式即可。设立操作数栈 OPND 用来存放表达式中扫描到的数。具体求值步骤为：初始化 OPND 为空栈，从左到右扫描后缀表达式，若遇操作数，则进栈；若遇运算符，则从栈中退出两个元素。先退出的放到运算符的右边，后退出的放到运算符左边，与此运算符进行相应的运算，结果再压入 OPND 栈中。直到从表达式中读出结束符$，此时 OPND 栈中只有一个元素，即为后缀表达式的值。

在扫描表达式时遇到数字或小数点要完成拼数功能，可以将读入的数字或小数点依次保存在一个字符串数组 data 中，然后利用 C 语言提供的库函数 atof 即可将 data 中的字符串转换成一个浮点数。

【算法描述】

```
double Postfix( )
{//计算以'$'结尾的后缀表达式的值。
   InitStack(OPND);                          //操作数栈初始化
   num=0.0;                                  //数字初始化
   ch=getchar();                             //读入后缀式的第一个字符
   while(ch!='$')                            //表达式没有扫描完毕
   {
      i=0;
      while((ch>='0'&&ch<='9')||ch=='.')
      {//拼数，将读入的数字或小数点依次保存在字符数组 data 中
         data[i]=ch;
         i++;
         ch=getchar();
      }
      num=atof(data);                        //将字符串转换成浮点数
      Push(OPND,num);                        //操作数进栈
      switch(ch)
      {//遇'+'、'-'、'*'、'/'，弹出两个数据进行运算后再压栈
         case ' ':break;                     //遇到空格，继续读入表达式中下一个字符
         case'+':Pop(OPND,b);Pop(OPND,a);Push(OPND,a+b);break;
         case'-':Pop(OPND,b);Pop(OPND,a);Push(OPND,a-b);break;
         case'*':Pop(OPND,b);Pop(OPND,a);Push(OPND,a*b);break;
         case'/':Pop(OPND,b);Pop(OPND,a);Push(OPND,a/b);break;
       }
      ch=getchar();                          //读入表达式中下一个字符
   }
   return GetTop(OPND);                      //输出运算结果
}
```

5. 解答

（1）在入栈出栈的过程中，需要满足在入栈出栈序列的任一位置，入栈次数即“*I*”的个数，必须大于或等于出栈次数即“*O*”的个数，否则视作非法序列。经检验，*A* 和 *D* 是合法序列，*B* 和 *C* 是非法序列。

（2）【算法思想】

假定被判定的操作序列存入字符串数组中 A 中，从数组中的第一个元素开始对数组中的元素逐一进行判断，直到字符串末尾。如果当前字符为“I”，则入栈 j 次数增 1，如果为“O”，则出栈次数 k 增 1。在判断过程中，如果出现 k 大于 j，则序列非法，不必继续判断，返回 false；在到达字符串末尾后，如果 k 不等于 j，则序列非法，返回 false；否则序列合法，返回 true。

【算法描述】

```
bool Judge(char A[])
{//判断 A 中的输入输出序列是否是合法序列。如果是，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')                             //未到字符数组尾时判断序列是否合法
   {
      switch(A[i])
      {
         case'I':j++;break;                      //入栈次数增 1
         case'O':k++;
         if(k>j)
         {
           cout<<"序列非法"<<endl;
           return false;
         }
      }
      i++;                                       //不论 A[i]是'I'或'O'，指针 i 均后移
   }
   if(k!=j)
   {
      cout<<"序列非法"<<endl;
      return false;
   }
   else
   {
      cout<<"序列合法"<<endl;
      return true;
   }
}
```

6. 解答

【算法思想】

置空队列：使队尾指针指向头结点，当队列非空时将队中元素逐个出队，释放相应的结点空间。

判断队空：队列只有一个头结点，即当头结点的指针域指向自己时，队为空。

入队操作：将新的结点插入到链队的尾部，同时将尾指针指向这个结点。

出队操作：当队列非空时，删除队头元素。当出队元素为最后一个元素时，要注意修改队尾指针，将其指向头结点。

【算法描述】

```
//- - - - - 队列的链式存储结构- - - - -
typedef struct QNode
{
   QElemType data;
```

```
  struct QNode *next;
}QNode, *QueuePtr;
typedef struct
{
  QueuePtr rear;                          //只设一个队尾指针
}LinkQueue;
void InitQueue(LinkQueue &Q)
{//置空队列
  Q->rear=Q->rear->next;                  //将队尾指针指向头结点
  while(Q->rear!=Q->rear->next)           //当队列非空时，将队中元素逐个出队
  {
    s=Q->rear->next;                      //s 指向队头元素
    Q->rear->next=s->next;                //尾结点的指针域指向新的队头元素
    delete s;                             //释放结点的空间
  }
}
int EmptyQueue(LinkQueue Q)
{//判断队是否为空，空返回 1，否则返回 0
 //队列只有一个头结点，即当头结点的指针域指向自己时，队为空
  return Q->rear->next->next==Q->rear->next;
}
Status EnQueue(LinkQueue &Q,QElemType e)
{//入队，插入元素 e 为 Q 的新的队尾元素
  p=new QueueNode;                        //申请新结点
  p->data=e;                              //将新结点数据域置为 e
  p->next=Q->rear->next;                  //将新结点插入到队尾
  Q-rear->next=p;
  Q->rear=p;                              //将尾指针移至新结点
  return OK;
}
Status DeQueue(LinkQueue &Q,QElemType &e)
{//出队，删除 Q 的队头元素，用 e 返回其值
  if(Q->rear->next->next==Q->rear->next)
    return ERROR;                         //若队列空，则返回 ERROR
  p=Q->rear->next->next;                  //p 指向队头元素
  e=p->data;                              //e 保存队头元素的值
  if(p==Q->rear)                          //当队列中只有一个结点
  {
    Q->rear=Q->rear->next;                //修改尾指针，使其指向头结点
    Q->rear->next=p->next;
  }
  else
    Q->rear->next->next=p->next;          //摘下结点 p
  delete p;                               //释放原队头元素的空间
  return OK;
}
```

7. 解答

【算法思想】

在循环队列中，头、尾指针“依环状增 1”的操作可用“模”运算来实现，通过取模，头指

针和尾指针就可以在顺序表空间内以头尾衔接的方式“循环”移动。因此循环队列不能以头、尾指针的值是否相同来判别队列空间是“满”还是“空”。在这种情况下，为了区别队满还是队空，通常有两种处理方案。

（1）方案一：少用一个元素空间（常用的方法，为各大教材所采用），即队列空间大小为 m 时，有 $m-1$ 个元素就认为是队满。这样判断队空的条件不变，即当头、尾指针的值相同时，则认为队空；而当尾指针在循环意义上加 1 后是等于头指针，则认为队满。

（2）方案二：另设一个标志位以区别队列是“空”还是“满”，即本题所要求的方案。

在第二种方案下，入队时，尾指针加 1，同时更新 tag 值；出队时，队头指针加 1，同时更新 tag 值。

【算法描述】

```
typedef struct{
   ElemType *base;
   int front,rear,tag;
}SqQueue;
Status InitQueue(SqQueue &Q)
{//构造一个空队列Q
   Q.base=new QElemType[M]                    //为队列分配一个最大容量为MAXSIZE的数组空间
   if(!Q.base) exit(OVERFLOW);                //存储分配失败
   Q.front=Q.rear=0;                          //头指针和尾指针置为零，队列为空
   Q.tag=0;                                   //标志初始化为0，队列为空
   return OK;
}
Status EnQueue(SqQueue &Q,QElemType e)
{//插入元素e为Q的新的队尾元素
   if((Q->tag==1)&&(Q->rear==Q.front))        //队满
      return ERROR;
   Q.base[Q.rear]=e;                          //新元素插入队尾
   Q.rear=(Q.rear+1)%M;                       //队尾指针加1
   if(Q->tag==0) Q->tag=1;                    //标志改1，表示队列非空
   return OK;
}
Status DeQueue(SqQueue &Q,QElemType &e)
{//删除Q的队头元素，用e返回其值
   if((Q->tag==0)&&(Q->rear==Q.front))        //队空
      return ERROR;
   e=Q.base[Q.front];                         //保存队头元素
   Q.front=(Q.front+1)%M;                     //队头指针加1
   if(Q->tag==1) Q->tag=0;                    //标志改0，表示队列非满
   return OK;
}
```

8. 解答

【算法思想】

用长度为 M 的一维数组 base[0...M-1]实现循环队列。其中队头指针和队尾指针分别为 front 和 rear，约定 front 指向队头元素的前一位置，rear 指向队尾元素。当 front==rear 时为队空，(rear+1)%m==front 为队满。从队头插入元素时向下标小的方向发展，从队尾入队则向下标大的方向发展。

```
typedef struct
{
  QElemType *base;                    //存储空间的基地址
  int front;                          //头指针
  int rear;                           //尾指针
}SqQueue;
Status EnQueue(SqQueue &Q,QElemType e)
{//在Q的队头插入新元素e
  if(Q.rear==(Q.front-1+M)%M)         //队满
    return ERROR;
  Q.base[Q.front]=e;                  //新元素插入队头
  Q.front=(Q.front-1+M)%M;            //修改队头指针
  return OK;
}
Status DeQueue(SqQueue &Q,QElemType &e)
{//删除Q的队尾元素，用e返回其值
  if(Q.front==Q.rear)                 //队空
    return ERROR;
  e=Q.base[Q.rear];                   //保存队尾元素
  Q.rear=(Q.rear-1+M)%M;              //队尾指针减1
  return OK;
}
```

9. 解答

（1）【算法思想】

编写递归函数时，关键点在于找到递归的结束条件和递归体。递归结束的条件是 m=0，此时返回 n+1。递归体包括两部分：当 $m\neq0$ 且 n=0 时条件成立时，调用递归函数 $Ack(m-1,1)$；当 $m\neq0$ 且 $n\neq0$ 时，调用递归函数 $Ack(m-1)$，$Ack(m,n-1)$)。

【算法描述】

```
int Ack(int m,n)
{//Ack(m,n)的递归算法
  if(m==0)
    return(n+1);                      //递归结束
  else if(m!=0&&n==0)
    return(Ack(m-1,1));               //调用递归函数Ack(m-1,1)
  else
    return(Ack(m-1,Ack(m,n-1)));      //调用递归函数Ack(m-1, Ack(m,n-1))
}
```

当 m=2，n=1 时，调用 $Ack(m-1,Ack(m,n-1))$，然后根据条件逐层递归，具体计算过程如下：

```
Ack(2,1)=Ack(1,Ack(2,0))              //因m≠0,n≠0而得
        =Ack(1,Ack(1,1))              //因m≠0,n=0而得
        =Ack(1,Ack(0,Ack(1,0)))       //因m≠0,n≠0而得
        =Ack(1,Ack(0,Ack(0,1)))       //因m≠0,n=0而得
        =Ack(1,Ack(0,2))              //因m=0而得
        =Ack(1,3)                     //因m=0而得
```

```
        =Ack(0,Ack(1,2))                          //因 m≠0,n≠0 而得
        =Ack(0,Ack(0,Ack(1,1)))                   //因 m≠0,n≠0 而得
        =Ack(0,Ack(0,Ack(0,Ack(1,0))))            //因 m≠0,n≠0 而得
        =Ack(0,Ack(0,Ack(0,Ack(0,1))))            //因 m≠0,n=0 而得
        =Ack(0,Ack(0,Ack(0,2)))                   //因 m=0 而得
        =Ack(0,Ack(0,3))                          //因 m=0 而得
        =Ack(0,4)                                 //因 n=0 而得
        =5                                        //因 n=0 而得
```

（2）【算法思想】

利用非递归的算法计算 *Ack*(*m*, *n*)时，可以根据 Ackermann 函数的定义，利用迭代的方法循环进行计算。首先计算得到 *Ack*(*0*, *n*)的值，然后逐个循环计算其他的值。

【算法描述】

```
int Ackerman(int m,int n)
{//Ack(m,n)的非递归算法
  for(j=0;j<n;j++) akm[0][j]=j+1;          //得到 Ack(0,n)的值
  for(i=1;i<m;i++)
  {
    akm[i][0]=akm[i-1][1];
    for(j=1;j<n;j++)
      akm[i][j]=akm[i-1][akm[i][j-1]];
  }
    return(akm[m][n]);
}
```

10. 解答

（1）【算法思想】

递归结束条件：当链表只有一个结点时（尾结点），返回当前结点的值即为最大值。

递归体：递归求解除尾结点之外的其余结点的最大值，并与尾结点的值进行比较，返回较大者。

【算法描述】

```
int GetMax(LinkList p)
{//递归求解链表的最大整数
 if(!p->next)                              //p 指向表尾，返回其数值域
    return p->data;
  else
  {
    int max=GetMax(p->next);               //递归
    return p->data>=max?p->data:max;       //返回较大者
  }
}
```

（2）【算法思想】

递归结束条件：当链表只有一个结点时（尾结点），返回 1。

递归体：递归求解除尾结点之外的其余结点个数，记为 *num*，则整个链表元素个数为 *num* 加 1。

【算法描述】

```
int GetLength(LinkList p)
{//递归求解链表的结点个数
```

```
    if(!p->next)                                    //p 指向表尾，返回 1
       return 1;
     else                                           //每递归一次，结点个数加 1
     {
       return GetLength(p->next)+1;
     }
}
```

（3）【算法思想】

递归结束条件：当链表只有一个结点时（尾结点），返回当前结点的值即为平均值。

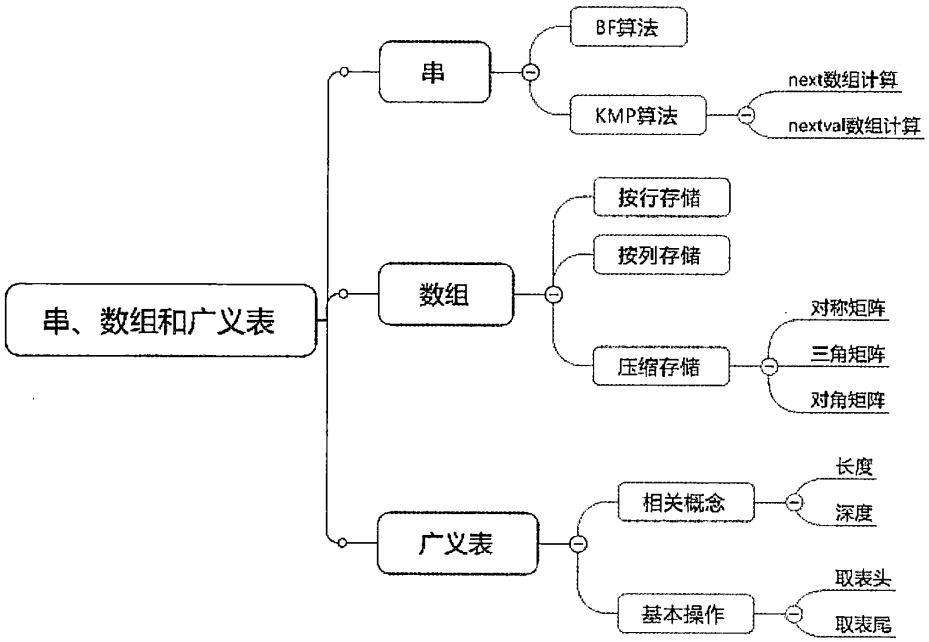
递归体：递归求解除尾结点之外的其余结点的平均值 *ave*，链表的平均值即为尾结点的值与 $ave\times(n-1)$的和，再除以 n。

【算法描述】

```
double GetAverage(LinkList p,int n)
{//递归求解链表中所有整数的平均值
  if(!p->next)                                      //p 指向表尾，返回其数值
     return p->data;
   else
   {
     double ave=GetAverage(p->next,n-1);
     // 递归求解除尾结点之外的其余 n-1 个结点的平均值
     return (ave*(n-1)+p->data)/n;                  //返回平均值
   }
}
```

第4章 串、数组和广义表

【知识导图】



【学习目标】

1. 串是内容受限的线性表，它限定了表中的元素为字符。串有两种基本存储结构：顺序存储和链式存储，但多采用顺序存储结构。串的常用算法是模式匹配算法，主要有 BF 算法和 KMP 算法。掌握 BF 算法和 KMP 算法的具体实现，明确 KMP 对 BF 的改进之处，掌握 KMP 算法中 next 数组和 nextval 数组的计算方法。

2. 多维数组可以看成线性表的推广，数组一般采用顺序存储结构，存储多维数组时，应先将其确定转换为一维结构，有按“行”转换和按“列”转换两种，掌握两种不同转换方法数组地址的计算方法（通常考查二维数组）。对于几种常见形式的特殊矩阵，比如对称矩阵、三角矩阵和对角矩阵，在存储时可进行压缩存储，掌握特殊矩阵的压缩存储方法。

3. 广义表是另外一种线性表的推广形式，线性表可以看成广义表的特例。掌握广义表的相关概念，主要包括长度和深度。掌握广义表的基本操作，主要包括取表头和取表尾的操作。

4.1　习题

一．选择题

1．串是一种特殊的线性表，其特殊性体现在（　　）。
A．可以顺序存储　　B．数据元素是一个字符
C．可以链式存储　　D．数据元素可以是多个字符

2．下面关于串的叙述中，（　　）是不正确的。
A．串是字符的有限序列
B．空串是由空格构成的串
C．模式匹配是串的一种重要运算
D．串既可以采用顺序存储，也可以采用链式存储

3．串“ababaaababaa”的 next 数为（　　）。
A．012345678999　　B．0121111212　　C．011234223456　　D．0123012322345

4．串“ababaabab”的 nextval 数为（　　）。
A．010104101　　B．002101　　C．010100011　　D．010101011

5．串的长度是指（　　）。
A．串中所含不同字母的个数　　B．串中所含字符的个数
C．串中所含不同字符的个数　　D．串中所含非空格字符的个数

6．假设以行序为主序存储二维数 array[1，…，100，1，…，100]，设每个数据元
存储单元，基地址为 10，则 [5,5]=（　　）。
A．808　　B．8　　C．1 010　　D．1 02

7．设有数组 A[i,j]，数组的每个元素长度为 3 字节，i 的值为 1 到 8，j 的值为 1
存首地址 BA 开始顺序存放，列为主存放时，元素 A[5,8]的存储首地
A．BA+141　　B．BA　　C．BA+222　　D．

8．设有一个 10 阶的对称矩阵 A，采用压缩存储方式，以行序为主存储，a
地址为 1，每个元素占一个地址，则 a_{85} 的地址为（　　）。
A．13　　B．32　　C．33

9．若对 n 阶对称矩阵 A 以行序为主序将其下三角形的元素（包
次存放于一维数组 B[1...(n(n+1)/2)]则在 B 中确定 a_{ij}（i<j）
A．i*(i-1)/2+j　　B．j*(j　　C．i*(i+1)/2+j

10．二维数组 A 的每个元素是由 10 个字符组成的串，其行下标
10。若 A 按行先存储，元素 A[
址相同。设每个字符占 1 字节。
A．A[8,5]　　B．A[3,1　　C．A[

11．二维数组 A[1，…，m，1，…，n 列
维数组元素 A[i,j]在一维数组 B 中
A．(i-1)*n+j　　B．(i-1)*

72

…)将字符串 t 插入到
t 插入。（说明：不得使

73

12. [0，…，4，-1，…，-3，5，…，7]中含有元素的个数（　　）。
A. 55　　B. 45　　C. 36　　D.

13. 广义表 A=(a,b,(c,d),(e,(f,g)))，则 Head(Tail(Head(Tail(Tail(A)))))的值
A. (g)　　B. (d)　　C. c　　D.

14. 广义表((a,b,c,d))的表头是（　　），表尾是（　　）。
A. a　　B. ()　　C. (a

15. 设广义表 L=((a,b,c))，则 L 的长度和深度分别
A. 1 和 1　　B. 1 和 3

16.【2015 年第 8 题】已知字符串 s 为“
算法进行匹配，第一次出现“
别是（　　）。
A. i=1，j=0

17.【2016 年第 4
先次序
A

18.
A

19. 设主
间复杂
A. O(m,

20. 设有两个串
A. 求子串

21. 若串 S=“softw
A. 8

将三对角矩阵 A[1,
素 A[66,65]在 B 中的位
A. 198　　B

二维数组 A 的元素都是 6
10，从供选择的答案中分
）存放 A 至少需要（　　）
A 的第 8 列和第 5 行共占（
择的答案：
A. 90
. 108　　B. 180
B. 114

组之所以有行优先顺序和列优先顺
的元素处在行和列两个关系中
的元素必须从左到右顺序排列
的元素之间存在次序关系
多维结构，内存是一维结构

4. 已知字符串 s1 中存放一段英文，写出算法 Format(s1, s2, s3, n)，将其按给定的长度 n 格式化成两端对齐的字符串 s2（即长度为 n 且首尾字符不得为空格字符），其多余的字符送 s3。
5. 设二维数组 a[1，…，m，1，…，n]含有 m×n 个整数。
（1）写一个算法判断 a 中所有元素是否互不相同？输出相关信息（yes/no）；
（2）试分析算法的时间复杂度。
6. 设任意 n 个整数存放于数组 A(1:n)中，试编写算法，将所有正数排在所有负数前面（要
算法复杂度为 0(n)）。

4. 已知字符串 $s1$ 中存放一段英文，写出算法 Format($s1$, $s2$, $s3$, n)，将其按给定的长度 n 格式化成两端对齐的字符串 $s2$（即长度为 n 且首尾字符不得为空格字符），其多余的字符送 $s3$。

5. 设二维数组 a[1，…，m，1，…，n]含有 $m \times n$ 个整数。

（1）写一个算法判断 a 中所有元素是否互不相同？输出相关信息（yes/no）；

（2）试分析算法的时间复杂度。

6. 设任意 n 个整数存放于数组 $A(1{:}n)$中，试编写算法，将所有正数排在所有负数前面（要求算法复杂度为 $0(n)$）。

25．对特殊矩阵采用压缩存储的主要目的是（　　）。

A．表达变得简单　　B．对矩阵元素的存取变得简单

C．去掉矩阵中多余的元素　　D．减少不必要的存储空间

26．对 n 阶对称矩阵压缩存储时，需要表长为（　　）的顺序表。

A．$n/2$　　B．$n^2/2$　　C．$n(n+1)/2$　　D．$n(n-1)/2$

27．设有一个 $n*n$ 的对称矩阵 A，将其下三角部分按行存放在一维数组 B 中，而 $A[0][0]$存放于 $B[0]$中，那么，第 i 行对角线元素 $A[i][i]$存放于 B 中（　　）处。

A．$(i+3)i/2$　　B．$(i+1)i/2$　　C．$(2n-i+1)i/2$　　D．$(2n-i-1)i/2$

28．下列说法正确的是（　　）。

（1）稀疏矩阵压缩存储后，必会失去随机存取功能。

（2）若一个广义表的表头为空表，则此广义表亦为空表。

（3）广义表的取表尾运算，其结果通常是个表，但有时也可是个单元素值。

（4）从逻辑结构上看，n 维数组是由多个 $n-1$ 维的数组构成。

A．仅（1）（2）　　B．仅（1）（4）　　C．仅（2）（3）　　D．仅（3）（4）

29．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

30．广义表 $L=(a,(b,c))$，进行 Tail(L)操作后的结果为（　　）。

A．c　　B．b,c　　C．(b,c)　　D．$((b,c))$

二．应用题

1．已知模式串 t= “abcaabbabcab” 写出用 KMP 法求得的每个字符对应的 next 和 nextval 函数值。

2．设目标为 t= “abcaabbabcabaacbacba”，模式为 p= “abcabaa”。

（1）计算模式 p 的 naxtval 函数值；

（2）画出利用 KMP 算法进行模式匹配时每一趟的匹配过程。

3．数组 A 中，每个元素 $A[i,j]$的长度均为 32 个二进位，行下标从-1 到 9，列下标从 1 到 11，从首地址 S 开始连续存放主存储器中，主存储器字长为 16 位。求：

（1）存放该数组所需多少单元?

（2）存放数组第 4 列所有元素至少需多少单元?

（3）数组按行存放时，元素 $A[7,4]$的起始地址是多少?

（4）数组按列存放时，元素 $A[4,7]$的起始地址是多少?

4．请将香蕉 banana 用工具 H( )—Head( )，T( )—Tail( )从 L 中取出。

L = (apple, (orange, (strawberry, (banana)), peach), pear)。

三．算法设计题

1．写一个算法，统计在输入字符串中各个不同字符出现的频度（字符串中的合法字符为 A～Z 这 26 个字母和 0～9 这 10 个数字）。

2．写一个递归算法来实现字符串逆序存储，要求不另设串存储空间。

3．编写算法，实现下面函数的功能。函数 void Insert(char*s,char*t,int pos)将字符串 t 插入到字符串 s 中，插入位置为 pos。假设分配给字符串 s 的空间足够让字符串 t 插入。（说明：不得使用任何库函数。）

12. [0，…，4，–1，…，–3，5，…，7]中含有元素的个数（　　）。
A．55　　B．45　　C．36　　D．16

13. 广义表 $A=(a,b,(c,d),(e,(f,g)))$，则 Head(Tail(Head(Tail(Tail(A)))))的值为（　　）。
A．(g)　　B．(d)　　C．c　　D．d

14. 广义表((a,b,c,d))的表头是（　　），表尾是（　　）。
A．a　　B．( )　　C．(a,b,c,d)　　D．(b,c,d)

15. 设广义表 $L=((a,b,c))$，则 L 的长度和深度分别为（　　）。
A．1 和 1　　B．1 和 3　　C．1 和 2　　D．2 和 3

16.【2015 年第 8 题】已知字符串 s 为“abaabaabacacaabaabcc”，模式串 t 为“abaabc”。采用 KMP 算法进行匹配，第一次出现“失配”（s[i]!=t[j]）时，$i=j=5$，则下次开始匹配时，i 和 j 的值分别是（　　）。
A．$i=1$，$j=0$　　B．$i=5$，$j=0$　　C．$i=5$，$j=2$　　D．$i=6$，$j=2$

17.【2016 年第 4 题】有一个 100 阶的三对角矩阵 M，其元素 $m_{i,j}(1\leqslant i\leqslant 100,\ 1\leqslant j\leqslant 100)$按行优先次序压缩存入下标从 0 开始的一维数组Ⅳ中。元素 $m_{30,30}$ 在 N 中的下标是（　　）。
A．86　　B．87　　C．88　　D．89

18. 对于 KMP 算法，在模式匹配时指示主串匹配位置的指针（　　）。
A．不会变大　　B．不会变小　　C．都有可能　　D．无法判断

19. 设主串的长度为 n，子串的长度为 m，那么 BF 算法的时间复杂度为（　　），KMP 算法的时间复杂度为（　　）。
A．O(m)　　B．O(n)　　C．O(n×m)　　D．O(n+m)

20. 设有两个串 S1 和 S2，求 S2 在 S1 中首次出现的位置的运算称作（　　）。
A．求子串　　B．判断是否相等　　C．模式匹配　　D．连接

21. 若串 S=“software”，其子串的数目是（　　）。
A．8　　B．37　　C．36　　D．9

22. 将三对角矩阵 A[1，…，100，1，…，100]按行优先存入一维数组 B[1，…，298]中，A 中元素 A[66,65]在 B 中的位置 K 为（　　）。
A．198　　B．195　　C．197　　D．199

23. 二维数组 A 的元素都是 6 个字符组成的串，行下标 i 的范围从 0 到 8，列下标 j 的范围从 1 到 10，从供选择的答案中分别选出正确答案。
（1）存放 A 至少需要（　　）字节；
（2）A 的第 8 列和第 5 行共占（　　）字节。
供选择的答案：
（1）A．90　　B．180　　C．270　　D．540
（2）A．108　　B．114　　C．54　　D．60

24. 多维数组之所以有行优先顺序和列优先顺序两种存储方式是因为（　　）。
A．数组的元素处在行和列两个关系中
B．数组的元素必须从左到右顺序排列
C．数组的元素之间存在次序关系
D．数组是多维结构，内存是一维结构

4.1　习题

一．选择题

1．串是一种特殊的线性表，其特殊性体现在（　　）。

A．可以顺序存储　　B．数据元素是一个字符

C．可以链式存储　　D．数据元素可以是多个字符

2．下面关于串的叙述中，（　　）是不正确的。

A．串是字符的有限序列

B．空串是由空格构成的串

C．模式匹配是串的一种重要运算

D．串既可以采用顺序存储，也可以采用链式存储

3．串“ababaaababaa”的 next 数组为（　　）。

A．012345678999　　B．012121111212　　C．011234223456　　D．0123012322345

4．串“ababaabab”的 nextval 数组为（　　）。

A．010104101　　B．010102101　　C．010100011　　D．010101011

5．串的长度是指（　　）。

A．串中所含不同字母的个数　　B．串中所含字符的个数

C．串中所含不同字符的个数　　D．串中所含非空格字符的个数

6．假设以行序为主序存储二维数组 A=array[1，…，100，1，…，100]，设每个数据元素占 2 个存储单元，基地址为 10，则 LOC[5,5]=（　　）。

A．808　　B．818　　C．1 010　　D．1 020

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

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

8．设有一个 10 阶的对称矩阵 A，采用压缩存储方式，以行序为主存储，a_{11}为第一元素，其存储地址为 1，每个元素占一个地址空间，则 a_{85}的地址为（　　）。

A．13　　B．32　　C．33　　D．40

9．若对 n 阶对称矩阵 A 以行序为主序方式将其下三角形的元素（包括主对角线上所有元素）依次存放于一维数组 $B[1...(n(n+1))/2]$中，则在 B 中确定 a_{ij}（$i<j$）的位置 k 的关系为（　　）。

A．$i*(i-1)/2+j$　　B．$j*(j-1)/2+i$　　C．$i*(i+1)/2+j$　　D．$j*(j+1)/2+i$

10．二维数组 A 的每个元素是由 10 个字符组成的串，其行下标 i=0，1，…，8，列下标 j=1，2，…，10。若 A 按行先存储，元素 $A[8,5]$的起始地址与当 A 按列先存储时的元素（　　）的起始地址相同。设每个字符占 1 字节。

A．$A[8,5]$　　B．$A[3,10]$　　C．$A[5,8]$　　D．$A[0,9]$

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

A．$(i-1)*n+j$　　B．$(i-1)*n+j-1$　　C．$i*(j-1)$　　D．$j*m+i-1$

4.2　答案及解析

一、选择题

1. B	2. B	3. C	4. A	5. B	6. B	7. B	8. C	9. B	10. B
11. A	12. B	13. D	14. C、B	15. C	16. C	17. B	18. B	19. C、D	20. C
21. B	22. B	23. D、A	24. D	25. D	26. C	27. A	28. B	29. A	30. D

1.【答案】B

【考点】串的定义

【解析】

字符串和栈、队列一样，都是特殊的线性表，栈和队列的特殊性体现在操作受限，而字符串的特殊性体现在内容受限。字符串中的每个数据元素只能为一个字符，而不能为其他类型的数据。所以答案选择 B。

2.【答案】B

【考点】字符串的相关概念

【解析】

空格常常是串的字符集合中的一个元素，由一个或多个空格组成的串称为空格串，并非空串，其长度就是其所包括的空格的个数。而空串是指包括零个字符的串，空串的长度为零。所以答案选择 B。

3.【答案】C

【考点】KMP 算法中 next 数组的计算

【解析】

对于模式串 t，求解 next 数组的基本步骤为：

（1）next[1]=0，next[2]=1。

（2）以后求解每一位的 next[j]值时，根据 j 的前一位进行比较，令 k=next[j−1]。

（3）将 $t[j-1]$与 $t[k]$进行比较，如果相等，则 next[j]=k+1；如果不等，令 k=next[k]，若 k 不等于 0，重复步骤（3），直到 k 等于 0，next[j]=1。

根据上述步骤，对于所给的模式串“ababaaababaa”，next[1]=0，next[2]=1，其余 next[j]值求解过程如下：

（1）当 j=3 时，k=next[j−1]=next[2]=1，观察 $t[2]$与 $t[k]$是否相等，$t[2]=b$，$t[1]=a$，不等，此时 k=next[k]=0，所以 next[j]=1。

（2）当 j=4，k=next[j−1]=next[3]=1，观察 $t[3]$与 $t[k]$是否相等，$t[3]=a$，$t[1]=a$，相等，所以 next[j]=k+1=2。

（3）当 j=5，k=next[j−1]=next[4]=2，观察 $t[4]$与 $t[k]$是否相等，$t[4]=b$，$t[2]=b$，相等，所以 next[j]=k+1=3。

（4）当 j=6，k=next[j−1]=next[5]=3，观察 $t[5]$与 $t[k]$是否相等，$t[5]=a$，$t[3]=a$，相等，所以 next[j]=k+1=4。

以此类推，可以推出模式串“ababaaababaa”的 next 数组，如图 4.1 所示，所以答案选择 C。

另外，当求解出 next[6]后，可观察 next 数组的前 6 项为 011234，此时即可排除选项 A、B 和 D，选择 C。

j	1	2	3	4	5	6	7	8	9	10	11	12
模式串 t	a	b	a	b	a	a	a	b	a	b	a	a
next[j]	0	1	1	2	3	4	2	2	3	4	5	6

图 4.1　串“ababaaababaa”的 next 数组的计算

4.【答案】A

【考点】KMP 算法中 nextval 数组的计算

【解析】

nextval 的计算需要根据 next 的值进行计算。对于模式串 t，具体求解方法为：nextval[1]=0，从第二位开始，将 nextval[next[j]]与 next[j]的值进行比较，若相等，则 nextval[j]=nextval[next[j]]；若不等，则 nextval[j]=next[j]。根据此规则，答案选择 A。

5.【答案】B

【考点】字符串的相关概念

【解析】

串中字符的数目 n 称为串的长度，这些字符可能有相同的，也可能包括空格。

6.【答案】B

【考点】二维数组地址的计算

【解析】

假设二维数组 $A[s\cdots m,t\cdots n]$每个数据元素占 L 个存储单元，LOC(i,j)是 a_{ij} 的存储位置，LOC(s, t)是 a_{st} 的存储位置，即数组的起始存储位置。当数组以行序为主序进行存储时，则元素 a_{ij} 的前面存储了 $i-s$ 行元素，每行有 $n-t+1$ 个元素，a_{ij} 所在行的前面则存储了 $j-t$ 个元素，因此 a_{ij} 的位置可由下式确定。

$$\mathrm{LOC}(i,j)=\mathrm{LOC}(s,t)+((i-s)\times(n-t+1)+(j-t))\times L$$

根据上式，LOC[5,5]的值为：

$$\mathrm{LOC}(5,5)=\mathrm{LOC}(1,1)+((5-1)\times(100-1+1)+(5-1))\times 2=818$$

7.【答案】B

【考点】二维数组地址的计算

【解析】

假设二维数组 $A[s\cdots m,t\cdots n]$每个数据元素占 L 个存储单元，LOC(i,j)是 a_{ij} 的存储位置，LOC(s, t)是 a_{st} 的存储位置，即数组的起始存储位置。当数组以列序为主序进行存储时，则元素 a_{ij} 的前面存储了 $j-t$ 列元素，每列有 $m-s+1$ 个元素，a_{ij} 所在行的前面则存储了 $i-s$ 个元素，因此 a_{ij} 的位置可由下式确定。

$$\mathrm{LOC}(i,j)=\mathrm{LOC}(s,t)+((j-t)\times(m-s+1)+(i-s))\times L$$

根据上式，元素 A[5,8]的存储首地址为：

$$\mathrm{LOC}(5,8)=\mathrm{LOC}(1,1)+((8-1)\times(8-1+1)+(5-1))\times 3=\mathrm{BA}+180$$

8.【答案】C

【考点】矩阵的压缩存储

【解析】

对于对称矩阵，当以行序为主序存储其下三角中的元素时，存储方式如图 4.2 所示，可以看

出，a_{85} 的地址为：1+2+3+4+5+6+7+5=33。

$$
\begin{array}{lllll}
a11 & & & & \\
a21 & a22 & & & \\
a31 & a32 & a33 & & \\
a41 & a42 & a43 & a44 & \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
a81 & a82 & a83 & a84 & a85
\end{array}
$$

图 4.2　对称矩阵的压缩存储

9.【答案】B

【考点】矩阵的压缩存储

【解析】

此题要求行序为主序存储其下三角中的元素，存储方式与图 4.2 类似。对于上三角中的元素 a_{ij}（$i<j$），其存储地址同下三角中的元素 a_{ji}。设存储在 a_{ji} 前面的元素总计为 m 个，则

$$m=1+2+3+\cdots+(j-1)+i-1=j(j-1)/2+i-1$$

而由于一维数组 $B[1...(n(n+1))/2]$的下标从 1 开始，因此，a_{ij}（$i<j$）在 B 中的位置 k 为 $m+1$，即 $j\times(j-1)/2+i$。

10.【答案】B

【考点】二维数组地址的计算

【解析】

若 A 按行先存储，根据第 6 题给出的计算公式，可以得到，LOC[8,5]的值为：

$$\mathrm{LOC}(8,5)=\mathrm{LOC}(0,1)+((8-0)\times(10-1+1)+(5-1))\times 10=\mathrm{LOC}(0,1)+840$$

若 A 按列先存储，则可根据第 6 题给出的计算公式，计算 LOC[3,10]的值为：

$$\mathrm{LOC}(3,10)=\mathrm{LOC}(0,1)+((10-1)\times(8-0+1)+(3-0))\times 10=\mathrm{LOC}(0,1)+840$$

因此，答案选择 B。

11.【答案】A

【考点】二维数组地址的计算

【解析】

类似第 6 题的计算方法，当数组 A 以行序为主序进行存储时，则元素 $A[i,j]$的前面存储了 $i-1$ 行元素，每行有 n 个元素，则 $A[i,j]$所在行的前面存储了 $j-1$ 个元素，因此，$A[i,j]$前面总计存储的元素个数 k 为：

$$k=(i-1)\times n+j-1$$

因为一维数组的下标从 1 开始，所以二维数组元素 $A[i,j]$在一维数组 B 中的下标为 $k+1$，即 $(i-1)\times n+j$，所以答案选择 A。

12.【答案】B

【考点】数组的存储

【解析】

这是一个三维数组，设每维的长度分别为 r、s 和 t，则数组元素个数 $n=r\times s\times t$，有 $r=4-0+1=5$，$s=-1-(-3)+1=3$，$t=7-5+1=3$，因此，$n=5\times 3\times 3=45$。

13.【答案】D

【考点】广义表的基本操作

【解析】

Head(A)表示取出的表头为非空广义表的第一个元素。Tail(A)表示取出的表尾为除去表头之外，由其余元素构成的表。Tail(A)=(b,(c,d),(e,(f,g)))；Tail(Tail(A))=((c,d),(e,(f,g)))；Head(Tail(Tail(A)))=(c,d)；Tail(Head(Tail(Tail(A))))=(d)；Head(Tail(Head(Tail(Tail(A)))))=d。因此答案选择 D。

14.【答案】C、B

【考点】广义表的基本操作

【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，广义表((a,b,c,d))的表头为一个子表(a,b,c,d)；表尾为除去表头之外，由其余元素构成的表，表尾一定是个广义表，广义表((a,b,c,d))的表尾为空表( )。

15.【答案】C

【考点】广义表的相关概念

【解析】

广义表的长度是指广义表中所含元素的个数，深度是指广义表中括号的层数。根据定义易知，L 的长度为 1，深度为 2，所以答案选择 C。

16.【答案】C

【考点】KMP 算法中 next 数组的计算

【解析】

方法一：排除法。

此题可以通过排除法快速求解。因为 KMP 算法在匹配过程中产生“失配”时，主串中的指针 i 是不移动的，这样便排除了选项 A 和 D。对于选项 B，因为指针 j 不移动，显然是不对的，所以选择答案 C。

方法二：手工求解 next 数组法。

根据 KMP 算法，当匹配过程中产生“失配”时，指针 i 不变，指针 j 退回到 next[j] 所指示的位置上重新进行比较。根据第 3 题给出的求解 next 数组的步骤（注意：此题中子串和模式串的位置都是从 0 开始），可以推出模式串 t 的 next 数组，如图 4.3 所示。因为出现“失配”时，i=j=5，next[j]=next[5]=2，所以下次开始匹配时，i=5，j=2。因此选择答案 C。

j	0	1	2	3	4	5
模式串 t	a	b	a	a	b	c
next[j]	-1	0	0	1	1	2

图 4.3　串“abaabc”的 next 数组的计算

17.【答案】B

【考点】矩阵的压缩存储

【解析】

三对角矩阵如图 4.4 所示，矩阵的第一行和最后一行有 2 个元素，其余行有 3 个元素。前 29 行有 2+3×28=86 个元素，$m_{30,30}$ 是第 30 行的第二个元素,又因为下标从 0 开始，所以元素 $m_{30,30}$ 在 N 中的下标是 87，答案选择 B。

$$A_{n\times n}=\begin{bmatrix} m_{1,1} & m_{1,2} & & & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & & \\ & m_{3,2} & m_{3,3} & m_{3,4} & \\ & & \cdots & \cdots & \cdots \\ 0 & & & m_{n,n-1} & m_{n,n} \end{bmatrix}$$

图 4.4　三对角矩阵

18.【答案】B

【考点】KMP 算法

【解析】

与 BF 算法相比，KMP 算法的特点就是在模式匹配的过程中，主串指针不会回溯，所以指针不会变小。

19.【答案】C、D

【考点】BF 算法和 KMP 算法

【解析】

对于 BF 算法，当匹配失败时，主串的指针 i 总是回溯到 $i-j+2$ 位置，模式串的指针总是回溯到首字符位置 $j=1$。在最坏情况下，每趟不成功的匹配都发生在模式串的最后一个字符与主串中相应字符的比较，其时间复杂度为 $O(n\times m)$。KMP 算法对对于 BF 算法进行了改进，每当一趟匹配过程中出现字符比较不等时，不需回溯 i 指针，而是利用已经得到的“部分匹配”的结果将模式向右“滑动”尽可能远的一段距离后，继续进行比较，其时间复杂度为 $O(n+m)$。

20【答案】C

【考点】字符串的相关运算

【解析】

求子串是指从字符串 $S1$ 中截取自第 i 个字符开始的长度为 L 的子串 $S2$，所以排除选项 A。模式匹配是指在字符串 $S1$ 中寻找子串 $S2$ 首次出现的位置，因此选项 C 正确，而选项 B 和 D 明显错误。

21.【答案】B

【考点】字符串的相关概念

【解析】

“software”没有重复的字符，其长度为 8。其中包括 1 个字符的子串有 8 个，包括 2 个字符的子串有 7 个，以此类推，包括 8 个字符的子串有 1 个。因此，S 的非空子串总计有：8+7+6+5+4+3+2+1=36。另外，因为空串是任何串的子串，所以 S 的子串数目总计为 37 个。

22.【答案】B

【考点】矩阵的压缩存储

【解析】

三对角矩阵的特点是：只有对角线以及对角线的两侧有值，其余元素均为 0，如图 4.5 所示。将三对角矩阵 A 存入数组 B 中的元素依次为：$A[1][1]$，$A[1][2]$，A[2][1]，$A[2][2]$，$A[2][3]$，$A[3][2]$，$A[3][3]$，$A[3][4]$……，当存储到元素 $A[66,65]$时，总计有 65×3−1=194 个元素，而 $A[66,65]$是数组 A 中第 66 行中的第一个元素，故其在 B 数组中的下标为 195。

本题也可根据公式直接进行计算，$B[K]$和$A[i,j]$之间的对应关系为：$K=2\times i+j-2$，当$i=66$，$j=65$时，$K=2\times66+65-2=195$。

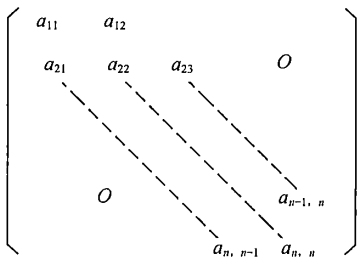


图 4.5　三对角矩阵

23.【答案】D、A

【考点】数组的存储

【解析】

（1）根据行、列下标的范围可知数组A有 9 行 10 列，总计包括 90 个元素，因为每个元素是 6 个字符组成的串，则数组A总计包括$6\times90=540$个字符，又因为每个字符占用一个字节，所以存放A至少需要 540 字节。

（2）第 8 列有 9 个元素，第 5 行有 10 个元素，但第 8 列和第 5 行重复计算了一个元素，所以总计包括 18 个元素，所占字节数总计为$6\times18=108$。

24.【答案】D

【考点】数组的存储

【解析】

由于存储单元是一维的结构，而数组可能是多维的结构，所以用一组连续存储单元存放数组的数据元素就有次序约定问题。因此多维数组有行优先顺序和列优先顺序两种存储方式，因此答案选择 D。

25.【答案】D

【考点】特殊矩阵的压缩存储

【解析】

特殊矩阵中包括很多值相同的元素或者是零元素。为了节省存储空间，可以对这类矩阵进行压缩存储。所谓压缩存储，是指为多个值相同的元素只分配一个存储空间，对零元不分配空间。因此答案选择 D。

26.【答案】C

【考点】特殊矩阵的压缩存储

【解析】

对称矩阵只需要存储一半元素，即存储其下（上）三角（包括对角线）中的元，总计包括元素个数为：$1+2+3+\cdots+(n-1)+n=n(n+1)/2$。

27.【答案】A

【考点】特殊矩阵的压缩存储

【解析】

第 0 行有 1 个元素，第 1 行有 2 个元素，则第 i 行有 i+1 个元素，因此，从 $A[0][0]$到 $A[i][i]$总计包括元素个数为：$1+2+\cdots+(i+1)=(i+1)(i+2)/2$。又因为数组 B 的起始坐标为 0，所以 $A[i][i]$在数组 B 中的下标为：$(i+1)\times(i+2)/2-1=(i+3)i/2$。因此答案选择 A。

这种题目也可采用特殊值法进行验证，排除错误选项。例如，取 $n=2$，$i=1$，则 $A[1][1]=B[2]$，用选项 A 中的公式进行验证是正确的，而用选项 B、C、D 中的公式进行验证是错误的。

28.【答案】B

【考点】数组和压缩矩阵的存储、广义表的基本操作

【解析】

（1）正确。在稀疏矩阵中，非零元素的分布是没有规律的，为了进行压缩存储，需要将每一个非零元素的值和其所在的行号、列号作为一个结点存放在一起，这样的结点组成的线性表称作三元组表。因为该结点已不是简单的向量，是无法根据下标进行存取的，所以说稀疏矩阵压缩存储后，必会失去随机存取功能。

（2）错误。广义表的表头为空，并不代表该广义表为空表。例如 $L=(())$，L 的表头为空，但 L 的长度是 1，并非空表。

（3）错误。表尾是指除去表头后剩下的元素组成的表，即表尾一定是一个广义表，表头可以为表或单元素值。

（4）正确。二维数组可以转换为线性表结构，每个元素本身又是一个线性表。三维数组则可以看成多个二维数组构成的线性表，以此类推，n 维数组则可以看成有多个 n–1 维数组构成的线性表。

29.【答案】A

【考点】广义表的相关概念

【解析】

任何一个非空广义表的表头元素可能是原子元素，也可能是表元素，但其表尾元素一定是广义表。由于广义表中的数据元素既可以是单个元素，也可以是子表，因此广义表难以用顺序存储结构来表示，通常采用链式存储结构来表示。表中的每个元素可用一个结点来表示，广义表中有两类结点：一类是单个元素结点，一类是子表结点。因此答案选择 A。

30.【答案】D

【考点】广义表的基本操作

【解析】

Tail(L)表示取出的表尾为除去表头之外，由其余元素构成的表。Tail(L)=((b,c))，因此答案选择 D。

二、应用题

1．解答

根据选择题第 3 题给出的求解 next 和第 4 题给出的求解 nextval 函数值的方法，可以推出模式串“abcaabbabcab”的 next 和 nextval 对应的值，如图 4.6 所示。

j	1	2	3	4	5	6	7	8	9	10	11	12
模式串 t	a	b	c	a	a	b	b	a	b	c	a	b
next[j]	0	1	1	1	2	2	3	1	2	3	4	5
nextval[j]	0	1	1	0	2	1	3	0	1	1	0	5

图 4.6

2．解答

（1）p 的 next 函数值为 0111232，进一步求得 p 的 nextval 函数值为 0110132。

（2）利用 KMP(改进的 nextval)算法，每趟匹配过程如下：

第一趟匹配：abcaabbabcabaacbacba

abcab(i=5，j=5)

第二趟匹配：abcaabbabcabaacbacba

abc(i=7，j=3)

第三趟匹配：abcaabbabcabaacbacba

a(i=7，j=1)

第四趟匹配：abcaabbabcabaacbacba

(成功) abcabaa(i=15，j=8)

3．解答

每个元素占 32 个二进制位，主存字长 16 位，故每个元素占 2 个字长，行下标可平移至 1 到 11。

（1）242。该数组包括的元素个数总计有：(9–(–1)+1)×(11–1+1)=121，因为每个元素占 2 个字长，存放该数组所需单元总计为 121×2=242 个。

（2）22。第 4 列包括的元素个数总计有：(9–(–1)+1)=11，每个元素占 2 个字长，因此，存放第 4 列所有元素至少需单元为 11×2=22 个。

（3）S+182。LOC=S+((7–(–1))×(11–1+1)+(4–1))×2=S+182。

（4）S+142。LOC=S+((7–1)×(9–(–1)+1)+(4–(–1)))×2=S+142。

4．解答

$H(H(T(H(T(H(T(L)))))))$。

$T(L)$=((orange,(strawberry,(banana)),peach),pear)

$H(T(L))$=(orange,(strawberry,(banana)),peach)

$T(H(T(L)))$=((strawberry,(banana)),peach)

$H(T(H(T(L))))$=(strawberry,(banana))

$T(H(T(H(T(L)))))$=((banana))

$H(T(H(T(H(T(L))))))$=(banana)

$H(H(T(H(T(H(T(L)))))))$=banana

三．算法设计题

1．解答

【算法思想】

由于字符串中只可能出现 26 个字母和 10 个数字，所以可以利用一个长度为 36 的整型数组 *num* 记录各个字符出现的次数，前 10 个元素记录数字字符出现的次数，后 26 个元素记录字母出现的次数。依次读入字符串中的每个字符，当遇到数字字符时，则将相应数字的 ASCII 代码值减去数字字符‘0’的 ASCII 代码值（48），得出的数值（0，…，9）作为该数字字符在 *num* 的下标；当遇到字母字符时，则将相应字母的 ASCII 代码值减去字符‘A’的 ASCII 代码值（65），然后加上 10，得出的数值（10，…，25）作为该字母字符在 *num* 的下标；当遇到其他符号时则不做任何处理，直至输入字符串结束。

【算法描述】

```
void Count()
{//统计输入字符串中数字字符和字母字符的个数
   for(i=0;i<36;i++) num[i]=0;        //初始化
   while((ch=getchar())!='\0')        //依次读入字符进行判断
      if('0'<=ch<='9')                //数字字符
      {
         i=ch-48;
         num[i]++;
      }
      else if('A'<=ch<='Z')           //字母字符
      {
         i=ch-65+10;
         num[i]++;
      }
   for(i=0;i<10;i++)                  //输出数字字符的个数
      cout<<"数字"<<i<<"的个数="<<num[i]<<endl;
   for(i=10;i<36;i++)                 //输出字母字符的个数
      cout<<"字母字符"<<char(i+55)<<"的个数="<<num[i]<<endl;
}
```

2．解答

【算法思想】

字符串的逆序存储即要求将第一个输入的字符最后存储，而最后输入的字符先存储。利用递归方法实现时，可以在读入一个字符后进行递归，递归返回后再进行存储。这样相当于将读入的字符逐个压栈，全部读入后再逐个出栈，依次存储在数组中，从而完成了逆序存储。

【算法描述】

```
void Inverse(char A[])
{//递归实现字符串的逆序存储
   static int i=0;                    //静态变量记录字符数组的下标
   cin>>ch;
   if(ch!='.')                        //'.'是字符串输入结束标志
   {
      Inverse(A);                     //递归
      A[i++]=ch;                      //递归返回后存储字符串
   }
   A[i]='\0';                         //字符串最后加上结尾标记
}
```

3．解答

【算法思想】

此算法是字符串的插入问题，要求在字符串 s 的 pos 位置，插入字符串 t，可以分两大步来完成：(1)查找字符串 s 的 pos 位置，将第 pos 个字符到字符串 s 尾的子串向后移动字符串 t 的长度；(2)将字符串 t 复制到字符串 s 的第 pos 个位置后。

注意，因为题目已假设分配给字符串 s 的空间足够大，所以不存在插入造成空间溢出的问题。但算法在插入前要判断插入位置 pos 的合法性，pos 小于 1 或大于串 s 的长度均为非法。

【算法描述】

```
void Insert(char *s,char *t,int pos)
{
   p=s;q=t;                          //p、q分别为字符串s和t的工作指针
   if(pos<1)                         //插入位置pos合法性的判断
   {
      cout<<"pos参数位置非法"<<endl;
      exit(0);
   }
   while(*p!='\0'&&i<pos)            //查找pos的位置，找到时i=pos
   {
      p++;
      i++;
   }
   if(*p=='\0')                      //pos的位置大于字符串s的长度
   {
      cout<<pos<<"位置大于字符串s的长度";
      exit(0);
   }
   else                              //查找字符串的尾
      while(*p!='\0')                //查到串尾时，i为字符'\0'的下标，p指向'\0'
      {
         p++;
         i++;
      }
   while(*q!='\0')                   //查找字符串t的长度x，循环结束时q指向'\0'
   {
      q++;
      x++;
   }
   for(j=i;j>=pos;j--)               //串s的pos后的子串右移，空出串t的位置
   {
      *(p+x)=*p;
      p--;
   }
   q--;                              //指针q回退到串t的最后一个字符
   p=p+x;                            //指针p重新置回插入的位置
   for(j=1;j<=x;j++)                 //将t串插入到s的pos位置上
      *p--=*q--;
};
```

4. 解答

【算法思想】

本题相当于滤掉字符串 $s1$ 中某些位置的空格，将其拆分成两个字符串 $s2$ 和 $s3$。由于要求 $s2$ 长度为 n 且首尾字符不得为空格字符，因此，可以首先从左到右扫描字符串 $s1$，找到第一个非空格字符，计数到 n；然后判断第 n 个拷入字符串 s2 的字符不得为空格；最后将余下字符复制到字符串 $s3$ 中。

【算法描述】

```
void Format(char *s1,*s2,*s3)
{//将s1拆分成s2和s3，要求s2长度为n且两端对齐
```

```
    p=s1;q=s2;                          //指针 p 和 q 分别为 s1 和 s2 的工作指针
    while(*p!='\0'&&*p==' ') p++;       //滤掉 s1 左端所有的空格
    if(*p=='\0')
    {
      cout<<"字符串 s1 为空串或空格串"<<endl;
      exit(0);
    }
    i=0;
    while(*p!='\0'&&i<n)                //将 s1 中的 n 个字符复制到 s2
    {
      *q=*p;q++;p++;i++;
    }
    p--;
    q--;
    if(*p=='\0')
    {
      cout<<"字符串 s1 没有"<<n<<"个有效字符"<<endl;
      exit(0);
    }
    if(*q==' ')                         //若最后一个字符为空格
    {
      while(*p==' '&&*p!='\0')          //往后查找一个非空格字符作为 s2 的尾字符
        p++;
      if(*p=='\0')
      {
        cout<<"字符串 s1 没有"<<n<<"个两端对齐的字符串"<<endl;
        exit(0);
      }
    *q=*p;                              //复制 s2 的最后一个非空字符
    }
    *(++q)='\0';                        //置 s2 结束标记
    q=s3;                               //q 重量为 s3
    p++;                                //将 s1 其余部分送 s3
    while(*p!='\0')
    {*q=*p;q++;p++;}
    *q='\0';                            //置 s3 结束标记
}
```

5. 解答

(1)【算法思想】

判断二维数组中的元素是否互不相同，需要逐个比较。但为了提高算法效率，应当避免重复的比较，确保每个元素同其他元素比较一次且只一次。基本思想为：依次遍历数组中每个元素，对于第 i 行的每个元素，先同本行后面的元素逐个比较，然后再同第 $i+1$ 行及其后各行元素逐个比较。在比较过程中，只要找到一对相等的元素，就可断定不是互不相同，不必继续比较，返回 0；否则继续比较，最后返回 1，表明数组所有元素互不相同。

【算法描述】

```
int IsEqual(int a[m][n],int m,int n)
{//判断二维数组中 a 所有元素是否互不相同，如是，返回 1；否则，返回 0
  for(i=0;i<m;i++)
    for(j=0;j<n-1;j++)
    {
```

```
        for(p=j+1;p<n;p++;)          //和同行其他元素比较
          if(a[i][j]==a[i][p])       //只要有一个相同，则不是互不相同
          {
            cout<<"no";
            return 0;
          }
        for(k=i+1;k<m;k++)           //和第 i+1 行及以后元素比较
          for(p=0;p<n;p++)
            if(a[i][j]==a[k][p])
            {
              cout<<"no";
              return 0;
            }
      }
    cout<<"yes";return 1;            //元素互不相同
  }
```

（2）【算法分析】

二维数组中的每一个元素同其他元素都比较一次，数组总计包括 $m\times n$ 个元素，第 1 个元素同其她 $m\times n-1$ 个元素比较，第 2 个元素同其它 $m\times n-2$ 个元素比较……第 $m\times n-1$ 个元素同最后一个元素$(m\times n)$比较一次。因此，在元素互不相等的情况下，总的比较次数为：$(m\times n-1)+(m\times n-2)+\cdots+2+1=(m\times n)(m\times n-1)/2$。在存在相同元素的情况下，可能第一次比较时相同，也可能最后一次比较时相同，假设在每个位置上均可能相同，总计包括$(m\times n-1)$个位置，这时的平均比较次数约为$(m\times n)(m\times n-1)/4$。因此，算法的时间复杂度是 $O(n^4)$。

6．解答

【算法思想】

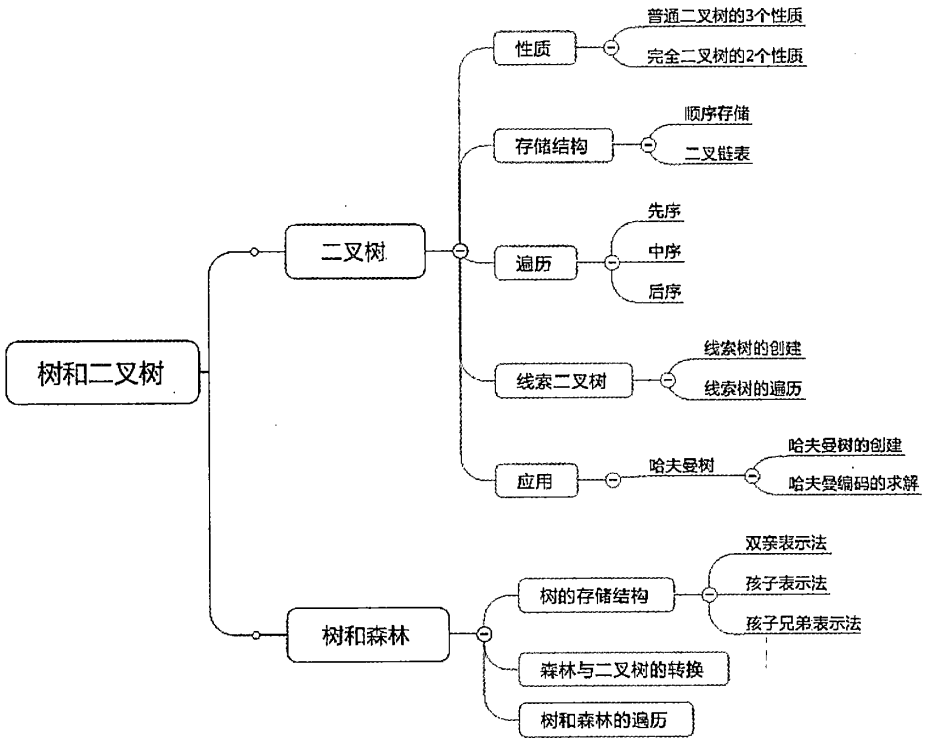
本题属于数列的划分问题，将数组 A 调整为左右两部分，每部分中的元素并不要求有序。可在数组首尾各设一个指针 low 和 high，low 从左至右搜索，遇到负数则停止；high 从右至左搜索，遇到正数则停止。然后将 low 和 high 所指向的数据进行交换。重复以上过程，直到 low 与 high 相等为止。

【算法描述】

```
void Partition(int A[],int n)
{//数组 A 中存储 n 个整数，将 A 中所有正数排在所有负数的前面
  while(low<high)
  {
    while(low<high&&A[low]>0)
      low++;
    while(low<high&&A[high]<0)
      high--;
    if(low<high)                 //交换 A[low]与 A[high]
    {
      t=A[low];
      A[low++]=A[high];
      A[high--]=t;
    }
  }
}
```

第 5 章 树和二叉树

【知识导图】



【学习目标】

1. 二叉树是一种最常用的树形结构，而满二叉树和完全二叉树又是两种特殊形态的二叉树。普通二叉树具有 3 个性质，完全二叉树具有两个性质。掌握二叉树的 5 个性质。

2. 二叉树有两种常用的存储表示：顺序存储和链式存储，链式存储是二叉树常用的存储结构，也称二叉链表存储表示法。掌握二叉树的两种存储表示法，尤其是二叉链表存储表示法，掌握不同存储结构的特点和适用场合。

3. 二叉树的遍历算法是其他运算的基础，掌握二叉树的先序、中序和后序遍历算法，掌握在三种基本遍历算法的基础上实现二叉树的其他算法。例如，二叉树的创建算法，计算二叉树的高度，计算二叉树的结点总数，计算叶子结点总数，计算度为 1 或度为 2 的结点总数，复制二叉树，交换二叉树等。

4. 在线索二叉树中，利用二叉链表中的 $n+1$ 个空指针域来存放指向某种遍历次序下的前驱结点和后继结点的指针，这些附加的指针就称为“线索”。引入二叉线索树的目的是加快查找结点前驱或后继的速度。掌握先序、中序和后序三种线索化二叉树的创建方法，掌握线索化的 3 种遍历的方法。

5. 哈夫曼树在通信编码技术上有广泛的应用，只要构造了哈夫曼树，按分支情况在左路径上写代码 0，右路径上写代码 1，然后从上到下叶子结点相应路径上的代码序列就是该叶子结点的最优前缀码，即哈夫曼编码。掌握哈夫曼树和哈夫曼编码的构造算法。

6. 树的存储结构有 3 种：双亲表示法、孩子表示法和孩子兄弟表示法。孩子兄弟表示法是常用的表示法，任意一棵树都能通过孩子兄弟表示法转换为二叉树进行存储。掌握将一般的树结构转换为二叉树的转换方法。森林与二叉树之间也存在相应的转换方法，通过这些转换，可以利用二叉树的操作解决一般树的有关问题。掌握森林与二叉树之间的相互转换方法。

5.1 习题

一、单项选择题

1. 把一棵树转换为二叉树后，这棵二叉树的形态是（　　）。
   A. 唯一的　　B. 有多种
   C. 有多种，但根结点都没有左孩子　　D. 有多种，但根结点都没有右孩子

2. 由 3 个结点可以构造出多少种不同的二叉树？（　　）。
   A. 2　　B. 3　　C. 4　　D. 5

3. 一棵完全二叉树上有 1 001 个结点，其中叶子结点的个数是（　　）。
   A. 250　　B. 500　　C. 254　　D. 501

4. 一个具有 1 025 个结点的二叉树的高 h 为（　　）。
   A. 11　　B. 10　　C. 11 至 1 025 之间　　D. 10 至 1 024 之间

5. 深度为 h 的满 m 叉树的第 k 层有（　　）个结点($1 \leqslant k \leqslant h$)。
   A. m^{k-1}　　B. m^k-1　　C. m^{h-1}　　D. m^h-1

6. 利用二叉链表存储树，则根结点的右指针是（　　）。
   A. 指向最左孩子　　B. 指向最右孩子　　C. 空　　D. 非空

7. 对二叉树的结点从 1 开始进行连续编号，要求每个结点的编号大于其左、右孩子的编号，同一结点的左右孩子中，其左孩子的编号小于其右孩子的编号，可采用（　　）遍历实现编号。
   A. 先序　　B. 中序
   C.后序　　D. 从根开始按层次遍历

8. 【2010 年第 5 题】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 的叶子结点个数是（　　）。
   A. 41　　B. 82　　C. 113　　D. 122

9. 在下列存储形式中，（　　）不是树的存储形式。
   A. 双亲表示法　　B. 孩子链表表示法　　C. 孩子兄弟表示法　　D. 顺序存储表示法

10. 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（　　）。
    A. 所有的结点均无左孩子　　B. 所有的结点均无右孩子

C．只有一个叶子结点　　D．是任意一棵二叉树

11．设哈夫曼树中有 199 个结点，则该哈夫曼树中有（　　）个叶子结点。

A．99　　B．100　　C．101　　D．102

12．若 X 是二叉中序线索树中一个有左孩子的结点，且 X 不为根，则 X 的前驱为（　　）。

A．X 的双亲　　B．X 的右子树中最左的结点

C．X 的左子树中最右结点　　D．X 的左子树中最右叶子结点

13．引入二叉线索树的目的是（　　）。

A．加快查找结点的前驱或后继的速度　　B．为了能在二叉树中方便地进行插入与删除

C．为了能方便的找到双亲　　D．使二叉树的遍历结果唯一

14．设 F 是一个森林，B 是由 F 变换得的二叉树。若 F 中有 n 个非终端结点，则 B 中右指针域为空的结点有（　　）个。

A．$n-1$　　B．n　　C．$n+1$　　D．$n+2$

15．【2010 年第 6 题】n（$n\geqslant 2$）个权值均不相同的字符构成哈夫曼树，关于该树的叙述中，错误的是（　　）。

A．该树一定是一棵完全二叉树

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．树中任一非叶子结点的权值一定不小于下一层任一结点的权值

16．【2009 年第 3 题】给定二叉树如图 5.1 所示。设 N 代表二叉树的根，L 代表根结点的左子树，R 代表根结点的右子树。若遍历后的结点序列为 3，1，7，5，6，2，4，则其遍历方式是（　　）。

A．*LRN*　　B．*NRL*　　C．*RLN*　　D．*RNL*

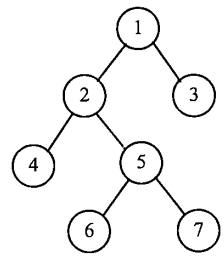


图 5.1　二叉树

17．【2009 年第 5 题】已知一棵完全二叉树的第 6 层（设根为第 1 层）有 8 个叶子结点，则该完全二叉树的结点个数最多是（　　）。

A．39　　B．52　　C．111　　D．119

18．【2009 年第 6 题】将森林转换为对应的二叉树，若在二叉树中，结点 u 是结点 v 的父结点的父结点，则在原来的森林中，u 和 v 可能具有的关系是（　　）。

I．父子关系　　II．兄弟关系　　III．u 的父结点与 v 的父结点是兄弟关系

A．只有 II　　B．I 和 II　　C．I 和 III　　D．I、II 和 III

19．【2010 年第 3 题】对于图 5.2 所示的线索二叉树(用虚线表示线索)，符合后序线索树定义的是（　　）。

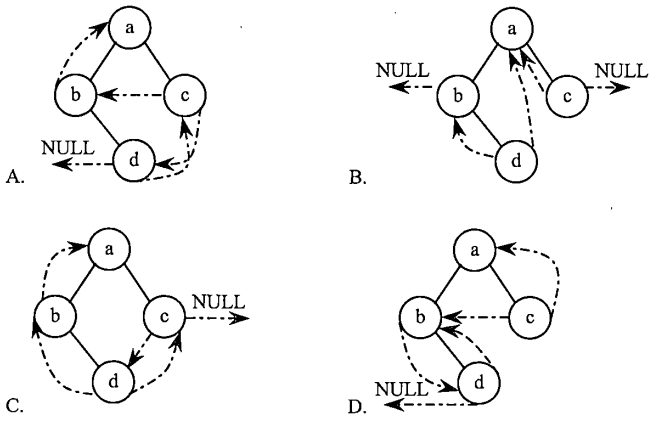


图 5.2　线索二叉树

20.【2011 年第 4 题】若一棵完全二叉树有 768 个结点，则该二叉树中叶子结点的个数是（　　）。

A．257　　B．258　　C．384　　D．385

21.【2011 年第 5 题】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

22.【2011 年第 6 题】已知一棵有 2 011 个结点的树，其叶子结点个数为 116，该树对应的二叉树中无右孩子的结点个数是（　　）。

A．115　　B．116　　C．1 895　　D．1 896

23.【2012 年第 3 题】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

24.【2013 年第 4 题】已知三叉树 T 中 6 个叶子结点的权分别是 2、3、4、5、6、7，T 的带权（外部）路径长度最小是（　　）。

A．27　　B．46　　C．54　　D．56

25.【2013 年第 5 题】若 X 是后序线索二叉树中的叶子结点，且 X 存在左兄弟结点 Y，则 X 的右线索指向的（　　）。

A．X 的父结点　　B．以 Y 为根的子树的最左下结点

C．X 的左兄弟结点 Y　　D．以 Y 为根的子树的最右下结点

26.【2014 年第 4 题】若对图 5.3 所示的二叉树进行中序线索化，则结点 x 的左、右线索指向的结点分别是（　　）。

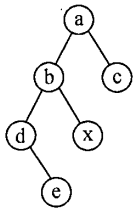


图 5.3　二叉树

A．e、c　　B．e、a　　C．d、c　　D．b、a

27.【2014 年第 5 题】将森林 F 转换为对应的二叉树 T，F 中叶子结点的个数等于（　　）。

A．T 中叶子结点的个数　　B．T 中度为 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

28.【2014 年第 6 题】5 个字符有如下 4 种编码方案，不是前缀编码的是（　　）。

A．01，0000，0001，001，1　　B．011，000，001，010，1

C．000，001，010，011，100　　D．0，100，110，1110，1100

29.【2015 年第 2 题】先序序列为 a、b、c、d 的不同二叉树的个数是（　　）。

A．13　　B．14　　C．15　　D．16

30.【2015 年第 3 题】下列选项给出的是从根分别到达两个叶子结点路径上的权值序列，能属于同一棵哈夫曼树的是（　　）。

A．24，10，5 和 24，10，7　　B．24，10，5 和 24，12，7

C．24，10，10 和 24，14，11　　D．24，10，5 和 24，14，6

31.【2016 年第 5 题】若森林 F 有 15 条边、25 个结点，则 F 包含树的个数是（　　）。

A．8　　B．9　　C．10　　D．11

二、应用题

1．试找出满足下列条件的二叉树。

（1）先序序列与后序序列相同。

（2）中序序列与后序序列相同。

（3）先序序列与中序序列相同。

（4）中序序列与层次遍历序列相同。

2．设一棵二叉树的先序序列：$ABDFCEGH$，中序序列：$BFDAGEHC$。

（1）画出这棵二叉树。

（2）画出这棵二叉树的后序线索树。

（3）将这棵二叉树转换成对应的树（或森林）。

3．假设用于通信的电文仅由 8 个字母组成，字母在电文中出现的频率分别为 0.07、0.19、0.02、0.06、0.32、0.03、0.21 和 0.10。

（1）试为这 8 个字母设计哈夫曼编码。

（2）试设计另一种由二进制表示的等长编码方案。

（3）对于上述实例，比较两种方案的优缺点。

4．已知下列字符 A、B、C、D、E、F、G 的权值分别为 3、12、7、4、2、8、11，试填写出其对应哈夫曼树 HT 存储结构的初态和终态。

5.【2012 年第 41 题】设有 6 个有序表 A、B、C、D、E、F，分别含有 10、35、40、50、60 和 200 个数据元素，各表中元素按升序排序。要求通过 5 次两两合并，将 6 个表最终合并成一个升序表，并在最坏情况下比较的总次数达到最小。请回答下列问题：

（1）给出完整的合并过程，并求出最坏情况下比较的总次数。

（2）根据你的合并过程，描述 n（$n\geqslant 2$）个不等长升序表的合并策略，并说明理由。

6.【2016 年第 42 题】如果一棵非空 $k(k\geqslant 2)$叉树 T 中每个非叶子结点都有 k 个孩子，则称 T 为正则后 k 树。请回答下列问题并给出推导过程。

（1）若 T 有 m 个非叶子结点，则 T 中的叶子结点有多少个?

（2）若 T 的高度为 h(单结点的树 h=1)，则 T 的结点数最多为多少个?最少为多少个?

三、算法设计题

1．以二叉链表作为二叉树的存储结构，统计二叉树的叶子结点个数。

2．以二叉链表作为二叉树的存储结构，判别两棵树是否相等。。

3．以二叉链表作为二叉树的存储结构，交换二叉树每个结点的左孩子和右孩子。

4．设计二叉树的双序遍历算法（双序遍历是指对于二叉树的每一个结点来说，先访问这个结点，再按双序遍历它的左子树，然后再一次访问这个结点，接下来按双序遍历它的右子树）。

5．计算二叉树的最大宽度（二叉树的最大宽度是指二叉树所有层中结点个数的最大值）。

6．用按层次顺序遍历二叉树的方法，统计树中具有度为 1 的结点数目。

7．求任意二叉树中第一条最长的路径长度，并输出此路径上各结点的值。

8．输出二叉树中从每个叶子结点到根结点的路径。

9.【2014 年第 41 题】二叉树的带权路径长度（*WPL*）是二叉树中所有叶子结点的带权路径长度之和。给定一棵二叉树 T，采用二叉链表存储，结点结构为（left、weight、right），其中叶子结点的 weight 域保存该结点的非负权值。设 root 为指向 T 的根结点的指针，请设计求 T 的 *WPL* 的算法。要求：

（1）给出算法的基本设计思想；

（2）使用 C 或 C++语言，给出二叉树结点的数据类型定义；

（3）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

5.2　答案及解析

一、单项选择题

1. A	2. D	3. D	4. C	5. A	6. C	7. C	8. B
9. D	10. C	11. B	12. C	13. A	14. C	15. A	16. D
17. C	18. B	19. D	20. C	21. C	22. D	23. A	24. B
25. A	26. D	27. C	28. D	29. B	30. D	31. C	

1.【答案】A

【考点】树的存储结构

【解析】

树常用的存储结构为孩子兄弟表示法，可以利用这种结构将一般的树结构转换为二叉树。这种存储结构的优点是它和二叉树的二叉链表表示完全一样，链表中结点的两个链域分别指向该结点的第一个孩子结点和下一个兄弟结点，因为转换规则是唯一的，所以转换成的二叉树是唯一的。

2.【答案】D

【考点】二叉树的定义

【解析】

可以构造出图 5.4 所示的五种情况。

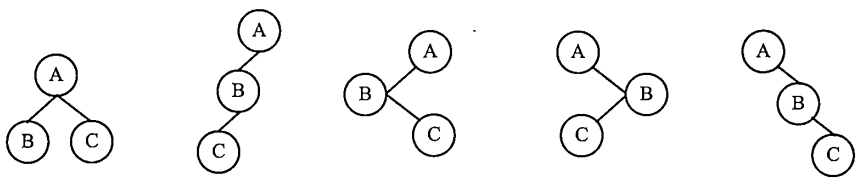


图 5.4　3 个结点构造的 5 种不同的二叉树

3.【答案】D

【考点】完全二叉树的性质

【解析】

方法一：

根据完全二叉树的性质可知，最后一个分支结点的序号为$\lfloor 1\,001/2 \rfloor$=500，所以叶子结点的个数为 1 001−500=501。

方法二：

设度为 0 结点（叶子结点）个数为 n_0，度为 1 的结点个数为 n_1，度为 2 的结点个数为 n_2，则 $n_0+n_1+n_2=1\,001$。由二叉树的性质可知，$n_0=n_2+1$，因此，可得 $2n_2+n_1=1\,000$。对于完全二叉树来说，n_1 只有两种可能，n_1 为 0 或 1。又因为 n_2 为整数，所以 n_1 只可能为 0，这样可得，$n_2=500$，$n_0=501$，即有 501 个叶子结点，因此答案选择 D。

4.【答案】C

【考点】二叉树的性质

【解析】

若二叉树每层仅有一个结点，则高度 h 最大，h 为 1 025；若为完全二叉树，则高度 h 最小 $h=\lfloor \log_2 1\,025 \rfloor + 1=11$。因此，$h$ 为 11 至 1 025 之间，答案选择 C。

5.【答案】A

【考点】树的性质

【解析】

深度为 h 的满 m 叉树共有 m^h-1 个结点，第 k 层有 m^{k-1} 个结点。

6.【答案】C

【考点】二叉树的存储结构

【解析】

树的二叉链表存储结构就是指树的孩子兄弟表示法，链表中结点的左指针指向该结点的第一个孩子结点，右指针指向其下一个兄弟结点。而树的根结点没有兄弟结点，故根结点的右指针指向空。所以答案选择 C。

7.【答案】C

【考点】二叉树的遍历

【解析】

按照先左孩子、再右孩子、最后根结点的顺序遍历二叉树可以满足题目要求，即后序遍历二叉树符合题目要求，所以答案选择 C。

8.【答案】B

【考点】树的性质

【解析】

设叶子结点的个数为 n_0。树 T 的结点个数总计为：$n=20+10+1+10+n_0=41+n_0$，而树 T 的分支个数总计为：$B=20\times4+10\times3+1\times2+10\times1=122$。因为 $n=B+1$，所以 $n_0=122+1-41=82$。

9.【答案】D

【考点】树的存储结构

【解析】

树的存储结构有 3 种：双亲表示法、孩子表示法、孩子兄弟表示法，其中孩子兄弟表示法是常用的表示法，任意一棵树都能通过孩子兄弟表示法转换为二叉树进行存储。所以答案选择 D。

10.【答案】C

【考点】二叉树的遍历

【解析】

因为先序遍历顺序是“中左右”，后序遍历顺序是“左右中”，当没有左子树时，先序遍历顺序变为“中右”和后序遍历变为“右中”，符合题意；当没有右子树时，先序遍历顺序变为“中左”和后序遍历变为“左中”，符合题意。只有一个叶子结点说明二叉树没有左子树或者没有右子树，所以答案选择 C。

11.【答案】B

【考点】哈夫曼树的性质

【解析】

在哈夫曼树中没有度为 1 的结点，只有度为 0（叶子结点）和度为 2 的结点。设叶子结点的个数为 n_0，度为 2 的结点的个数为 n_2，由二叉树的性质 $n_0=n_2+1$，则总结点数 $n=n_0+n_2=2\times n_0-1$，得到 $n_0=100$。

12.【答案】C

【考点】线索二叉树

【解析】

线索二叉树利用二叉链表的空链域来存放结点的前驱和后继信息。因为中序遍历的顺序为“左中右”，所以 X 的前驱为 X 左子树中最右的结点。所以答案选择 C。

13.【答案】A

【考点】线索二叉树

【解析】

当以二叉链表作为存储结构时，只能找到结点的左、右孩子信息，而不能直接得到结点在任一序列中的前驱和后继信息，这种信息只有在遍历的动态过程中才能得到，为此引入线索二叉树来保存这些在动态过程中得到的有关前驱和后继的信息。所以答案选择 A。

14.【答案】C

【考点】森林与二叉树的转换

【解析】

计算步骤如下：

（1）若 F 中有 t 个终端结点，则结点总数为：$N=t+n$，左右指针域的个数分别为：$N_{左}=N_{右}=t+n$，其中非空链域个数（树中的分支个数）为：$N_{非空}=N-1=t+n-1$。

（2）森林转换为二叉树的规则为“左孩子右兄弟”，因为有 n 个非终端结点，即有且仅有 n 个结点有孩子，当转化成二叉树后则有 n 个非空的左指针域，记为：$N_{左非空}=n$。

（3）非空的右指针域个数为：$N_{右非空}=N_{非空}-N_{左非空}=t+n-1-n=t-1$。

（4）空的右指针域个数为：$N_{右空}=N_{右}-N_{右非空}=t+n-(t-1)=n+1$。

所以答案选择 C。

15.【答案】A

【考点】哈夫曼树的构造

【解析】

哈夫曼树在构造时，重复在森林 F 中选取两棵根结点的权值最小的树作为左右子树，然后构造一棵新的二叉树，且置新的二叉树的根结点的权值为其左、右子树上根结点的权值之和。因此，选项 B、C 和 D 中的结论是正确的，而哈夫曼树与完全二叉树没有任何关联，选项 A 中的结论是错误的，答案选择选项 A。

16.【答案】D

【考点】二叉树的遍历

【解析】

根据遍历结果，可以看出先访问右子树，然后访问根结点，最后访问左子树，所以为 RNL，答案选择选项 D。

17.【答案】C

【考点】二叉树的性质

【解析】

根据完全二叉树的定义，第六层有叶子结点说明此完全二叉树的高度可能为 6 或者 7，当树的高度为 7 时结点最多。高度为 7 时，第 1 层到第 6 层的结点都是满的，总计有 $2^6-1=63$ 个结点。由已知，第 6 层有 8 个叶子结点，可计算得到第 6 层的非叶子结点个数为 $2^5-8=24$。而每个非叶子

结点最多各有 2 个孩子结点，所以该完全二叉树的结点个数最多为 63+24×2=111。因此答案选择 C。

18.【答案】B

【考点】森林和二叉树的转换。

【解析】

由已知条件中结点 u 是结点 v 的父结点的父结点，可以画出图 5.5 所示的 4 种不同的二叉树的形态。基于森林转换为二叉树的“左孩子右兄弟”这一转换规则，对于这 4 种形态，可以分别对应原来的森林的 4 种不同的形态，如图 5.6 所示。

（1）v 是 u 的第一子树的第一子树的根结点，如图 5.6（a）所示。

（2）v 是 u 的第二子树的根结点，如图 5.6（b）所示，这种情况下 u 和 v 是父子关系。

（3）u 和 v 是兄弟关系，如图 5.6（c）所示，其中，u 为它们共同双亲结点的第一子树的根结点，而 v 是第三子树的根结点。

（4）v 是 u 的兄弟的第一子树的根结点，如图 5.6（d）所示。

在原来的森林中，结点 u 和结点 v 只可能为父子关系或兄弟关系，因此答案选择 B。

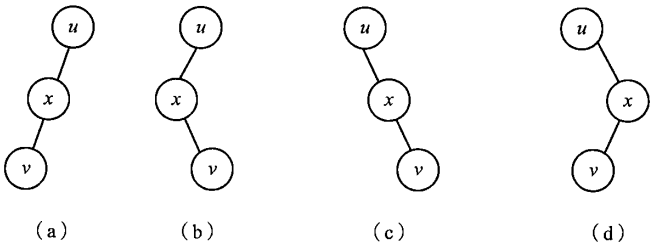


图 5.5　结点 u 是结点 v 的父结点的父结点的二叉树

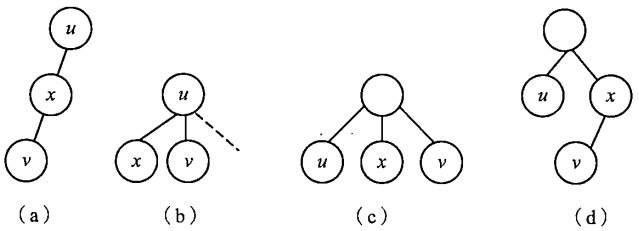


图 5.6　图 5.3 中 4 种不同情况对应的森林

19.【答案】D

【考点】线索二叉树

【解析】

线索二叉树利用二叉链表的空链域来存放结点的前驱和后继信息。题中所给二叉树的后序序列为 d，b，c，a。结点 d 为叶子结点，无前驱结点，后继结点为 b，则左链域空，右链域指针指向其后继结点 b；结点 b 有右子树无左子树，其前驱结点为 d，后继结点为 c，则其左链域指针指向其前驱结点 d；结点 c 无左子树无右子树，其前驱结点为 b，后继结点为 c，则左链域指向其前驱结点 b，右链域指向其后继结点 a。所以答案选择 D。

20.【答案】C

【考点】完全二叉树的性质

【解析】

此题与第 3 题的计算方法相同，按第 3 题给出的两种计算方法均可得到答案 C。

21.【答案】C

【考点】二叉树的遍历

【解析】

由前序遍历序列和后序遍历序列恰好相反可知，不可能存在一个结点同时有左、右孩子，即可断定此二叉树必为单支树，高度为 4。由前序遍历序列可知，1 为根结点，则在中序遍历序列中，1 只能在序列首或序列尾且不能同时有左、右孩子，此时 ABCD 都满足。再考虑以 1 的孩子 2 为根结点的子树，在以 2 为根结点的子树的中序遍历序列中，2 只能在序列首或序列尾且不能同时有左、右孩子，ABD 满足，C 选项不满足，所以答案选择 C。

22.【答案】D

【考点】树与二叉树的转换。

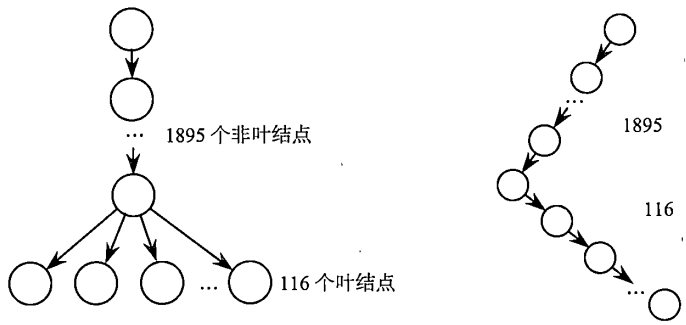
【解析】

方法一：

利用孩子兄弟表示法可以将树转换为二叉树，树转换为二叉树后，树中每一个分支结点的所有子结点中的最右子结点没有右孩子。根结点也无右孩子。所以，对应的二叉树中无右孩子的结点个数为分支结点数+1=2 011−116+1=1 896。

方法二：

可利用特殊值法进行计算。假设题意中的树是图 5.7（a）所示的结构，对应的二叉树中仅有前 115 个叶子结点有右孩子，则该树对应的二叉树中无右孩子的结点个数是：2 011−115=1 896，如图 5.7（b）所示。



（a）转换前　　（b）转换后

图 5.7　树和二叉树的对应关系

23.【答案】A

【考点】二叉树的遍历

【解析】

根据前序遍历序列可知根结点是 a 且 e 为 a 的孩子结点，当两个结点的前序序列为 XY 且后序序列为 YX 时，可知 X 是 Y 的祖先。由 a 的孩子结点的前序序列 e，b，d，c 和后序序列 b，c，d，e 可知，bcd 的祖先是 e，所以根结点 a 的孩子结点只有 e，答案选择 A。

24.【答案】B

【考点】哈夫曼树的构造

【解析】

通常的哈夫曼树是指二叉树，实际上哈夫曼树也可以是 k 叉的，只是在构造 k 叉哈夫曼树时需要先进行一些调整。构造哈夫曼 k 叉树的思想是每次选 k 个权重最小的结点来合成一个新的结点，该结点权重为 k 个结点权重之和。但是当 k 大于 2 时，按照这个步骤做下去可能到最后剩下的结点少于 k，无法继续构造。为解决此问题，可以先计算出 k 叉哈夫曼树的叶子结点数目为 $(k-1)m+1$，其中 m 为分支结点数为 k 的结点个数。然后对给定的 n 个权值构造 k 叉哈夫曼树时，可以先考虑增加一些权值为 0 的叶子结点，使得叶子结点总数满足 $(k-1)m+1$ 这种形式，最后再按照哈夫曼树的方法进行构造即可。

对于题目要求根据给定的 6 个叶子结点构造三叉树，因为不满足 $(k-1)m+1$ 这种形式，需要补上一个权值为 0 的叶子结点，此时结点序列为：0，2，3，4，5，6，7。这样就可利用类似构造二叉哈夫曼树的方法进行构造了，每次选取权值最小的三个结点组成一个新结点，以此类推，直到所有结点都被加入三叉树中，最后构造的三叉哈夫曼树如图 5.8 所示，$WPL=(2+3)\times3+(4+5)\times2+(6+7)\times1=46$。

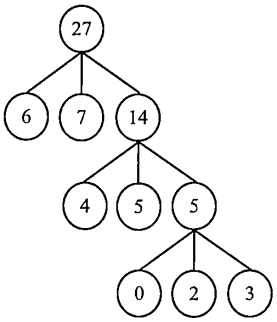


图 5.8　三叉哈夫曼树

25.【答案】A

【考点】线索二叉树

【解析】

根据后序线索二叉树的定义，X 为叶子结点且有左兄弟，可断定 X 为右孩子结点，利用后序遍历的方式可知 X 的父结点是其后序后继结点，所以 X 的右线索指向 X 的父结点，答案选择 A。

26.【答案】D

【考点】线索二叉树

【解析】

题中所给二叉树的中序遍历序列为 e，d，b，x，a，c，因为结点 x 的前驱为 b，后继为 a，所以结点 x 的左、右线索指向的结点分别是 b 和 a，答案选择 D。

27.【答案】C

【考点】森林与二叉树的转换

【解析】

森林与二叉树的转换规则是“左孩子右兄弟”。在转化过程中，原森林某结点的第一个孩子结点作为其左孩子，它的兄弟作为其右孩子。因为森林中的叶子结点没有孩子结点，所以森林中的叶子结点转化为二叉树时没有左孩子，因此，F 中叶子结点的个数就等于 T 中左孩子指针为空的结点个数，答案选择 C。

28.【答案】D

【考点】哈夫曼编码

【解析】

如果在一个编码方案中，任一个编码都不是其他任何编码的前缀（最左子串），则称编码是前缀编码。选项 D 中编码“110”是编码“1100”的前缀，不满足前缀编码的定义，所以选项 D 不是前缀编码。

29.【答案】B

【考点】二叉树的遍历

【解析】

由先序遍历和中序遍历的遍历过程可知，先序遍历和中序遍历相当于以先序序列为入栈次序，以后序序列为出栈次序。又因为由先序序列和中序序列能唯一确定一棵二叉树，所以题目可看作“以序列 a，b，c，d 为入栈次序，则出栈个数是多少？”。计算 n 个数的出栈序列的种数时可以利用卡特兰数的递推公式 $f(n)=C(2n,n)/(n+1)$ 来进行计算（参见第 3 章选择题第 1 题的解析），本题 n 为 4，出栈序列个数为 70/5=14。

30.【答案】D

【考点】哈夫曼树的构造

【解析】

在构造哈夫曼树的过程中，父结点的权值为左、右孩子权值之和。选项 A 中，若两个 10 属于不同的两棵子树，根的权值为 24 与左、右孩子权值之和 20(10+10=20)不相等；若两个 10 属于同一棵子树，其权值 10 与其两个孩子的权值之和 12(5+7=12)不相等，所以 A 选项不正确。同理，选项 B 和 C 也不正确，选项 D 是正确的。

31.【答案】C

【考点】森林与二叉树的转换

【解析】

由森林中树的个数与结点数的关系可以进行推导。

对于一棵树而言，边数=结点数-1。每多一棵树，结点数就少一个。即：

一棵树时，边数=结点数-1

两棵树时，边数=结点数-2

⋮

n 棵树时，边数=结点数-n

因此，根据本题题意，F 包含树的个数 n=25-15=10。

本题也可以利用特殊值法进行求解。当 15 条边全是一棵树的，则该棵树有 16 个结点，剩下 9 个结点都不再形成边，即一个结点算一棵树，这样总计包括 1+9=10 棵树。

二、应用题

1．解答

先序遍历二叉树的顺序是“根—左子树—右子树”，中序遍历的顺序是“左子树—根—右子树”，后序遍历的顺序是“左子树—右子树—根”。

（1）或为空树，或为只有根结点的二叉树。

（2）或为空树，或为任一结点至多只有左子树的二叉树。

（3）或为空树，或为任一结点至多只有右子树的二叉树。

（4）或为空树，或为任一结点至多只有右子树的二叉树。

2．解答

（1）二叉树如图 5.9（a）所示。

（2）二叉树的后序序列 F D B G H E C A，可画出后序线索树如图 5.9（b）所示。

（3）这棵二叉树转换成对应的森林如图 5.9（c）所示。

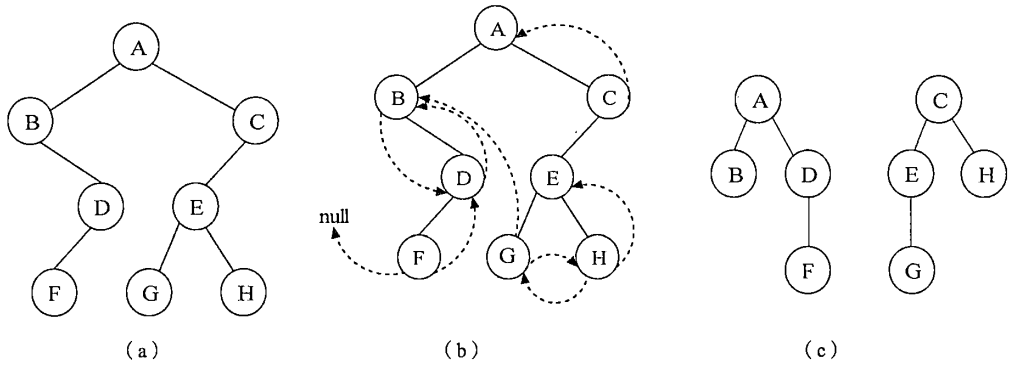


图 5.9　第 2 题对应的二叉树及其森林

3．解答

（1）可以将频率放大 100 倍，以方便进行计算，不影响哈夫曼树的构造。

w={7，19，2，6，32，3，21，10}，根据哈夫曼树的构造规则构造图 5.10 所示的哈夫曼树。8 个字母对应的哈夫曼编码如表 5.1 中的第 3 列所示。

（2）等长编码如表 5.1 中的第 4 列所示。

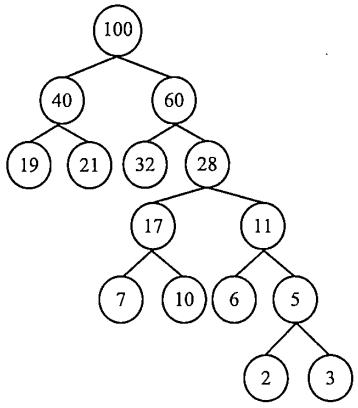


图 5.10　第 3 题对应的哈夫曼树

表 5.1　　哈夫曼编码和等长编码

字母编号	出现频率	哈夫曼编码	等长编码
1	0.07	1100	000
2	0.19	00	001
3	0.02	11110	010
4	0.06	1110	011
5	0.32	10	100
6	0.03	11111	101
7	0.21	01	110
8	0.10	1101	111

（3）对于上述两种方案，等长编码的构造显然比哈夫曼编码的构造简单。但是，哈夫曼编码是最优前缀编码。对于包括 n 个字符的数据文件，分别以它们的出现次数为权值构造哈夫曼树，则利用该树对应的哈夫曼编码对文件进行编码，能使该文件压缩后对应的二进制文件的长度最短。

哈夫曼编码对应二叉树的 *WPL* 为：

$WPL_1=2\times(0.19+0.32+0.21)+4\times(0.07+0.06+0.10)+5\times(0.02+0.03)=1.44+0.92+0.25=2.61$

等长编码对应二叉树的 *WPL* 为：

$WPL_2=3\times(0.19+0.32+0.21+0.07+0.06+0.10+0.02+0.03)=3$

4．解答

哈夫曼树如图 5.11 所示，其存储结构 *HT* 的初始状态如表 5.2（a）所示，其终结状态如表 5.2（b）所示。

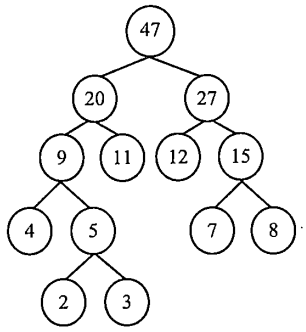


图 5.11　第 4 题对应的哈夫曼树

表 5.2　　例 5.2 的存储结构

（a）*HT* 的初态

结点 *i*	weight	parent	lchild	rchild
1	3	0	0	0
2	12	0	0	0
3	7	0	0	0
4	4	0	0	0
5	2	0	0	0
6	8	0	0	0
7	11	0	0	0
8	—	0	0	0
9	—	0	0	0
10	—	0	0	0
11	—	0	0	0
12	—	0	0	0
13	—	0	0	0

（b）*HT* 的终态

结点 *i*	weight	parent	lchild	rchild
1	3	8	0	0
2	12	12	0	0
3	7	10	0	0
4	4	9	0	0
5	2	8	0	0
6	8	10	0	0
7	11	11	0	0
8	5	9	5	1
9	9	11	4	8
10	15	12	3	6
11	20	13	9	7
12	27	13	2	10
13	47	0	11	12

5．解答

（1）6 个表的合并过程实际上相当于以各有序表的长度为权值，构建一棵哈夫曼树的过程，如图 5.12 所示。

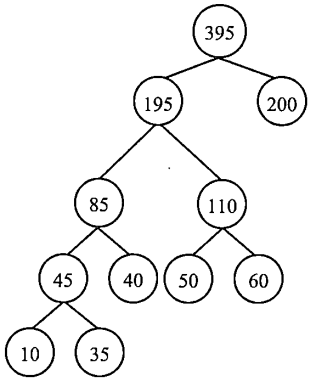


图 5.12　第 5 题对应的哈夫曼树

根据该哈夫曼树，6 个序列的合并过程如下所示。

第 1 次合并：表 A 与表 B 合并，生成含有 45 个元素的表 AB；

第 2 次合并：表 AB 与表 C 合并，生成含有 85 个元素的表 ABC；

第 3 次合并：表 D 与表 E 合并，生成含有 110 个元素的表 DE；

第 4 次合并：表 ABC 与表 DE 合并，生成含有 195 个元素的表 ABCDE；

第 5 次合并：表 ABCDE 与表 F 合并，生成含有 395 个元素的表 ABCDEF。

由于合并两个长度分别为 m 和 n 的有序表，最坏情况下需要比较 $m+n-1$ 次，故最坏情况下比较的总次数计算如下所示。

第 1 次合并：最多比较次数=10+35–1=44；

第 2 次合并：最多比较次数=45+40–1=84；

第 3 次合并：最多比较次数=50+60–1=109；

第 4 次合并：最多比较次数=85+110–1=194；

第 5 次合并：最多比较次数=195+200–1=394；

比较的总次数最多为：44+84+109+194+394=825。

（2）各表的合并策略：在对多个有序表进行两两合并时，若表长不同，则最坏情况下总的比较次数依赖于表的合并次序，可以借用哈夫曼树的构造思想，依次选择最短的两个表进行合并，这样可以获得最坏情况下最佳的合并效率。

6．解答

（1）根据定义，正则 k 叉树中仅含有两类结点：叶子结点（个数记为 n_0）和度为 k 的分支结点（个数记为 n_k）。树 T 中的结点总数 $n=n_0+n_k=n_0+m$。树中所含的边数 $e=n-1$，这些边均为 m 个度为 k 的结点发出的，即 $e=m\times k$。整理得：$n_0+m=m\times k+1$，故 $n_0=(k-1)\times m+1$。

（2）高度为 h 的正则 k 叉树 T 中，含最多结点的树形为：除第 h 层外，第 1 到第 $h-1$ 层的结点都是度为 k 的分支结点，而第 h 层均为叶子结点，即树是“满”树。此时第 $j(1\leqslant j\leqslant h)$层结点数为 k^{j-1}，结点总数 M_1 为：

$$M_1=\sum_{j-1}^{h}k^{j-1}=\frac{k^h-1}{k-1}$$

含最少结点的正则 k 叉树的树形为：第 1 层只有根结点，第 2 到第 $h-1$ 层仅含 1 个分支结点

和 k–1 个叶子结点，第 h 层有 k 个叶子结点。即除根外第 2 到第 h 层中每层的结点数均为 k，故 T 中所含结点总数 M_2 为：

$$M_2=1+(h-1)\times k$$

三、算法设计题

1．解答

【算法思想】

利用递归实现，递归结束的条件有两个：一个是二叉树为空，另一个是遇到叶子结点。当二叉树为空时，递归结束返回 0；当二叉树不为空且左、右子树为空（叶子结点）时，递归结束返回 1；当二叉树不为空，且左、右子树不同时为空（非叶子结点），递归调用统计函数，返回左子树中叶子结点个数加上右子树中叶子结点个数。

【算法描述】

```
int LeafNode(BiTree T)
{//统计二叉树 T 中叶子结点的个数
  if(T==NULL)
    return 0;                          //空树，返回 0
  else if(T->lchild==NULL&&T->rchild==NULL)
    return 1;                          //叶子结点，返回 1
  else                                 //递归
    return LeafNode(T->lchild)+LeafNode(T->rchild);
}
```

2．解答

【算法思想】

利用递归实现，递归结束的条件有 3 个：一是两棵树均为空，返回相等；二是只有一棵树为空，返回不等；三是根结点的数据域不等，返回不等。其他情况下则递归判断相应的孩子结点是否相等。

【算法描述】

```
int CmpTree(BiTree T1,BiTree T2)
{//判别两棵二叉树是否相等
  if(T1==NULL&&T2==NULL)
    return 1;                          //都是 NULL，相等
  else if(T1==NULL||T2==NULL)
    return 0;                          //只有一个为 NULL，不等
  if(T1->data!=T2->data)
    return 0;                          //根结点不相等，直接返回不等，否则递归
  left=right=0;
  left=CmpTree(T1->lchild,T2->lchild);
  right=CmpTree(T1->rchild,T2->rchild);
  return left&&right;
}
```

3．解答

【算法思想】

如果某结点的左、右子树均为空，则返回，否则交换该结点的左、右孩子结点，然后递归交换左、右子树。

【算法描述】

```
void ChangeLR(BiTree &T)
{//交换二叉树每个结点的左孩子和右孩子
  if(T==NULL) return;
  if(T->lchild==NULL&&T->rchild==NULL)
    return;                                  //左右子树有一个为空，返回
  else //交换左、右孩子结点
  {
    temp=T->lchild;
    T->lchild=T->rchild;
    T->rchild=temp;
  }
  ChangeLR(T->lchild);                       //递归交换左子树
  ChangeLR(T->rchild);                       //递归交换右子树
}
```

4. 解答

【算法思想】

双序遍历与先序遍历相比，只是多了一次访问，即在递归遍历其左子树后再访问一次该结点。因此，步骤如下：

若二叉树为空，则空操作；否则：

（1）访问根结点；

（2）先序遍历左子树；

（3）访问根结点；

（4）先序遍历右子树。

【算法描述】

```
void DoubleTraverse(BiTree T)
{//双序遍历二叉树 T 的递归算法
  if(T)//若二叉树非空
  {
    cout<<T->data;                           //访问根结点
    DoubleTraverse(T->lchild);               //双序遍历左子树
    cout<<T->data;                           //访问根结点
    DoubleTraverse(T->rchild);               //双序遍历右子树
  }
}
```

5. 解答

【算法思想】

计算二叉树最大的宽度可采用层次遍历的方法，利用队列来实现。首先判断是否为空树，如果为空树则宽度为 0；不为空则分别记录局部的宽度和当前的最大宽度，逐层遍历结点，如果结点有孩子结点，则将其孩子加入队尾；每层遍历完毕之后，若局部宽度大于当前的最大宽度，则修改最大宽度。

【算法描述】

```
int Width(BiTree T)
{//求二叉树 T 的最大宽度
  if(T==NULL)
    return 0;                                //空二叉树宽度为 0
  else
```

```
    {
      BiTree Q[];                          //Q 是队列，元素为二叉树结点指针，容量足够大
      front=1;                             //队头指针
      rear=1;                              //队尾指针
      last=1;                              //同层最右结点在队列中的位置
      temp=0;                              //局部宽度
      maxw=0;                              //最大宽度
      Q[rear]=T;                           //根结点入队
      while(front<=last)
      {
        p=Q[front++];
        temp++;                            //同层元素数加 1
        if(p->lchild!=NULL)
          Q[++rear]=p->lchild;             //左子女入队
        if(p->rchild!=NULL)
          Q[++rear]=p->rchild;             //右子女入队
        if(front>last)                     //一层结束
        {
          last=rear;                       //last 指向下层最右元素
          if(temp>maxw) maxw=temp;         //更新当前最大宽度
             temp=0;
        }                                  //if
      }                                    //while
      return maxw;
    }
}
```

6. 解答

【算法思想】

可以借助队列实现层次顺序遍历二叉树。从根结点开始，首先将结点进队，然后依次访问队列中的结点，访问之后将其出队。若该结点左子树为空、右子树非空或者右子树为空、左子树非空，则该结点为度为 1 的结点，计数加 1，并将该结点的非空孩子结点入队，直到队列为空则统计结束。

【算法描述】

```
int Level(BiTree T)
{//层次遍历二叉树，并统计度为 1 的结点的个数
  num=0;                                   //num 为 1 的结点的个数
  if(T)
  {
    InitQueue(Q);                          //Q 是以二叉树结点指针为元素的队列,初始化为空
    EnQueue(Q,T);                          //根结点进队
    while(!QueueEmpty(Q))
    {
      DeQueue(Q,p);                        //p 出队,访问结点
      cout<<p->data;                       //访问出队的结点
      if((p->lchild&&!p->rchild)||(!p->lchild&&p->rchild))
        num++;//度为 1 的结点，计数加 1
      if(p->lchild) EnQueue(Q,p->lchild); //非空左子女入队
      if(p->rchild) EnQueue(Q,p->rchild); //非空右子女入队
    }
```

```
    }
    return num;
}
```

7. 解答

【算法思想】

本题可以借助栈来实现。因为在后序遍历栈中可以保留当前结点的祖先信息，可设一变量longest保存栈的最高栈顶指针。每当退栈时，栈顶指针高于longest的值时，则将该栈存入辅助栈中。辅助栈始终保存最长路径长度上的结点，直至后序遍历完毕，则辅助栈中的内容即为最长的路径长度。

【算法描述】

```
void LongestPath(BiTree T)
{//求二叉树中的第一条最的长路径长度
  BiTree p=T,l[],s[];    //l,s是栈，元素是二叉树结点的指针，l中保留当前最长路径中的结点
  top=0;longest=0;
  while(p||top>0)
  {
    while(p)                              //沿左分枝向下
    {
      s[++top]=p;
      tag[top]=0;
      p=p->Lc;
    }
    if(tag[top]==1)                       //当前结点的右分枝已遍历
    {
      if(!s[top]->Lc&&!s[top]->Rc)        //只有到叶子结点时，才查看路径长度
        if(top>longest)
        {
          for(i=1;i<=top;i++) l[i]=s[i];
          longest=top;
        }
        top--;                            //保留当前最长路径到l栈，记住最高栈顶指针，退栈
    }
    else if(top>0)                        //沿右子分枝向下
    {
        tag[top]=1;
        p=s[top]->Rc;
    }
  }
}
```

8. 解答

【算法思想】

可采用递归的方法进行先序遍历，设数组path用来保存遍历时经过的路径。从根结点开始向下先序遍历二叉树，当遇到叶子结点*T时，由于叶子结点*T尚未添加到path中，因此先输出T->data的值，然后再依次输出保存在path中的结点。当遇到非叶子结点时，则将其加入path中，然后递归遍历左、右子树。

【算法描述】

```
void AllPath(BiTree T,ElemType path[],int pathlen)
{//输出二叉树中从每个叶子结点到根结点的路径，pathlen初始为0
```

```
    if(T!=NULL)
    {
      if(T->lchild==NULL&&T->rchild==NULL)          //叶子结点
      {
        cout<<" "<<T->data<<"到根结点路径:"<<T->data;
        for(i=pathlen-1;i>=0;i--)
          cout<<path[i]<<endl;
      }
      else
      {
        path[pathlen]=T->data;                      //将当前结点放入路径中
        pathlen++;                                  //路径长度增1
        AllPath(T->lchild,path,pathlen);            //递归遍历左子树
        AllPath(T->rchild,path,pathlen);            //递归遍历右子树
        pathlen--;                                  //恢复环境
      }
    }
}
```

9. 解答

(1)【算法思想】

利用递归的算法可以求解 *WPL*，已知：

WPL=树中所有叶子结点的带权路径长度之和=左子树中所有叶子结点的带权路径长度之和+右子树中所有叶子结点的带权路径长度之和。

叶子结点的带权路径长度=该结点的 weight 域的值×该结点的深度。

设根结点的深度为 0，若某结点的深度为 d 时，则其孩子结点的深度为 d+1。

算法的具体递归思路如下：

① 如果是空树，则递归结束，返回 0；

② 如果是叶子结点，则返回 weight×d；

③ 如果是其他情况，则返回左子树中所有叶子结点的带权路径长度之和+右子树中所有叶子结点的带权路径长度之和。

(2) 二叉树结点的数据类型定义如下：

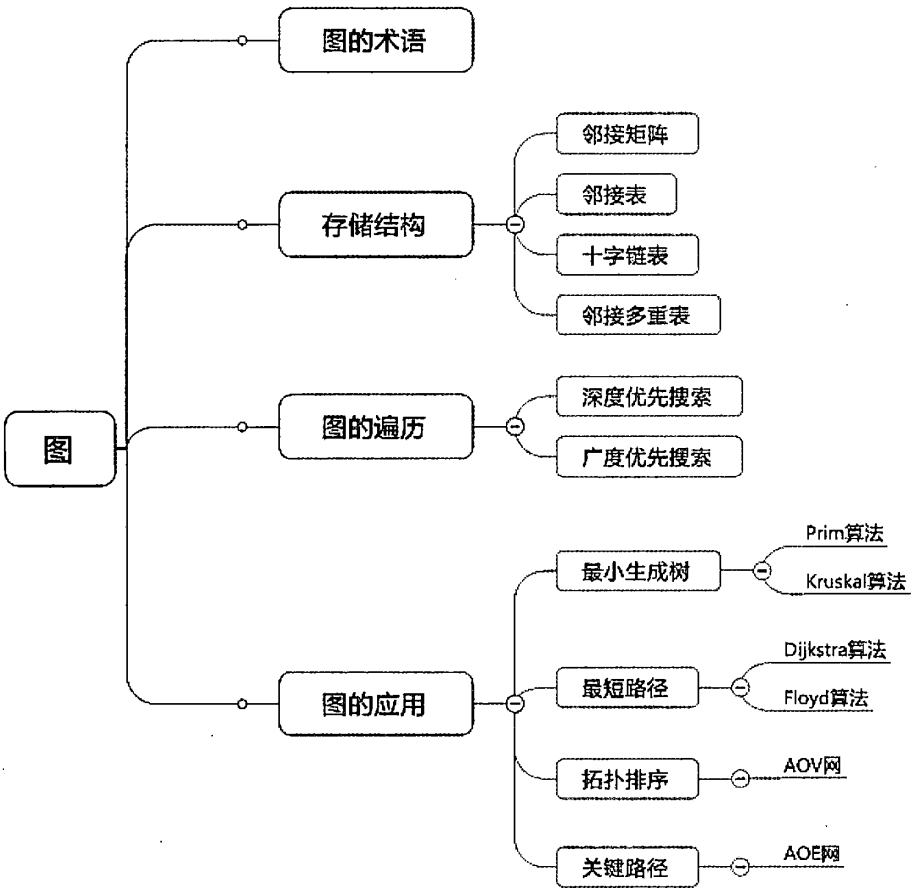
```
typedef struct BiTNode
{
  int weight;
  struct BiTNode *left,*right;
}BiTNode,*BiTree;
```

(3)【算法描述】

```
int WPL(BiTree &T,int d)
{//求二叉树T的带权路径长度
 //调用时T指向二叉树的根结点，d为0
  if(T==NULL)
    return 0;                          //空树，递归结束，返回0
  if(T->left==NULL&&T->right==NULL)
    return(T->weight*d);               //叶子结点，则返回weight*d
  else                                 //递归
    return(WPL(T->left,d+1)+WPL(T->right,d+1));
}
```

第6章 图

【知识导图】



【学习目标】

1. 根据不同的分类规则，图分为多种类型：无向图、有向图、完全图、连通图、强连通图、带权图（网）、稀疏图和稠密图等。邻接点、路径、回路、度、连通分量、生成树等是在图的算法设计中常用的重要术语，掌握图的基本术语，掌握与这些术语相联系的相关计算题。

2. 图的存储方式有两大类：以边集合方式的表示法和以链接方式的表示法。其中，以边集合方式表示的为邻接矩阵，以链接方式表示的包括邻接表、十字链表和邻接多重表。邻接矩阵表示

法借助二维数组来表示元素之间的关系，实现起来较为简单；邻接表、十字链表和邻接多重表都属于链式存储结构，实现起来较为复杂。在实际应用中具体采取哪种存储表示，可以根据图的类型和实际算法的基本思想进行选择。其中，邻接矩阵和邻接表是两种常用的存储结构，重点掌握这两种存储方法。掌握不同存储结构的特点和适用场合。

3. 图的遍历算法是实现图的其他运算的基础。图的遍历方法有两种：深度优先搜索遍历和广度优先搜索遍历。深度优先搜索遍历类似于树的先序遍历，借助栈结构来实现（递归）；广度优先搜索遍历类似于树的层次遍历，借助队列结构来实现。掌握图的两种遍历算法的实现以及两种遍历算法的应用。

4. 图的很多算法与实际应用密切相关，比较常用的算法包括最小生成树算法、最短路径算法、拓扑排序和求解关键路径算法。

（1）最小生成树算法：连通图的最小生成树是不唯一的，但最小生成树的权值之和是唯一的。构造最小生成树有 Prim 算法和 Kruskal 算法，两者都能达到同一目的。但前者算法思想的核心是归并点，时间复杂度是 $O(n^2)$，适用于稠密网；后者是归并边，时间复杂度是 $O(e\log_2 e)$，适用于稀疏网。掌握两种构造方法的特点和适用场合，掌握根据这两种算法思想用图示法求出给定网的一棵最小生成树的过程。

（2）最短路径算法：一种是迪杰斯特拉算法，求从某个源点到其余各顶点的最短路径，求解过程是按路径长度递增的次序产生最短路径，时间复杂度是 $O(n^2)$；另一种是弗洛伊德算法，求每一对顶点之间的最短路径，时间复杂度是 $O(n^3)$，从实现形式上来说，这种算法比以图中的每个顶点为源点 n 次调用迪杰斯特拉算法更为简洁。掌握两种方法的实现过程，并能手工模拟实现其求解过程，尤其是迪杰斯特拉算法。

（3）拓扑排序和关键路径都是有向无环图的应用。拓扑排序是基于用顶点表示活动的有向图，即 AOV-网。对于不存在环的有向图，图中所有顶点一定能够排成一个线性序列，即拓扑序列，拓扑序列是不唯一的。掌握拓扑序列的求解过程。

（4）关键路径算法是基于用弧表示活动的有向图，即 AOE-网。关键路径上的活动叫作关键活动，这些活动是影响工程进度的关键，它们的提前或拖延将使整个工程提前或拖延。关键路径是不唯一的，关键路径算法是在拓扑排序的基础上实现的，掌握关键路径的求解过程。

6.1 习题

一. 单项选择题

1. 在一个图中，所有顶点的度数之和等于图的边数的（　　）倍。

A. 1/2　　B. 1　　C. 2　　D. 4

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

A. 1/2　　B. 1　　C. 2　　D. 4

3. 具有 n 个顶点的有向图最多有（　　）条边。

A. n　　B. $n(n-1)$　　C. $n(n+1)$　　D. n^2

4. n 个顶点的连通图用邻接矩阵表示时，该矩阵至少有（　　）个非零元素。

A. n　　B. $2(n-1)$　　C. $n/2$　　D. n^2

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

A. 7　　B. 8　　C. 9　　D. 10

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

A. 非连通　　B. 连通　　C. 强连通　　D. 有向

7. 下面（　　）适合构造一个稠密图 G 的最小生成树。

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

8. 用邻接表表示图进行广度优先遍历时，通常借助（　　）来实现算法。

A. 栈　　B. 队列　　C. 树　　D. 图

9. 用邻接表表示图进行深度优先遍历时，通常借助（　　）来实现算法。

A. 栈　　B. 队列　　C. 树　　D. 图

10. 深度优先遍历类似于二叉树的（　　）。

A. 先序遍历　　B. 中序遍历　　C. 后序遍历　　D. 层次遍历

11. 广度优先遍历类似于二叉树的（　　）。

A. 先序遍历　　B. 中序遍历　　C. 后序遍历　　D. 层次遍历

12. 图的 BFS 生成树的树高比 DFS 生成树的树高（　　）。

A. 小　　B. 相等　　C. 小或相等　　D. 大或相等

13. 已知图的邻接矩阵如图 6.1 所示，则从顶点 v_0 出发按深度优先遍历的结果是（　　）。

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

图 6.1　第 13 题邻接矩阵

A. 0243156　　B. 0136542　　C. 0134256　　D. 0361542

14. 已知图的邻接表如图 6.2 所示，则从顶点 v_0 出发按广度优先遍历的结果是（　　），按深度优先遍历的结果是（　　）。

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

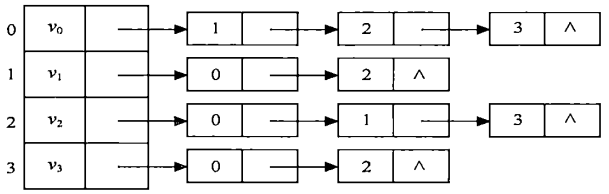


图 6.2　第 14 题邻接表

15. 下面（　　）方法可以判断出一个有向图是否有环。

A. 广度优先遍历　　B. 拓扑排序　　C. 求最短路径　　D. 求关键路径

16.【2009 年第 7 题】下列关于无向连通图特性的叙述中，正确的是（　　）。

I. 所有顶点的度之和为偶数。

II. 边数大于顶点个数减1。

III. 至少有一个顶点的度为1。

A. 只有I　　B. 只有II　　C. I和II　　D. I和III

17.【2010年第7题】若无向图$G=(V,E)$中含7个顶点，要保证图G在任何情况下都是连通的，则需要的边数最少是（　　）。

A. 6　　B. 15　　C. 16　　D. 21

18.【2010年第8题】对图6.3所示的有向图进行拓扑排序，可以得到不同的拓扑序列的个数是（　　）。

A. 4　　B. 3　　C. 2　　D. 1

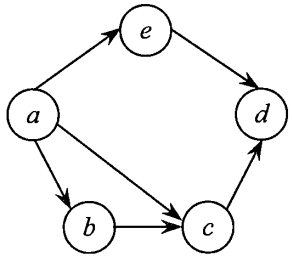


图6.3　第18题有向图

19.【2011年第8题】下列关于图的叙述中，正确的是（　　）。

I. 回路是简单路径。

II. 存储稀疏图，用邻接矩阵比邻接表更省空间。

III. 若有向图中存在拓扑序列，则该图不存在回路。

A. 仅II　　B. 仅I、II　　C. 仅III　　D. 仅I、III

20.【2012年第5题】对于有n个顶点、e条边且使用邻接表存储的有向图进行广度优先遍历，其算法的时间复杂度是（　　）。

A. O(n)　　B. O(e)　　C. O($n+e$)　　D. O($n\times e$)

21.【2012年第6题】若用邻接矩阵存储有向图，矩阵中主对角线以下的元素均为零，则关于该图拓扑序列的结论是（　　）。

A. 存在，且唯一　　B. 存在，且不唯一

C. 存在，可能不唯一　　D. 无法确定是否存在

22.【2012年第7题】对图6.4的有向带权图，若采用迪杰斯特拉（Dijkstra）算法求源点a到其他各顶点的最短路径，则得到的第一条最短路径的目标顶点是b，第二条最短路径的目标顶点是c，后序得到的其余各最短路径的目标顶点依次是（　　）。

A. d, e, f　　B. e, d, f　　C. f, d, e　　D. f, e, d

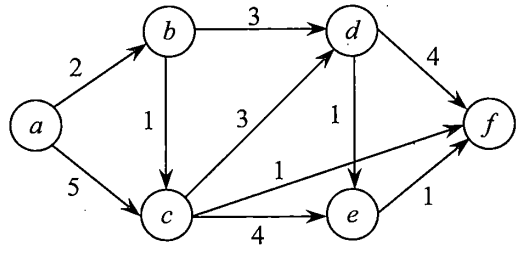


图6.4　第22题有向带权图

23.【2012 年第 8 题】下列关于最小生成树的说法中，正确的是（　　）。

I. 最小生成树的代价唯一。

II. 权值最小的边一定会出现在所有的最小生成树中。

III. 用普里姆（Prim）算法从不同顶点开始得到的最小生成树一定相同。

IV. 使用普里姆和克鲁斯卡尔（Kruskal）算法得到的最小生成树总不相同。

A. 仅 I　　B. 仅 II　　C. 仅 I、III　　D. 仅 II、IV

24.【2013 年第 7 题】设图的邻接矩阵 A 如图 6.5 所示。各顶点的度依次是（　　）。

A. 1, 2, 1, 2　　B. 2, 2, 1, 1

C. 3, 4, 2, 3　　D. 4, 4, 2, 2

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

图 6.5　第 24 题邻接矩阵

25.【2013 年第 8 题】若对图 6.6 所示的无向图进行遍历，则下列选项中，不是广度优先遍历序列的是（　　）。

A. h, c, a, b, d, e, g, f　　B. e, a, f, g, b, h, c, d

C. d, b, c, a, h, e, f, g　　D. a, b, c, d, h, e, f, g

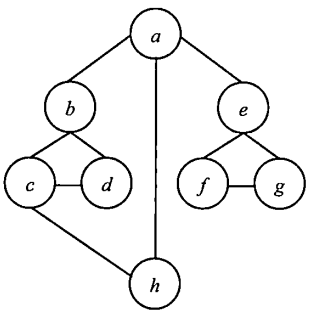


图 6.6　第 25 题无向图

26.【2013 年第 9 题】图 6.7 所示的 AOE 网表示一项包含 8 个活动的工程。通过同时加快若干活动的进度可以缩短整个工程的工期。下列选项中，加快其进度就可以缩短工程工期的是（　　）。

A. c 和 e　　B. d 和 e　　C. f 和 d　　D. f 和 h

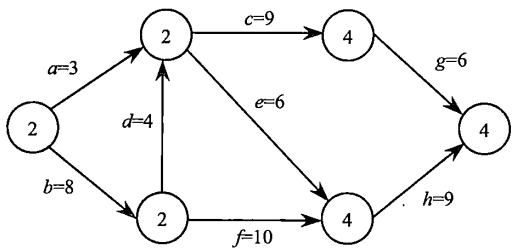


图 6.7　AOE 网

27.【2014年第7题】对如图6.8所示的有向图进行拓扑排序，得到的拓扑序列可能是（　　）。

A. 3，1，2，4，5，6　　B. 3，1，2，4，6，5

C. 3，1，4，2，5，6　　D. 3，1，4，2，6，5

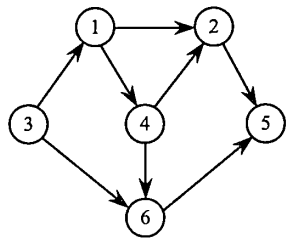


图6.8　第27题有向图

28.【2015年第5题】设有向图 $G=(V,E)$，顶点集 $V=\{V_0,V_1,V_2,V_3\}$ E=$\{<V_0,V_1>,<V_0,V_2>,<V_0,V_3>,<V_1,V_3>\}$，若从顶点 V_0 开始对图进行深度优先遍历，则可能得到的不同遍历序列个数是（　　）。

A. 2　　B. 3　　C. 4　　D. 5

29.【2015年第6题】求图6.9所示带权图的最小（代价）生成树时，可能是克鲁斯卡（Kruskal）算法第二次选中但不是普里姆（Prim）算法（从 V_4 开始）第2次选中的边是（　　）。

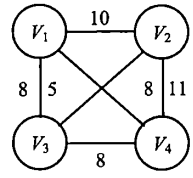


图6.9　第29题有向图

A. (V_1,V_3)　　B. (V_1,V_4)　　C. (V_2,V_3)　　D. (V_3,V_4)

30.【2016年第6题】下列选项中，不是如图6.10所示图的深度优先搜索序列的是（　　）。

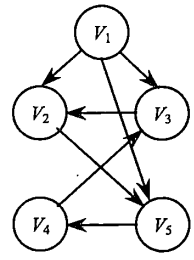


图6.10　第30题有向图

A. v_1，v_5，v_4，v_3，v_2　B. v_1，v_3，v_2，v_5，v_4　C. v_1，v_2，v_5，v_4，v_3　D. v_1，v_2，v_3，v_4，v_5

31.【2016年第7题】对于有 n 个顶点 e 条边的有向图采用邻接表存储，则拓扑排序算法的时间复杂度是（　　）。

A. $O(n)$　　B. $O(n+e)$　　C. $O(n^2)$　　D. $O(n\times e)$

32.【2016年第8题】使用迪杰斯特拉（Dijkstra）算法求图6.11中从顶点1到其他各顶点的最短路径，依次得到的各最短路径的目标顶点是（　　）。

A. 5，2，3，4，6　B. 5，2，3，6，4　C. 5，2，4，3，6　D. 5，2，6，3，4

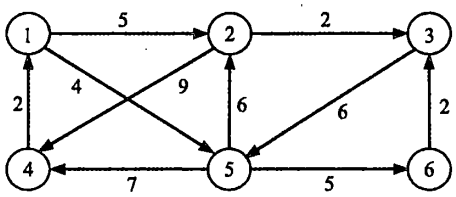


图 6.11　第 32 题有向图

二. 应用题

1. 已知图 6.12 所示的有向图，请给出：

（1）每个顶点的入度和出度；

（2）邻接矩阵；

（3）邻接表；

（4）逆邻接表。

2. 已知图 6.13 所示的无向网，请给出：

（1）邻接矩阵；

（2）邻接表；

（3）最小生成树。

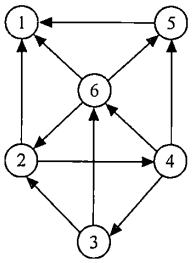


图 6.12　应用题第 1 题有向图

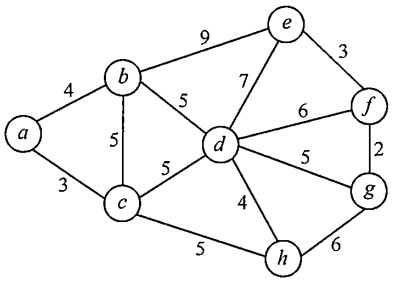


图 6.13　应用题第 2 题无向网

3. 已知图的邻接矩阵如图 6.14 所示。试分别画出自顶点 1 出发进行遍历所得的深度优先生成树和广度优先生成树。

4. 有向网如图 6.15 所示，试用迪杰斯特拉算法求出从顶点 a 到其他各顶点间的最短路径，完成表 6.1。

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

图 6.14　应用题第 3 题邻接矩阵

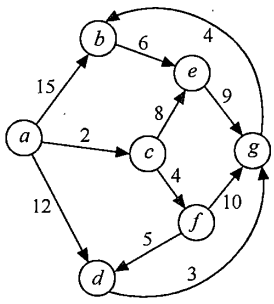


图 6.15　应用题第 4 题有向网

表 6.1　　最短路径的求解

终点 \ d	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
b	15 (a,b)					
c	**2** **(a,c)**					
d	12 (a,d)					
e	∞					
f	∞					
g	∞					
S 终点集	$\{a,c\}$					

5. 试对图 6.16 所示的 AOE-网：

（1）求这个工程最早可能在什么时间结束；

（2）求每个活动的最早开始时间和最迟开始时间；

（3）确定哪些活动是关键活动。

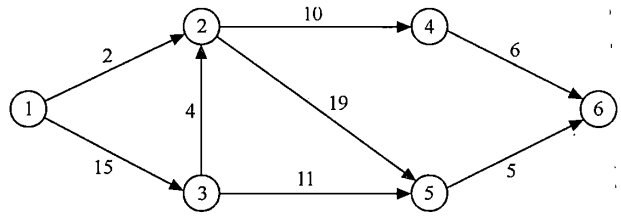


图 6.16　应用题第 5 题 AOE-网

6.【2015 年第 42 题】已知有 5 个顶点的图 G 如图 6.17 所示。

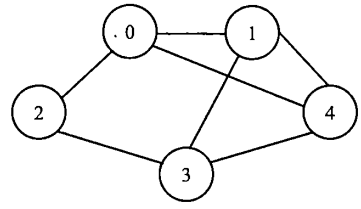


图 6.17　应用题第 6 题无向图

请回答下列问题。

（1）写出图 G 的邻接矩阵 $\boldsymbol{A}$（行、列下标从 0 开始）。

（2）求 $\boldsymbol{A}^2$，矩阵 $\boldsymbol{A}^2$ 中位于 0 行 3 列元素值的含义是什么？

（3）若已知具有 $n(n\geqslant 2)$个顶点的邻接矩阵为 $\boldsymbol{B}$，则 $\boldsymbol{B}^m(2\leqslant m\leqslant n)$非零元素的含义是什么？

7.【2011 年第 41 题】已知有 6 个顶点（顶点编号为 0～5）的有向带权图 G，其邻接矩阵 A 为上三角矩阵，按行为主序（行优先）保存在如下的一维数组中。

4	6	∞	∞	∞	5	∞	∞	∞	4	3	∞	∞	3	3

要求：

（1）写出图 G 的邻接矩阵 A；

（2）画出有向带权图 G；

（3）求图 G 的关键路径，并计算该关键路径的长度。

8.【2009 年第 41 题】带权图（权值非负，表示边连接的两顶点间的距离）的最短路径问题是找出从初始顶点到目标顶点之间的一条最短路径。假定从初始顶点到目标顶点之间存在路径，现有一种解决该问题的方法：

（1）设最短路径初始时仅包含初始顶点，令当前顶点 u 为初始顶点；

（2）选择离 u 最近且尚未在最短路径中的一个顶点 v，加入到最短路径中，修改当前顶点 $u=v$；

（3）重复步骤②，直到 u 是目标顶点时为止。

请问上述方法能否求得最短路径？若该方法可行，请证明之；否则，请举例说明。

三．算法设计题

1．分别以邻接矩阵和邻接表作为存储结构，实现以下图的基本操作：

（1）增加一个新顶点 v，InsertVex(G, v)；

（2）删除顶点 v 及其相关的边，DeleteVex(G, v)；

（3）增加一条边<v, w>，InsertArc(G, v, w)；

（4）删除一条边<v, w>，DeleteArc(G, v, w)。

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

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

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

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

6.2 答案及解析

一. 单项选择题

1. C	2. B	3. B	4. B	5. C	6. B	7. A	8. B
9. A	10. A	11. D	12. C	13. C	14. D	15. B	16. A
17. C	18. B	19. C	20. C	21. C	22. C	23. A	24. C
25. D	26. C	27. D	28. D	29. C	30. D	31.B	32.B

1.【答案】C

【考点】图的基本术语

【解析】

一般地，如果顶点 v_i 的度记为 $TD(v_i)$，那么一个有 n 个顶点、e 条边的图，满足如下关系 $e=\frac{1}{2}\sum_{i=1}^{n}TD(v_i)$，因此所有顶点的度数之和等于图的边数的 2 倍。

2.【答案】B

【考点】图的基本术语

【解析】

有向图所有顶点入度之和等于所有顶点出度之和。

3.【答案】B

【考点】图的基本术语

【解析】

对于有 n 个顶点有向图，边数最多的情况是：任意一个点到其他 n–1 个点都有一条有向边，即最多有 $n(n-1)$条边。也可以换个角度考虑，因为有向图的边有方向之分，最多的边数即为从 n 个顶点中选取 2 个顶点的有序排列，结果为 $n(n-1)$。

4.【答案】B

【考点】邻接矩阵

【解析】

n 个顶点的连通图至少需要 n–1 条边，由于无向图的每条边同时关联两个顶点，因此邻接矩阵中每条边被存储了两次，即矩阵中至少有 $2(n-1)$个非零元素。

5.【答案】C

【考点】图的基本术语

【解析】

8 个顶点的连通无向图最多有 8×7/2=28 条边，再添加一个点即构成非连通无向图，故该图至少有 9 个顶点。

6.【答案】B

【考点】深度优先搜索

【解析】

即从该无向图任意一个顶点出发有到其他各个顶点的路径，所以该图一定是连通图。

7.【答案】A

【考点】最小生成树

【解析】

四个选项中，选项 C 和 D 是用来求解最短路径的算法。选项 A 和 B 都是用来构造最小生成树的算法。其中，Prim 算法通过不断归并点来完成的，也称为“加点法”，时间复杂度为 $O(n^2)$，与图中的边数无关，因此适用于求稠密图的最小生成树；而 Kruskal 算法通过不断归并边来完成的，也称为“加边法”，时间复杂度为 $O(e\log_2 e)$，与图中的边数有关，因此适用于求稀疏图的最小生成树。

记忆小窍门：Prim 人名短，通过归并点来完成，Kruskal 人名长，通过归并边来完成。

8.【答案】B

【考点】广度优先搜索

【解析】

广度优先搜索遍历的特点是：尽可能先对横向进行搜索。从一个顶点出发，开始访问该顶点的所有邻接点，然后再依次向下，一层一层的访问，即先访问的顶点其邻接点亦先被访问。为此，算法实现时，不论采用邻接矩阵还是邻接表表示图，广度优先搜索都必须引进队列保存已被访问过的顶点。

9.【答案】A

【考点】深度优先搜索

【解析】

深度优先搜索遍历的特点是：尽可能先对纵向进行搜索。从顶点的第一个邻接点一直访问下去再访问顶点的第二个邻接点，即先访问的顶点其邻接点会后被访问，其遍历过程是一个递归的过程。在算法实现时，不论采用邻接矩阵还是邻接表表示图，深度优先搜索都可以借助栈来实现。

10.【答案】A

【考点】深度优先搜索

【解析】

深度优先搜索遍历类似于树的先序遍历，是树的先序遍历的推广。

11.【答案】D

【考点】广度优先搜索

【解析】

广度优先搜索遍历类似于树的层次遍历，是树的层次遍历的推广。

12.【答案】C

【考点】图的遍历

【解析】

对于一些特殊的图，比如只有一个顶点的图，其 BFS 生成树的树高和 DFS 生成树的树高相等。而对于一般的图，根据图的 BFS 生成树和 DFS 生成树的构造过程，BFS 生成树的树高比 DFS 生成树的树高小。

13.【答案】C

【考点】深度优先搜索

【解析】

深度优先搜索遍历的过程如下。

（1）从顶点 v_0 出发，访问 v_0。

（2）在访问了顶点 v_0 之后，选择 v_0 第一个未被访问的邻接点 v_1，访问 v_1。以 v_1 为新顶点，重复此步，访问 v_3、v_4、v_2。在访问了 v_2 之后，由于 v_2 的邻接点都已被访问，此步结束。

（3）搜索从 v_2 回到 v_4，此时由于 v_4 的另一个邻接点 v_5 未被访问，访问 v_5，再继续进行下去访问 v_5 未被访问的邻接点 v_6。直至图中所有顶点都被访问过，搜索结束。

由此，得到的顶点访问序列为：

$$v_0,\ v_1,\ v_3,\ v_4,\ v_2,\ v_5,\ v_6$$

14.【答案】D

【考点】广度优先搜索和深度优先搜索

【解析】

广度优先搜索遍历的过程为如下。

（1）从顶点 v_0 出发，访问 v_0。

（2）依次访问 v_0 的各个未曾访问过的邻接点 v_1、v_2 和 v_3。

（3）依次访问 v_1 的邻接点 v_0 和 v_2，以及 v_2 的邻接点 v_0、v_1 和 v_3，最后访问 v_3 的邻接点 v_0 和 v_2。由于这些顶点的邻接点均已被访问，并且图中所有顶点都被访问，完成了图的遍历。

由此得到的顶点访问序列为：

$$v_0,\ v_1,\ v_2,\ v_3$$

深度优先搜索遍历的过程为如下。

（1）从顶点 v_0 出发，访问 v_0。

（2）在访问了顶点 v_0 之后，选择第一个未被访问的邻接点 v_1，访问 v_1。以 v_1 为新顶点，重复此步，访问 v_2、v_3。在访问了 v_3 之后，由于 v_3 的邻接点都已被访问，此步结束。

（3）搜索从 v_3 回到 v_2，再依次回到 v_1 和 v_0，图中所有顶点都被访问过，搜索结束。由此，得到的顶点访问序列为：

$$v_0,\ v_1,\ v_2,\ v_3$$

注意：虽然本题答案中两处均为选项 D，但遍历过程是截然不同的。

15.【答案】B

【考点】拓扑排序

【解析】

拓扑排序是通过在有向图中反复选择无前驱的顶点来完成的，对于有环的图进行拓扑排序时，最后是无法找到一个无前驱的顶点。因此，如果能够用拓扑排序完成对图中所有结点的排序，则说明图中没有环，反之，则说明有环。

16.【答案】A

【考点】图的基本术语

【解析】

无向连通图所有顶点度数之和为边的两倍，因此所有顶点的度之和必为偶数，I 的正确的。无向连通图对应的生成树也是无向连通图，此时边数等于顶点个数减 1，因此 II 是不正确的。III 也是不正确的，例如，如果图中只包含一个连接所有顶点的环，图的边数等于顶点个数，每个顶点的度就都是 2。因此答案选择 A。

17.【答案】C

【考点】图的基本术语

【解析】

要保证无向图 G 在任何情况下都是连通的，即任意变动图 G 中的边，G 始终保持连通。考虑极端的情况，图 G 的 6 个顶点构成完全连通子图 $G1$，则 $G1$ 有 6×5/2=15 条边，再添一条边第 7 个顶点必然与 $G1$ 构成连通图，因此边数最少为 16。

18.【答案】B

【考点】拓扑排序

【解析】

拓扑排序的步骤如下。

（1）在有向图中选一个无前驱的顶点且输出它。

（2）从图中删除该顶点和所有以它为尾的弧。

（3）重复（1）和（2），直至不存在无前驱的顶点。

（4）若此时输出的顶点数小于有向图中的顶点数，则说明有向图中存在环，否则输出的顶点序列为一个拓扑序列。

因为图中无前驱的顶点可能不唯一，所以拓扑排序的结果是不唯一的。对于题中的图，顶点 a 没有前驱，先输出 a，在删除 a 及弧<a,b>、<a,c>、<a,e>之后，顶点 b 和 e 没有前驱，可任选一个。假设先输出 b，在删除 b 及弧<b,c>之后，则输出 c 且删去 c 及弧<c,d>，输出 e 且删去 e 及弧<e, d>，则最后输出 d。最后得到该有向图的拓扑有序序列为：

$$a,\ b,\ c,\ e,\ d$$

模仿上述过程，可以得到另外两个拓扑序列：

$$a,\ b,\ e,\ c,\ d$$

和：

$$a,\ e,\ b,\ c,\ d$$

因此，可以得到 3 个不同的拓扑序列，选择 B 选项。

19.【答案】C

【考点】图的基本术语 图的存储结构拓扑排序

【解析】

（1）第一个顶点和最后一个顶点相同的路径称为回路，序列中顶点不重复出现的路径称为简单路径。显然，回路不是简单路径，所以Ⅰ是错误的。

（2）利用邻接矩阵存储图时，不论稀疏图还是稠密图，n 个顶点都需要 n^2 个单元存储边。而稀疏图中的边数较少，利用邻接矩阵存储稀疏图显然浪费了大量空间，应选邻接表进行存储，所以Ⅱ是错误的。

（3）用拓扑排序的方法可以判断图中是否存在回路，如果对一个图可以完成拓扑排序，则此图不存在回路，所以Ⅲ是正确的。

故答案选择 C。

20.【答案】C

【考点】广度优先搜索

【解析】

广度优先搜索需要借助队列实现。邻接表的结构包括：顶点表和边表。当使用邻接表存储方式时，在对图进行广度优先遍历时每个顶点均需入队一次（顶点表遍历），故时间复杂度为 O(n),

在搜索所有顶点的邻接点的过程中，每条边至少访问一次（出边表遍历），故时间复杂度为 O(e)，因此算法总的时间复杂度为 O(n+e)。

21.【答案】C

【考点】邻接矩阵 拓扑排序

【解析】

对角线以下元素均为零，表明只有顶点 i 到顶点 j（$i<j$）可能有边，而顶点 j 到顶点 i 一定没有边。也是说，这样的有向图中一定没有回路，所以可以产生拓扑序列，但拓扑序列不一定唯一。

22.【答案】C

【考点】迪杰斯特拉算法

【解析】

用集合 S（一维数组）记录从源点 a 到各个终点是否已被确定最短路径长度，true 表示确定，false 表示尚未确定，某个终点确定后即加入集合 S 中。从 a 到各终点的最短路径的求解过程如表 6.2 所示。

表 6.2 迪杰斯特拉算法的求解过程

终点	从 a 到各终点的最短路径长度 D 值和最短路径的求解过程				
	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$
a	∞	∞	∞	∞	∞
b	**2**(a, b)				
c	5(a, c)	**3** (a, b, c)			
d	∞	5(a, b, d)	**5**(a, b, d)	**5**(a, b, d)	
e	∞	∞	7$(a, b,$c, $e)$		
f	∞	∞	4$(a, b,$c,$f)$	7$(a, b,$c, $e)$	6$(a, b,$d, $e)$
S	$\{a, b\}$	$\{a, b, c\}$	$\{a, b, c, f\}$	$\{a, b, c, f, d\}$	$\{a, b, c, f,$ d,$e\}$

从表 6.2 的求解过程可以看出，从源点 a 出发，最短路径的目标顶点依次是 a、b、c、e、d、f，因此答案选择 C。

23.【答案】A

【考点】最小生成树

【解析】

（1）最小生成树的形状不唯一（因为可能存在权值相同的边），但是代价一定是唯一的，所以 I 正确。

（2）对于 II 如果权值最小的边有多条并且构成环状，则总有权值最小的边将不能出现在最小生成树中，所以 II 不正确。

（3）当存在多条权值相同的边时，用 Prim 算法从不同顶点开始得到的最小生成树不一定相同，III 不正确。

（4）在构造最小生成树时，Prim 算法是通过不断归并点来完成的，Kruskal 算法是通过不断归并边来完成的，二者构造的最小生成树可能是相同的，也可能是不同的，所以 IV 不正确。

故答案选择 A。

24.【答案】C

【考点】邻接矩阵

【解析】

由于矩阵 A 不是对称矩阵，所以矩阵 A 对应的图 G 是有向图。有向图中结点编号对应的行中非零元素的个数为该结点的出度，对应的列中非零元素的个数为该结点的入度，即各顶点的度是矩阵中此结点对应的横行和纵列非零元素之和。综上所述，第一个顶点的出度为 2，入度为 1，所以度为 3。同理，第 2 个结点度为 4，第 3 个结点度为 2，第 4 个结点度为 3。因此答案选择 C。

25.【答案】D

【考点】广度优先搜索

【解析】

因为从图的逻辑结构出发，在遍历时可以选择任何一个邻接点进行访问，因此广度优先搜索序列是不唯一的。这种题目可以逐一进行验证，根据第 14 题中给出的广度优先搜索过程，对于选项 A 中的序列，从顶点 h 出发，逐层向下访问。

（1）第 1 层，访问顶点 h；

（2）第 2 层，访问 h 未被访问的邻接点 c 和 a；

（3）第 3 层，访问 c 的未被访问的邻接点 b 和 d，访问 a 的未被访问的邻接点 e；

（4）第 4 层，b 和 d 的邻接点均被访问，继续访问 e 的未被访问的邻接点 g 和 f。

可以看出选项 A 的序列 h，c，a，b，d，e，g，f 是从顶点 h 出发的广度优先遍历序列，同理验证选项 B 和 C 也是正确的，而选项 D 是不正确的，是深度优先搜索遍历的结果。故答案选择 D。

26.【答案】C

【考点】关键路径。

【解析】

从源点到汇点的带权路径长度最长的路径，称为关键路径。此 AOE 网的关键路径有三条：*bdcg*、*bdeh* 和 *bfh*。根据定义，只有关键路径上的活动时间同时减少时，才能可以缩短工期。选项 A、B 和 D 都并不包含在所有的关键路径中，只有选项 C 包含，因此加快 f 和 d 的进度才能缩短工期。所以答案选择 C。

27.【答案】D

【考点】拓扑排序

【解析】

根据第 18 题中所给的拓扑排序过程可以对各个选项中的序列进行一一验证。对于本题中的图，顶点 3 没有前驱，先输出 3；在删除 3 及弧<1,3>，<3,6>之后，顶点 1 没有前驱，先输出 1；在删除 1 及弧<1,2>，<1,4>之后，顶点 4 没有前驱，先输出 4；在删除 4 及弧<4,2>，<4,6>之后顶点 2 和 6 没有前驱，可任选一个，假设先输出 2；在删除 2 及弧<2,5>之后，则输出 6 且删去 6 及弧<6,5>，最后输出 5。最后得到该有向图的拓扑有序序列为：

3，1，4，2，6，5

模仿上述过程，可以得到另外两个拓扑序列：

3，1，4，6，2，5

28.【答案】D

【考点】深度优先搜索

【解析】

画出该有向图如图 6.18 所示。

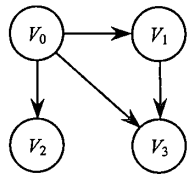


图 6.18 第 28 题有向图

从顶点 V_0 开始对图进行深度优先遍历，共有 5 种可能：$<V_0,V_1,V_3,V_2>$，$<V_0,V_2,V_3,V_1>$，$<V_0,V_3,V_2,V_1>$，$<V_0,V_2,V_1,V_3>$，$<V_0,V_3,V_1,V_2>$，所以答案选择 D。

29.【答案】C

【考点】最小生成树

【解析】

Kruskal 算法是按权值升序选择边的，首先选权值为 5 的边(V_1,V_4)，然后选择权值为 8 的边。此时有 3 种选择：(V_1,V_3)、(V_3,V_4)和(V_2,V_3)。Prim 算法从 V_4 开始，首先选择权值为 5 的边(V_1, V_4)，接着可以选择(V_1,V_3)或(V_3,V_4)，不可能选择(V_2,V_3)，所以答案选择 C。

30.【答案】D

【考点】深度优先搜索

【解析】

因为从图的逻辑结构出发，在遍历时可以选择任何一个邻接点进行访问，因此深度优先搜索序列是不唯一的。根据第 13 题给出的深度优先搜索遍历的具体步骤，可以对各个选项中的序列进行验证。

对于选项 A 中的序列，访问过程如下。

（1）从顶点 v_1 出发，访问 v_1。

（2）在访问了顶点 v_1 之后，选择第一个未被访问的邻接点 v_5，访问 v_5。以 v_5 为新顶点，重复此步，访问 v_4，v_3、v_2。在访问了 v_2 之后，由于 v_2 的邻接点都已被访问，此步结束。

（3）搜索从 v_2 回到 v_3，由于同样的理由，搜索继续回到 v_4，直至 v_1，由此，得到的顶点访问序列为：

$$v_1，v_5，v_4，v_3，v_2。$$

因此，选项 A 的序列是正确的。同理，可得到选项 B 和 C 中的序列也是正确的，而选项 D 中的序列是错误的，因此答案选择 D。

31.【答案】B

【考点】拓扑排序

【解析】

采用邻接表实现拓扑排序时，首先要通过建立入度数组来计算出各顶点的入度。对有 n 个顶点和 e 条边的有向图而言，建立入度数组的复杂度为 $O(e)$，建立零入度顶点栈的时间复杂度为 $O(n)$。在拓扑排序过程中，若有向图无环，则每个顶点进一次栈，出一次栈，入度减 1 的操作在循环中总共执行 e 次，所以，总的时间复杂度为 $O(n+e)$。

32.【答案】B

【考点】迪杰斯特拉算法

【解析】

用集合 S（一维数组）记录从源点 a 到各个终点是否已被确定最短路径长度，某个终点确定后即加入集合 S 中。从 a 到各终点的最短路径的求解过程如表 6.3 所示。

表 6.3　　迪杰斯特拉算法的求解过程

终点	从 a 到各终点的最短路径长度 D 值和最短路径的求解过程				
	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$
1	∞	∞	∞	∞	∞
2	**5**(1, 2)	**5**(1, 2)			
3	∞	∞	**7**(1, 2, 3)		
4	∞	11(1, 5, 4)	11(1, 5, 4)	11(1, 5, 4)	
5	**4**(1, 5)				
6	∞	9(1, 5, 6)	9(1, 5, 6)	9(1, 5, 6)	
S	{1, 5}	{1, 5, 2}	{1, 5 2, 3}	{1, 5, 2, 3, 6}	{1, 5, 2, 3, 6, 4}

从表 6.3 的求解过程可以看出，从源点 a 出发，最短路径的目标顶点依次是 5，2，3，6，4，因此答案选择 B。

二. 应用题

1. 解答

（1）每个顶点的入度和出度如图 6.19（a）所示；

（2）邻接矩阵如图 6.19（b）所示；

（3）邻接表如图 6.19（c）所示；

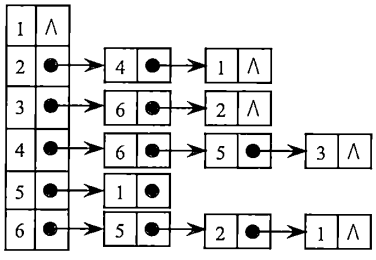
（4）逆邻接表如图 6.19（d）所示。

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

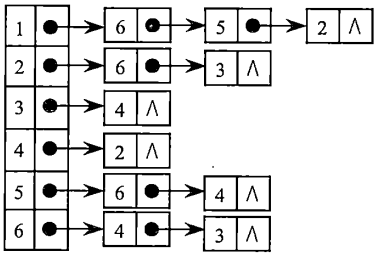
（a）顶点的入度和出度

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

（b）邻接矩阵



（c）邻接表



（d）逆邻接表

图 6.19　第 1 题答案

2．解答

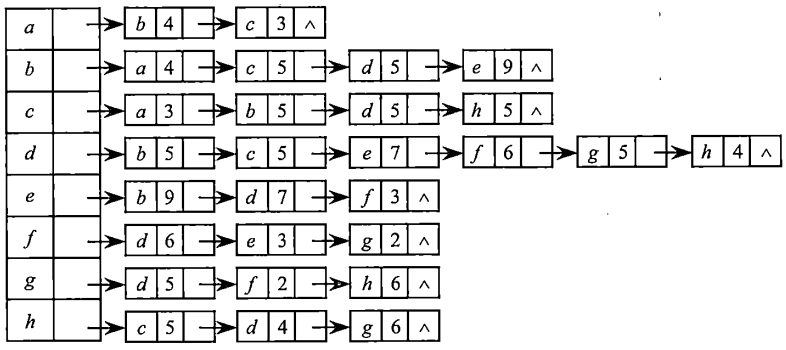
（1）邻接矩阵如图 6.20（a）所示；

（2）邻接表如图 6.20（b）所示；

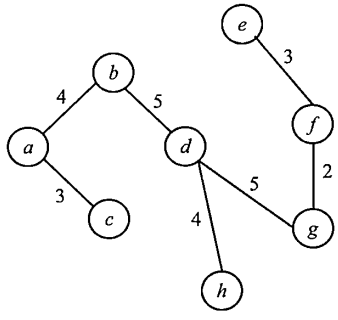
（3）最小生成树如图 6.20（c）所示。

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

（a）邻接矩阵



（b）邻接表



（c）最小生成树

图 6.20 第 2 题答案

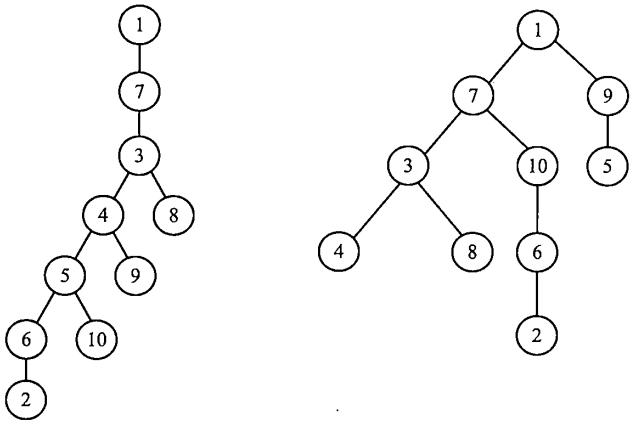
3．解答

深度优先生成树和广度优先生成树分别如图 6.21（a）和（b）所示。

4．解答

用集合 S（一维数组）记录从源点 a 到各个终点是否已被确定最短路径长度，true 表示确定，false 表示尚未确定，某个终点确定后即加入集合 S 中。用 k 记录当前求得的终点，则从 a 到各终

点的最短路径的求解过程如表 6.4 所示。



（a）深度优先生成树　　（b）广度优先生成树

图 6.21　第 3 题答案

表 6.4　　最短路径的求解

终点 \ d	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
b	15 (a,b)	15 (a,b)	15 (a,b)	15 (a,b)	15 (a,b)	**15** **(a,b)**
c	**2** **(a,c)**					
d	12 (a,d)	12 (a,d)	11 (a,c,f,d)	**11** **(a,c,f,d)**		
e	∞	10 (a,c,e)	**10** **(a,c,e)**			
f	∞	**6** **(a,c,f)**				
g	∞	∞	16 (a,c,f,g)	16 (a,c,f,g)	**14** **(a,c,f,d,g)**	
S 终点集	$\{a,c\}$	$\{a,c,f\}$	$\{a,c,f,e\}$	$\{a,c,f,e,d\}$	$\{a,c,f,e,d,g\}$	$\{a,c,f,e,d,g,b\}$

5．解答

求解关键活动的步骤是：首先对图中的顶点进行排序，在排序过程中按拓扑序列求出每个事件的最早发生时间 $ve(i)$和最迟发生时间 $vl(i)$；然后求出每个活动 a_i的最早开始时间 $e(i)$和最晚开始时间 $l(i)$；最后找出 $e(i)=l(i)$的活动，即为关键活动。

其中，主要的计算量为依次求解 $ve(i)$和 $vl(i)$以及 $e(i)$和 $l(i)$四个描述量的值。

- 计算事件 v_i的最早发生时间 $ve(i)$。

进入事件 v_i的每一活动都结束，v_i才可发生，所以 $ve(i)$是从源点到 v_i的最长路径长度。

求 $ve(i)$的值，可根据拓扑顺序从源点开始向汇点递推。通常将工程的开始顶点事件 v_0的最早发生时间定义为 0，即：

$$ve(0)=0$$

$$ve(i) = \mathrm{Max}\{ve(k) + w_{k,i}\} \quad <v_k, v_i> \in T, \ 1 \leqslant i \leqslant n-1$$

其中，T 是所有以 v_i 为头的弧的集合，$w_{k,i}$ 是弧<v_k, v_i>的权值，即对应活动<v_k, v_i>的持续时间。

- 计算事件 v_i 的最迟发生时间 $vl(i)$。

事件 v_i 的发生不得延误 v_i 的每一后继事件的最迟发生时间。为了不拖延工期，v_i 的最迟发生时间不得迟于其后继事件 v_k 的最迟发生时间减去活动<v_i, v_k>的持续时间。

求出 $ve(i)$后，可根据逆拓扑顺序从汇点开始向源点递推，求出 $vl(i)$。

$$vl(n-1) = ve(n-1)$$

$$vl(i) = \mathrm{Min}\{vl(k) - w_{i,k}\} \quad <v_i, v_k> \in S, \ 0 \leqslant i \leqslant n-2$$

其中，S 是所有以 v_i 为尾的弧的集合，$w_{i,k}$ 是弧<v_i, v_k>的权值。

- 计算活动 a_i = <v_j, v_k>的最早开始时间 $e(i)$。

只有事件 v_j 发生了，活动 a_i 才能开始。所以，活动 a_i 的最早开始时间等于事件 v_j 的最早发生时间 $ve(j)$，即：

$$e(i) = ve(j)$$

- 计算活动 a_i = <v_j, v_k>的最晚开始时间 $l(i)$。

活动 a_i 的开始时间需保证不延误事件 v_k 的最迟发生时间。所以活动 a_i 的最晚开始时间 $l(i)$等于事件 v_k 的最迟发生时间 $vl(k)$减去活动 a_i 的持续时间 $w_{j,k}$，即：

$$l(i) = vl(k) - w_{j,k}$$

根据上述计算方法，计算得到顶点事件的发生时间和活动的开始时间分别汇总为表 6.5（a）和表 6.5（b）。

（1）由表 6.5（a）可以看出，这个工程最早可能结束时间为 43。

（2）每个活动的最早开始时间和最迟开始时间如表 6.5（b）所示。

（3）关键活动包括<1,3>、<3,2>、<2,5>、<5,6>。

表 6.5　　第 5 题关键活动求解的中间结果

（a）顶点事件的发生时间

顶点 i	$ve(i)$	$vl(i)$
1	0	0
2	19	19
3	15	15
4	29	37
5	38	38
6	43	43

（b）活动的开始时间

活动 a_i	$e(i)$	$l(i)$	$l(i)-e(i)$
<1,2>	0	17	17
<1,3>	0	0	0
<3,2>	15	15	0
<2,4>	19	27	8
<2,5>	19	19	0
<3,5>	15	27	12
<4,6>	29	37	8
<5,6>	38	38	0

6．解答

（1）图 G 的邻接矩阵 $\boldsymbol{A}$ 如图 6.22（a）所示。

（2）矩阵 $\boldsymbol{A}^2$ 如图 6.22（b）所示。$\boldsymbol{A}^2$ 中位于 0 行 3 列的元素值 3 表示从顶点 0 到顶点 3 之间长度为 2 的路径共有 3 条。

（3）$\boldsymbol{B}^m(2 \leqslant m \leqslant n)$非零元素的含义是：图中从顶点 i 到顶点 j 长度为 m 的路径条数。

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

（a）矩阵 A

$$A^2=\begin{bmatrix}3&1&0&3&1\\1&3&2&1&2\\0&2&2&0&2\\3&1&0&3&1\\1&2&2&1&3\end{bmatrix}$$

（b）矩阵 A^2

图 6.22　无向图

7．解答

（1）由题目分析可写出一个待定的上三角矩阵，如图 6.23（a）所示（其中“?”为待定元素）。可以看出，第 1 行至第 5 行主对角线上方的元素分别为 5 个、4 个、3 个、2 个、1 个，这样便可画出压缩存储数组中元素所属行的情况，如图 6.23（b）所示。

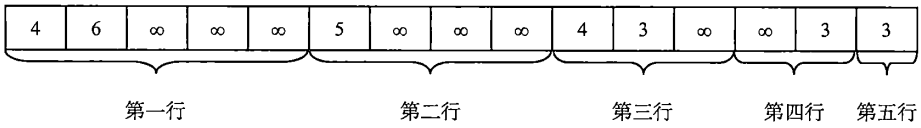
将各元素填入相应行即得图 G 的邻接矩阵 A 如图 6.23（c）所示。

（2）根据第 1 步中的邻接矩阵 A 即可做出有向带权图 G，如图 6.23（d）所示。

（3）根据第 5 题求解关键活动的步骤可以求出关键活动，由关键活动形成的由源点到汇点的最长路径就是关键路径，图 6.23（e）中双线箭头所标识的 4 个活动组成图 G 的关键路径，其关键路径长度为 4+5+4+3=16。

$$\begin{bmatrix}0&?&?&?&?&?\\\infty&0&?&?&?&?\\\infty&\infty&0&?&?&?\\\infty&\infty&\infty&0&?&?\\\infty&\infty&\infty&\infty&0&?\\\infty&\infty&\infty&\infty&\infty&0\end{bmatrix}\begin{matrix}5\\4\\3\\2\\1\\\end{matrix}$$

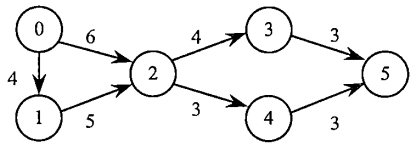
（a）待定的上三角矩阵



（b）压缩存储数组中元素所属行的情况

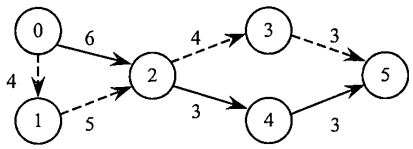
$$A=\begin{bmatrix}0&4&6&\infty&\infty&\infty\\\infty&0&5&\infty&\infty&\infty\\\infty&\infty&0&4&3&\infty\\\infty&\infty&\infty&0&\infty&3\\\infty&\infty&\infty&\infty&0&3\\\infty&\infty&\infty&\infty&\infty&0\end{bmatrix}$$

（c）邻接矩阵 A



（d）有向带权图 G

图 6.23　第 7 题答案



（e）关键路径

图 6.23 第 7 题答案（续）

8．解答

该方法求得的路径不一定是最短路径。例如，对于如图 6.24 所示的带权图，如果按照题中的求解原则，从 A 到 C 的最短路径为 $A\rightarrow B\rightarrow C$，事实上其最短路径为 $A\rightarrow D\rightarrow C$。

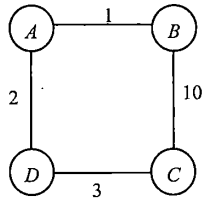


图 6.24 第 8 题示例

三．算法设计题

1．解答

假设图 G 为无向无权图，以邻接矩阵作为存储结构时，各个基本操作的实现如下。

（1）增加一个新顶点 v。

【算法思想】

首先判断插入的合法性，然后将顶点的数量加 1，并将新顶点 v 存入顶点表，邻接矩阵相应位置的元素置为 0。

【算法描述】

```
Status InsertVex(AMGraph &G,int v)
{//在以邻接矩阵形式存储的无向图 G 上插入顶点 v
  if((G.vexnum+1)>MVNUM)
    return INFEASIBLE;                          //判断插入操作是否合法
  G.vexnum++;                                   //增加图的顶点数量
  G.vexs[G.vexnum]=v;                           //将新顶点 v 存入顶点表
  for(k=0;k<G.vexnum;k++)                       //邻接矩阵相应位置的元素置为 1
    G.arcs[G.vexnum][k]=G.arcs[k][G.vexnum]=0;
  return OK;
}
```

（2）删除顶点 v 及其相关的边。

【算法思想】

如果采用类似顺序表的删除方法，则需要移动后面的顶点和边，这里为了避免移动，采用交换的思想。首先判断删除的合法性，然后将待删除顶点交换到最后一个顶点，将邻接矩阵中相应边的关系随之交换，最后图中顶点总数减 1。

【算法描述】

```
Status DeleteVex(AMGraph &G,int v)
{//在以邻接矩阵形式存储的无向图 G 上删除顶点 v
```

```
    n=G.vexnum;
    if((m=LocateVex(G,v))<0)
      return ERROR;                          //判断删除操作是否合法，即 v 是否在 G 中
    G.vexs[m]<->G.vexs[n];                   //将待删除顶点交换到最后一个顶点
    for(i=0;i<n;i++)                         //将边的关系随之交换
    {
      G.arcs[m][i]<->G.arcs[n][i];
      G.arcs[i][m]<->G.arcs[i][n];
    }
    G.vexnum--;                              //顶点总数减 1
    return OK;
  }
```

（3）增加一条边<*v*, *w*>。

【算法思想】

首先判断新增边涉及的两个顶点是否存在，如果存在且并非同一顶点，则在邻接矩阵上增加对应的边，最后将图的边数加 1 。

【算法描述】

```
Status InsertArc(AMGraph &G,int v,int w)
{//在以邻接矩阵形式存储的无向图 G 上插入边(v,w)
  if((i=LocateVex(G,v))<0) return ERROR;     //判断插入位置是否合法
  if((j=LocateVex(G,w))<0) return ERROR;
  if(i==j) return ERROR;
  G.arcs[i][j]=G.arcs[j][i]=1;               //在邻接矩阵上增加对应的边
  G.arcnum++;                                //边数加 1
  return OK;
}
```

（4）删除一条边<*v*, *w*>。

【算法思想】

首先判断待删除的边所涉及的两个顶点是否存在，如果存在，则在邻接矩阵上删除对应的边，最后将图的边数减 1 。

【算法描述】

```
Status DeleteArc(AMGraph &G,int v,int w)
{//在以邻接矩阵形式存储的无向图 G 上删除边(v,w)
  if((i=LocateVex(G,v))<0) return ERROR;     //判断删除位置是否合法
  if((j=LocateVex(G,w))<0) return ERROR;
  G.arcs[i][j]=G.arcs[j][i]=0;               //在邻接矩阵上删除边
  G.arcnum--;//边数减 1
  return OK;
}
```

以邻接表作为存储结构时，各个基本操作的实现如下。

（1）增加一个新顶点 *v*。

【算法思想】

首先判断插入的合法性，然后将顶点的数量加 1，并将新顶点对应的链表的头结点数据域赋值为 *v*，指针域赋值为 NULL。

【算法描述】

```
Status InsertVex(AMGraph &G,int v)
{//在以邻接表形式存储的无向图 G 上插入顶点 v
  if((G.vexnum+1)>MVNUM)
    return INFEASIBLE;                          //判断插入操作是否合法
  G.vexnum++;                                   //增加图的顶点数量
  G.vertices[G.vexnum].data=v;                  //新顶点对应的链表的头结点数据域赋值为 v
  G.vertices[G.vexnum].firstarc=NULL;           //指针域赋值为 NULL
  return OK;
}
```

（2）删除顶点 v 及其相关的边

【算法思想】

①判断顶点 v 是否存在，不存在则退出，存在则将顶点 v 从图中删除，顶点数减 1。②删除与 v 相关联的边。依次遍历邻接表，如果当前结点的第一条边关联的结点为 v，则直接删除该边，图的边数目减 1；否则继续比较后面的结点，寻找是否存在与结点 v 关联的边，存在则删除该边，图的边数目减 1。

【算法描述】

```
Status DeleteVex(ALGraph &G,int v)
{在以邻接表形式存储的无向图 G 上删除顶点 v
  n=LocateVex(G,v);
  if(n==0)                                      //如果 n 的值为 0，表示删除的顶点 v 不存在
    return ERROR;
  for(i=n;i<G.vexnum;i++)                       //把 v 的顶点信息删除，后面顶点向前移动
    G.vertices[i]=G.vertices[i+1];
  G.vexnum--;                                   //顶点数减 1
  for(i=1;i<=G.vexnum;i++)                      //遍历图，删除顶点 v 相关的边
  {
    p=G.vertices[i];q=G.vertices[i];            //p 指向当前结点的前驱,q 指向当前结点
    if(p.firstarc->adjvex==v)
    {//如果当前结点的第一条边关联的结点为 v，则直接删除该边
      G.vertices[i].firstarc=G.vertices[i].firstarc->nextarc;
      G.arcnum--;
      continue;
    }
    q.firstarc=q.firstarc->nextarc;             //继续比较后面的边结点
    while(q.firstarc)
    {
      if(q.firstarc->adjvex==v)
      {//存在与 v 相关联的边，则删除该边
        p.firstarc->nextarc=q.firstarc->nextarc;
        G.arcnum--;
        break;
      }
      p.firstarc=q.firstarc;                    //没找到与 v 相关联的边，指针后移
      q.firstarc=q.firstarc->nextarc;
    }
  }
  return OK;
}
```

（3）增加一条边<v,w>。

【算法思想】

首先确定插入边(v,w)的两个顶点是否合法，如果合法则生成新的边结点，将新结点插入顶点 v 所在的边链表的头结点之后。因为 G 是无向图，需要同时将新结点插入顶点 w 所在的边链表的头结点之后，即采用头插法插入对应链表。

【算法描述】

```
Status InsertArc(ALGraph &G,int v,int w)
{//在以邻接表形式存储的无向图 G 上插入边(v,w)
  i=LocateVex(G,v);
  j=LocateVex(G,w);                          //确定 v 和 w 在 G 中的位置
  if(i==0) return ERROR;                     //判断插入位置是否合法
  if(j==0) return ERROR;
  p1=new ArcNode;                            //生成一个新的边结点*p1
  p1->adjvex=j;                              //邻接点序号为 j
  p1->nextarc=G.vertices[i].firstarc;
  G.vertices[i].firstarc=p1;                 //将新结点*p1 插入顶点 v 的边表头部
  p2=new ArcNode;
  p2->adjvex=i;
  p2->nextarc=G.vertices[j].firstarc;        //将新结点*p1 插入顶点 w 的边表头部
  G.vertices[j].firstarc=p2;
  G.arcnum++;
  return OK;
}
```

（4）删除一条边<v,w>。

【算法思想】

首先确定待删除的边(v,w)的两个顶点是否合法，然后分别针对 v 和 w 所在的边链表删除对应的边。从顶点 v 所在的边链表依次遍历链表进行查找，如果找到边结点(v,w)，则将其删除。由于 G 为无向图，需要再从顶点 w 所在的边链表依次遍历链表进行查找，如果找到边结点(w,v)，则将其删除。最后将图中边的数目减 1.

【算法描述】

```
Status DeleteArc(ALGraph &G,int v,int w)
{//在以邻接表形式存储的无向图 G 上删除边<v, w>
  i=LocateVex(G,v);
  j=LocateVex(G,w);                          //确定 v 和 w 在 G 中的位置
  if(i==0) return ERROR;                     //判断删除位置是否合法
  if(j==0) return ERROR;
  if(G.vertices[i].firstarc->adjvex==w)
  //从顶点 v 所在的边链表遍历链表进行查找，第一个边结点是(v,w)，直接删除
      G.vertices[i].firstarc=G.vertices[i].firstarc->nextarc;
  else
  {//第一个边结点不是(v,w)，继续向后查找(v,w)
    p1=G.vertices[i].firstarc;
    p2=G.vertices[i].firstarc;
    while(p2->nextarc)
```

```
    {
      if(p2->adjvex==w)                          //在链表中找到边结点(v,w)
      {
        p1->nextarc=p2->nextarc;
        break;
      }
      p1=p2;
      p2=p2->nextarc;
    }
  }
  if(G.vertices[j].firstarc->adjvex==v)
  //从顶点w所在的边链表遍历链表进行查找，第一个边结点是(w,v)，直接删除
    G.vertices[j].firstarc=G.vertices[j].firstarc->nextarc;
  else
  {//第一个边结点不是(w,v)，继续向后查找(w,v)
    p1=G.vertices[j].firstarc;p2=G.vertices[j].firstarc;
    while(p2->nextarc)
    {
      if(p2->adjvex==v)                          //在链表中找到边结点(w,v)
      {
        p1->nextarc=p2->nextarc;
        break;
      }
      p1=p2;
      p2=p2->nextarc;
    }
  }
  G.arcnum--;                                    //边的数目减1
  return OK;
}
```

2．解答【算法思想】

要实现深度优先遍历的非递归过程，需要借助一个栈来保存访问过的顶点。首先将栈初始化为空，然后将起始顶点 v 进栈。判断栈是否为空，若不为空则重复进行下列操作：首先，取栈顶元素，如果该顶点未被访问，则访问该顶点，将访问标志改为“已访问”；然后，将该顶点的所有未访问过的邻接点进栈；直至栈为空时，表明图中所有的顶点都被访问。

【算法描述】

```
void DFS(ALGraph G,int v)
{//从第v个顶点出发非递归实现深度优先遍历图G
  InitStack(&S);                                 //构造一个空栈
  Push(S,v);                                     //顶点v进栈
  while(!StackEmpty(S))                          //栈非空
  {
    Pop(S,k);                                    //栈顶元素k出栈
    if(!visited[k])                              //k未被访问
    {
```

```
        cout<<k;                                    //访问第 k 个顶点
        visited[k]=true;
        p=G.vertices[k].firstarc;                   //p 指向 k 的边链表的第一个边结点
        while(p!=NULL)                              //边结点非空
         {
           w=p->adjvex;                             //表示 w 是 k 的邻接点
           if(!visited[w]) Push(S,w);               //如果 w 未访问，w 进栈
           p=p->nextarc;                            //p 指向下一个边结点
         }
      }//if
   }//while
}
```

3. 解答【算法思想】

首先利用 Dijkstra 算法求 v_0 到其他所有顶点的最短路径，分别保存在数组 $D[i]$ 中，然后求出 $D[i]$ 中值最大的数组下标 m 即可。

Dijkstra 算法的主要步骤为：（1）初始化。将 v_0 加到顶点集 S 中，即 $S[v_0]$=true；将 v_0 到各个终点之间的最短路径长度初始化为权值，即 $D[v]$=G.arcs$[v_0][v]$；如果 v_0 和 v 之间有弧，则将 v 的前驱置为 v_0，否则将 v 的前驱置为-1。（2）循环计算每个顶点 v 到 v_0 之间的最短路径。首先，选择一条当前的最短路径，终点为 v，将 v 加入 S 中，即 $S[v]$=true；然后，根据条件更新从 v_0 出发到集合 $V-S$ 上任一顶点的最短路径的长度，若条件 $D[k]+G$.arcs $[k][i]<D[i]$ 成立，则更新 $D[i]=D[k]+$G.arcs$[k][i]$，同时更改 v_i 的前驱为 v_k，Path$[i]=k$。

【算法描述】

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

```
        if(!S[w]&&(D[v]+G.arcs[v][w]<D[w]))
        {
           D[w]=D[v]+G.arcs[v][w];          //更新 D[w]
           Path[w]=v;                       //更改 w 的前驱为 v
        }//if
    }//for
/*-----------最短路径求解完毕，设距离顶点 v0 的最短路径长度最大的一个顶点为 m--------------*/
    Max=D[0];
    m=0;
    for(i=1;i<n;i++)
      if(Max<D[i]) m=i;
    return m;                               //返回顶点下标
}
```

4. 解答【算法思想】

可以利用深度优先搜索的算法递归遍历邻接表来进行判断。引入变量 *level* 来记录递归的层数，设置 visited[*n*]为访问标志数组，已访问顶点的值记为 true。该递归算法结束的条件为顶点 v_i 等于顶点 v_j，表明首尾相遇，存在路径，返回 1。如果递归结束的条件不成立，则从 v_i 的边链表的第一个边结点开始，获取 v_i 的邻接点，对 v_i 的尚未访问的邻接点递归调用。递归进行一次，*level* 的值则增加 1。如果递归之后 *level* 的值为 1，表明顶点 v_i 到顶点 v_j 不存在路径，返回 0。

【算法描述】

```
int visited[MVNum];                         //访问标志数组，其初值为 false
int level=1;                                //递归进行的层数
int PathDFS(ALGraph G,int i,int j)
{//基于深度优先搜索判断有向图 G 中顶点 i 到顶点 j 是否有路径,是则返回 1,否则返回 0
   if(i==j) return 1;                       //递归结束的条件，首尾相遇，存在路径
   else
   {
      visited[i]=true;                      //顶点 i 访问标志为 true
      for(p=G.vertices[i].firstarc;p;p=p->nextarc, level--)
      {
         level++;                           //递归层次增 1
         k=p->adjvex;
         if(!visited[k]&&PathDFS(G,k,j))
            return 1;                       //i 下游的顶点到 j 有路径
      }//for
   }//else
   if(level==1) return 0;                   //i 到 j 没有路径
}
```

5. 解答【算法思想】

可以利用深度优先搜索的算法递归遍历邻接表来进行判断。设置 visited[*n*]为访问标志数组，已访问顶点的值记为 true。该算法每递归调用一次，长度 *k* 减 1，递归结束的条件为首尾顶点相遇且 *k* 为 0，此时表明两个顶点之间存在一条长度为 *k* 的简单路径，返回 1。如果递归结束的条件不成立，则从 v_i 的边链表的第一个边结点开始，获取 v_i 的邻接点，对 v_i 的尚未访问的邻接点递归调

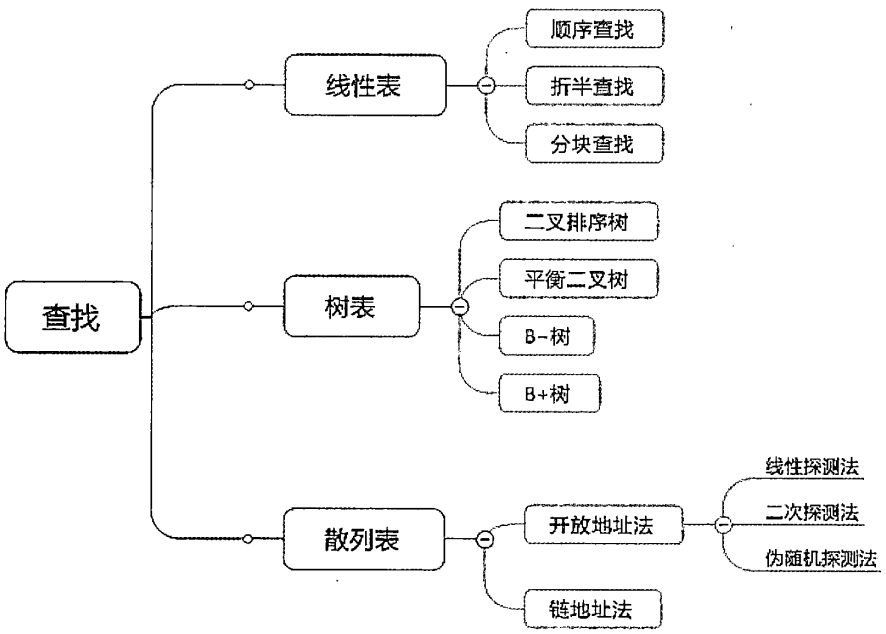
用，长度 k 减 1。如果最后递归结束的条件不成立且无法继续递归下去，表明两个顶点之间不存在一条长度为 k 的简单路径，返回 0。(注意，本题允许曾经被访问过的结点出现在另一条路径中，故在遍历之后需要设置 visited[i]的值为 0。)

【算法描述】

```
int visited[MAXSIZE];
int PathLenK(ALGraph G,int i,int j,int k)
{//判断邻接表方式存储的有向图 G 的顶点 i 到 j 是否存在长度为 k 的简单路径
  if(i==j&&k==0) return 1;                    //找到了一条路径,且长度符合要求
  else if(k>0)
  {
    visited[i]=true;
    for(p=G.vertices[i].firstarc;p;p=p->nextarc)
    {//从结点 i 开始遍历, p 为 i 的边链表的第一个边结点
      v=p->adjvex;
      if(!visited[v])                         //v 未被访问过
        if(PathLenK(G,v,j,k-1))
          return 1;                           //递归继续判断 1 到 j, 且剩余路径长度减 1
    }
      visited[i]=false;                       //允许曾经被访问过的结点出现在另一条路径中
  }
  return 0;                                   //没找到
}
```

第 7 章 查找

【知识导图】



【学习目标】

1. 线性表的查找。线性表的查找主要包括顺序查找、折半查找和分块查找。顺序查找既适用于线性表的顺序存储结构，又适用于线性表的链式存储结构，效率较低；折半查找只能用于有序的顺序表，效率较高；分块查找综合了上述两者的优点，既能较快查找，又能适用于动态变化的要求。掌握顺序查找和折半查找算法的实现，掌握描述折半查找过程的判定树的构造方法，明确三者的适用场合，掌握三种查找平均查找长度（ASL）的计算方法。

2. 树表的查找。树表的结构主要包括二叉排序树、平衡二叉树、B–树和 B + 树。

（1）二叉排序树的查找过程与折半查找过程类似，查找效率在平均情况下相同，但因采用树的二叉链表表示，因此适合经常做插入和删除的动态查找表。二叉排序树在形态均匀时性能最好，而形态为单支树时其查找性能则退化为与顺序查找相同。掌握二叉排序树的查找、创建、插入、删除算法，掌握二叉排序树 ASL 的计算方法。

（2）平衡二叉树的平衡调整方法可以确保二叉排序树在任何情况下的深度均为 $O(\log_2 n)$，掌

握平衡调整的4种方法：LL型、RR型、LR型和RL型。

（3）B-树是一种平衡的多叉查找树，是一种在外存文件系统中常用的动态索引技术。在B-树上进行查找的过程和二叉排序树类似，是一个顺指针查找结点和在结点内的关键字中查找交叉进行的过程。为了满足B-树的定义，在B-树中插入一个关键字，可能产生结点的“分裂”，而删除一个关键字，可能产生结点的“合并”。掌握B-树的查找、插入和删除的方法。

（4）B+树是B-树的一种变型树，更适合做文件系统的索引。B+树所有的叶子结点中包含了全部关键字的信息，以及指向含这些关键字记录的指针，且叶子结点本身依关键字的大小，按自小而大顺序链接，这就为顺序查找提供了方便。而B-树只适用于随机查找，不适用于顺序查找。明确B-树和B+树二者的区别。

3．散列表的查找。散列表不是以关键字比较为基础进行查找的，而是通过一种散列函数把记录的关键字和它在表中的位置建立起对应关系。散列查找法主要研究两方面的问题：如何构造散列函数和如何处理冲突。构造散列函数的方法很多，除留余数法是最常用的构造散列函数的方法。处理冲突的方法通常分为两大类：开放地址法和链地址法。掌握散列函数设计的原则，掌握不同冲突解决方法的特点和适用场合，掌握冲突处理的具体过程，掌握不同冲突解决方案下的ASL的计算方法。

7.1 习题

一、单项选择题

1．对n个元素的表进行顺序查找时，若查找每个元素的概率相同，则平均查找长度为（　　）。

A．$(n-1)/2$　　B．$n/2$　　C．$(n+1)/2$　　D．n

2．适用于折半查找的表的存储方式及元素排列要求为（　　）。

A．链接方式存储，元素无序　　B．链接方式存储，元素有序

C．顺序方式存储，元素无序　　D．顺序方式存储，元素有序

3．如果要求一个线性表既能较快地查找，又能适应动态变化的要求，最好采用（　　）查找法。

A．顺序查找　　B．折半查找　　C．分块查找　　D．哈希查找

4．折半查找有序表{4, 6, 10, 12, 20, 30, 50, 70, 88, 100}。若查找表中元素58，则它将依次与表中（　　）比较大小，查找结果是失败。

A．20, 70, 30, 50　　B．30, 88, 70, 50　　C．20, 50　　D．30, 88, 50

5．对22个记录的有序表进行折半查找，当查找失败时，至少需要比较（　　）次关键字。

A．3　　B．4　　C．5　　D．6

6．折半搜索与二叉排序树的时间性能（　　）。

A．相同　　B．完全不同　　C．有时不相同　　D．数量级都是$O(\log_2 n)$

7．分别以下列序列构造二叉排序树，与用其他三个序列所构造的结果不同的是（　　）。

A．100, 80, 90, 60, 120, 110, 130

B．100, 120, 110, 130, 80, 60, 90

C．100, 60, 80, 90, 120, 110, 130

D．100, 80, 60, 90, 120, 130, 110

8．在平衡二叉树中插入一个结点后造成了不平衡，设最低的不平衡结点为A，并已知A的左孩子

的平衡因子为0右孩子的平衡因子为1，则应做（　　）型调整以使其平衡。

A．LL　　B．LR　　C．RL　　D．RR

9．下列关于 m 阶B–树的说法错误的是（　　）。

A．根结点至多有 m 棵子树

B．所有叶子都在同一层次上

C．非叶子结点至少有 $m/2$ (m 为偶数)或 $m/2+1$（m 为奇数）棵子树

D．根结点中的数据是有序的

10．下面关于B–和B+树的叙述中，不正确的是（　　）。

A．B–树和B+树都是平衡的多叉树　　B．B–树和B+树都可用于文件的索引结构

C．B–树和B+树都能有效地支持顺序查找　　D．B–树和B+树都能有效地支持随机查找

11．m 阶B–树是一棵（　　）。

A．m 叉排序树　　B．m 叉平衡排序树

C．m–1 叉平衡排序树　　D．m+1 叉平衡排序树

12．下面关于散列查找的说法，正确的是（　　）。

A．散列函数构造得越复杂越好，因为这样随机性好、冲突小

B．除留余数法是所有散列函数中最好的

C．不存在特别好与坏的散列函数，要视情况而定

D．散列表的平均查找长度有时也和记录总数有关

13．下面关于哈希查找的说法，不正确的是（　　）。

A．采用链地址法处理冲突时，查找一个元素的时间是相同的

B．采用链地址法处理冲突时，若插入规定总是在链首，则插入任一个元素的时间是相同的

C．用链地址法处理冲突，不会引起二次聚集现象

D．用链地址法处理冲突，适合表长不确定的情况

14．设哈希表长为14，哈希函数是 $H(key)=key\%11$，表中已有数据的关键字为15、38、61、84，现要将关键字为49的元素存储到表中，用二次探测法解决冲突，则放入的位置是（　　）。

A．8　　B．3　　C．5　　D．9

15．采用线性探测法处理冲突，可能要探测多个位置，在查找成功的情况下，所探测的这些位置上的关键字（　　）。

A．不一定都是同义词　　B．一定都是同义词

C．一定都不是同义词　　D．都相同

16．【2009年第4题】在图7.1所示的二叉排序树中，满足平衡二叉树定义的是（　　）。

A.　B.　C.　D.

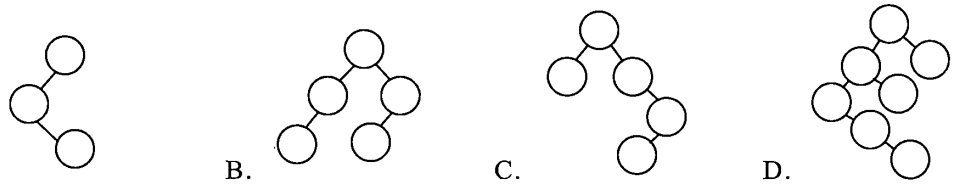


图7.1　二叉排序树

17．【2009年第8题】下列叙述中，不符合 m 阶B–树定义要求的是（　　）。

A．根结点最多有 m 棵子树　　B．所有的叶子结点都在同一层上

C．各结点内关键字均升序或降序排列　　D．叶子结点之间通过指针链接

18.【2010 年第 4 题】在图 7.2 所示的平衡二叉树中插入关键字 48 后得到一棵新平衡二叉树，在新平衡二叉树中，关键字 37 所在结点的左、右子结点中保存的关键字分别是（　　）。

A．13，48　　B．24，48　　C．24，53　　D．24，90

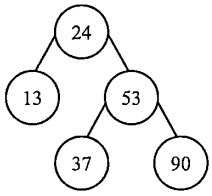


图 7.2　平衡二叉树

19.【2010 年第 9 题】已知一个长度为 16 的顺序表 L，其元素按关键字有序排列，若采用折半查找法查找一个不存在的元素，则比较次数最多的是（　　）。

A．4　　B．5　　C．6　　D．7

20.【2011 年第 7 题】对于下列关键字序列，不可能构成某二叉排序树中一条查找路径的序列是（　　）。

A．95, 22, 91, 24, 94, 71　　B．92, 20, 91, 34, 88, 35

C．21, 89, 77, 29, 36, 38　　D．12, 25, 71, 68, 33, 34

21.【2011 年第 9 题】为提高散列（Hash）表的查找效率，可以采取的正确措施是（　　）。

Ⅰ．增大装填（载）因子

Ⅱ．设计冲突（碰撞）少的散列函数

Ⅲ．处理冲突（碰撞）时避免产生聚集（堆积）现象

A．仅Ⅰ　　B．仅Ⅱ　　C．仅Ⅰ、Ⅱ　　D．仅Ⅱ、Ⅲ

22.【2012 年第 4 题】若平衡二叉树的高度为 6，且所有非叶子结点的平衡因子均为 1，则该平衡二叉树的结点总数为（　　）。

A．12　　B．20　　C．32　　D．33

23.【2012 年第 9 题】有一棵 3 阶 B–树，如图 7.3 所示。删除关键字 78 得到一棵新 B–树，其最右叶子结点中所含的关键字是（　　）。

A．60　　B．60, 62　　C．62, 65　　D．65

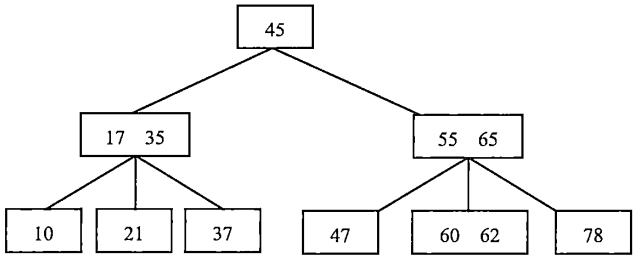


图 7.3　3 阶 B–树

24.【2013 年第 3 题】若将关键字 1、2、3、4、5、6、7 依次插入到初始为空的平衡二叉树 T 中，则 T 中平衡因子为 0 的分支结点的个数是（　　）。

A．0　　B．1　　C．2　　D．3

25.【2013 年第 6 题】在任意一棵非空二叉排序树 $T1$ 中，删除某结点 v 之后形成二叉排序树 $T2$，再将 v 插入 $T2$ 形成二叉排序树 $T3$。下列关于 $T1$ 与 $T3$ 的叙述中，正确的是（　　）。

I. 若 v 是 $T1$ 的叶子结点，则 $T1$ 与 $T3$ 不同

II. 若 v 是 $T1$ 的叶子结点，则 $T1$ 与 $T3$ 相同

III. 若 v 不是 $T1$ 的叶子结点，则 $T1$ 与 $T3$ 不同

IV. 若 v 不是 $T1$ 的叶子结点，则 $T1$ 与 $T3$ 相同

A. 仅 I、III　　B. 仅 I、IV　　C. 仅 II、III　　D. 仅 II、IV

26.【2013 年第 10 题】在一株高度为 2 的 5 阶 B–树中，所含关键字的个数最少是（　　）。

A. 5　　B. 7　　C. 8　　D. 14

27.【2014 年第 8 题】用哈希（散列）方法处理冲突（碰撞）时可能出现聚集（堆积）现象，下列选项中，会受堆积现象直接影响的是（　　）。

A. 存储效率　　B. 散列函数　　C. 装填（装载）因子 D. 平均查找长度

28.【2014 年第 9 题】在一棵具有 15 个关键字的 4 阶 B–树中，含关键字的结点个数最多是（　　）。

A. 5　　B. 6　　C. 10　　D. 15

29.【2015 年第 4 题】现有一棵无重复关键字的平衡二叉树（AVL 树），对其进行中序遍历可得到一个降序序列。下列关于该平衡二叉树的叙述中，正确的是（　　）。

A. 根结点的度一定为 2　　B. 树中最小元素一定是叶子结点

C. 最后插入的元素一定是叶子结点　　D. 树中最大元素一定无左子树

30.【2015 年第 7 题】下列选项中，不能构成折半查找中关键字比较序列的是（　　）。

A. 500, 200, 450, 180　　B. 500, 450, 200, 180

C. 180, 500, 200, 450　　D. 180, 200, 500, 450

31.【2016 年第 9 题】在有 n（n>1 000）个元素的升序数组 A 中查找关键字 x。查找算法的伪代码如下所示。

```
k=0;
while(k<n且A[k]<x) k=k+3;
if(k<n且A[k]==x) 查找成功;
else if(k-1<n且A[k-1]==x) 查找成功;
   else if(k-2<n且A[k-2]==x) 查找成功;
      else 查找失败;
```

本算法与折半查找算法相比，有可能具有更少比较次数的情形是（　　）。

A. 当 x 不在数组中　　B. 当 x 接近数组开头处

C. 当 x 接近数组结尾处　　D. 当 x 位于数组中间位置

32.【2016 年第 10 题】B+树不同于 B–树的特点之一是（　　）。

A. 能支持顺序查找　　B. 结点中含有关键字

C. 根结点至少有两个分支　　D. 所有叶子结点都在同一层上

二、应用题

1. 假定对有序表{3, 4, 5, 7, 24, 30, 42, 54, 63, 72, 87, 95}进行折半查找，试回答下列题。

（1）画出描述折半查找过程的判定树。

（2）若查找元素 54，需依次与哪些元素比较?

（3）若查找元素 90，需依次与哪些元素比较?

（4）假定每个元素的查找概率相等，求查找成功时的平均查找长度。

2．在一棵空的二叉排序树中依次插入关键字序列为{12, 7, 17, 11, 16, 2, 13, 9, 21, 4}，请画出所得到的二叉排序树。

3．已知长度为 12 的表：（Jan, Feb, Mar, Apr, May, June, July, Aug, Sep, Oct, Nov, Dec）。

（1）试按表中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，并求其在等概率的情况下查找成功的平均查找长度。

（2）若对表中元素先进行排序构成有序表，求在等概率的情况下对此有序表进行折半查找时查找成功的平均查找长度。

（3）按表中元素顺序构造一棵平衡二叉排序树，并求其在等概率的情况下查找成功的平均查找长度。

4．对图 7.4 所示的 3 阶 B–树，依次执行下列操作，画出各步操作的结果。

（1）插入 90。

（2）插入 25。

（3）插入 45。

（4）删除 60。

（5）删除 80。

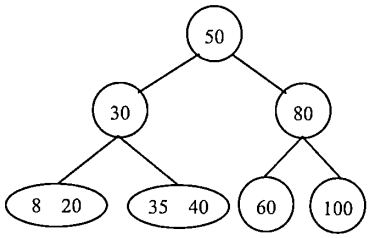


图 7.4　3 阶 B–树

5．设散列表的地址范围为 0～17，散列函数为：$H(key) = key\%16$。用线性探测法处理冲突，输入关键字序列：（10, 24, 32, 17, 31, 30, 46, 47, 40, 63, 49），构造散列表，试回答下列问题：

（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）若查找关键字 60，需要依次与哪些关键字进行比较?

（4）假定每个关键字的查找概率相等，求查找成功时的平均查找长度。

6．设有一组关键字（9, 1, 23, 14, 55, 20, 84, 27），采用散列函数：$H(key) = key\%7$，表长为 10，用开放地址法的二次探测法处理冲突。要求：对该关键字序列构造散列表，并计算查找成功的平均查找长度。限定 d_i 取值为 $1^2, 2^2, \cdots, k^2 (k \leqslant m/2)$。

7．设散列函数 $H(K) = 3K \% 11$，散列地址空间为 0～10，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，并分别求出等概率下查找成功时和查找失败时的平均查找长度 ASL_{succ} 和 ASL_{unsucc}。

（1）线性探测法。

（2）链地址法。

8．【2010 年第 41 题】将关键字序列{7, 8, 30, 11, 18, 9, 14}散列存储到散列表中，散列表的存储空间是一个下标从 0 开始的一维数组散列，函数为：$H(key)=(key \times 3)$MOD T，处理冲突采用线

性探测再散列法，要求装载因子为 0.7。问题：

（1）请画出所构造的散列表。

（2）分别计算等概率情况下，查找成功和查找不成功的平均查找长度。

9.【2013 年第 42 题】设包含 4 个数据元素的集合 S={"do", "for", "repeat", "while"}，各元素的查找概率依次为：$p1$=0.35，$p2$=0.15，$p3$=0.15，$p4$=0.35。将 S 保存在一个长度为 4 的顺序表中，采用折半查找法，查找成功时的平均查找长度为 2.2。请回答：

（1）若采用顺序存储结构保存 S，且要求平均查找长度更短，则元素应如何排列？应使用何种查找方法？查找成功时的平均查找长度是多少？

（2）若采用链式存储结构保存 S，且要求平均查找长度更短，则元素应如何排列？应使用何种查找方法？查找成功时的平均查找长度是多少？

三、算法设计题

1．试写出折半查找的递归算法。

2．试写一个判别给定二叉树是否为二叉排序树的算法。

3．已知二叉排序树采用二叉链表存储结构，根结点的指针为 T，链结点的结构为（lchild, data, rchild），其中 lchild、rchild 分别指向该结点左、右孩子的指针，data 域存放结点的数据信息。请写出递归算法，从小到大输出二叉排序树中所有数据值 $\geq x$ 的结点的数据。要求先找到第一个满足条件的结点后，再依次输出其他满足条件的结点。

4．已知二叉树 T 的结点形式为（llink, data, count, rlink），在树中查找值为 X 的结点，若找到，则记数（count）加 1；否则，作为一个新结点插入树中，插入后仍为二叉排序树，写出其非递归算法。

5．假设一棵平衡二叉树的每个结点都表明了平衡因子 b，试设计一个算法，求平衡二叉树的高度。

6．分别写出在散列表中插入和删除关键字为 K 的一个记录的算法，设散列函数为 H，解决冲突的方法为链地址法。

7.2 答案及解析

一、单项选择题

1. C	2. D	3. C	4. A	5. B	6. C	7. C	8. C
9. C	10. C	11. B	12. C	13. A	14. D	15. A	16. B
17. D	18. C	19. B	20. A	21. B	22. B	23. D	24. D
25. C	26. A	27. D	28. D	29. D	30. A	31. B	32. A

1.【答案】C

【考点】顺序查找

【解析】

对于含有 n 个记录的表，查找成功时的平均查找长度为 $ASL=\sum_{i=1}^{n}P_iC_i$，其中，P_i为查找表中第 i 个记录的概率，且 $\sum_{i=1}^{n}P_i=1$；C_i为找到表中其关键字与给定值相等的第 i 个记录时，和给定值已进行过比较的关键字个数。由题目可知，$P_i=1/n$，$C_i=i$；即 $ASL=\frac{1}{n}\sum_{i=1}^{n}i=\frac{n+1}{2}$。

2.【答案】D

【考点】折半查找

【解析】

折半查找在查找过程中需要定位待查找表的上界、下界和中间位置，因此线性表必须采用顺序存储结构，而且表中元素按关键字有序排列。故答案选择 D。

3.【答案】C

【考点】分块查找

【解析】

在平均情况下，顺序查找效率较低，折半查找效率较高，但同顺序查找一样，折半查找不能适应动态变化的要求。哈希查找效率较高，但是，当使用开哈希数组进行存储时，也不能适应动态变化的要求。分块查找的优点是：在表中插入和删除数据元素时，只要找到该元素对应的块，就可以在该块内进行插入和删除运算。由于块内是无序的，故插入和删除比较容易，无须进行大量移动。如果线性表既要快速查找又经常进行动态变化，则可采用分块查找。因此答案选择 C。

4.【答案】A

【考点】折半查找。

【解析】

表中共 10 个元素，第一次取⌊(1+10)/2⌋=5，与第 5 个元素 20 比较，58>20，在右子表{30, 50, 70, 88, 100}中进行查找；对于右子表{30, 50, 70, 88, 100}，取⌊(6+10)/2⌋=8，与第 8 个元素 70 比较，

58<70，在左子表{30, 50}中进行查找；对于左子表{30, 50}，取⌊(6+7)/2⌋=6，与第 6 个元素 30 比较，58>30，在右子表{50}中进行查找；对于右子表{50}，取⌊(7+7)/2⌋=7，与第 7 个元素 50 比较，58>50，所以最终查找失败。在查找过程中，依次与表中 20、70、30、50 进行了比较，因此答案选择 A。

5.【答案】B

【考点】折半查找

【解析】

22 个记录的有序表，其折半查找的判定树深度为$\lfloor \log_2 22 \rfloor + 1 = 5$，且该判定树不是满二叉树，即查找失败时至多比较 5 次，至少比较 4 次。

6.【答案】C

【考点】折半查找 二叉排序树的查找

【解析】

折半查找的时间复杂度为 $O(\log_2 n)$，二叉排序树在形态均匀时查找性能最好，时间复杂度为 $O(\log_2 n)$，而形态为单支树时其查找性能则退化为与顺序查找相同，因此，折半查找与二叉排序树查找的时间性能有时不相同。

7.【答案】C

【考点】二叉排序树的创建

【解析】

方法一：

根据题目中的选项 A、B、D 构造二叉排序树，构造结果相同，如图 7.5（a）所示，而选项 C 构造的二叉排序树如图 7.5（b）所示，所以答案选择 C。

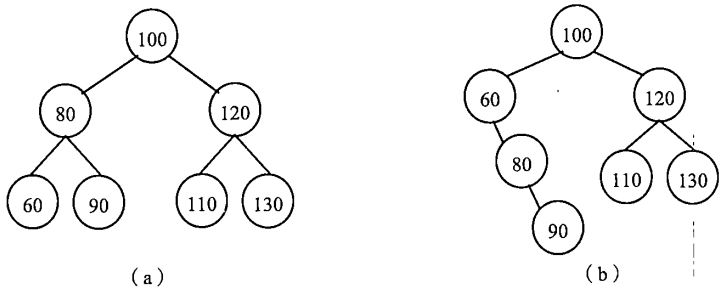


图 7.5　二叉排序树

方法二：

A、B、C、D 四个选项构造二叉排序树都以 100 为根，易知 A、B、D 三个序列中第一个比 100 小的关键字为 80，即 100 的左孩子为 80，而 C 选项中 100 的左孩子为 60，故答案选择 C。

8.【答案】C

【考点】平衡二叉树

【解析】

由题目知，在平衡二叉树中插入一个结点后造成了不平衡，即二叉树上的结点的平衡因子绝对值超过 1。题目中已知最低的不平衡结点为 A，其左孩子平衡因子为 0，右孩子平衡因子为 1，当把结点插入到 A 的右子树根结点的左子树上时，结点 A 的平衡因子绝对值超过 1，即应做 RL 调整使其平衡。

9.【答案】C

【考点】B–树

【解析】

若根结点不是叶子结点，则至少有两棵子树，选项 C 与此矛盾，因此答案选择 C。

10.【答案】C

【考点】B–和 B+树

【解析】

B–树只适用于随机查找，不适用于顺序查找；而 B+树为叶子结点增加链表指针，所有关键字都在叶子结点中出现，非叶子结点作为叶子结点的索引，这就为顺序查找提供了方便。

11.【答案】B

【考点】B–树

【解析】

一棵 m 阶的 B–树，或为空树，或为每个结点至多有 m 棵子树的 m 叉树。B–树具有平衡、有序、多路的特点。

（1）所有叶子结点均在同一层次，这体现出其平衡的特点。

（2）树中每个结点中的关键字都是有序的，且关键字 K_i “左子树”中的关键字均小于 K_i，而其“右子树”中的关键字均大于 K_i，这体现出其有序的特点。

（3）除叶子结点外，有的结点中有一个关键字、两棵子树，有的结点中有两个关键字、三棵子树，这种 4 阶的 B–树最多有三个关键字、四棵子树，这体现出其多路的特点。

综上所述，答案选择 B。

12.【答案】C

【考点】散列表的查找

【解析】

构造散列函数的方法很多，一般来说，应根据具体问题选用不同的散列函数，而不能说哪种散列函数绝对好或绝对不好。因此，选项 A 和 B 是错误的，而选项 C 是正确的。散列表的平均查找长度是装填因子 α 的函数，而不是记录个数 n 的函数，显然，选项 D 也是错误的。

13.【答案】A

【考点】散列表的查找

【解析】

链地址法的基本思想是：把具有相同散列地址的记录放在同一个单链表中，称为同义词链表。在同义词构成的单链表中，查找该单链表中的不同元素所消耗的时间是不同的，因此，A 是不正确的。当规定插入总是在链首时，则插入任一个元素的时间复杂度都是 O(1)的，时间是相同的。因此选项 B 是正确的，选项 C 也是正确的，因为具有相同散列地址的记录放在同一个单链表，这样不会发生两个散列地址不同的记录争夺同一个后继散列地址的现象，即不会引起二次聚集现象。另外因为链地址法的空间是动态分配的，适合表长不确定的情况，显然选项 D 也是正确的。因此答案选择 A。

14.【答案】D

【考点】散列表的查找

【解析】

关键字 15 放入位置是 $H(15)=15\%11=4$，同理，关键字 38 放入位置 5，关键字 61 放入位

置 6，关键字 84 放入位置 7。当添加关键字 49 时，地址为 5，发生了冲突。利用二次探测法解决冲突得到新地址 $H_1=(5+1^2)\%14=6$，仍冲突，再次探测新地址 $H_2=(5-1^2)\%14=4$，仍冲突，继续探测新地址 $H_3=(5+2^2)\%14=9$，此时不冲突，即将关键字 49 放入位置 9，因此答案选 D。

15.【答案】A

【考点】散列表的查找

【解析】

所探测的这些关键字可能是在处理其他关键字冲突过程中放入该位置的，因此不一定是同义词，所以答案选择 A。

16.【答案】B

【考点】平衡二叉树

【解析】

根据平衡二叉树的定义，左子树和右子树的深度之差的绝对值不超过 1，只有选项 B 对应的二叉树满足这个条件，因此答案选择 B。

17.【答案】D

【考点】B–树

【解析】

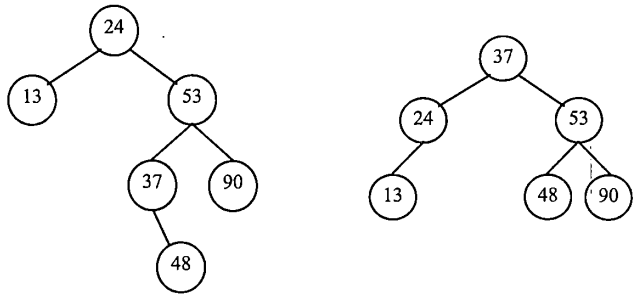
选项 D 是 B+树的特点，而不是 B–树的特点。

18.【答案】C

【考点】平衡二叉树

【解析】

插入 48 得到图 7.6（a）所示的二叉树，此时二叉树根结点的平衡因子由–1 变为–2，失去平衡，需要进行平衡调整，调整后得到图 7.6（b）所示的平衡二叉树。



（a）不平衡　　（b）平衡

图 7.6　平衡二叉树插入 48 后的情况

19.【答案】B

【考点】折半查找

【解析】

折半查找法在查找不成功时，和给定值进行比较的关键字个数最多为 $\lfloor \log_2 n \rfloor + 1$，由题目可知 $n=16$，因此比较次数最多为 5，答案选择 B。

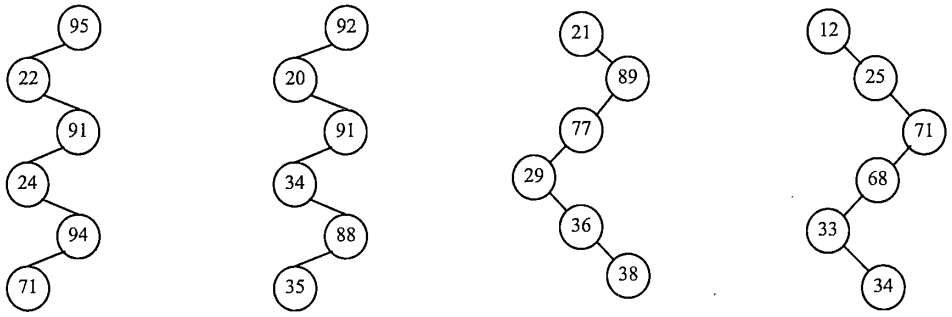
20.【答案】A

【考点】二叉排序树

【解析】

根据二叉排序树的定义，二叉排序树中各结点的值满足“左小右大”的关系。二叉排序树的查找过程是沿从根到某个叶子结点的一条路径进行关键字比较的过程，因此，对于一组给定的关键字序列，在判断是否可以构成二叉排序树中的一条查找路径时，可将该序列按“左小右大”的规则画出对应的二叉树。题目四个选项对应的二叉树如图 7.7（a）~（d）所示。

图 7.7（a）中的 94 位于 91 的左子树，不满足二叉排序树的定义，而其他三个选项对应的二叉排序树均是正确的。因此，答案选择 A。



（a）　（b）　（c）　（d）

图 7.7　第 20 题对应的二叉树

21.【答案】B

【考点】散列表的查找

【解析】

散列表的装填因子 α 定义为：

$$\alpha=\frac{\text{表中填入的记录数}}{\text{散列表的长度}}$$

α 标志散列表的装满程度。直观地看，α 越小，发生冲突的可能性就越小；反之，α 越大，表中已填入的记录越多，再填记录时，发生冲突的可能性就越大，则查找时，给定值需与之进行比较的关键字的个数也就越多。因此，Ⅰ是错误的。而聚集现象是不可避免的，显然，Ⅲ是也错误的。一般来说，可以根据具体问题构造一个“好”的散列函数，使计算出的散列地址的分布尽量均匀，尽可能减少冲突，所以Ⅱ是正确的。故答案选择 B。

22.【答案】B

【考点】平衡二叉树

【解析】

可以根据递归的思路进行求解，将“高度为 n 且所有非叶子结点的平衡因子均为 1 的平衡二叉树”递归分解为以下问题：

（1）只有一个根结点；

（2）左子树为“高度为 n–1 且所有非叶子结点的平衡因子均为 1 的平衡二叉树”；

（3）右子树为“高度为 n–2 且所有非叶子结点的平衡因子均为 1 的平衡二叉树”。

假设用 $f(n)$表示平衡二叉树的高度，根据题目可知 n=6，因此二叉树的结点总数为：

$$\begin{aligned}f(6)&=1+f(5)+f(4)\\&=1+(1+f(4)+f(3))+f(4)\\&=2+2\times(f(3)+f(2))+1)+f(3)\end{aligned}$$

$=4+3\times f(3)+2\times f(2)$

$=4+3\times 4+2\times 2$

$=20$

23.【答案】D

【考点】B–树

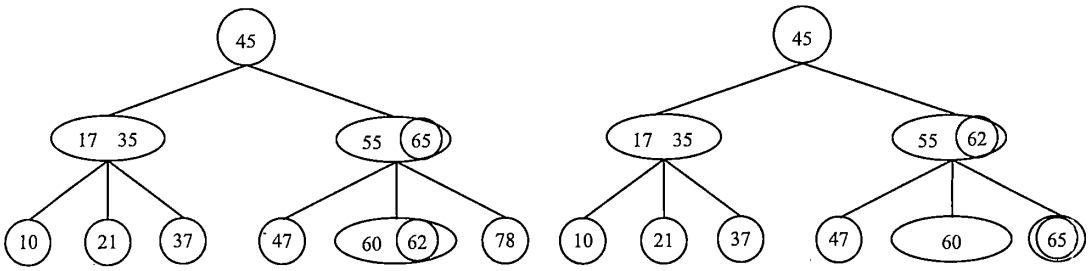
【解析】

根据 3 阶 B–树的定义，结点中所含关键字最多为 2 个，最少为 1 个，题目中要删除的结点 78 位于 B–树的叶子结点中，因此直接删除。因为删除关键字后此结点内的关键字个数为 0，低于最低限，所以需要进行结点中关键字的调整。结点 78 的兄弟结点有两个关键字，可以借一个关键字给原关键字 78 所在的结点。最靠近 78 所在结点的父结点的关键字 65，其兄弟中最靠近它的关键字是 62，用从兄弟结点借来的关键字 62 与父结点的关键字 65 交换，如图 7.8（a）所示，交换后的结果如图 7.8（b）所示。

24.【答案】D

【考点】平衡二叉树

【解析】



（a）删除前　　（b）删除后

图 7.8　B–树的删除

利用 7 个关键字构建平衡二叉树 T，平衡因子为 0 的分支结点个数为 7，构建平衡二叉树过程如图 7.9 所示。

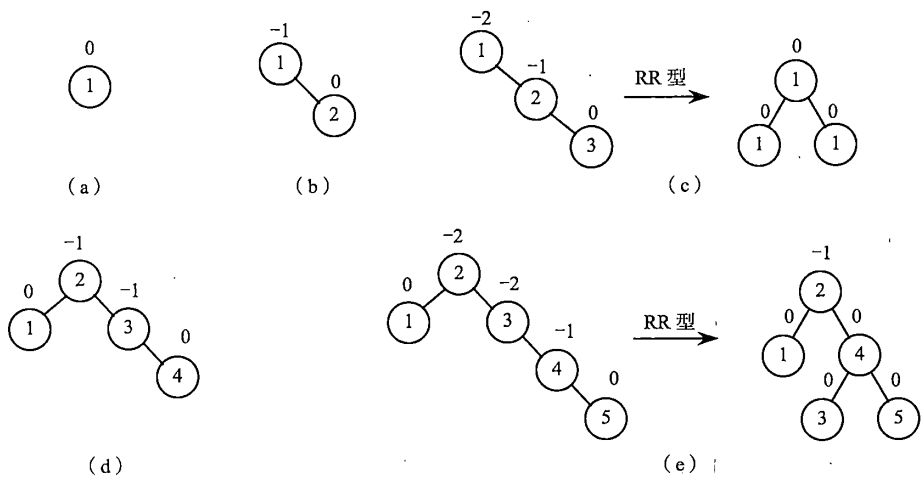


图 7.9　平衡二叉树的创建

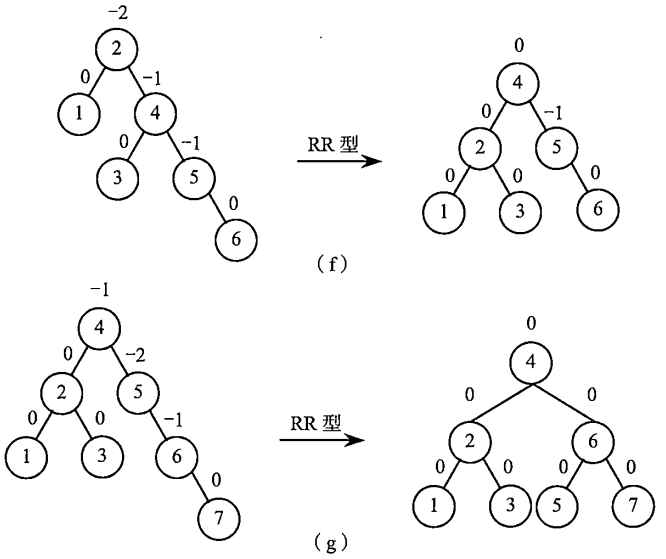


图 7.9　平衡二叉树的创建（续）

25.【答案】C

【考点】二叉排序树

【解析】

（1）当删除一棵二叉排序树的某个叶子结点 v 时，除 v 的父结点的指向 v 的指针被修改为 NULL 外，其他结点均不受影响。此时，如果再将结点 v 插入到原来的二叉排序树中，被删结点仍插到原位，因此结论 II 正确。（2）当删除一棵二叉排序树的某个非叶子结点 v 后，如果再将结点 v 插入到原来的二叉排序树中，结点 v 一定会成为新的二叉排序树的叶子结点，此时的二叉排序树与删除结点之前不同，因此结论 III 正确。基于上述两点分析，答案选择 C。

26.【答案】A

【考点】B–树

【解析】

关键字最少时，应该是结点刚好能分裂的时候。一棵高度为 2 的 5 阶 B–树，根结点只有到达 5 个关键字的时候才能产生分裂，成为高度为 2 的 B–树。

27.【答案】D

【考点】散列表的查找

【解析】

产生堆积现象，即产生了冲突，它对存储效率、散列函数和装填因子均不会有影响，而平均查找长度会因为堆积现象而增大，因此答案选择 D。

28.【答案】D

【考点】B–树

【解析】

对于关键字数量固定的 B–树，要使结点个数达到最多，则使每个结点中的关键字个数尽量少即可。根据 4 阶 B–树的定义，根结点最少含 1 个关键字，非根结点中最少含 4/2–1=1 个关键字，所以每个结点中，关键字数量最少都为 1 个，即每个结点都有 2 个分支。这与二叉排序树很类似，15 个结点正好可以构造一个 4 层的 4 阶 B–树，使得叶子结点全在第 4 层，符合 B–树定义，因此答案选择 D。

29.【答案】D

【考点】平衡二叉树　中序遍历

【解析】

根据中序遍历得到降序序列可以知道，每个结点的左子树的结点的值比该结点的值大，因为没有重复的关键字，所以树中最大元素一定无左子树。

30.【答案】A

【考点】折半查找

【解析】

本题可以借助折半查找的判定树进行直观地分析。根据判定树的定义，树中各结点的值满足“左小右大”的关系。折半查找的过程是沿从根到某个叶子结点的一条路径的关键字比较过程，因此，对于一个正确的折半查找关键字比较序列，其所对应的判定树上一定存在此序列所对应的一条从根到某个叶子结点的查找路径。题目四个选项对应的判定树分别如图 7.10（a）~（d）所示。选项 A 中最后一个关键字 180 位于 200 的右子树，不满足折半查找判定树的定义，而其他三个选项对应的判定树均是正确的。因此答案选择 A。

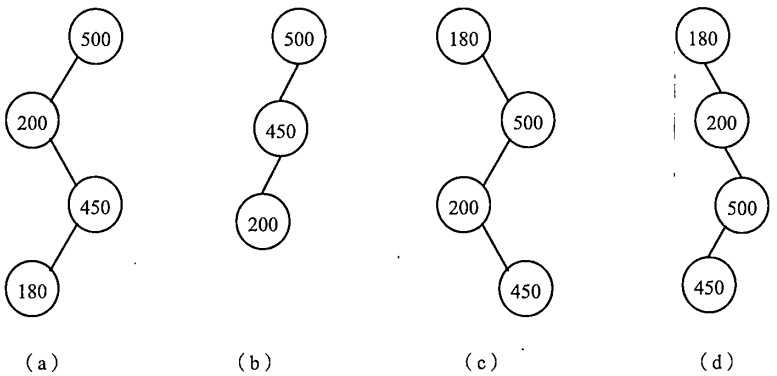


图 7.10　折半查找判定树

31.【答案】B

【考点】折半查找

【解析】

在任何情况下，折半查找都从表的中间记录开始进行比较。而本算法通过 while 循环可以定位到与待查关键字 x 值相隔位置最大为 3 的记录上，这样当 x 接近数组的开头处时，while 循环执行一次便可跳出循环，然后利用之后的判断语句即可完成查找工作，此时比折半查找的比较次数少。

32.【答案】A

【考点】B+树　B–树

【解析】

B+树所有的叶子结点中包含了全部关键字的信息，以及指向含这些关键字记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接，这就为顺序查找提供了方便。而 B–树只适用于随机查找，不适用于顺序查找。

二. 应用题

1. 解答

（1）先画出判定树如图 7.11 所示（$mid=\lfloor(1+12)/2\rfloor=6$）。

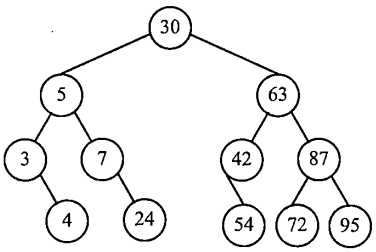


图 7.11　折半查找判定树

（2）查找元素 54，需依次与 30、63、42、54 元素比较。

（3）查找元素 90，需依次与 30、63、87、95 元素比较。

（4）求 *ASL* 之前，需要统计查找每个元素时的比较次数。查找时元素的比较次数总计为：1+2×2+4×3+5×4=37，所以 *ASL*=37/12≈3.08。（注意：此处不能利用教材中的公式（7-3）进行计算，因为公式（7-3）是在判定树为满二叉树的情况下得到的。）

2．解答

从一棵空树开始，按照所给关键字的顺序依次将结点插入到二叉排序树中，具体的插入过程如图 7.12（a）~（k）所示。

在插入完成后，可以进一步验证下插入的结果是否正确。通过对图（k）所示的二叉排序树进行中序遍历，得到了一个升序序列：2，4，7，9，11，12，13，16，17，21，进而说明所创建的二叉排序树是正确的。

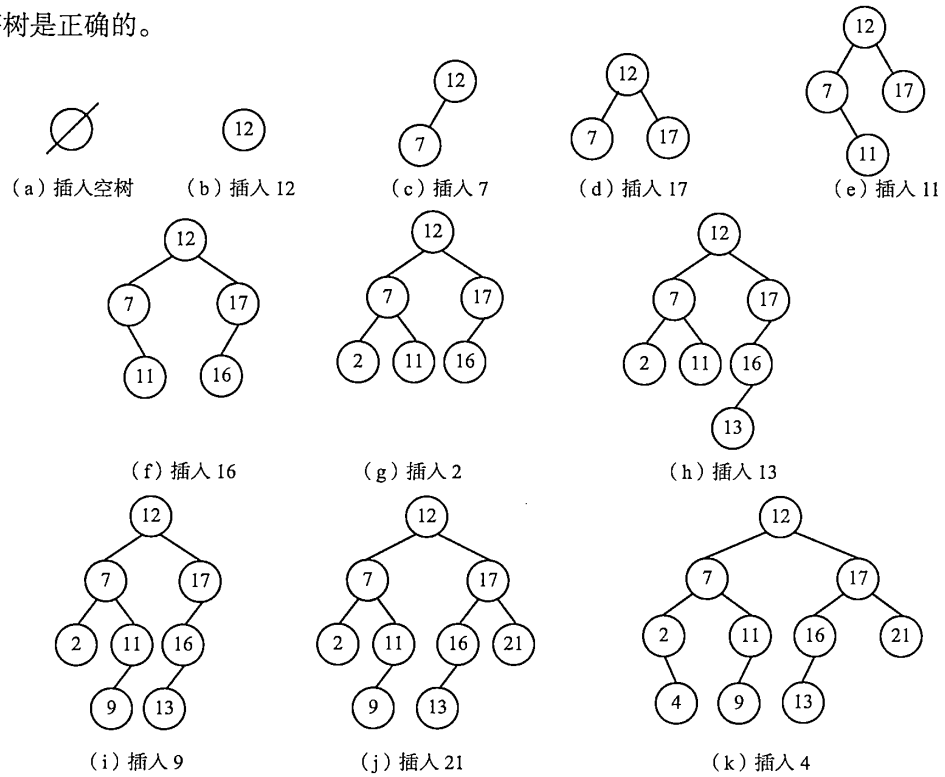


图 7.12　折半查找判定树

3．解答

（1）求得的二叉排序树如图 7.13（a）所示，在等概率情况下查找成功的平均查找长度为：

$$ASL=(1\times1+2\times2+3\times3+4\times3+5\times2+6\times1)/12=42/12=7/2$$

（2）经排序后的表及在折半查找时找到表中元素所经比较的次数对照如下：

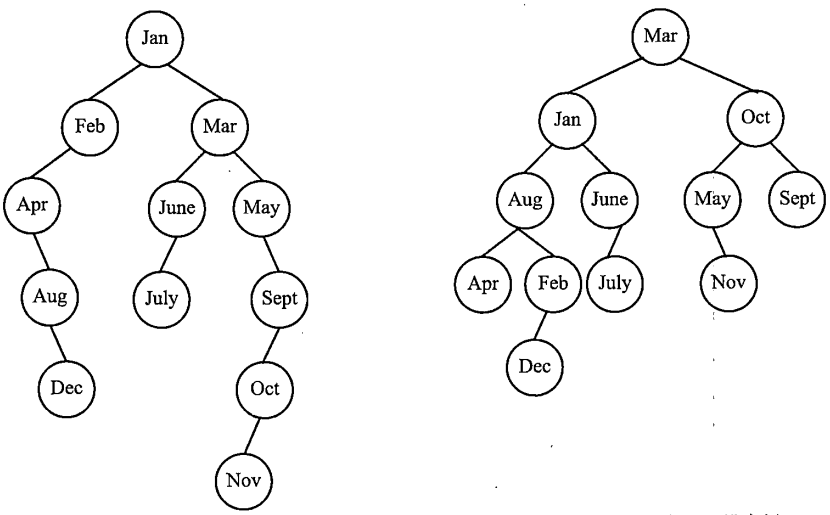
Apr	Aug	Dec	Feb	Jan	July	June	Mar	May	Nov	Oct	Sept
3	4	2	3	4	1	3	4	2	4	3	4

在等概率情况下查找成功的平均查找长度为：

$$ASL=(1\times1+2\times2+3\times4+4\times5)/12=37/12$$

（3）平衡二叉排序树如图 7.13（b）所示，在等概率情况下查找成功的平均查找长度为：

$$ASL=(1\times1+2\times2+3\times4+4\times4+5\times1)/12=19/6$$

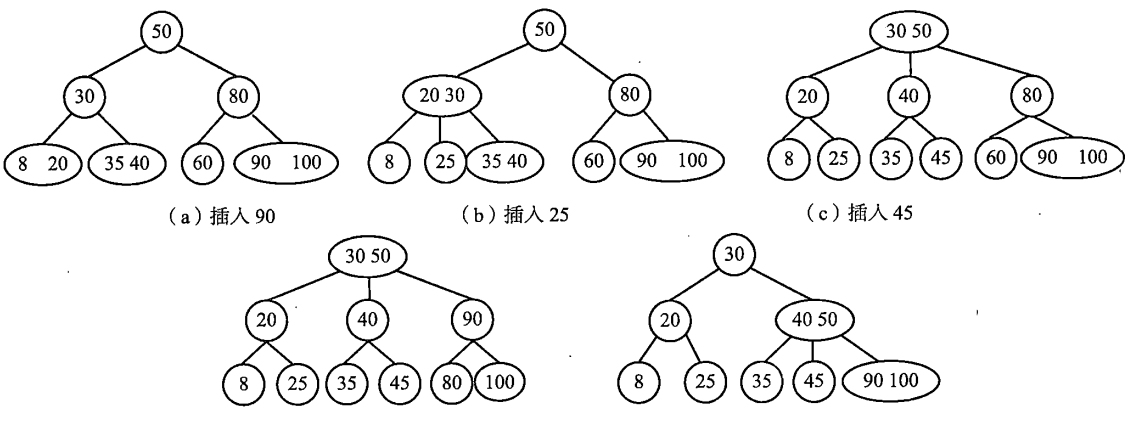


（a）二叉排序树　　（b）平衡二叉排序树

图 7.13　第 3 题答案

4．解答

（1）~（5）各步的操作结果如图 7.14（a）~（e）所示。



（a）插入 90　（b）插入 25　（c）插入 45

（d）删除 60　（e）删除 80

图 7.14　第 4 题答案

5．解答

（1）

$H(10)=10\%16=10$

H（24）=24%16=8

H（32）=32%16=0

H（17）=17%16=1

H（31）=31%16=15

H（30）=30%16=14

H（46）=46%16=14

H（47）=47%16=15

H（40）=40%16=8

H（63）=63%16=15

H（49）=49%16=1

依次计算各个关键字的散列地址，如果没有冲突，将关键字直接存放在相应的散列地址所对应的单元中；否则，用线性探测法处理冲突，直到找到相应的存储单元中。

如对于前 6 个关键字进行计算，$H(10)=10$，$H(24)=8$，$H(32)=0$，$H(17)=1$，$H(31)=15$，$H(30)=14$ 所得散列地址均没有冲突，直接填入所在单元。

而对于第 7 个关键字 46，$H(46)=14$，发生冲突，根据线性探测法，求得下一个地址(14 + 1)%18 = 15，发生冲突，继续探测下一个地址(15 + 1)%18 = 16，此单元为空，所以 46 填入序号为 16 的单元。

同理，可依次填入其他关键字，最终构造结果如表 7.1 所示。

表 7.1　　用线性探测法处理冲突时的散列表

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
比较次数	1	1	6	3					1	2	1				1	1	3	3

（2）要查找一个关键字 *key*，首先用散列函数计算 $H_0=H(key)$，然后进行比较，比较的次数和创建散列表时放置此关键字的比较次数是相同的。

查找 63 时，首先要与 $H(63)$=63%16=15 号单元的内容比较，即 63 与 31 比较，不匹配，然后顺移，与 46、47、32、17、63 相比，一共比较了 6 次。

（3）查找 60 时，首先要与 $H(60)$=60%16=12 号单元的内容比较，但因为 12 号单元为空，所以只比较这一次即可，查找失败。

（4）假定每个关键字的查找概率相等，查找成功时的平均查找长度为：

ASL=（1+1+6+3+1+2+1+1+1+3+3）/11=23/11

6．解答

H（9）=9 % 7=2

H（1）=1 % 7=1

H（23）=23 % 7=2

H（14）=14% 7=0

H（55）=55% 7=6

H（20）=20 % 7=6

H（84）=84 % 7=0

H（27）=27 % 7=6

如对于前两个关键字进行计算，$H(9)=2$，$H(1)=1$，所得散列地址均没有冲突，直接填入所

在单元。

而对于第 3 个关键字，$H(23)=2$，发生冲突，根据二次探测法，求得下一个地址$(2+1^2)\%10=3$，没有冲突，所以 23 填入序号为 3 的单元。

同理，可依次填入其他关键字，最终构造结果如表 7.2 所示。

表 7.2　　用二次探测法处理冲突时的散列表

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

假定每个关键字的查找概率相等，求查找成功时的平均查找长度为：

$$ASL=(1+1+1+2+3+4+1+2)/8=15/8$$

7．解答

（1）

$H(32)=3\times 32 \text{ MOD } 11=8$

$H(13)=3\times 13 \text{ MOD } 11=6$

$H(49)=3\times 49 \text{ MOD } 11=4$

$H(24)=3\times 24 \text{ MOD } 11=6$

$H(38)=3\times 38 \text{ MOD } 11=4$

$H(4)=3\times 4 \text{ MOD } 11=1$

$H(12)=3\times 12 \text{ MOD } 11=3$

如对于前 3 个关键字进行计算，$H(32)=8$，$H(13)=6$，$H(49)=4$，所得散列地址均没有冲突，直接填入所在单元。

而对于第 4 个关键字，$H(24)=6$，发生冲突，根据线性探测法，求得下一个地址$(6+1)\%11=7$，没有冲突，所以 24 填入序号为 7 的单元。

同理，可依次填入其他关键字，最终构造结果如表 7.3 所示。

表 7.3　　用线性探测法处理冲突时的散列表

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	
比较次数		1		1	1	2	1	2	1	2	

等概率下查找成功时和查找失败时的平均查找长度为：

$$ASL_{succ}=(1+1+1+2+1+2+1+2)/8=11/8$$

$$ASL_{unsucc}=(1+2+1+8+7+6+5+4+3+2+1)/11=40/11$$

（2）由散列函数哈希函数 $H(K)=3K \text{ MOD } 11$ 得知散列地址的值域为 0～10，故整个散列表由 11 个单链表组成，用数组 $HT[0...10]$存放各个链表的头指针。如散列地址为 1 的关键字 4 构成一个单链表，链表的头指针保存在 $HT[1]$中；散列地址均为 4 的同义词 49、38 构成一个单链表，链表的头指针保存在 $HT[4]$中。同理，可以构造其他几个单链表，整个散列表的结构如图 7.15 所示。

等概率下查找成功时和查找失败时的平均查找长度为：

$$ASL_{succ}=(1\times 5+2\times 3)/8=11/8$$

$$ASL_{unsucc}=(1+2+1+2+3+1+3+1+3+1+1)/11=19/11$$

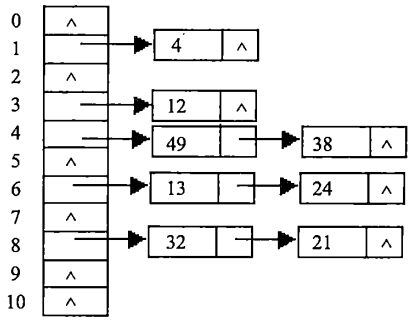


图 7.15　用链地址法处理冲突时的散列表

8．解答

（1）因为装填因子为 0.7，关键字总个数为 7，所以存储空间长度为 L=7/0.7=10，构造的散列函数为：$H(key)=(key \times 3)$MOD7

线性探测再散列函数为：$H(key)=(H(key)+d_i)$ MOD 10, $(d_i=1,2,3,\cdots,9)$。

因此，各关键字在散列表的下标为：

$H(7)=(7 \times 3)$ MOD 7=0

$H(8)=(8 \times 3)$ MOD 7=3

$H(30)=(30 \times 3)$ MOD 7=6

$H(11)=(11 \times 3)$ MOD 7 =5

$H(18)=(18 \times 3)$ MOD 7 =5

$H(9)=(9 \times 3)$ MOD 7= 6

$H(14)=(14 \times 3)$ MOD 7=0

根据上述计算结果，最终构造如表 7.4 所示。

表 7.4　　用线性探测法处理冲突时的散列表

散列地址	0	1	2	3	4	5	6	7	8	9	10
关键字	7	14		8		11	30	18	9		
比较次数	1	2		1		1	1	3	3		

（2）由上表可得，等概率下查找成功时和查找失败时的平均查找长度为：

ASL_{succ}=(1+2+1+1+1+3+3)/7=12/7

ASL_{unsucc}==(3+2+1+2+1+5+4)/7=18/7

9．解答

【答案一】

（1）采用顺序存储结构，数据元素按其查找概率降序排列。

采用顺序查找方法。查找成功时的平均查找长度为

$$ASL=0.35 \times 1+0.35 \times 2+0.15 \times 3+0.15 \times 4=2.1。$$

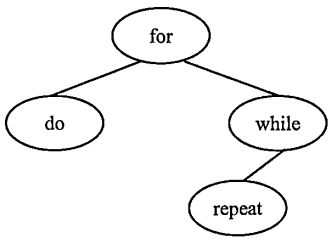
（2）采用链式存储结构，数据元素按其查找概率降序排列，构成单链表。

采用顺序查找方法，查找成功时的平均查找长度为

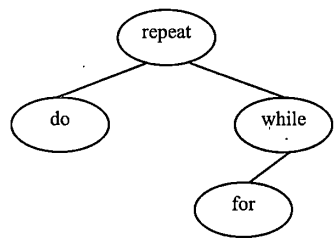
$$ASL=0.35 \times 1+0.35 \times 2+0.15 \times 3+0.15 \times 4=2.1。$$

【答案二】

采用二叉链表存储结构，构造二叉排序树，可得图 7.16 所示的两棵不同形状的二叉排序树。



（a）二叉排序树 1



（b）二叉排序树 2

图 7.16　第 9 题答案

采用二叉排序树的查找方法，查找成功时的平均查找长度为：

$$ASL=0.15\times 1+0.35\times 2+0.35\times 2+0.15\times 3=2.0。$$

三. 算法设计题

1．解答

【算法思想】

为了标记折半查找过程中每一次的查找区间，分别用 low 和 high 来表示当前查找区间的下界和上界，mid 为区间的中间位置。

递归结束的条件有两个：一个是查找失败，一个是查找成功。当 low 大于 high 时，查找失败，返回 0；当 low 小于等于 high 时，将给定值 *key* 与 mid 对应记录的关键字 *R*[mid]进行比较，若相等则查找成功，返回中间位置 mid；若 *key* 小于 *R*[mid]，则对左子表递归查找，否则对右子表递归查找。

【算法描述】

```
int BinSearch_Cur(SSTable ST,KeyType key, int low, int high)
{//在有序表 ST 中折半查找其关键字等于 key 的数据元素，若查找成功，返回 k 所在位置，查找失败返回 0
  if(low>high) return 0;                          //查找不到时返回 0
  if(low<=high)                                   //low 和 high 分别是有序表的下界和上界
  {
    mid=(low+high)/2;                             //mid 取值为 low 和 high 的中间值
    if(ST.R[mid]==key)
      return mid;
    else if(key<ST.R[mid])
      return BinSearch_Cur(ST,key,low,mid-1);     //对左子表递归查找
    else
      return BinSearch_Cur(ST,key,mid+1,high);    //对右子表递归查找
  }
}
```

2．解答

【算法思想】

根据二叉排序树的定义，对二叉树进行递归遍历，左子树的关键字比根节点的关键字小，右子树的关键字比根节点的关键字大，一旦有不满足条件则可判定不是二叉排序树。

通过参数 flag 的值来判定，flag 为 1 表示是二叉排序树，为 0 表示为非二叉排序树，flag 初值为 1。设全局指针变量 pre（初值为 NULL）用来指向遍历过程结点的前驱。

（1）首先递归遍历左子树。（2）如果 pre 为 NULL，则使其指向根结点（中序遍历的第一个结点不必判断）。否则，将当前结点与 pre 所指的前驱结点作比较，如果前驱结点的值小于当前结点的值，则令 pre 指向当前结点；如果前驱结点的值大于当前结点的值，则表示不是二叉排序树，

flag 置为 0。（3）最后递归遍历右子树。

【算法描述】

```
typedef struct BiTNode
{
   TElemtype data;
   struct BiTNode *lchild,*rchild;
}BiTNode,*BiTree;
BiTree pre=NULL;                                    //前驱指针
void JudgeBST(BiTree T,int &flag)
{//判断二叉树 T 是否是二叉排序树，flag 初值为 1
   if(T!=NULL&&flag)
   {
      JudgeBST(T->lchild,flag);                     //中序遍历左子树
      if(pre==NULL) pre=T;                          //中序遍历的第一个结点不必判断
      else if(pre->data<T->data) pre=T;             //前驱指针指向当前结点
      else flag=0;                                  //不是二叉排序树
      JudgeBST(T->rchild,flag);                     //中序遍历右子树
   }
}
```

3. 解答

【算法思想】

首先从根结点开始查找，找到数据值小于 x 的结点后，将其与双亲断开；如果根结点的值小于 x，则沿右子树查找第一个大于等于 x 的结点，找到后与上面同样处理，输出处理后的二叉排序树。

【算法描述】

```
void Print(BSTree T)
{//中序输出以 T 为根的二叉排序树的所有结点
   if(T)
   {
      Print(T->lchild);
      cout<<T->data;
      Print(T->rchild);
   }
}
void PrintAllx(BSTree T, ElemType x)
{//在二叉排序树 T 中，查找值≥x 的结点并输出
   p=T;
   if(p)
   {
      while(p&&p->data<x)                           //沿右分枝向下，找到第一个值≥x 的结点
        p=p->rchild;
          T=p;                                      //T 所指结点是值≥x 的结点的树的根
        if(p)                                       //找到第一个值<x 的结点
        {
           f=p;p=p->lchild;
           while(p&&p->data>=x)                     //沿左分枝向下，找到第一个值<x 的结点
           {
              f=p;                                  //f 是 p 的双亲结点的指针，且指向第一个值≥x 的结点
              p=p->lchild;
```

```
            }
            if(p) f->lchild=NULL;          //双亲与找到的第一个值<x 的结点断开
            Print(T);                      //输出以 T 为根的子树
         }                                 //内层 if(p)
      }                                    //外层 if(p)
   }
```

4. 解答

【算法思想】

根据二叉排序树的特点，首先利用循环进行遍历查找，如果找到，则使其计数加 1；否则判断 X 与当前结点值的大小，小则使指针指向左子树，大则使指针指向右子树。如果遍历结束也无法找到值为 X 的结点，则生成一个新结点，并根据二叉排序树的要求将结点插入树中，即判断遍历结束时结点的前驱结点的值与 X 的大小，大于 X 则插入左子树，小于 X 则插入右子树。

【算法描述】

```
void SearchBST(BiTree &T,int X)
{
   s=new BiTNode;                          //生成一个数据域为 X 的新结点 s
   s->data=X;
   s->count=0;
   s->lchild=s->rchild=NULL;
   if(!T)                                  //如果该树为空则结束该函数
   {
      T=s;
      return;
   }
   f=NULL;
   q=T;
   while(q)
   {
      if(q->data==X)                       //如果找到该值则计数加 1
      {
         q->count++;
         return ;
      }
      f=q;
      if(X<q->data=                        //X 小于当前结点值，使指针指向左子树
         q=q->lchild;
      else                                 //X 大于当前结点值，使指针指向右子树
         q=q->rchild;
   }
   if(f->data>X)                           //如果找不到，就将新结点插入树中
      f->lchild=s;
   else
      f->rchild=s;
}
```

5. 解答

【算法思想】

设根结点的层次为 1，循环遍历平衡二叉树，每遍历一层，高度加 1，直到遍历到层数最大的

子树的叶子结点，所得高度值就是平衡二叉树的高度。遍历原则为当结点的平衡因子 b 为 0 时，任选左右某一分支向下遍历；若 b 不为 0，则当 b=1 时沿左分支向下遍历，b=—1 时沿右分支向下遍历。

【算法描述】

```
int Height(BSTree T)
{//求平衡二叉树 T 的高度
  level=0;                                    //记录树的高度
  p=T;
  while(p)
  {
    level++;                                  //树的高度增 1
    if(p->b<0)p=p->rchild;                    //b=-1 沿右分支向下
    else p=p->lchild;                         //b>=0 沿左分支向下
  }
  return level;                               //返回平衡二叉树的高度
}
```

6. 解答

【算法思想】

对于插入算法：首先计算 K 的散列地址，若该单元为空，则将此元素放入此空位中；若单元非空，但与给定的 K 值不等，则采用链地址法处理冲突，将该元素插入该地址所表示的链表末尾。

对于删除算法：首先计算 K 的散列地址，若该单元为空，则表明该元素不存在，返回 false；若单元非空，但与给定的 K 值不等，遍历对应的单链表进行查找，查找成功后在链表中删除此结点。

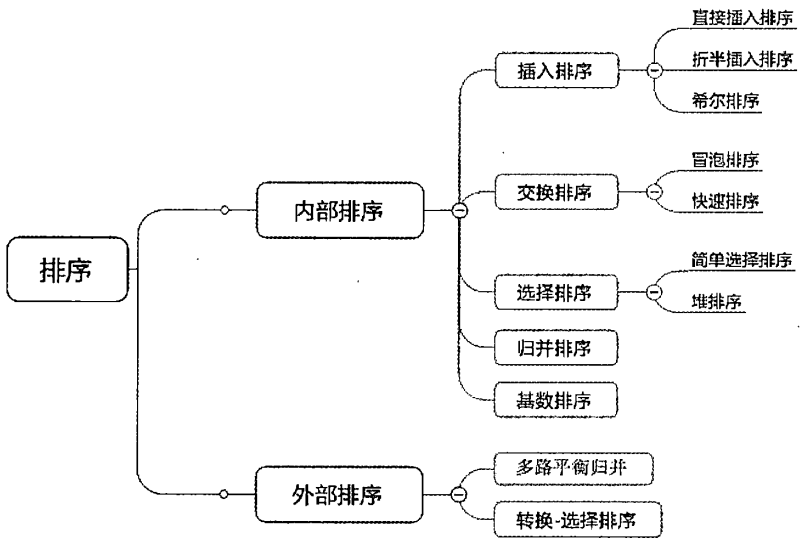
【算法描述】

```
bool Insert_K( )
{
  cin>>data;
  ant=H(data);
  p=HT[ant];                                  //找到该散列地址代表的位置
  while(p->next)                              //确定插入位置
  {
    if(p->next->data==data)
      return false;
    p=p->next;
  }
  s=newLNode;
  s->data=data;
  s->next=p->next;
  p->next=s;                                  //插入该结点置于链表尾部
  return true;
}
bool Delete_K( )
{
  cin>>data;
  ant=H(data);
  p=HT[ant];                                  //找到该散列地址代表的位置
  while(p->next)                              //确定删除位置
  {
```

```
        if(p->next->data==data)
        {
            s=p->next;
            p->next=s->next;
            delete s;                                    //删除该结点
            return true;
            }
            p=p->next;                                   //遍历下一个结点
    }
    return false;
}
```

第8章 排序

【知识导图】



【学习目标】

1. 插入排序的基本思想是：每一趟将一个待排序的记录，按其关键字的大小插入到已经排好序的一组记录的适当位置上，直到所有待排序记录全部被插入为止。根据查找待排序记录插入位置的方法的不同，又可分为不同的插入排序方法。直接插入排序采用顺序查找法，折半查找采用折半查找法，希尔排序从“减少记录个数”和“序列基本有序”两个方面对直接插入排序进行了改进。就平均性能来说，折半插入排序优于直接插入排序。但记录的初始排列为正序或接近正序时，直接插入排序比折半插入排序执行的关键字比较次数要少。希尔排序的时间复杂度较直接插入排序低，但希尔排序的时间复杂度是所取“增量”序列的函数，这涉及一些数学上尚未解决的难题，到目前为止尚未有人求得一种最好的增量序列。

2. 交换排序的基本思想是：两两比较待排序记录的关键字，一旦发现两个记录不满足次序要求时则进行交换，直到整个序列全部满足要求为止。冒泡排序是一种最简单的交换排序方法，快速排序是一种先进的交换排序方法，其适合初始记录无序、n 较大时的情况。平均情况下，快速排序的时间复杂度为 $O(n\log_2 n)$，当 n 较大时，在平均情况下快速排序是所有内部排序方法中速度最快的一种。

3. 选择排序的基本思想是：每一趟从待排序的记录中选出关键字最小的记录，按顺序放在已排序的记录序列的最后，直到全部排完为止。选择排序包括简单选择排序和堆排序，这两种方法的主要区别是选取最小的数所依据的规则不同。简单选择排序是通过简单的顺序遍历方法确定最小数，其时间复杂度为 $O(n^2)$；堆排序是利用堆这种数据结构的性质，通过堆顶元素交换、堆调整等一系列操作来完成选择，其时间复杂度为 $O(n\log_2 n)$，属于先进的排序方法。

4. 归并排序是将两个或两个以上的有序表合并成一个有序表的过程。将两个有序表合并成一个有序表的过程称为 2-路归并，2-路归并最为简单和常用。归并排序时间复杂度为 $O(n\log_2 n)$，属于先进的排序方法。

5. 基数排序不需要比较关键字的大小，它是根据关键字中各位的值，通过对待排序记录进行若干趟“分配”与“收集”来实现排序的，是一种借助于多关键字排序的思想对单关键字排序的方法。

掌握与排序相关的基本概念，如：关键字比较次数、数据移动次数、稳定性、内部排序、外部排序。深刻理解各种内部排序方法的基本思想、特点、算法的实现及其性能分析，能从时间、空间、稳定性各个方面对各种排序方法做综合比较，并能加以灵活应用。掌握外部排序方法中败者树的建立及归并方法，掌握置换-选择排序的过程和最佳归并树的构造方法。

8.1　习题

一、单项选择题

1. 从未排序序列中依次取出元素与已排序序列中的元素进行比较，将其放入已排序序列的正确位置上的方法，这种排序方法称为（　　）。

A. 归并排序　　B. 冒泡排序　　C. 插入排序　　D. 选择排序

2. 从未排序序列中挑选元素，并将其依次放入已排序序列（初始时为空）的一端的方法，称为（　　）。

A. 归并排序　　B. 冒泡排序　　C. 插入排序　　D. 选择排序

3. 对 n 个不同的关键字由小到大进行冒泡排序，在下列（　　）情况下比较的次数最多。

A. 从小到大排列好　　B. 从大到小排列好的　　C. 元素无序　　D. 元素基本有序

4. 对 n 个不同的排序码进行冒泡排序，在元素无序的情况下比较的次数最多为（　　）。

A. $n+1$　　B. n　　C. $n-1$　　D. $n(n-1)/2$

5. 快速排序在下列（　　）情况下最易发挥其长处。

A. 被排序的数据中含有多个相同排序码

B. 被排序的数据已基本有序

C. 被排序的数据完全无序

D. 被排序的数据中的最大值和最小值相差悬殊

6. 对 n 个关键字作快速排序，在最坏情况下，算法的时间复杂度是（　　）。

A. $O(n)$　　B. $O(n^2)$　　C. $O(n\log_2 n)$　　D. $O(n^3)$

7. 若一组记录的排序码为{46, 79, 56, 38, 40, 84}，则利用快速排序的方法，以第一个记录为基准得到的一次划分结果为（　　）。

A. 38, 40, 46, 56, 79, 84　　B. 40, 38, 46, 79, 56, 84

C. 40, 38, 46, 56, 79, 84　　D. 40, 38, 46, 84, 56, 79

8．下列关键字序列中，（　　）是堆。

A．16, 72, 31, 23, 94, 53　　B．94, 23, 31, 72, 16, 53

C．16, 53, 23, 94, 31, 72　　D．16, 23, 53, 31, 94, 72

9．堆是一种（　　）排序。

A．插入　　B．选择　　C．交换　　D．归并

10．堆的形状是一棵（　　）。

A．二叉排序树　　B．满二叉树　　C．完全二叉树　　D．平衡二叉树

11．若一组记录的排序码为{46, 79, 56, 38, 40, 84}，则利用堆排序的方法建立的初始堆为（　　）。

A．79, 46, 56, 38, 40, 84　　B．84, 79, 56, 38, 40, 46

C．84, 79, 56, 46, 40, 38　　D．84, 56, 79, 40, 46, 38

12．下述几种排序方法中，要求内存最大的是（　　）。

A．希尔排序　　B．快速排序　　C．归并排序　　D．堆排序

13．下述几种排序方法中，（　　）是稳定的排序方法。

A．希尔排序　　B．快速排序　　C．归并排序　　D．堆排序

14．数据表中有 10 000 个元素，如果仅要求求出其中最大的 10 个元素，则采用（　　）算法最节省时间。

A．冒泡排序　　B．快速排序　　C．简单选择排序　　D．堆排序

15．下列排序算法中，（　　）不能保证每趟排序至少能将一个元素放到其最终的位置上。

A．希尔排序　　B．快速排序　　C．冒泡排序　　D．堆排序

16.【2009 年第 9 题】已知关键字序列{5,8,12,19,28,20,15,22}是小根堆，插入关键字 3，调整后得到的小根堆是（　　）。

A．3，5，12，8，28，20，15，22，19　　B．3，5，12，19，20，15，22，8，28

C．3，8，12，5，20，15，22，28，19　　D．3，12，5，8，28，20，15，22，19

17.【2009 年第 10 题】若数据元素序列{11,12,13,7,8,9,23,4,5}是采用下列排序方法之一得到的第二趟排序后的结果，则该排序算法只能是（　　）。

A．冒泡排序　　B．插入排序　　C．选择排序　　D．归并排序

18.【2010 年第 10 题】采用递归方式对顺序表进行快速排序，下列关于递归次数的叙述中，正确的是（　　）。

A．递归次数与初始数据的排列次数无关

B．每次划分后，先处理较长的分区可以减少递归次数

C．每次划分后，先处理较短的分区可以减少递归次数

D．递归次数与每次划分后得到的分区处理顺序无关

19.【2010 年第 11 题】对一组数据{2,12,16,88,5,10}进行排序，若前三趟排序结果如下:

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}。

则采用的排序方法可能是（　　）。

A．冒泡排序法　　B．希尔排序法　　C．归并排序法　　D．基数排序法

20.【2011 年第 10 题】为实现快速排序算法，待排序序列宜采用的存储方式是（　　）。

A．顺序存储　　B．散列存储　　C．链式存储　　D．索引存储

21.【2011 年第 11 题】已知序列{25,13,10,12,9}是大根堆，在序列尾部插入新元素 18，将其再调整为大根堆，调整过程中元素之间进行的比较次数是（　　）。

A．1　　B．2　　C．4　　D．5

22.【2012 年第 10 题】排序过程中，对尚未确定最终位置的所有元素进行一遍处理称为一趟排序。下列排序方法中，每一趟排序结束时都至少能够确定一个元素最终位置的方法是（　　）。

I．简单选择排序　　II．希尔排序　　III．快速排序

IV．堆排序　　V．归并排序

A．仅 I、III、IV　　B．仅 I、III、V　　C．仅 II、III、IV　　D．仅 III、IV、V

23.【2012 年第 11 题】对同一待排序序列分别进行折半插入排序和直接插入排序，两者之间可能的不同之处是（　　）。

A．排序的总趟数　　B．元素的移动次数

C．使用辅助空间的数量　　D．元素之间的比较次数

24.【2013 年第 11 题】若对给定的关键字序列{110,119,007,911,114,120,122}进行基数排序，则第 2 趟分配收集后得到的关键字序列是（　　）。

A．{007,110,119,114,911,120,122}

B．{007,110,119,114,911,122,120}

C．{007,110,911,114,119,120,122}

D．{110,120,911,122,114,007,119}

25.【2014 年第 10 题】用希尔排序方法对一个数据序列进行排序时，若第 1 趟排序结果为 9，1，4，13，7，8，20，23，15，则该趟排序采用的增量（间隔）可能是（　　）。

A．2　　B．3　　C．4　　D．5

26.【2014 年第 11 题】下列选项中，不可能是快速排序第 2 趟排序结果的是（　　）。

A．{2,3,5,4,6,7,9}　　B．{2,7,5,6,4,3,9}　　C．{3,2,5,4,7,6,9}　　D．{4,2,3,5,7,6,9}

27.【2015 年第 9 题】下列排序算法中元素的移动次数和关键字的初始排列次序无关的是（　　）。

A．直接插入排序　　B．冒泡排序　　C．基数排序　　D．快速排序

28.【2015 年第 10 题】已知小根堆为{8,15,10,21,34,16,12}，删除关键字 8 之后需重建堆，在此过程中，关键字之间的比较数是（　　）。

A．1　　B．2　　C．3　　D．4

29.【2015 年第 11 题】希尔排序的组内排序采用的是（　　）。

A．直接插入排序　　B．折半插入排序　　C．快速排序　　D．归并排序

30.【2016 年第 11 题】对 10TB 的数据文件进行排序，应使用的方法是（　　）。

A．希尔排序　　B．堆排序　　C．快速排序　　D．归并排序

二、应用题

1．设待排序的关键字序列为{12, 2, 16, 30, 28, 10, 16*, 20, 6, 18}，试分别写出使用以下排序方法，每趟排序结束后关键字序列的状态。

（1）直接插入排序。

（2）折半插入排序。

（3）希尔排序（增量选取 5、3 和 1）。

（4）冒泡排序。

（5）快速排序。

（6）简单选择排序。

（7）堆排序。

（8）二路归并排序。

2. 给出如下关键字序列{321, 156, 57, 46, 28, 7, 331, 33, 34, 63}，试按链式基数排序方法，列出每一趟分配和收集的过程。

3. 对输入文件（101，51，19，61，3，71，31，17，19，100，55，20，9，30，50，6，90）；当 k=6 时，使用置换-选择算法，写出建立的初始败者树及生成的初始归并段。

三、算法设计题

1. 试以单链表为存储结构，实现简单选择排序算法。

2. 有 n 个记录存储在带头结点的双向链表中，现用双向冒泡排序法对其按升序进行排序，请写出这种排序的算法。（注：双向冒泡排序即相邻两趟排序向相反方向冒泡。）

3. 设有顺序放置的 n 个桶，每个桶中装有一粒砾石，每粒砾石的颜色是红、白、蓝之一。求重新安排这些砾石，使得所有红色砾石在前，所有白色砾石居中，所有蓝色砾石在后，重新安排时对每粒砾石的颜色只能看一次，并且只允许交换操作来调整砾石的位置。

4. 编写算法，对 n 个关键字取整数值的记录序列进行整理，以使所有关键字为负值的记录排在关键字为非负值的记录之前，要求：

（1）采用顺序存储结构，至多使用一个记录的辅助存储空间；

（2）算法的时间复杂度为 O(n)。

5. 借助于快速排序的算法思想，在一组无序的记录中查找给定关键字值等于 *key* 的记录。设此组记录存放于数组 r[1...n]中。若查找成功，则返回该记录在 r 数组中的位置，否则显示“not find”信息。

6. 有一种简单的排序算法，叫做计数排序。这种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。必须注意的是，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字比该记录的关键字小。假设针对某一个记录，统计出的计数值为 c，那么，这个记录在新的有序表中的合适的存放位置即为 c。

（1）给出适用于计数排序的顺序表定义；

（2）编写实现计数排序的算法；

（3）对于有 n 个记录的表，关键字比较次数是多少？

（4）与简单选择排序相比较，这种方法是否更好？为什么？

8.2　答案及解析

一. 单项选择题

1. C	2. D	3. B	4. D	5. C	6. B	7. C	8. D	9. B	10. C
11. B	12. C	13. C	14. D	15. A	16. A	17. B	18. D	19. A	20. A
21. B	22. A	23. D	24. C	25. B	26. C	27. C	28. C	29. A	30. D

1.【答案】C

【考点】插入排序

【解析】

插入排序的基本思想是：每一趟将一个待排序的记录，按其关键字的大小插入到已经排好序的一组记录的适当位置上，直到所有待排序记录全部插入为止。

2.【答案】D

【考点】选择排序

【解析】

选择排序的基本思想：每一趟从待排序的记录中选出关键字最小的记录，按顺序放在已排序的记录序列的最后，直到全部排完为止。

3.【答案】B

【考点】冒泡排序

【解析】

冒泡排序的最坏情况是当初始序列为逆序时的情况，这种情况下，需进行 $n-1$ 趟排序，总的关键字比较次数 KCN 和记录移动次数 RMN（每次交换都要移动 3 次记录）分别为：

$$KCN=\sum_{i=n}^{2}(i-1)=n(n-1)/2\approx n^2/2$$

$$RMN=3\sum_{i=n}^{2}(i-1)=3n(n-1)/2\approx 3n^2/2$$

4.【答案】D

【考点】冒泡排序

【解析】

见第 3 题解析。

5.【答案】C

【考点】快速排序

【解析】

从快速排序算法的递归树可知，快速排序的趟数取决于递归树的深度。

最好情况：当被排序的数据完全无序时，每一趟排序后都能将记录序列均匀地分割成两个长度大致相等的子表，类似折半查找。在 n 个元素的序列中，对枢轴定位所需时间为 $O(n)$。若设 $T(n)$

是对 n 个元素的序列进行排序所需的时间，而且每次对枢轴正确定位后，正好把序列划分为长度相等的两个子表，$T(n)=O(n\log_2 n)$。

最坏情况：在待排序序列已经排好序的情况下，其递归树成为单支树，每次划分只得到一个比上一次少一个记录的子序列。这样，必须经过 $n-1$ 趟才能将所有记录定位，而且第 i 趟需要经过 $n-i$ 次比较。这时，$T(n)=O(n^2)$。这种情况下，快速排序的速度已经退化到简单排序的水平。

6.【答案】B

【考点】快速排序

【解析】

快速排序的平均时间复杂度为 $O(n\log_2 n)$，但在最坏情况下，即关键字基本排好序的情况下，其递归树成为单支树，每次划分只得到一个比上一次少一个记录的子序列。这样，必须经过 $n-1$ 趟才能将所有记录定位，而且第 i 趟需要经过 $n-i$ 次比较。这时，$T(n)=O(n^2)$。

7.【答案】C

【考点】快速排序

【解析】

第一趟快速排序过程如图 8.1 所示。

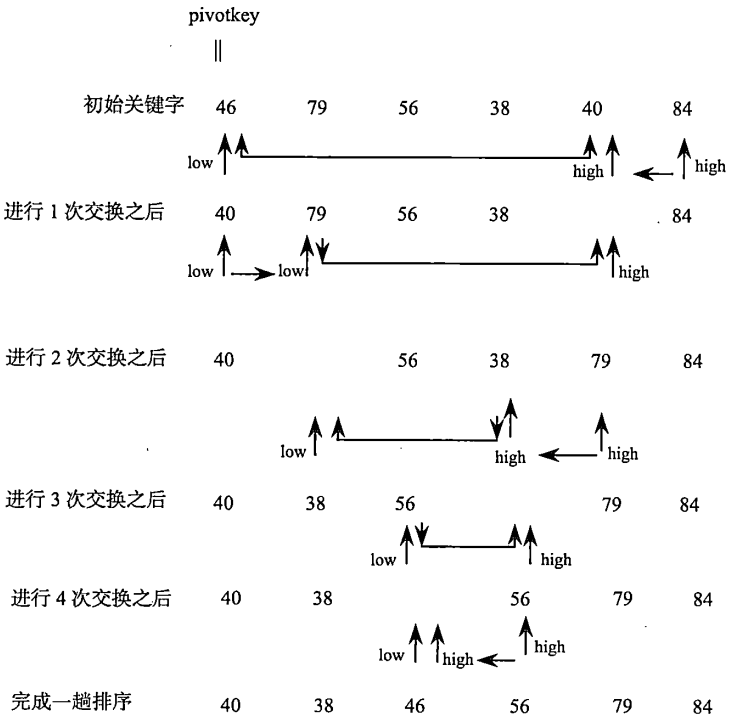


图 8.1 第一趟快速排序

8.【答案】D

【考点】堆排序

【解析】

n 个元素的序列 $\{k_1,k_2,\cdots,k_n\}$ 称之为堆，当且仅当满足以下条件时：

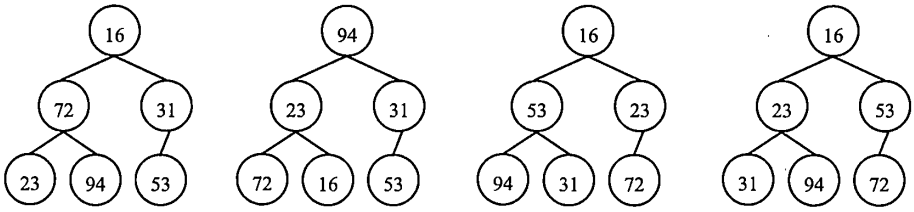
（1）$k_i \geqslant k_{2i}$ 且 $k_i \geqslant k_{2i+1}$。

或：

（2）$k_i \leqslant k_{2i}$ 且 $k_i \leqslant k_{2i+1}$ $(1 \leqslant i \leqslant \lfloor n/2 \rfloor)$。

若将和此序列对应的一维数组（即以一维数组做此序列的存储结构）看成一个完全二叉树，则堆实质上是满足如下性质的完全二叉树：树中所有非终端结点的值均不大于（或不小于）其左、右孩子结点的值。

四个选项所对应的完全二叉树分别如图 8.2（a）~（d）所示。



（a）　（b）　（c）　（d）

图 8.2　四个选项对应的完全二叉树

9.【答案】B

【考点】堆排序

【解析】

堆排序是一种树形选择排序，在排序过程中，将待排序的记录 $r[1, \cdots, n]$ 看成一棵完全二叉树的顺序存储结构，利用完全二叉树中双亲结点和孩子结点之间的内在关系，在当前无序的序列中选择关键字最大（或最小）的记录。

10.【答案】C

【考点】堆排序

【解析】

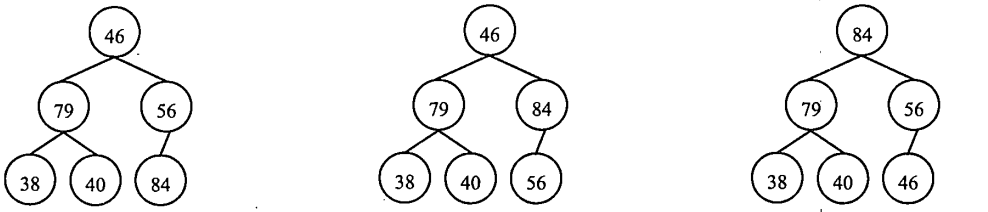
见第 8 题解析。

11.【答案】B

【考点】建初堆

【解析】

初始序列对应的完全二叉树如图 8.3（a）所示，从其最后一个非终端结点即元素 56 开始筛选，依次将以 56、79 和 46 为根的子树都调整为堆。首先从第 3 个元素 56 开始，由于 56<84，需要交换，被筛选之后序列的状态如图 8.3（b）所示；对于第 2 个元素 79，因其值不小于左右子树的根的值，则无须交换；最后对根元素 46 进行筛选，将其与两个子树的根的较大者即 84 交换，46 再与 56 交换，最后得到如图 8.3（c）所示的大根堆。



（a）无序序列　（b）56 被筛选之后的状态　（c）46 被筛选之后建成的大根堆

图 8.3　建初堆的过程

12.【答案】C

【考点】内部排序方法的比较。

【解析】

堆排序、希尔排序的空间复杂度是 O(1)，快速排序的空间复杂度是 $O(\log_2 n)$，归并排序的空间复杂度是 $O(n)$，因此答案选择 C。

13.【答案】C

【考点】内部排序方法的比较

【解析】

不稳定的排序方法包括：快速排序、希尔排序、堆排序；稳定的排序方法包括：直接插入排序、折半插入排序、冒泡排序、简单选择排序、归并排序、基数排序。因此正确答案选 C。

记忆小窍门：不稳定的三个排序方法可以简记为“快些（希）堆”，可以想成当我们试图将很多材料堆积起来时，如果只为了追求“快一些”，可能会导致堆积起来的材料不稳定，容易倒塌。除了简记为“快些（希）堆”的三个排序方法，其他所有的排序方法都是稳定的，因此答案选择 C。

14.【答案】D

【考点】内部排序方法的比较

【解析】

平均情况下，冒泡排序和简单选择排序的时间复杂度是 $O(n^2)$，快速排序和堆排序的的时间复杂度是 $O(n\log_2 n)$，但快速排序每趟排序并不能确定一个最大数，所以答案选择 D。

15.【答案】A

【考点】内部排序

【解析】

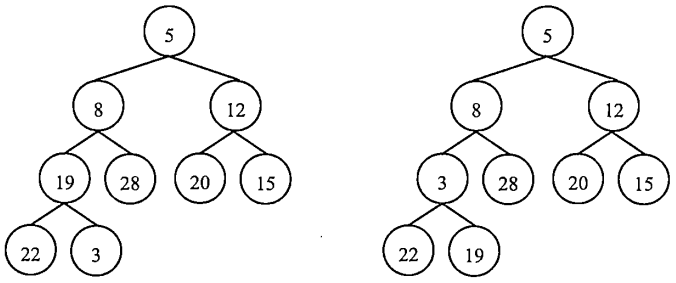
快速排序的每趟排序将作为枢轴的元素放到最终位置，冒泡排序和堆排序的每趟排序可以将最大或最小的元素放到最终位置。所以答案选择 A。

16.【答案】A

【考点】堆的调整

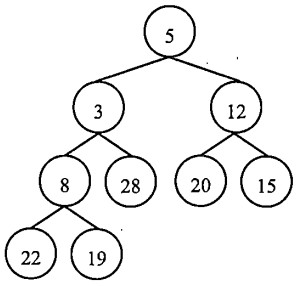
【解析】

原始的小根堆在插入关键字 3 之后，如图 8.4（a）所示，不再满足小根堆的要求，需要利用筛选法依次进行调整。首先从最后一个非终端结点开始筛选，即从第 4 个元素 19 开始，由于 19>3，19 与 3 交换，调整后得到图 8.4（b）；之后对于第 3 个元素 12 不大于其左、右子树根的值，无须交换；对于第 2 个元素 8，由于 8>3，8 与 3 交换，调整后得到图 8.4（c）；最后，对于第 1 个元素 5，由于 5>3，5 与 3 交换，调整后得到小根堆如图 8.4（d）。因此，答案选择 A。

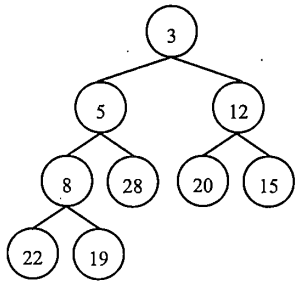


（a）插入 3　　（b）19 被筛选之后的状态

图 8.4　调整堆的过程



（c）8 被筛选之后的状态



（d）5 被筛选之后的小根堆

图 8.4　调整堆的过程（续）

17.【答案】B

【考点】内部排序的比较

【解析】

此题可用排除法。

（1）假设是冒泡排序或者选择排序，则第二趟排序后的结果应有 2 个最大或者最小的元素，题目中所给的序列与此不符，因此，排除选项 A 和 C。

（2）假设是归并排序，则第二趟排序后，应形成 2 个长度为 4 的和 1 个长度为 1 的有序子序列，题目中所给的序列与此不符，因此，排除选项 D。

（3）假设是插入排序，则第二趟排序后，可能出现题目中的情况，因此，选择答案 B。

18.【答案】D

【考点】快速排序。

【解析】

快速排序的递归次数是与初始数据的排列次序有关的。最坏情况是在待排序序列已经排好序的情况下，其递归树成为单支树；最好情况是在序列基本无序的情况下，每一趟排序后都能将记录序列均匀地分割成两个长度大致相等的子表，类似折半查找。因此，选项 A 是错误的。对于选项 B、C 和 D，可以进行如下分析。

快速排序的递归函数可以简写为：

```
void QSort(SqList &L,int low,int high)
{
    QSort(L,l,p-1);          //① 对左子表递归排序
    QSort(L,p+1,h);          //② 对右子表递归排序
}
```

此函数的递归次数由 l 和 h 决定的。设快速排序的递归次数设为 $F(l,h)$，则按照上述代码中①②句的执行次序有：

递归次数 $F(l,h)=F(l,p-1)+F(p+1,h)$　③

如果将①②句颠倒，则有：

递归次数 $F(l,h)=F(p+1,h)+F(l,p-1)$　④

显然③和④式是相等的，因此递归次数与每次划分后得到的分区处理顺序无关，选项 B 和 C 都是错误的。因此，答案选择 D。

19.【答案】A

【考点】内部排序的比较

【解析】

（1）假设是冒泡排序，每一趟排序均能确定一个最大（或最小）的排序码并放到目标位置，因此，采用冒泡排序可能出现题目中的情况，选项 A 正确。

（2）假设是希尔排序，从三趟排序结果中找不出希尔排序使用的增量序列，因此，排除选项 B。

（3）假设是归并排序，第一趟归并的结果应该是 2，12，16，88，5，10，因此，排除选项 C。

（4）假设是基数排序，第一趟分配收集的结果应该是 10，2，12，5，16，88，因此，排除选项 D。

20.【答案】A

【考点】快速排序

【解答】

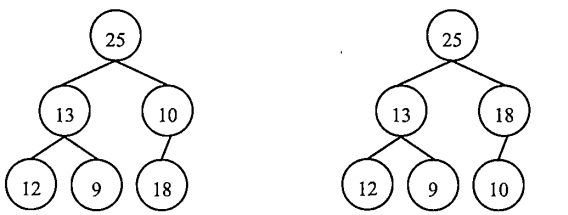
快速排序采用顺序存储结构，在选定枢轴记录后，可以利用数组的下标定位到某个记录，高效地以左右交替的形式将待排序记录分成左、右两个子表，然后分别对左、右子表进行递归，完成排序。而采用链式或其他存储结构，无法利用数组的下标快速定位到某个记录，排序效率较低，所以不宜采用。

21.【答案】 B

【考点】堆的调整

【解析】

原始的大根堆在插入关键字 18 之后，如图 8.5（a）所示，不再满足大根堆的要求，需要进行调整。首先 18 与 10 比较，由于 18>10，18 与 10 交换，调整后得到图 8.5（b）；之后 25 与 18 比较，不交换，总计比较 2 次。



（a）插入 18　　（b）18 被筛选之后的状态

图 8.5 调整堆的过程

22.【答案】A

【考点】内部排序的比较

【解析】

简单选择排序每次选择未排序的序列中最小的元素，并将其放入最终位置；希尔排序每次是对划分的子表进行排序，使得局部有序，不能保证每一趟排序结束都能确定一个元素的最终位置；快速排序每一趟排序结束后都将枢轴元素放入最终位置；堆排序每次都将大根堆的根结点与表尾结点交换，再对堆进行调整，确定其最终位置；归并排序每趟对子表进行两两归并得到若干局部有序的结果，无法确定最终位置。所以答案选择 A。

23.【答案】D

【考点】折半插入排序 直接插入排序

【解析】

折半插入排序和直接插入排序都是将待插入元素插入前面的有序子序列，区别是：确定当前记录在前面有序序列中的位置时，直接插入排序采用顺序查找，折半插入排序采用折半查找。折

半插入排序的比较次数与序列初态无关，而直接插入排序的比较次数与序列初态有关。

24.【答案】C

【考点】基数排序

【解析】

基数排序是一种最低位优先的分配排序算法，第 1 趟排序是按照个位数字来排序的，结果为：{110,120,911,122,114,007,119}；第 2 趟排序是按照十位数字的大小进行排序的，结果为：{007,110,911,114,119,120,122}。所以答案选择 C。

25.【答案】B

【考点】希尔排序

【解析】

希尔排序将相隔增量整数倍的元素组成一组，并使用插入排序在组内进行排序。一趟排序后，各组内元素已有序。假设增量为 d，当 d=2 时，将元素分为两个子序列分别为{9,4,7,20,15}和{1,13,8,23}，显然两个子序列均是无序的，因此选项 A 是错误的。当 d=3 时，将元素分为三个子序列分别为{9,13,20}、{1,7,23}和{4,8,15}，这三个子序列都是递增有序的，故 B 选项符合题目要求。当 d=4 时，将元素分为四个子序列分别为{9,7,15}、{1,8}、{4,20}和{13,23}，显然{9,7,15}是无序的，选项 C 是错误的。当 d=5 时，将元素分为五个子序列分别为{9,8}、{1,20}、{4,23}、{13,15}和{7}，显然子序列{9,8}不是一个递增的，因此选项 D 也是错误的。

26.【答案】C

【考点】快速排序

【解析】

快速排序每一趟排序结束后都将枢轴元素放入最终位置，且枢轴之前的所有元素均小于它，其后的元素均大于它。第 i 趟排序结束，至少有 i 个元素放入最终位置，因此第 2 趟排序结束，至少有 2 个元素放入最终位置。选项 A 和 B 中，2 和 9 可以充当枢轴；选项 D 中，5 和 9 可以充当枢轴；选项 C 中只能找到一个元素 9 可以充当枢轴元素，枢轴个数不符合题目要求，因此答案选择 C。

27.【答案】C

【考点】内部排序的比较

【解析】

（1）对于其中的某一趟直接插入排序，内层的循环次数取决于待插记录的关键字与前 i–1 个记录的关键字之间的关系。在最好情况（正序：待排序序列中记录按关键字非递减有序排列）下，比较 1 次，不移动；在最坏情况（逆序：待排序序列中记录按关键字非递增有序排列）下，比较 i 次，移动 i+1 次。对于整个排序过程需执行 n–1 趟，最好情况下，总的比较次数达最小值 n–1，记录不需移动；最坏情况下，总的关键字比较次数 KCN 和记录移动次数 RMN 均达到最大值，均约为 $n^2/2$。

（2）冒泡排序最好情况（初始序列为正序）：只需进行一趟排序，在排序过程中进行 n–1 次关键字间的比较，且不移动记录。

最坏情况（初始序列为逆序）：需进行 n–1 趟排序，总的关键字比较次数 KCN 和记录移动次数 RMN（每次交换都要移动 3 次记录）分别为 $n^2/2$ 和 $3n(n-1)/2$。

（3）基数排序不需要比较关键字的大小，它是根据关键字中各位的值，通过对待排序记录进行若干趟“分配”与“收集”来实现排序的，是一种借助于多关键字排序的思想对单关键字排序的方法，元素的移动次数和关键字的初始排列次序是无关的。

（4）快速排序最好情况：元素基本无序时，选第一个元素为枢轴记录，每一趟排序后都能将记录序列均匀地分割成两个长度大致相等的子表。快速排序最坏情况：在待排序序列已经排好序的情况下，其递归树成为单支树，每次划分只得到一个比上一次少一个记录的子序列。

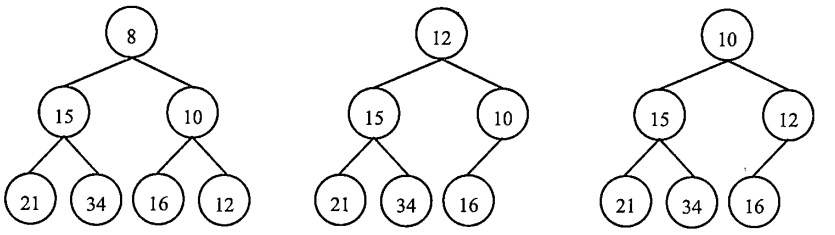
因此，只有基数排序元素的移动次数和关键字的初始排列次序无关，答案选择 C。

28.【答案】C

【考点】堆的调整

【解析】

原始的小根堆如图 8.6（a）所示，在删除关键字 8 之后，最后一个关键字 12 顶替 8 的位置，如图 8.6（b）所示，此时不再满足小根堆的要求，需要自顶向下进行调整。首先，12 的两个孩子结点 15 和 10 相比较（第 1 次），10 较小，12 与较小的孩子结点 10 相比较（第 2 次），12>10，两者交换，10 成为堆顶。之后 12 再与其孩子结点 16 比较（第 3 次），12<16，此时不需要交换。堆调整结束，因此答案选择 C。



（a）原始堆　（b）删除 8 之后的状态　（c）10 被筛选之后的状态

图 8.6　调整堆的过程

29.【答案】A

【考点】希尔排序

【解析】

希尔排序将相隔增量整数倍的元素组成一组，并使用直接插入排序在组内进行排序。然后依次缩减增量再进行排序，待整个序列中的元素基本有序（即增量足够小）时，再对全体元素进行一次直接插入排序。因此，答案选择 A。

30.【答案】D

【考点】内部排序

【解析】

当待排序的记录数目很大，无法一次性调入内存时，整个排序过程就必须借用外存分批调入内存才能完成，即利用外部排序的方法，2-路平衡归并是一种常用的外部排序方法。因此，答案选择 D。

二、应用题

1．解答

（1）直接插入排序。

[2　12]　16　30　28　10　16*　20　6　18

[2　12　16]　30　28　10　16*　20　6　18

[2　12　16　30]　28　10　16*　20　6　18

[2　12　16　28　30]　10　16*　20　6　18

[2　10　12　16　28　30]　16*　20　6　18
[2　10　12　16　16*　28　30]　20　6　18
[2　10　12　16　16*　20　28　30]　6　18
[2　6　10　12　16　16*　20　28　30]　18
[2　6　10　12　16　16*　18　20　28　30]

（2）折半插入排序：排序过程同（1）。

（3）希尔排序（增量选取 5，3，1）。

10　2　16　6　18　12　16*　20　30　28　（增量选取 5）
6　2　12　10　18　16　16*　20　30　28　（增量选取 3）
2　6　10　12　16　16*　18　20　28　30　（增量选取 1）

（4）冒泡排序。

2　12　16　28　10　16*　20　6　18　[30]
2　12　16　10　16*　20　6　18　[28　30]
2　12　10　16　16*　6　18　[20　28　30]
2　10　12　16　6　16*　[18　20　28　30]
2　10　12　6　16　[16*　18　20　28　30]
2　10　6　12　[16　16*　18　20　28　30]
2　6　10　[12　16　16*　18　20　28　30]
2　6　[10　12　16　16*　18　20　28　30]

（5）快速排序。

12　[6　2　10]　**12**　[28　30　16*　20　16　18]
6　[2]　**6**　[10]　12　[28　30　16*　20　16　18]
28　2　6　10　12　[18　16　16*　20]　**28**　[30]
18　2　6　10　12　[16*　16]　**18**　[20]　28　30
16*　2　6　10　12　16*　[16]　18　20　28　30

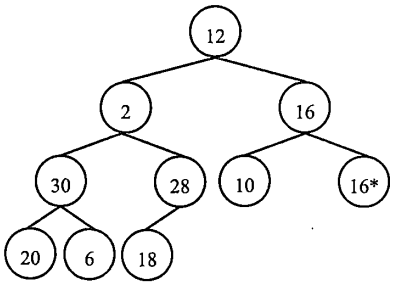
左子序列递归深度为 1，右子序列递归深度为 3。

（6）简单选择排序。

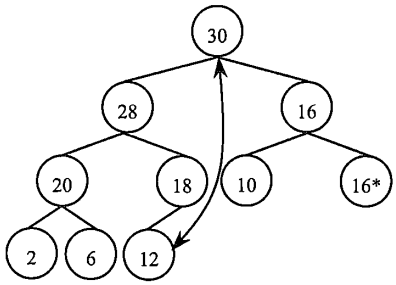
2　[12　16　30　28　10　16*　20　6　18]
2　6　[16　30　28　10　16*　20　12　18]
2　6　10　[30　28　16　16*　20　12　18]
2　6　10　12　[28　16　16*　20　30　18]
2　6　10　12　16　[28　16*　20　30　18]
2　6　10　12　16　16*　[28　20　30　18]
2　6　10　12　16　16*　18　[20　30　28]
2　6　10　12　16　16*　18　20　[30　28]
2　6　10　12　16　16*　18　20　28　[30]

（7）堆排序。

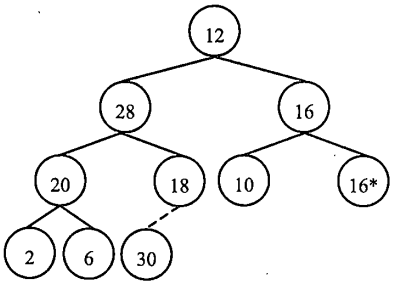
第一步，创建初始大根堆；第二步，做堆排序，具体过程如图 8.7 所示。



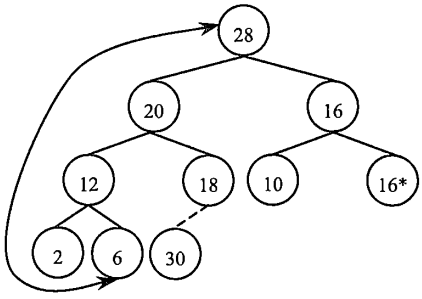
（a）初始序列（不是大根堆）



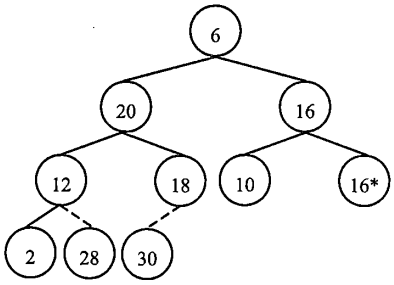
（b）创建初始大根堆 r[1...10]



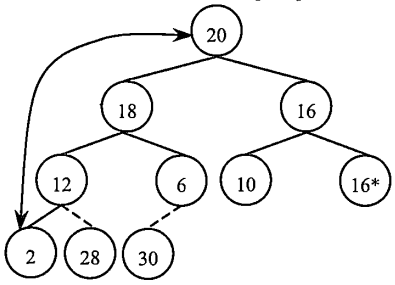
（c）第一趟排序的交换操作之后



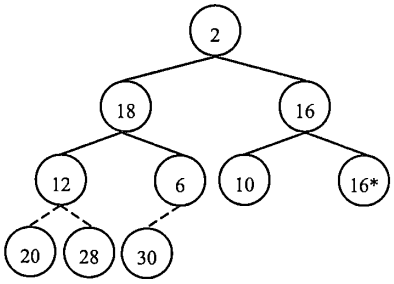
（d）第一趟重调整堆 r[1...9]之后



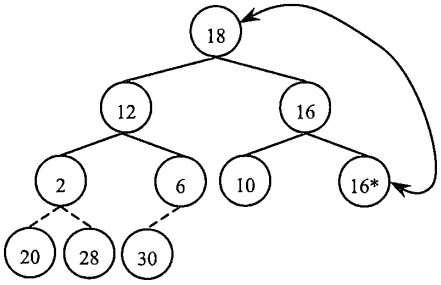
（e）第二趟排序的交换操作之后



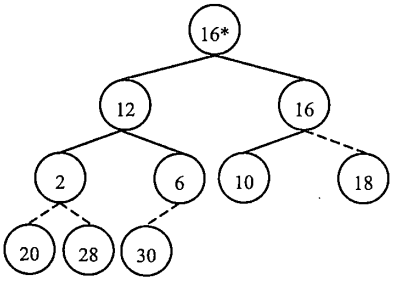
（f）第二趟重调整堆 r[1...8]之后



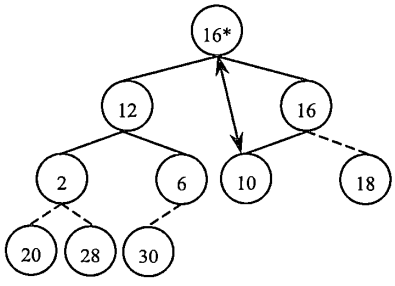
（g）第三趟排序的交换操作之后



（h）第三趟重调整堆 r[1...7]之后

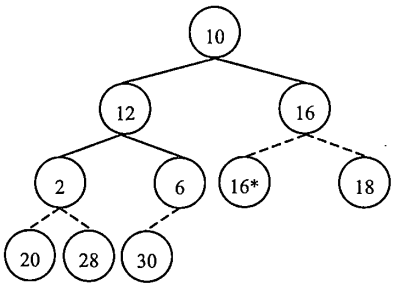


（i）第四趟排序的交换操作之后

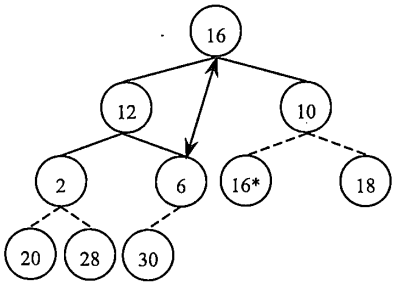


（j）第一趟重调整堆 r[1...6]之后

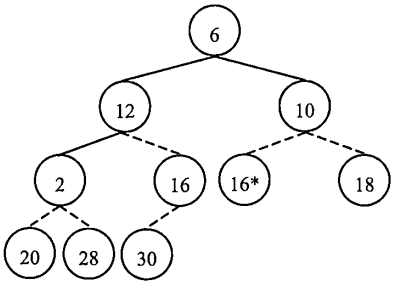
图 8.7　堆排序的过程



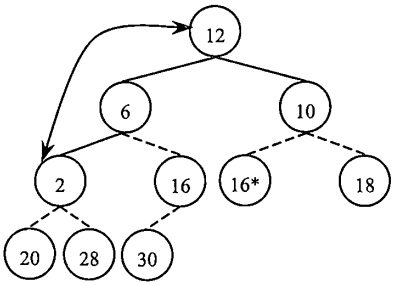
（k）第五趟排序的交换操作之后



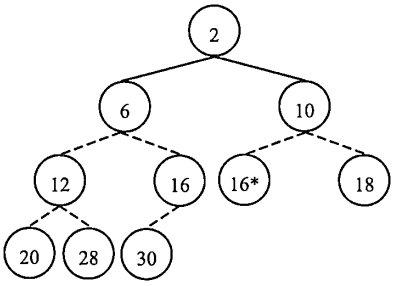
（l）第五趟重调整堆 r[1...5]之后



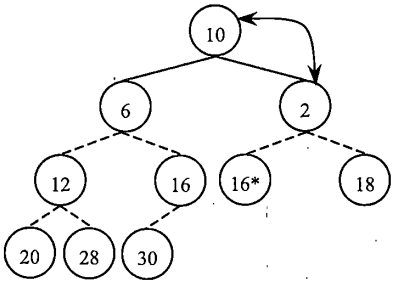
（m）第六趟排序的交换操作之后



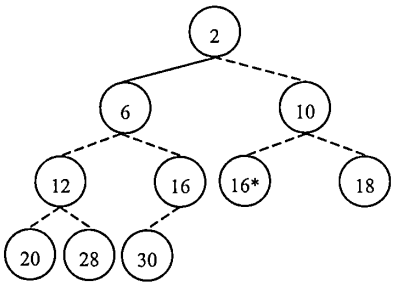
（n）第六趟重调整堆 r[1...4]之后



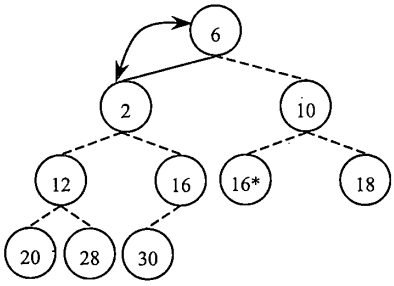
（o）第七趟排序的交换操作之后



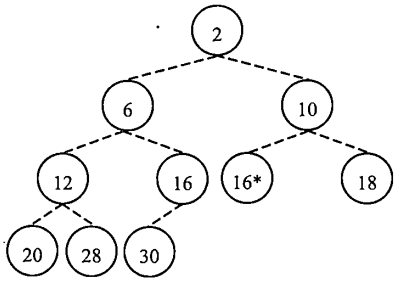
（p）第七趟重调整堆 r[1...3]之后



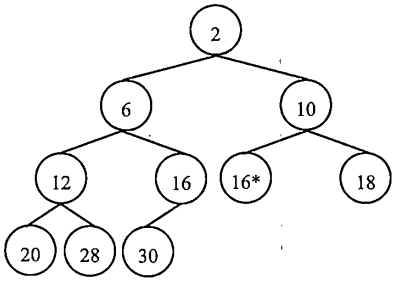
（q）第八趟排序的交换操作之后



（r）第八趟重调整堆 r[1...2]之后



（s）第九趟排序的交换操作之后



（t）最终排序结果

图 8.7　堆排序的过程（续）

（8）二路归并排序。

2 12　16 30　10 28　16 * 20　6 18

2 12 16 30　10 16* 20 28　6 18

2 10 12 16 16* 20 28 30　6 18

2 6 10 12 16 16* 18 20 28 30

2．解答

按最低位优先法：

→321→156→57→46→28→7→331→33→34→63

一趟分配：

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	321		33	34		156	57	28	
	331		63			46	7		

一趟收集：

→321→331→33→63→34→156→46→57→7→28

二趟分配：

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
7		321	331	46	156				
		28	33		57	63			
			34						

二趟收集：

→7→321→28→331→33→34→46→156→57→63

二趟分配：

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
7	156		321						
28			331						
33									
34									
46									
57									
63									

三趟收集：

→7→28→33→34→46→57→63→156→321→331

3．解答

初始败者树如图 8.8 所示。

初始归并段：

R_1:3,19,31,51,61,71,100,101

R_2:9,17,19,20,30,50,55,90

R_3:6

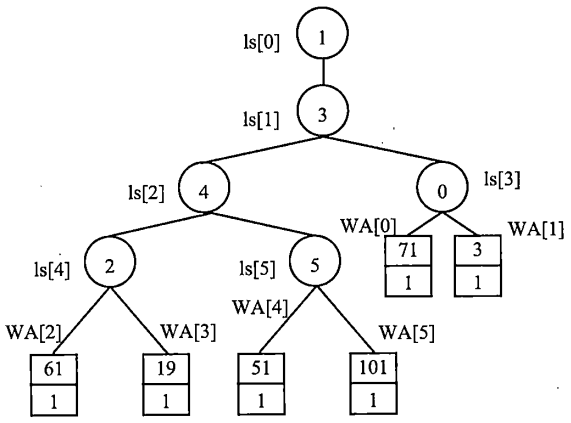


图 8.8　初始败者树

三. 算法设计题

1. 解答

【算法思想】

指针 p 初始化为首元结点，从前向后遍历链表，找到关键字最小的结点，用指针记录其位置；然后将关键字最小的结点的数据域与 p 的数据域交换。指针 p 后移，继续查找 p 之后的结点中关键字最小的结点，将其数据域与 p 的数据域交换。依此类推，重复上述操作，直到 p 为 NULL 时结束，排序完成。

【算法描述】

```
void SelectSort(LinkList &L)
{//基于单链表的简单选择排序
  p=L->next;                                    //p 指向首元结点
  while(p!=NULL)                                //顺链域向后扫描，直到 p 为空
  {
    q=p->next;
    r=p;                                        //r 是指向关键字最小的结点的指针
    while(q!=NULL)
    {
      if(q->data<r->data) r=q;
      q=q->next;
    }
    if(r!=p)
      r->data<-->p->data;                       //交换 r 和 p 的数据域
    p=p->next;                                  //p 指向下一个结点
  }
}
```

2. 解答

【算法思想】

从头结点出发开始顺序扫描链表，利用冒泡排序的思想使最大的结点沉底，然后反向冒泡使最小的结点冒出。重复上述步骤，直到不存在任何结点的交换时即完成排序。

【算法描述】

```
typedef struct DuLNode
{
```

```
    ElemType data;
    struct DuLNode *prior,*next;
}DuLNode, *DuLinkList;
void DuplexSort(DuLinkList &L)
{//对存储在带头结点的双向链表 L 中的元素进行双向冒泡排序
    exchange=1;                                    //是否发生交换的标记
    head=L;                                        //双向链表头，向下冒泡的开始结点
    tail=NULL;                                     //双向链表尾，向上冒泡的开始结点
    while(exchange)
    {
      p=head->next;                                //p 是工作指针，指向当前结点
      exchange=0;                                  //假定本趟无交换
      while(p->next!=tail)                         //向下冒泡，每一趟使一最大元素沉底
      {
        if(p->data>p->next->data)                  //交换两结点指针，涉及 6 条链
        {
          temp=p->next;exchange=1;                 //有交换
          p->next=temp->next;temp->next->prior=p;  //先将结点从链表上摘下
          temp->next=p;p->prior->next=temp;        //将 temp 插到结点*p 前
          temp->prior=p->prior;p->prior=temp;
        }
        else p=p->next;                            //无交换，指针后移
      }
      tail=p;                                      //准备向上起泡
      p=tail->prior;
      while(exchange&&p->prior!=head)
      {//向上冒泡，每一趟有一最小元素冒出
        if(p->data<p->prior->data)                 //交换两结点指针，涉及 6 条链
        {
          temp=p->prior;exchange=1;                //有交换
          p->prior=temp->prior;
          temp->prior->next=p;                     //先将 temp 结点从链表上摘下
          temp->prior=p;p->next->prior=temp;       //将 temp 插到 p 结点后(右)
          temp->next=p->next;p->next=temp;
        }
        else p=p->prior;                           //无交换，指针前移
      }
      head=p;                                      //准备向下起泡
    }                                              //while(exchange)
}
```

3．解答

【算法思想】

此题目可以借助快速排序的思想进行解决。由于题目要求“对每粒砾石的颜色只能看一次”，设 4 个指针 i、j 和 $k1$、$k2$，分别用来指向第一块白色石头、红色石头、最后一块白色石头和蓝色石头。首先找到从左向右第一块不为红的石头和从右向左第一块不为蓝的石头。如果从左向右第一块不为红的石头是蓝色的，则把它与从右向左第一块不为蓝的石头交换。如果从右向左第一块不为蓝的石头是红色的，则把它与从左向右第一块不为红的石头交换。如果这两块石头都是白色的，则把这两块石头之后出现的第一块红石头与第一块白石头交换或之后第一块蓝石头与最后一

个白石头交换。重复上述步骤，直至所有石头遍历完毕。

为方便处理，将三种砾石的颜色分别用字符 R、W 和 B 表示。

【算法描述】

```
void QkSort(char r[],int n)
{//r 为含有 n 个元素的线性表，元素是具有红、白和蓝色的砾石
//要求使所有红色砾石在前，白色居中，蓝色在最后
  i=1;j=1;                                    //i 指向第一块白石头，j 指向红石头
  k1=n;k2=n;                                  //k1 指向蓝石头，k2 指向最后一块白石头
  while(j<k1)                                 //遍历所有石头
  {
    while(r[k1]=='B')                         //从右向左找到第一块非蓝的石头
    {
      k1--;
      k2--;
    }
    while(r[i]=='R')                          //从左向右找到第一块非红的石头
    {
      i++;
      j++;
    }
    if(r[j]=='B')
    {
      temp=r[i];
      r[i]=r[k2];
      r[k2]=temp;                             //交换两块石头
      while(r[i]=='R')                        //继续找
      {
        i++;
        j++;
      }
      while(r[k1]=='B')
      {
        k1--;
        k2--;
      }
    }
    else if(r[k1]=='R')
    {
      temp=r[i];
      r[i]=r[k2];
      r[k2]=temp;
      while(r[i]=='R')
      {
        i++;
        j++;
      }
      while(r[k1]=='B')
      {
        k1--;
```

```
            k2--;
          }
        }
        else                                    //两块石头都是白色
        {
          while(r[j]=='W')                      //找到从左向右第一块非白的石头
          {
            j++;
          }
          if(j>=k1)                             //和从右向左遍历的会合，表示已找完，结束
            break;
          if(r[j]=='R')
          {//是红色，交换它和从左向右的第一块白色石头
            temp=r[j];
            r[j]=r[i];
            r[i]=temp;
            i++;
          }
          else
          {//蓝色，交换它和从右向左的第一块白色石头
            temp=r[k2];
            r[k2]=r[j];
            r[j]=temp;
            k2--;
          }
        }
      }
    }
```

4. 解答

【算法思想】

此题目可借助快速排序中子表划分的算法思想对表中的数据进行划分。附设两个指针 low 和 high，初始时分别指向表的下界和上界（low = 1，high = n–1）。从表的最左侧位置依次向右搜索（low++），找到第一个关键字为非负值的记录；然后再从表的最右侧位置，依次向左搜索（high++），找到第一个关键字为负值的记录；之后将找到的记录进行交换。重复上述步骤，直至 low 与 high 相等为止。

【算法描述】

```
void Process(int a[],int n)
{//对数组 a 中的 n 个关键字进行整理
//使所有关键字为负值的记录排在关键字为非负值的记录之前
  low=0;
  high=n-1;
  while(low<high)
  {
    while(low<high&&a[low]<0)                   //找到从左向右的非负值记录
      low++;
    while(low<high&&a[high]>0)                  //找到从右向左的负值记录
      high--;
    if(low<high)                                //如果需要交换，即 low<high
```

```
        {
          temp=a[low];
          a[low]=a[high];
          a[high]=temp;                                    //交换记录
          low++;                                           //继续向后找
          high--;
        }
      }
    }
```

5．解答

【算法思想】

借助于快速排序的算法思想，附设两个指针 low 和 high，初始时分别指向数组的下界和上界（low = 1，high = *n*）。首先,从数组右侧开始找到第一个不大于关键字的记录，其位置为 high。如果 *r*[high]等于 *key*，则查找成功返回其位置 high。然后再从数组左侧开始找到第一个不小于关键字的记录，其位置为 low。如果 *r*[low]等于 *key*，则查找成功返回其位置 low。重复上述步骤，直至 low 与 high 相等为止。循环结束后尚无返回则意味着查找失败。

【算法描述】

```
int Search(ElemType r[],int low,int high,ElemType key)
{//在数组 r[low..high]中查找关键字值等于 key 的记录
  while(low<high)
  {
    while(low<=high&&r[high]>key) high--;
    //从右侧开始找到第一个不大于关键字的记录，其位置为 high
    if(r[high]==key) return high;                          //查找成功返回其位置
    while(low<=high&&r[low]<key) low++;
    //从左侧开始找到第一个不小于关键字的记录，其位置为 low
    if(r[low]==key) return low;                            //查找成功返回其位置
  }
  cout<<"Not find";                                        //查找失败
  return 0;
}
```

6．解答

【算法思想】

针对数组中的每一个关键字 *a*[*i*]，从前向后扫描该表，统计关键字比 *a*[*i*]小的元素个数，记为 *c*。然后根据 *c* 的值将当前关键字存放在数组 *b* 中，即 *b*[*c*]=*a*[*i*]。

【算法描述】

（1）定义如下所示。

```
typedef struct
{
  int key;
  ElemType info;
}RecType;
```

（2）算法如下所示。

```
void CountSort(RecType a[],RecType b[],int n)
{//计数排序算法，将包括 n 个数据的数组 a 中的记录排序后放入数组 b 中
  for(i=0;i<n;i++)                          //针对数组中的每个关键字
```

```
    {
        for(j=0,c=0;j<n;j++)                    //统计关键字比当前关键字小的元素个数
            if(a[j].key<a[i].key) c++;
            b[c]=a[i];                          //根据比当前关键字小的元素个数将当前关键字存放在数组 b 中
    }
}
```

（3）对于有 n 个记录的表，关键码比较 n^2 次。

（4）简单选择排序算法优于本算法。简单选择排序的比较次数是 $n(n-1)/2$，且只需要一个辅助空间以完成记录的交换，即空间复杂度为 O(1)。而本题中的方法比较次数是 n^2，且需要另一同样大小的数组空间辅助完成，即空间复杂度为 O(n)。

第二篇　实验

实验 1　基于线性表的图书信息管理

实验 2　基于栈的中缀算术表达式求值

实验 3　基于栈的后缀算术表达式求值

实验 4　基于字符串模式匹配算法的病毒感染检测问题

实验 5　基于哈夫曼树的数据压缩算法

实验 6　基于二叉树的表达式求值算法

实验 7　基于 Dijsktra 算法的最短路径求解

实验 8　基于广度优先搜索的六度空间理论的验证

课程设计　基于不同策略的英文单词的词频统计和检索系统

实验 1
基于线性表的图书信息管理

【实验目的】

1．掌握线性表的顺序存储表示和链式存储表示。

2．掌握顺序表和链表的基本操作，包括创建、查找、插入和删除等算法。

3．明确线性表两种不同存储结构的特点及其适用场合，明确它们各自的优缺点。

【实验内容】

图书信息表包括以下 10 项常用的基本操作：图书信息表的创建和输出、排序、修改、逆序存储、最贵图书的查找、最爱图书的查找、最佳位置图书的查找、新图书的入库、旧图书的出库、图书去重。实验要求分别利用顺序表和链表实现上述 10 项操作，因此，实验内容总计包括以下 20 道题目，其中前 10 道要求基于顺序表实现相应的功能，后 10 道要求基于链表实现相应的功能。

1．基于顺序存储结构的图书信息表的创建和输出

问题描述

定义一个包含图书信息（书号、书名、价格）的顺序表，读入相应的图书数据来完成图书信息表的创建。然后，统计图书表中的图书个数，同时逐行输出每本图书的信息。

输入要求

输入 n+1 行，其中前 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。最后第 n+1 行是输入结束标志：0 0 0（空格分隔的三个 0）。其中，书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计 n+1 行，第 1 行是所创建的图书信息表中的图书个数，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版） 20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
0 0 0

输出样例

6

9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版） 20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00

2. 基于顺序存储结构的图书信息表的排序

问题描述

定义一个包含图书信息（书号、书名、价格）的顺序表，读入相应的图书数据完成图书信息表的创建。然后，将图书按照价格降序排序，逐行输出排序后每本图书的信息。

输入要求

输入 n+1 行，前 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。最后，第 n+1 行是输入结束标志：0 0 0（空格分隔的三个 0）。其中，书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计 n 行，每行是一本图书的信息（书号、书名、价格），书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
0 0 0

输出样例

9787302219972 单片机技术及应用 32.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787302203513 单片机原理与应用技术 26.00
9787302257646 程序设计基础 25.00
9787811234923 汇编语言程序设计教程 21.00
9787302164340 程序设计基础（第 2 版）20.00

3. 基于顺序存储结构的图书信息表的修改

问题描述

首先，定义一个包含图书信息（书号、书名、价格）的顺序表，读入相应的图书数据完成图书信息表的创建。然后，计算所有图书的平均价格，将所有低于平均价格的图书价格提高 20%，所有高于或等于平均价格的图书价格提高 10%。最后，逐行输出价格修改后的图书信息。

输入要求

输入 n+1 行，前 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。最后，第 n+1 行是输入结束标志：0 0 0（空格分隔

的三个 0）。其中书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计 n+1 行，第 1 行是修改前所有图书的平均价格，后 n 行是价格修改后 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
0 0 0

输出样例

25.50
9787302257646 程序设计基础 30.00
9787302164340 程序设计基础（第 2 版）24.00
9787302219972 单片机技术及应用 35.20
9787302203513 单片机原理与应用技术 28.60
9787810827430 工业计算机控制技术——原理与应用 31.90
9787811234923 汇编语言程序设计教程 25.20

4. 基于顺序存储结构的图书信息表的逆序存储

问题描述

定义一个包含图书信息（书号、书名、价格）的顺序表，读入相应的图书数据来完成图书信息表的创建。然后将读入的图书信息逆序存储，逐行输出逆序存储后每本图书的信息。

输入要求

输入 n+1 行，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中，书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计 n 行，第 i 行是原有图书信息表中第 n–i+2 行（此处不是 n–i+1，是因原有图书信息表中第 1 行不是图书信息而是图书数目）的图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00

9787811234923 汇编语言程序设计教程 21.00

输出样例

9787811234923 汇编语言程序设计教程 21.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787302203513 单片机原理与应用技术 26.00
9787302219972 单片机技术及应用 32.00
9787302164340 程序设计基础（第 2 版）20.00
9787302257646 程序设计基础 25.00

5. 基于顺序存储结构的图书信息表的最贵图书的查找

问题描述

定义一个包含图书信息（书号、书名、价格）的顺序表，读入相应的图书数据来完成图书信息表的创建。然后，查找价格最高的图书，输出相应图书的信息。

输入要求

总计输入 n+1 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中，书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计输出 m+1 行，其中，第一行是最贵图书的数目（价格最高的图书可能有多本），后 m 行是最贵图书的信息，每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811230710 C#程序设计易懂易会教程 32. 00

输出样例

2
9787302219972 单片机技术及应用 32.00
9787811230710 C#程序设计易懂易会教程 32. 00

6. 基于顺序存储结构的图书信息表的最爱图书的查找

问题描述

定义一个包含图书信息（书号、书名、价格）的顺序表，读入相应的图书数据来完成图书信息表的创建。然后，根据指定的最爱图书的名字，查找最爱的图书，输出相应图书的信息。

输入要求

总计 n+m+2 行。首先，输入 n+1 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中书号和书名为字符串类型，价格为浮点数类型。然后输入 m+1 行，其中，第一行是一个整数

m，代表查找 m 次，后 m 行是每次待查找的最爱图书名字。

输出要求

若查找成功：

总计输出 m（k+1）行，对于每一次查找，第一行是最爱图书数目（同一书名的图书可能有多本），后 k 行是最爱图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

若查找失败：

只输出以下提示：抱歉，没有你的最爱！

输入样例

6

9787302257646 程序设计基础 25.00

9787302164340 程序设计基础（第 2 版）20.00

9787302219972 单片机技术及应用 32.00

9787302203513 单片机原理与应用技术 26.00

9787810827430 工业计算机控制技术——原理与应用 29.00

9787811234923 汇编语言程序设计教程 21.00

2

数据结构

程序设计基础

输出样例

抱歉，没有你的最爱！

1

9787302257646 程序设计基础 25.00

7. 基于顺序存储结构的图书信息表的最佳位置图书的查找

问题描述

定义一个包含图书信息（书号、书名、价格）的顺序表，读入相应的图书数据来完成图书信息表的创建。然后根据指定的最佳位置的序号，查找该位置上的图书，输出相应图书的信息。

输入要求

总计 n+m+2 行。首先，输入 n+1 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中，书号和书名为字符串类型，价格为浮点数类型。然后，输入 m+1 行，其中，第一行是一个整数 m，代表查找 m 次，后 m 行每行内容为一个整数，代表待查找的图书的位置序号。

输出要求

输出 m 行。

若查找成功：

输出内容为第 i 次查询的指定位置上的一本图书的信息（书号、书名、价格），书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

若查找失败：

只输出以下提示：抱歉，最佳位置上的图书不存在！

输入样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
2
2
0

输出样例

9787302164340 程序设计基础（第 2 版） 20.00
抱歉，最佳位置上的图书不存在！

8. 基于顺序存储结构的图书信息表的新图书的入库

问题描述

首先，定义一个包含图书信息（书号、书名、价格）的顺序表，读入相应的图书数据来完成图书信息表的创建。然后，根据指定的待入库的新图书的位置和信息，将新图书插入到图书表中指定的位置上。最后，输出新图书入库后所有图书的信息。

输入要求

总计 n+3 行。首先，输入 n+1 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中，书号和书名为字符串类型，价格为浮点数类型。之后输入第 n+2 行，内容仅为一个整数，代表待入库的新图书的位置序号。最后，输入第 n+3 行，内容为新图书的信息，书号、书名、价格用空格分隔。

输出要求

若插入成功：

输出新图书入库后所有图书的信息（书号、书名、价格），总计 n+1 行，每行是一本图书的信息，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

若插入失败：

只输出以下提示：抱歉，入库位置非法！

输入样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
2
9787302265436 计算机导论实验指导 18. 00

输出样例

9787302257646 程序设计基础 25.00
9787302265436 计算机导论实验指导 18.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00

9. 基于顺序存储结构的图书信息表的旧图书的出库

问题描述

定义一个包含图书信息（书号、书名、价格）的顺序表，读入相应的图书数据来完成图书信息表的创建。然后根据指定的待出库的旧图书的位置，将该图书从图书表中删除。最后输出该图书出库后的所有图书的信息。

输入要求

总计 n+2 行。首先，输入 n+1 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中书号和书名为字符串类型，价格为浮点数类型。然后，输入第 n+2 行，内容仅为一个整数，代表待删除的旧图书的位置序号。

输出要求

若删除成功：

输出旧图书出库后所有图书的信息（书号、书名、价格），总计 n–1 行，每行是一本图书的信息，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

若删除失败：

只输出以下提示：抱歉，出库位置非法！

输入样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
2

输出样例

9787302257646 程序设计基础 25.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00

10. 基于顺序存储结构的图书信息表的图书去重

问题描述

出版社出版的任何一本图书的书号（ISBN）都是唯一的，即图书表中不允许包含书号重复的图书。定义一个包含图书信息（书号、书名、价格）的顺序表，读入相应的图书数据来完成图书信息表的创建（书号可能重复）。然后进行图书的去重，即删除书号重复的图书（只保留第一本）。最后输出去重后所有图书的信息。

输入要求

总计输入 n+1 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格（书号可能重复）。其中书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计输出 m+1 行（$m \leqslant n$），其中，第一行是去重后的图书数目，后 m 行是去重后图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302257646 程序设计基础 25.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787302219972 单片机技术及应用 32.00

输出样例

4
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787810827430 工业计算机控制技术——原理与应用 29.00

11. 基于链式存储结构的图书信息表的创建和输出

问题描述

定义一个包含图书信息（书号、书名、价格）的链表，读入相应的图书数据来完成图书信息表的创建。然后，统计图书表中的图书个数，同时逐行输出每本图书的信息。

输入要求

输入 n+1 行，其中前 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。最后第 n+1 行是输入结束标志：0 0 0（空格分隔的三个 0）。其中，书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计 n+1 行，第 1 行是所创建的图书信息表中的图书个数，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

9787302257646 程序设计基础 25.00

9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
0 0 0

输出样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00

12. 基于链式存储结构的图书信息表的排序

问题描述

定义一个包含图书信息（书号、书名、价格）的链表，读入相应的图书数据完成图书信息表的创建。然后，将图书按照价格降序排序，逐行输出排序后每本图书的信息。

输入要求

输入 n+1 行，前 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。最后第 n+1 行是输入结束标志：0 0 0（空格分隔的三个 0）。其中书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计 n 行，每行是一本图书的信息（书号、书名、价格），书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
0 0 0

输出样例

9787302219972 单片机技术及应用 32.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787302203513 单片机原理与应用技术 26.00
9787302257646 程序设计基础 25.00
9787811234923 汇编语言程序设计教程 21.00
9787302164340 程序设计基础（第 2 版）20.00

13. 基于链式存储结构的图书信息表的修改

问题描述

首先，定义一个包含图书信息（书号、书名、价格）的链表，读入相应的图书数据完成图书信息表的创建。然后，计算所有图书的平均价格，将所有低于平均价格的图书价格提高 20%，所有高于或等于平均价格的图书价格提高 10%。最后，逐行输出价格修改后的图书信息。

输入要求

输入 n+1 行，前 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。最后第 n+1 行是输入结束标志：0 0 0（空格分隔的三个 0）。其中，书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计 n+1 行，第 1 行是修改前所有图书的平均价格，后 n 行是价格修改后 n 本图书的信息，每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
0 0 0

输出样例

25.50
9787302257646 程序设计基础 30.00
9787302164340 程序设计基础（第 2 版）24.00
9787302219972 单片机技术及应用 35.20
9787302203513 单片机原理与应用技术 28.60
9787810827430 工业计算机控制技术——原理与应用 31.90
9787811234923 汇编语言程序设计教程 25.20

14. 基于链式存储结构的图书信息表的逆序存储

问题描述

定义一个包含图书信息（书号、书名、价格）的链表，读入相应的图书数据来完成图书信息表的创建。然后将读入的图书逆序存储，逐行输出逆序存储后每本图书的信息。

输入要求

输入 n+1 行，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中，书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计 n 行，第 i 行是原有图书信息表中第 n-i+2 行（此处不是 n-i+1，是因原有图书信息表中第 1 行不是图书信息而是图书数目）的图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00

输出样例

9787811234923 汇编语言程序设计教程 21.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787302203513 单片机原理与应用技术 26.00
9787302219972 单片机技术及应用 32.00
9787302164340 程序设计基础（第 2 版）20.00
9787302257646 程序设计基础 25.00

15. 基于链式存储结构的图书信息表的价格最贵图书的查找

问题描述

定义一个包含图书信息（书号、书名、价格）的链表，读入相应的图书数据来完成图书信息表的创建。然后，查找价格最高的图书，输出相应图书的信息。

输入要求

总计输入 n+1 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中，书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计输出 m+1 行，其中，第一行是最贵图书数目（价格最高的图书可能有多本），后 m 行是最贵图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811230710 C#程序设计易懂易会教程 32. 00

输出样例

2
9787302219972 单片机技术及应用 32.00
9787811230710 C#程序设计易懂易会教程 32. 00

16. 基于链式存储结构的图书信息表的最爱图书的查找

问题描述

定义一个包含图书信息（书号、书名、价格）的链表，读入相应的图书数据来完成图书信息表的创建。然后，根据指定的最爱图书的名字，查找最爱的图书，输出相应图书的信息。

输入要求

总计 $n+m+2$ 行。首先输入 $n+1$ 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中书号和书名为字符串类型，价格为浮点数类型。然后输入 $m+1$ 行，其中，第一行是一个整数 m，代表查找 m 次，后 m 行是每次待查找的最爱图书名字。

输出要求

若查找成功：

总计输出 m（$k+1$）行，对于每一次查找，第一行是最爱图书数目（同一书名的图书可能有多本），后 k 行是最爱图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，其中，价格输出保留两位小数。

若查找失败：

只输出以下提示：抱歉，没有你的最爱！

输入样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
2
数据结构
程序设计基础

输出样例

抱歉，没有你的最爱！
1
9787302257646 程序设计基础 25.00

17. 基于链式存储结构的图书信息表的最佳位置图书的查找

问题描述

定义一个包含图书信息（书号、书名、价格）的链表，读入相应的图书数据来完成图书信息表的创建。然后，根据指定的最佳位置的序号，查找该位置上的图书，输出相应图书的信息。

输入要求

总计 $n+m+2$ 行。首先，输入 $n+1$ 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中，书号和书名为字符串类型，价格为浮点数类型。然后输入 $m+1$ 行，其中，第一行是一个整数 m，代表查找 m 次，后 m 行每行内容为一个整数，代表待查找的图书的位置序号。

输出要求

输出 m 行

若查找成功：

输出内容为第 i 次查询的指定位置上的一本图书的信息（书号、书名、价格），书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

若查找失败：

只输出以下提示：抱歉，最佳位置上的图书不存在！

输入样例

6

9787302257646 程序设计基础 25.00

9787302164340 程序设计基础（第 2 版）20.00

9787302219972 单片机技术及应用 32.00

9787302203513 单片机原理与应用技术 26.00

9787810827430 工业计算机控制技术——原理与应用 29.00

9787811234923 汇编语言程序设计教程 21.00

2

2

0

输出样例

9787302164340 程序设计基础（第 2 版）20.00

抱歉，最佳位置上的图书不存在！

18. 基于链式存储结构的图书信息表的新图书的入库

问题描述

定义一个包含图书信息（书号、书名、价格）的链表，读入相应的图书数据来完成图书信息表的创建。然后，根据指定的待入库的新图书的位置和图书的信息，将新图书插入到图书表中指定的位置上。最后，输出新图书入库后的所有图书的信息。

输入要求

总计 n+3 行。首先，输入 n+1 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中，书号和书名为字符串类型，价格为浮点数类型。之后输入第 n+2 行，内容仅为一个整数，代表待入库的新图书的位置序号。最后，输入第 n+3 行，内容为新图书的信息，书号、书名、价格用空格分隔。

输出要求

若插入成功：

输出新图书入库后所有图书的信息（书号、书名、价格），总计 n+1 行，每行是一本图书的信息，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

若插入失败：

只输出以下一行提示：抱歉，入库位置非法！

输入样例

6

9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00
2
9787302265436 计算机导论实验指导 18. 00

输出样例

9787302257646 程序设计基础 25.00
9787302265436 计算机导论实验指导 18.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00

19. 基于链式存储结构的图书信息表的旧图书的出库

问题描述

定义一个包含图书信息（书号、书名、价格）的链表，读入相应的图书数据来完成图书信息表的创建。然后，根据指定的待出库的旧图书的位置，将该图书从图书表中删除。最后，输出该图书出库后的所有图书的信息。

输入要求

总计 n+2 行。首先输入 n+1 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格。其中书号和书名为字符串类型，价格为浮点数类型。之后输入第 n+2 行，内容仅为一个整数，代表待删除的旧图书的位置序号。

输出要求

若删除成功：

输出旧图书出库后所有图书的信息（书号、书名、价格），总计 n–1 行，每行是一本图书的信息，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

若删除失败：

只输出以下一行提示：抱歉，出库位置非法！

输入样例

6
9787302257646 程序设计基础 25.00
9787302164340 程序设计基础（第 2 版）20.00
9787302219972 单片机技术及应用 32.00
9787302203513 单片机原理与应用技术 26.00
9787810827430 工业计算机控制技术——原理与应用 29.00
9787811234923 汇编语言程序设计教程 21.00

2

输出样例

9787302257646 程序设计基础 25.00

9787302219972 单片机技术及应用 32.00

9787302203513 单片机原理与应用技术 26.00

9787810827430 工业计算机控制技术——原理与应用 29.00

9787811234923 汇编语言程序设计教程 21.00

20. 基于链存储结构的图书信息表的图书去重

问题描述

出版社出版的任何一本图书的书号（ISBN）都是唯一的，即图书表中不允许包含书号重复的图书。首先，定义一个包含图书信息（书号、书名、价格）的链表，读入相应的图书数据来完成图书信息表的创建（书号可能重复）。然后，进行图书的去重，即删除书号重复的图书（只保留第一本）。最后，输出去重后所有图书的信息。

输入要求

总计输入 n+1 行，其中，第一行是图书数目 n，后 n 行是 n 本图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔，价格之后没有空格（书号可能重复）。其中书号和书名为字符串类型，价格为浮点数类型。

输出要求

总计输出 m+1 行（$m \leqslant n$），其中，第一行是去重后的图书数目，后 m 行是去重后图书的信息（书号、书名、价格），每本图书信息占一行，书号、书名、价格用空格分隔。其中，价格输出保留两位小数。

输入样例

6

9787302257646 程序设计基础 25.00

9787302164340 程序设计基础（第 2 版）20.00

9787302219972 单片机技术及应用 32.00

9787302257646 程序设计基础 25.00

9787810827430 工业计算机控制技术——原理与应用 29.00

9787302219972 单片机技术及应用 32.00

输出样例

4

9787302257646 程序设计基础 25.00

9787302164340 程序设计基础（第 2 版）20.00

9787302219972 单片机技术及应用 32.00

9787810827430 工业计算机控制技术——原理与应用 29.00

【实验提示】

不同的存储结构对应的算法实现不同，下面分别给出图书信息表的顺序存储结构和链式存储结构的定义。

根据图书所包含的基本信息，可以对图书信息的结构体描述如下：

```
#define MAXSIZE 10000          //图书表可能达到的最大长度
```

```
typedef struct                  //图书信息定义
{
   char no[20];                 //图书 ISBN
   char name[50];               //图书名字
   float price;                 //图书价格
}Book;
```

基于上述图书信息的描述，下面分别给出图书信息表的顺序存储结构和链式存储结构的定义。

图书信息表的顺序存储结构如下：

```
typedef struct
{
   Book *elem;                  //存储空间的基地址
   int length;                  //图书表中当前图书个数
}SqList;                        //图书表的顺序存储结构类型为 SqList
```

链式存储结构的定义如下：

```
typedef struct  LNode
{
   Book data;                   //结点的数据域
   struct LNode *next;          //结点的指针域
}LNode,*LinkList;               //LinkList 为指向结构体 LNode 的指针类型
```

实验 2
基于栈的中缀算术表达式求值

【实验目的】

1．掌握栈的基本操作算法的实现，包括栈初始化、进栈、出栈、取栈顶元素等。

2．掌握利用栈实现中缀表达式求值的算法。

【实验内容】

问题描述

输入一个中缀算术表达式，求解表达式的值。运算符包括“+”、“-”、“*”、“/”、“(”“)”“=”，参加运算的数为 double 类型且为正数。（要求：直接使用中缀算术表达式进行计算，不能转换为后缀或前缀表达式再进行计算，只考虑二元运算即可。）

输入要求

多组数据，每组数据一行，对应一个算术表达式，每个表达式均以“=”结尾。当表达式只有一个“=”时，输入结束。参加运算的数为 double 类型。

输出要求

对于每组数据输出 1 行，为表达式的运算结果。输出保留两位小数。

输入样例

2+2=

20*(4.5-3)=

=

输出样例

4.00

30.00

【实验提示】

此实验内容即为主教材算法 3.22 的扩展内容，算法 3.22 只考虑个位数的运算，不具备拼数功能。拼数功能可以手工编写，也可以借助 C 语言的库函数 atof()函数来完成，其功能是将字符串转换为双精度浮点数（double）。

实验 3
基于栈的后缀算术表达式求值

【实验目的】

1．掌握中缀表达式转换为后缀表达式的算法。

2．掌握后缀表达式求值的算法。

【实验内容】

问题描述

输入一个中缀算术表达式，将其转换为后缀表达式，然后对后缀表达式进行求值。运算符包括“+”“-”“*”“/”“=”，参与运算的为小于 10 的自然数（只考虑二元运算即可）。

输入要求

多组数据，每组数据一行，对应一个算术表达式，每个表达式均以“=”结尾。当表达式只有一个“=”时，输入结束。

输出要求

对于每组数据输出 2 行，第 1 行为中缀表达式对应的后缀式，第 2 行为后缀式求值的结果。

输入样例

```
9+(3-1)*3+1/2=
1+2=
=
```

输出样例

```
931-3*+12/+
15
12+
3
```

【实验提示】

主教材算法 3.22 为中缀表达式的求值算法，算法借助两个栈分别存放运算符和运算数，直接对中缀表达式进行求值。

对于表达式的求值，还有另外一种重要的算法，即本次实验要求的内容。首先借助一个运算符栈将中缀表达式转换为后缀表达式，然后再借助一个运算数栈对后缀表达式进行求值。其中，将中缀表达式转换为后缀表达式的算法请参见第 3 章 22 题的解析部分，对转换后得到的后缀表达式进行计算的具体步骤如下所示。

借助一个工作栈 OPND，用以寄存操作数或运算结果。从左到右扫描后缀表达式，读入第一个字符 ch。若 ch 不是运算符，则压入 OPND 栈，读入下一字符；若 ch 是运算符，则从 OPND 栈中依次弹出两个数分别到 Y 和 X，然后以“X ch Y”的形式计算出结果，将结果压入 OPND 栈中。如果后缀表达式未读完，重复执行上面过程，最后 OPND 栈顶元素即为表达式的求值结果，返回此元素。

实验 4

基于字符串模式匹配算法的病毒感染检测问题

【实验目的】

1．掌握字符串的顺序存储表示方法。

2．掌握字符串模式匹配算法 BF 算法或 KMP 算法的实现。

【实验内容】

问题描述

医学研究者最近发现了某些新病毒，通过对这些病毒的分析，得知它们的 DNA 序列都是环状的。现在研究者已收集了大量的病毒 DNA 和人的 DNA 数据，想快速检测出这些人是否感染了相应的病毒。为了方便研究，研究者将人的 DNA 和病毒 DNA 均表示成由一些字母组成的字符串序列，然后检测某种病毒 DNA 序列是否在患者的 DNA 序列中出现过，如果出现过，则此人感染了该病毒，否则没有感染。例如，假设病毒的 DNA 序列为 baa，患者 1 的 DNA 序列为 aaabbba，则感染；患者 2 的 DNA 序列为 babbba，则未感染。（注意，人的 DNA 序列是线性的，而病毒的 DNA 序列是环状的。）

输入要求

多组数据，每组数据有 1 行，为序列 *A* 和 *B*，*A* 对应病毒的 DNA 序列，*B* 对应人的 DNA 序列。*A* 和 *B* 都为“0”时输入结束。

输出要求

对于每组数据输出 1 行，若患者感染了病毒输出“YES”，否则输出“NO”。

输入样例

```
abbab abbabaab
baa cacdvcabacsd
abc def
0 0
```

输出样例

```
YES
YES
NO
```

【实验提示】

此实验内容即要求实现主教材的案例 4.1，具体实现可参考算法 4.5。算法 4.5 是利用 BF 算法来实现字符串的模式匹配过程的，效率较低，可以利用 KMP 算法完成模式匹配以提高算法的效率。读者可以模仿算法 4.5，利用 KMP 算法来完成病毒感染检测的方案。

实验 5
基于哈夫曼树的数据压缩算法

【实验目的】

1．掌握哈夫曼树的构造算法。

2．掌握哈夫曼编码的构造算法。

【实验内容】

问题描述

输入一串字符串，根据给定的字符串中字符出现的频率建立相应的哈夫曼树，构造哈夫曼编码表，在此基础上可以对压缩文件进行压缩（即编码），同时可以对压缩后的二进制编码文件进行解压（即译码）。

输入要求

多组数据，每组数据 1 行，为一个字符串（只考虑 26 个小写字母即可）。当输入字符串为“0”时，输入结束。

输出要求

每组数据输出 $2n+3$ 行（n 为输入串中字符类别的个数）。第 1 行为统计出来的字符出现频率（只输出存在的字符，格式为:字符:频度），每两组字符之间用一个空格分隔，字符按照 ASCII 码从小到大的顺序排列。第 2 行至第 $2n$ 行为哈夫曼树的存储结构的终态（如主教材 139 页表 5.2（b），一行当中的数据用空格分隔）。第 $2n+1$ 行为每个字符的哈夫曼编码（只输出存在的字符，格式为：字符:编码），每两组字符之间用一个空格分隔，字符按照 ASCII 码从小到大的顺序排列。第 $2n+2$ 行为编码后的字符串，第 $2n+3$ 行为解码后的字符串（与输入的字符串相同）。

输入样例

```
aaaaaaabbbbbccdddd
aabccc
0
```

输出样例

```
a:7 b:5 c:2 d:4
1 7 7 0 0
2 5 6 0 0
3 2 5 0 0
4 4 5 0 0
5 6 6 3 4
6 11 7 2 5
```

7 18 0 1 6
a:0 b:10 c:110 d:111
00000001010101010110110111111111111
aaaaaaabbbbbccdddd
a:2 b:1 c:3
1 2 4 0 0
2 1 4 0 0
3 3 5 0 0
4 3 5 2 1
5 6 0 3 4
a:11 b:10 c:0
111110000
aabccc

【实验提示】

此实验内容即要求实现主教材的案例 5.1，具体实现可参考算法 5. 10 和算法 5.11。

首先，读入一行字符串，统计每个字符出现的频率；然后，根据字符出现的频率利用算法 5. 10 建立相应的哈夫曼树；最后，根据得到的哈夫曼树利用算法 5.11 求出每个字符的哈夫曼编码。

实验 6
基于二叉树的表达式求值算法

【实验目的】

1. 掌握二叉树的二叉链表存储表示和二叉树的遍历等基本算法。
2. 掌握根据中缀表达式创建表达式树的算法。
3. 掌握基于表达式树的表达式求值算法。

【实验内容】

问题描述

输入一个表达式（表达式中的数均为小于 10 的正整数），利用二叉树来表示该表达式，创建表达式树，然后利用二叉树的遍历操作求表达式的值。

输入要求

多组数据。每组数据 1 行，为一个表达式，表达式以“=”结尾。当输入只有一个“=”时，输入结束。

输出要求

每组数据输出 1 行，为表达式的值。

输入样例

```
2*(2+5)=
1+2=
=
```

输出样例

```
14
3
```

【实验提示】

此实验内容即要求实现主教材的案例 5.2，具体实现可参考算法 5.12 和算法 5.13。

首先，读入表达式，利用算法 5.12 创建一个基于二叉链表表示的表达式树；然后，利用算法 5.13 对表达式树进行后序遍历，得到表达式的值。

实验 7 基于 Dijsktra 算法的最短路径求解

【实验目的】

1. 掌握图的邻接矩阵表示法，掌握采用邻接矩阵表示法创建图的算法。
2. 掌握求解最短路径的 Dijsktra 算法。

【实验内容】

问题描述

一张地图包括 n 个城市，假设城市间有 m 条路径（有向图），每条路径的长度已知。给定地图的一个起点城市和终点城市，利用 Dijsktra 算法求出起点到终点之间的最短路径。

输入要求

多组数据，每组数据有 m+3 行。第一行为两个整数 n 和 m，分别代表城市个数 n 和路径条数 m。第二行有 n 个字符，代表每个城市的名字。第三行到第 m+2 行每行有两个字符 a 和 b 和一个整数 d，代表从城市 a 到城市 b 有一条距离为 d 的路。最后一行为两个字符，代表待求最短路径的城市起点和终点。当 n 和 m 都等于 0 时，输入结束。

输出要求

每组数据输出 2 行。第 1 行为一个整数，为从起点到终点之间最短路的长度。第 2 行为一串字符串，代表该路径。每两个字符之间用空格隔开。

输入样例

```
3 3
A B C
A B 1
B C 1
C A 3
A C
6 8
A B C D E F
A F 100
A E 30
A C 10
B C 5
C D 50
D E 20
```

E F 60
D F 10
A F
0 0

输出样例

2
A B C
60
A E D F

【实验提示】

此实验内容即为主教材算法 6.10 的扩展内容，算法 6.10 求出源点 v0 到图中其余所有顶点的最短路径。本实验要求求出一个指定起点到一个指定终点的最短路径。

为了提高算法的效率，在求解时，可以加以判断，当已求得的终点为指定终点时，则可以终止求解，按要求输出相应结果。

实验 8 基于广度优先搜索的六度空间理论的验证

【实验目的】

1．掌握图的邻接矩阵和邻接表表示法，掌握采用邻接矩阵和邻接表表示法创建图的算法。

2．掌握图的广度优先搜索算法。

3．掌握基于图的广度优先搜索的六度空间理论验证的算法。

【实验内容】

问题描述

“六度空间”理论又称作“六度分隔（Six Degrees of Separation）”理论。这个理论可以通俗地阐述为：“你和任何一个陌生人之间所间隔的人不会超过六个，也就是说，最多通过五个人你就能够认识任何一个陌生人。”假如给你一个社交网络图，请你对每个节点计算符合“六度空间”理论的结点占结点总数的百分比。

输入要求

多组数据，每组数据 m+1 行。第一行有两个数字 n 和 m，代表有 n 个人和 m 组朋友关系。n 个人的编号为 1 到 n。第二行到第 m+1 行每行包括两个数字 a 和 b，代表这两个人互相认识。当 n 和 m 都等于 0 时，输入结束。

输出要求

每组数据输出 n 行，对每个结点输出与该结点距离不超过 6 的结点数占结点总数的百分比，精确到小数点后 2 位。每个结节点输出一行，格式为“结点编号:（空格）百分比%”。

输入样例

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

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

输出样例

1: 70.00%
2: 80.00%
3: 90.00%
4: 100.00%
5: 100.00%
6: 100.00%
7: 100.00%
8: 90.00%
9: 80.00%
10: 70.00%
1: 70.00%
2: 80.00%
3: 80.00%
4: 80.00%
5: 80.00%
6: 80.00%
7: 80.00%
8: 70.00%
9: 20.00%
10: 20.00%

【实验提示】

此实验内容即要求实现主教材的案例 6.1，具体实现可参考算法 6.14。

首先，读入表达式，利用算法 5.12 创建一个基于二叉链表表示的表达式树；然后，利用算法 5.13 对表达式树进行后序遍历，得到表达式的值。

算法 6.14 给出了利用广度优先搜索方法进行验证的方案，实际上也可以利用求解最短路径的方法（迪杰斯特拉算法或弗洛伊德算法）对六度空间理论进行理论上的验证。读者可以根据算法 6.14 和最短路算法自行写出相应的验证方法。

课程设计

基于不同策略的英文单词的词频统计和检索系统

【实验目的】

1．掌握基于线性表、二叉排序树和散列表不同存储结构的查找算法。

2．掌握不同检索策略对应的平均查找长度（ASL）的计算方法，明确不同检索策略的时间性能的差别。

3．掌握相关的排序算法。

【实验内容】

一篇英文文章存储在一个文本文件中，分别基于线性表、二叉排序树和哈希表不同的存储结构，实现单词词频的统计和单词的检索功能。同时计算不同检索策略下的 ASL，通过比较 ASL 的大小，对不同检索策略的时间性能做出相应的比较分析（在课程设计报告中给出）。具体内容如下。

1．一篇包括标点符号的英文文章存储在文本文件 InFile.txt 中，假设文件中单词的个数最多不超过 5 000 个。从该文件中读取英文单词，过滤掉所有的标点。

2．分别基于线性表、二叉排序树和哈希表不同的存储结构，实现单词词频的统计和单词的检索功能。其中，线性表采用顺序表和链表两种不同的存储结构分别实现顺序查找，同时实现基于顺序表的折半查找；哈希表分别实现基于开放地址法的哈希查找和基于链地址法的哈希查找。因此，总计实现 6 种不同的检索策略。

3．不论采取哪种检索策略，实现的功能均相同。

（1）词频统计。

当读取一个单词后，若该单词还未出现，则在适当的位置上添加该单词，将其词频计为 1；若该单词已经出现过，则将其词频增加 1。统计结束后，将所有单词及其频率按照词典顺序写入文本文件中。其中，不同的检索策略分别写入 6 个不同的文件。

基于顺序表的顺序查找：OutFile1.txt。

基于链表的顺序查找：OutFile2.txt。

基于顺序表的折半查找：OutFile3.txt。

基于二叉排序树的查找：OutFile4.txt。

基于开放地址法的哈希查找：OutFile5.txt。

基于链地址法的哈希查找：OutFile6.txt。

注：如果实现方法正确，6 个文件的内容应该是一致的。

（2）单词检索。

输入一个单词，如果查找成功，则输出该单词对应的频率，同时输出查找成功的平均查找长度和查找所花费的时间。如果查找失败，则输出“查找失败”的提示。

【实验提示】

不同的检索策略所采取的数据结构不一样，算法实现的过程不一样，但查找结果是一样的。

假设存储英文文章的文本文件 InFile.txt 的内容如图 1 所示，下面给出系统运行的部分参考截图。

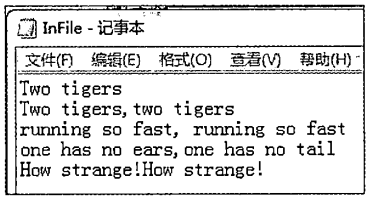


图 1　文件 InFile.txt 的内容

主界面结果如图 2 所示。

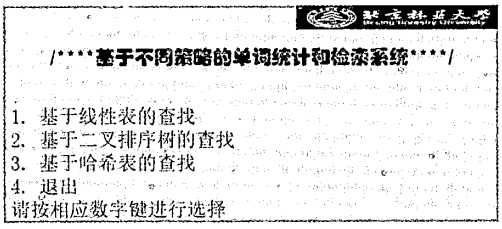


图 2　系统主界面

对于图 2，选择“1”后进入“基于线性表的查找”，结果如图 3 所示。

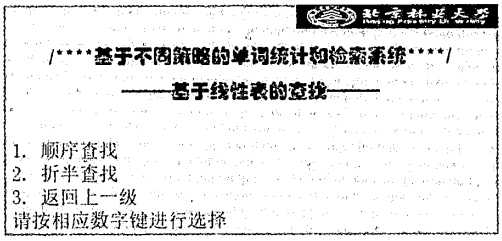


图 3　基于线性表的查找

对于图 3，选择“1”后进入“顺序查找”，结果如图 4 所示。

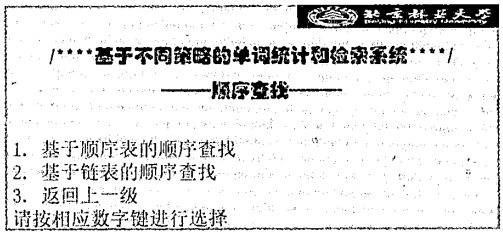


图 4　顺序查找

对于图 4，选择“1”后进入“基于顺序表的顺序查找”，结果如图 5 所示。

对于图 5，选择“1”后进入“词频统计”，实现词频统计功能。要求统计出所有单词的总数和每个单词的词频。统计结果全部输出到对应的文件 OutFile1.txt 中，该文件中的数据总计 n+1 行，第一行 n 为所有单词的总数，后面 n 行为每个单词及其出现的频率（单词和频率用空格分隔）。

文件 OutFile1.txt 的示例如图 6 所示。

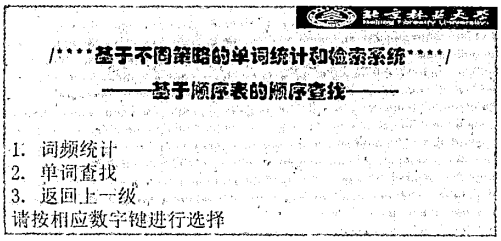


图 5　基于顺序表的顺序查找

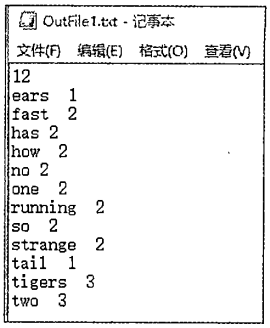


图 6　词频统计

对于图 5，选择“2”后进入“单词查找”，输入待查找的单词后，如果查找成功，结果如图 7 所示。首先显示此单词的词频，之后分别给出查找该单词所花的时间（单位为毫秒、微秒或纳秒，可以利用高精度的时间函数做计算来实现）和平均查找长度。如果查找失败，结果如图 8 所示。

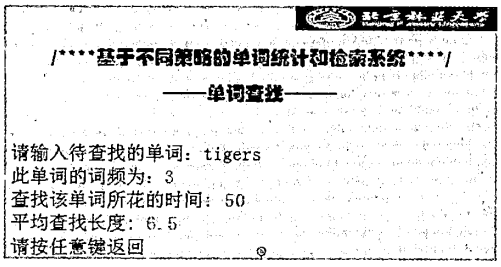


图 7　单词查找成功

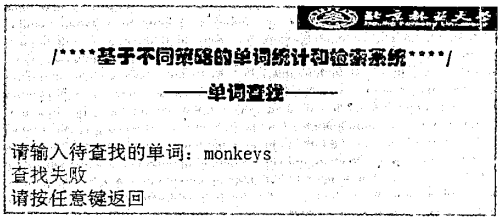


图 8　单词查找失败

对于其他 5 种检索策略，查找结果同图 7 或图 8 所示，只是查找成功时所花的时间和平均查找长度不同而已。

【实验选做】

1．用窗体版界面取代控制台界面。

2．程序中所用到的排序算法选用先进的排序算法（快速排序或归并排序或堆排序）。

3．实现低频词过滤功能，要求将出现频率（词频/单词总数）低于 10%的单词删除，可以选择不同的存储结构分别实现。

（1）顺序表。

（2）链表。

（3）二叉排序树。

（4）基于开放地址法的哈希表。

（5）基于链地址法的哈希表。