第 1 章 绪 论

课后习题讲解

1. 填空

⑴（ ）是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。
【解答】数据元素

⑵（ ）是数据的最小单位，（ ）是讨论数据结构时涉及的最小数据单位。
【解答】数据项，数据元素
【分析】数据结构指的是数据元素以及数据元素之间的关系。

⑶ 从逻辑关系上讲，数据结构主要分为（ ）、（ ）、（ ）和（ ）。
【解答】集合，线性结构，树结构，图结构

⑷ 数据的存储结构主要有（ ）和（ ）两种基本方法，不论哪种存储结构，都要存储两方面的内容：（ ）和（ ）。
【解答】顺序存储结构，链接存储结构，数据元素，数据元素之间的关系

⑸ 算法具有五个特性，分别是（ ）、（ ）、（ ）、（ ）、（ ）。
【解答】有零个或多个输入，有一个或多个输出，有穷性，确定性，可行性

⑹ 算法的描述方法通常有（ ）、（ ）、（ ）和（ ）四种，其中，（ ）被称为算法语言。
【解答】自然语言，程序设计语言，流程图，伪代码，伪代码

⑺ 在一般情况下，一个算法的时间复杂度是（ ）的函数。
【解答】问题规模

⑻ 设待处理问题的规模为 n，若一个算法的时间复杂度为一个常数，则表示成数量级的形式为（ ），若为 n*log25n，则表示成数量级的形式为（ ）。
【解答】O(1)，O(nlog2n)
【分析】用大 O 记号表示算法的时间复杂度，需要将低次幂去掉，将最高次幂的系数去掉。

2. 选择题

⑴ 顺序存储结构中数据元素之间的逻辑关系是由（ ）表示的，链接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
【解答】C，D
【分析】顺序存储结构就是用一维数组存储数据结构中的数据元素，其逻辑关系由存储位置（即元素在数组中的下标）表示；链接存储结构中一个数据元素对应链表中的一个结点，元素之间的逻辑关系由结点中的指针表示。

⑵ 假设有如下遗产继承规则：丈夫和妻子可以相互继承遗产；子女可以继承父亲或母亲的遗产；子女间不能相互继承。则表示该遗产继承关系的最合适的数据结构应该是（ ）。
A 树 B 图 C 线性表 D 集合

【解答】B

【分析】将丈夫、妻子和子女分别作为数据元素，根据题意画出逻辑结构图。

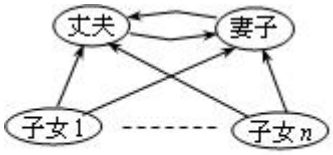


图 1-2　遗产继承逻辑结构图

⑶ 算法指的是（ ）。

A 对特定问题求解步骤的一种描述，是指令的有限序列。

B 计算机程序 C 解决问题的计算方法 D 数据处理

【解答】A

【分析】计算机程序是对算法的具体实现；简单地说，算法是解决问题的方法；数据处理是通过算法完成的。所以，只有 A 是算法的准确定义。

⑷ 下面（ ）不是算法所必须具备的特性。

A 有穷性 B 确切性 C 高效性 D 可行性

【解答】C

【分析】高效性是好算法应具备的特性。

⑸ 算法分析的目的是（ ），算法分析的两个主要方面是（ ）。

A 找出数据结构的合理性 B 研究算法中输入和输出的关系

C 分析算法的效率以求改进 D 分析算法的易读性和文档性

E 空间性能和时间性能 F 正确性和简明性

G 可读性和文档性 H 数据复杂性和程序复杂性

【解答】C，E

3. 判断题

⑴ 算法的时间复杂度都要通过算法中的基本语句的执行次数来确定。

【解答】错。时间复杂度要通过算法中基本语句执行次数的数量级来确定。

⑵ 每种数据结构都具备三个基本操作：插入、删除和查找。

【解答】错。如数组就没有插入和删除操作。此题注意是每种数据结构。

⑶ 所谓数据的逻辑结构指的是数据之间的逻辑关系。

【解答】错。是数据之间的逻辑关系的整体。

⑷ 逻辑结构与数据元素本身的内容和形式无关。

【解答】对。因此逻辑结构是数据组织的主要方面。

⑸ 基于某种逻辑结构之上的基本操作，其实现是唯一的。

【解答】错。基本操作的实现是基于某种存储结构设计的，因而不是唯一的。

4. 分析以下各程序段，并用大 O 记号表示其执行时间。

```
(1) i=1; k=0;
    while (i<n-1)
    {
      k=k+10*i;
      i++;
    }
```

```
(2) i=1; k=0;
    do
    {
      k=k+10*i;
      i++;
    } while (i<=n)
```

```
(3) i=1; j=0;
    while (i+j<=n)
      if (i>j) j++;
        else i++;
```

```
(4) y=0;
    while ((y+1)*(y+1)<=n)
        y=y+1;
```

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

【解答】⑴ 基本语句是 k=k+10*i，共执行了 n-2 次，所以 T(n)=O(n)。

⑵ 基本语句是 k=k+10*i，共执行了 n 次，所以 T(n)=O(n)。

⑶ 分析条件语句，每循环一次，i+j 整体加 1，共循环 n 次，所以 T(n)=O(n)。

⑷ 设循环体共执行 T(n)次，每循环一次，循环变量 y 加 1，最终 T(n)=y，即：(T(n)+1)2≤n，所以 T(n)=O(n 1/2)。

⑸ x++是基本语句，所以 $T(n)=\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=O(n^3)$

5．设有数据结构（D，R），其中 D={1, 2, 3, 4, 5, 6}，R={(1,2),(2,3),(2,4),(3,4),(3,5),(3,6),(4,5),(4,6)}。试画出其逻辑结构图并指出属于何种结构。

【解答】其逻辑结构图如图 1-3 所示，它是一种图结构。

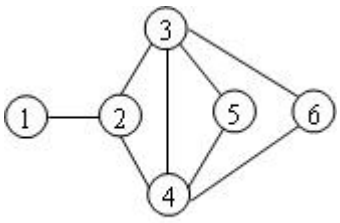


图 1-3　逻辑结构图

6. 为整数定义一个抽象数据类型，包含整数的常见运算，每个运算对应一个基本操作，每个基本操作的接口需定义前置条件、输入、功能、输出和后置条件。

【解答】整数的抽象数据类型定义如下：

ADT integer

Data

整数 a：可以是正整数(1, 2, 3, …)、负整数(-1, -2, -3, …)和零

Operation

Constructor

前置条件：整数 a 不存在

输入：一个整数 b

功能：构造一个与输入值相同的整数

输出：无

后置条件：整数 a 具有输入的值

Set

前置条件：存在一个整数 a

输入：一个整数 b

功能：修改整数 a 的值，使之与输入的整数值相同

输出：无

后置条件：整数 a 的值发生改变

Add

前置条件：存在一个整数 a

输入：一个整数 b

功能：将整数 a 与输入的整数 b 相加

输出：相加后的结果

后置条件：整数 a 的值发生改变

Sub

前置条件：存在一个整数 a

输入：一个整数 b

功能：将整数 a 与输入的整数 b 相减

输出：相减的结果

后置条件：整数 a 的值发生改变

Multi

前置条件：存在一个整数 a

输入：一个整数 b

功能：将整数 a 与输入的整数 b 相乘

输出：相乘的结果

后置条件：整数 a 的值发生改变

Div

前置条件：存在一个整数 a

输入：一个整数 b

功能：将整数 a 与输入的整数 b 相除

输出：若整数 b 为零，则抛出除零异常，否则输出相除的结果

后置条件：整数 a 的值发生改变

Mod

前置条件：存在一个整数 a

输入：一个整数 b

功能：求当前整数与输入整数的模，即正的余数

输出：若整数 b 为零，则抛出除零异常，否则输出取模的结果

后置条件：整数 a 的值发生改变

Equal

前置条件：存在一个整数 a

输入：一个整数 b

功能：判断整数 a 与输入的整数 b 是否相等

输出：若相等返回 1，否则返回 0

后置条件：整数 a 的值不发生改变

endADT

7. 求多项式 A(x)的算法可根据下列两个公式之一来设计：

⑴ A(x)=anxn+an-1xn-1+…+a1x+a0

⑵ A(x)=(…(anx+an-1)x+…+a1)x)+a0

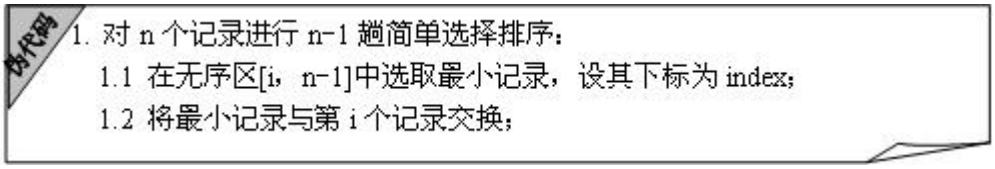
根据算法的时间复杂度分析比较这两种算法的优劣。

【解答】第二种算法的时间性能要好些。第一种算法需执行大量的乘法运算，而第二种算法进行了优化，减少了不必要的乘法运算。

8. 算法设计（要求：算法用伪代码和 C++描述，并分析最坏情况下的时间复杂度）

⑴ 对一个整型数组 A[n]设计一个排序算法。

【解答】下面是简单选择排序算法的伪代码描述。



下面是简单选择排序算法的 C++描述。

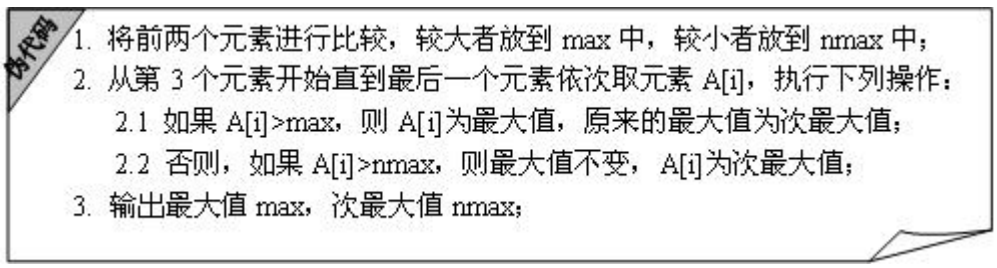
简单选择排序算法 SelectSort

```
void SelectSort(int r[ ], int n)
{
   for (i=0; i<n-1; i++)          //对 n 个记录进行 n-1 趟简单选择排序
   {
       index=i;
       for (j=i+1; j<n; j++)      //在无序区中选取最小记录
           if (r[j]<r[index]) index=j;
       if (index!=i) r[i]←→r[index];   //交换元素
   }
}
```

分析算法，有两层嵌套的 for 循环，所以，$T(n)=\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}1=O(n^2)$ 。

⑵ 找出整型数组 A[n]中元素的最大值和次最大值。

【解答】算法的伪代码描述如下：



算法的 C++描述如下：

最大值和次最大值算法 Max_NextMax

```
void Max_NextMax(int A[ ], int n, int & max, int & nmax)
{
   if (A[0]>= A[1]) {
                    max=A[0];
                    nmax=A[1];
   }
   else {
        max=A[1];
        nmax=A[0];
   }
   for (i=2; i<n; i++)
        if (A[i]>=max) {
                        nmax=max;
                        max=A[i];
        }
        else if (A[i]>nmax) nmax=A[i];
   cout<<"最大值为："<<max<<"\n 次最大值为："<<nmax<<endl;
}
```

分析算法，只有一层循环，共执行 n-2 次，所以，T(n)=O(n)。

学习自测及答案

1．顺序存储结构的特点是（ ），链接存储结构的特点是（ ）。

【解答】用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，用指示元素存储地址的指针表示数据元素之间的逻辑关系。

2. 算法在发生非法操作时可以作出处理的特性称为（ ）。

【解答】健壮性

3. 常见的算法时间复杂度用大O记号表示为：常数阶( )、对数阶( )、线性阶 ( )、平方阶( )和指数阶( )。

【解答】O(1)，O(log2n)，O(n)，O(n2)，O(2n)

4．将下列函数按它们在 n 时的无穷大阶数，从小到大排列。

n, n-n3+7n5, nlogn, 2n/2, n3, log2n, n1/2+log2n, (3/2)n, n!, n2+log2n

【解答】log2n, n1/2+log2n, n, nlog2n, n2+log2n, n3, n-n3+7n5, 2n/2, (3/2)n, n!

5．试描述数据结构和抽象数据类型的概念与程序设计语言中数据类型概念的区别。

【解答】数据结构是指相互之间存在一定关系的数据元素的集合。而抽象数据类型是指一个数据结构以及定义在该结构上的一组操作。程序设计语言中的数据类型是一个值的集合和定义在这个值集上一组操作的总称。抽象数据类型可以看成是对数据类型的一种抽象。

6. 对下列用二元组表示的数据结构,试分别画出对应的逻辑结构图，并指出属于何种结构。

⑴ A=(D，R)， 其中 D={a1, a2, a3, a4}，R={ }

⑵ B=(D，R)， 其中 D={a, b, c, d, e, f}，R={,,,,}

⑶ C=(D，R)，其中 D={a，b，c，d，e，f}，R={,,,,,}

⑷ D=(D，R)， 其中 D={1, 2, 3, 4, 5, 6}，

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

【解答】⑴ 属于集合，其逻辑结构图如图 1-4(a)所示；⑵ 属于线性结构，其逻辑结构图如图 1-4(b)所示；⑶ 属于树结构，其逻辑结构图如图 1-4(c)所示；⑷ 属于图结构，其逻辑结构图如图 1-4(d)所示。

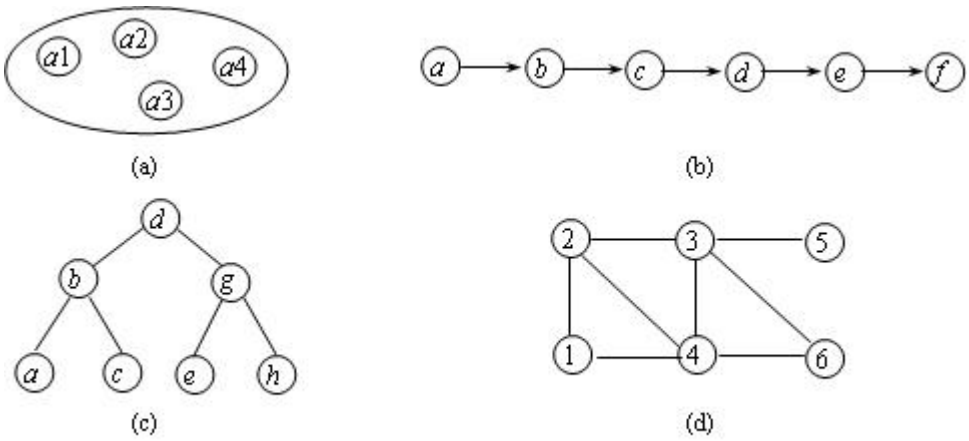


图 1-4　第 7 题对应的逻辑结构图

7. 求下列算法的时间复杂度。

```
count=0; x=1;
while (x {
x*=2;
count++;
}
return count;
```

【解答】O(log2n)

第 2 章 线性表

课后习题讲解

1. 填空

⑴ 在顺序表中，等概率情况下，插入和删除一个元素平均需移动（ ）个元素，具体移动元素的个数与（ ）和（ ）有关。

【解答】表长的一半，表长，该元素在表中的位置

⑵ 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 个元素的存储地址是（ ）。

【解答】108

【分析】第 5 个元素的存储地址=第 1 个元素的存储地址＋(5－1)×2=108

⑶ 设单链表中指针 p 指向结点 A，若要删除 A 的后继结点（假设 A 存在后继结点），则需修改指针的操作为（ ）。

【解答】p->next=(p->next)->next

⑷ 单链表中设置头结点的作用是（ ）。

【解答】为了运算方便

【分析】例如在插入和删除操作时不必对表头的情况进行特殊处理。

⑸ 非空的单循环链表由头指针 head 指示，则其尾结点（由指针 p 所指）满足（ ）。

【解答】p->next=head

【分析】如图 2-8 所示。

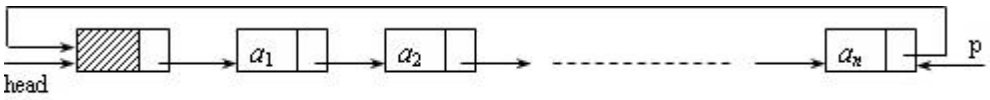


图 2-8 尾结点 p 与头指针 head 的关系示意图

⑹ 在由尾指针 rear 指示的单循环链表中，在表尾插入一个结点 s 的操作序列是（ ）；删除开始结点的操作序列为（ ）。

【解答】s->next =rear->next; rear->next =s; rear =s;

q=rear->next->next; rear->next->next=q->next; delete q;

【分析】操作示意图如图 2-9 所示：

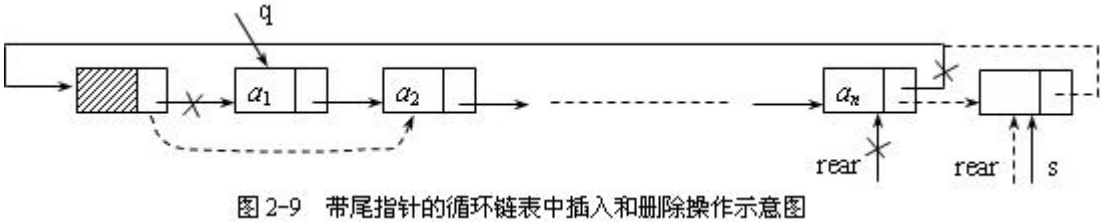


图 2-9 带尾指针的循环链表中插入和删除操作示意图

⑺ 一个具有 n 个结点的单链表，在指针 p 所指结点后插入一个新结点的时间复杂度为（ ）；在给定值为 x 的结点后插入一个新结点的时间复杂度为（ ）。

【解答】O(1)，O(n)

【分析】在 p 所指结点后插入一个新结点只需修改指针，所以时间复杂度为 O(1)；而在给定值为 x 的结点后插入一个新结点需要先查找值为 x 的结点，所以时间复杂度为 O(n)。

⑻ 可由一个尾指针唯一确定的链表有（ ）、（ ）、（ ）。

【解答】循环链表，循环双链表，双链表

2. 选择题

⑴ 线性表的顺序存储结构是一种（ ）的存储结构，线性表的链接存储结构是一种（ ）的存储结构。

A 随机存取 B 顺序存取 C 索引存取 D 散列存取

【解答】A，B

【分析】参见 2.2.1。

⑵ 线性表采用链接存储时，其地址（ ）。

A 必须是连续的　　B 部分地址必须是连续的

C 一定是不连续的　　D 连续与否均可以

【解答】D

【分析】线性表的链接存储是用一组任意的存储单元存储线性表的数据元素，这组存储单元可以连续，也可以不连续，甚至可以零散分布在内存中任意位置。

⑶ 单循环链表的主要优点是（ ）。

A 不再需要头指针了

B 从表中任一结点出发都能扫描到整个链表；

C 已知某个结点的位置后，能够容易找到它的直接前趋；

D 在进行插入、删除操作时，能更好地保证链表不断开。

【解答】B

⑷ 链表不具有的特点是（ ）。

A 可随机访问任一元素 B 插入、删除不需要移动元素

C 不必事先估计存储空间 D 所需空间与线性表长度成正比

【解答】A

⑸ 若某线性表中最常用的操作是取第 i 个元素和找第 i 个元素的前趋，则采用（ ）存储方法最节省时间。

A 顺序表 B 单链表 C 双链表 D 单循环链表

【解答】A

【分析】线性表中最常用的操作是取第 i 个元素，所以，应选择随机存取结构即顺序表，同时在顺序表中查找第 i 个元素的前趋也很方便。单链表和单循环链表既不能实现随机存取，查找第 i 个元素的前趋也不方便，双链表虽然能快速查找第 i 个元素的前趋，但不能实现随机存取。

⑹ 若链表中最常用的操作是在最后一个结点之后插入一个结点和删除第一个结点，则采用（ ）存储方法最节省时间。

A 单链表 B 带头指针的单循环链表 C 双链表 D 带尾指针的单循环链表

【解答】D

【分析】在链表中的最后一个结点之后插入一个结点需要知道终端结点的地址，所以，单链表、带头指针的单循环链表、双链表都不合适，考虑在带尾指针的单循环链表中删除第一个结点，其时间性能是 O(1)，所以，答案是 D 。

⑺ 若链表中最常用的操作是在最后一个结点之后插入一个结点和删除最后一个结点，则采用（ ）存储方法最节省运算时间。

A 单链表 B 循环双链表 C 单循环链表 D 带尾指针的单循环链表

【解答】B

【分析】在链表中的最后一个结点之后插入一个结点需要知道终端结点的地址，所以，单链表、单循环链表都不合适，删除最后一个结点需要知道终端结点的前驱结点的地址，所以，带尾指针的单循环链表不合适，而循环双链表满足条件。

⑻ 在具有 n 个结点的有序单链表中插入一个新结点并仍然有序的时间复杂度是（ ）。

A O(1) B O(n) C O(n2) D O(nlog2n)

【解答】B

【分析】首先应顺序查找新结点在单链表中的位置。

⑼ 对于 n 个元素组成的线性表，建立一个有序单链表的时间复杂度是（ ）。

A O(1) B O(n) C O(n2) D O(nlog2n)

【解答】C

【分析】该算法需要将 n 个元素依次插入到有序单链表中，而插入每个元素需 O(n)。

⑽ 使用双链表存储线性表，其优点是可以（ ）。

A 提高查找速度 B 更方便数据的插入和删除

C 节约存储空间 D 很快回收存储空间

【解答】B

【分析】在链表中一般只能进行顺序查找，所以，双链表并不能提高查找速度，因为双链表中有两个指针域，显然不能节约存储空间，对于动态存储分配，回收存储空间的速度是一样的。由于双链表具有对称性，所以，其插入和删除操作更加方便。

⑾ 在一个单链表中，已知 q 所指结点是 p 所指结点的直接前驱，若在 q 和 p 之间插入 s 所指结点，则执行（ ）操作。

A s->next=p->next; p->next=s; B q->next=s; s->next=p;

C p->next=s->next; s->next=p; D p->next=s; s->next=q;

【解答】B

【分析】注意此题是在 q 和 p 之间插入新结点，所以，不用考虑修改指针的顺序。

⑿ 在循环双链表的 p 所指结点后插入 s 所指结点的操作是（ ）。

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

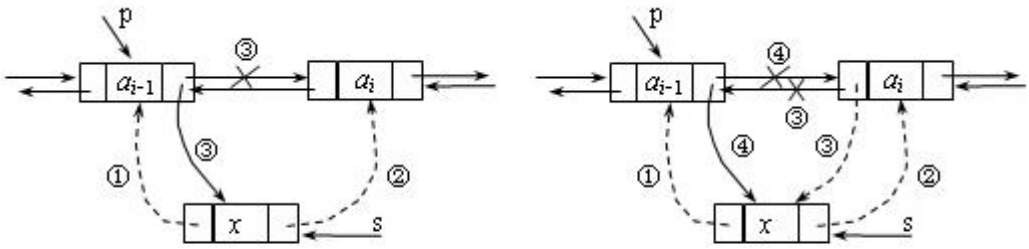
B p->next=s; p->next->prior=s; s->prior=p; s->next=p->next;

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

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

【解答】D

【分析】在链表中，对指针的修改必须保持线性表的逻辑关系，否则，将违背线性表的逻辑特征，图 2-10 给出备选答案 C 和 D 的图解。



(a) 备选答案 C 操作示意图(第 4 步指针修改无法进行)　　(b) 备选答案 D 操作示意图

图 2-10　双链表插入操作修改指针操作示意图

3. 判断题

⑴ 线性表的逻辑顺序和存储顺序总是一致的。

【解答】错。顺序表的逻辑顺序和存储顺序一致，链表的逻辑顺序和存储顺序不一定一致。

⑵ 线性表的顺序存储结构优于链接存储结构。

【解答】错。两种存储结构各有优缺点。

⑶ 设 p，q 是指针，若 p=q，则*p=*q。

【解答】错。p=q 只能表示 p 和 q 指向同一起始地址，而所指类型则不一定相同。

⑷ 线性结构的基本特征是：每个元素有且仅有一个直接前驱和一个直接后继。

【解答】错。每个元素最多只有一个直接前驱和一个直接后继，第一个元素没有前驱，最后一个元素没有后继。

⑸ 在单链表中，要取得某个元素，只要知道该元素所在结点的地址即可，因此单链表是随机存取结构。

【解答】错。要找到该结点的地址，必须从头指针开始查找，所以单链表是顺序存取结构。

4. 请说明顺序表和单链表各有何优缺点，并分析下列情况下，采用何种存储结构更好些。

⑴ 若线性表的总长度基本稳定，且很少进行插入和删除操作，但要求以最快的速度存取线性表中的元素。

⑵ 如果 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发生变化。

⑶ 描述一个城市的设计和规划。

【解答】顺序表的优点：① 无需为表示表中元素之间的逻辑关系而增加额外的存储空间；② 可以快速地存取表中任一位置的元素（即随机存取）。顺序表的缺点：① 插入和删除操作需移动大量元素；② 表的容量难以确定；③ 造成存储空间的“碎片”。

单链表的优点：① 不必事先知道线性表的长度；② 插入和删除元素时只需修改指针，不用移动元素。单链表的缺点：① 指针的结构性开销；② 存取表中任意元素不方便，只能进行顺序存取。

⑴ 应选用顺序存储结构。因为顺序表是随机存取结构，单链表是顺序存取结构。本题很少进行插入和删除操作，所以空间变化不大，且需要快速存取，所以应选用顺序存储结构。

⑵ 应选用链接存储结构。链表容易实现表容量的扩充，适合表的长度动态发生变化。

⑶ 应选用链接存储结构。因为一个城市的设计和规划涉及活动很多，需要经常修改、扩充和删除各种信息，才能适应不断发展的需要。而顺序表的插入、删除的效率低，故不合适。

5．算法设计

⑴ 设计一个时间复杂度为O(n)的算法，实现将数组A[n]中所有元素循环右移k个位置。

【解答】算法思想请参见主教材第一章思想火花。下面给出具体算法。

循环右移算法 Converse

```
void Converse(int A[ ], int n, int k)
{
    Reverse(A, 0, k-1);
    Reverse(A, k, n-1);
    Reverse(A, 0, n-1);
}
void Reverse(int A[ ], int from, int to)       //将数组 A 中元素从 from 到 to 逆置
{
    for (i=0; i<(to-from+1)/2; i++)
        A[from+i]←→A[to-i];       //交换元素
}
```

分析算法，第一次调用Reverse函数的时间复杂度为O(k)，第二次调用Reverse函数的时间复杂度为O(n-k)，第三次调用Reverse函数的时间复杂度为O(n)，所以，总的时间复杂度为O(n)。

⑵ 已知数组A[n]中的元素为整型，设计算法将其调整为左右两部分，左边所有元素为奇数，右边所有元素为偶数，并要求算法的时间复杂度为O(n)。

【解答】从数组的两端向中间比较，设置两个变量i和j，初始时i=0，j=n-1，若A[i]为偶数并且A[j]为奇数，则将A[i]与A[j]交换。具体算法如下：

数组奇偶调整算法 Adjust

```
void Adjust(int A[ ], n)
{
  i=0; j=n-1;
  while (i<j)
  {
    while (A[i] % 2!=0) i++;
    while (A[j] % 2==0) j--
    if (i<j) A[i]←→A[j];
  }
}
```

分析算法，两层循环将数组扫描一遍，所以，时间复杂度为O(n)。

⑶ 试编写在无头结点的单链表上实现线性表的插入操作的算法，并和带头结点的单链表上的插入操作的实现进行比较。

【解答】参见2.2.3。

⑷ 试分别以顺序表和单链表作存储结构，各写一实现线性表就地逆置的算法。

【解答】顺序表的逆置，即是将对称元素交换，设顺序表的长度为 length，则将表中第 i 个元素与第 length-i-1 个元素相交换。具体算法如下：

顺序表逆置算法 Reverse

```
template <class T>
void Reverse(T data[ ], int length)
{
    for (i=0; i<=length/2; i++)
    {
        temp=data[i];
        data[i]=data[length-i-1];
        data[length-i-1]=temp;
    }
}
```

单链表的逆置请参见 2.2.4 算法 2-4 和算法 2-6。

⑸ 假设在长度大于 1 的循环链表中，即无头结点也无头指针，s 为指向链表中某个结点的指针，试编写算法删除结点 s 的前趋结点。

【解答】利用单循环链表的特点，通过指针 s 可找到其前驱结点 r 以及 r 的前驱结点 p，然后将结点 r 删除，如图 2-11 所示，具体算法如下：

循环链表删除算法 Del

```
template <class T>
void Del(Node<T> *s)
{
  p=s;              //工作指针 p 初始化，查找 s 的前驱结点的前驱结点，用 p 指示
  while (p->next->next!=s)
      p=p->next;
  r=p->next;        //r 为 p 的前驱结点，q 为 r 的前驱结点
  p->next=s;        //删除 r 所指结点
  delete r;
}
```

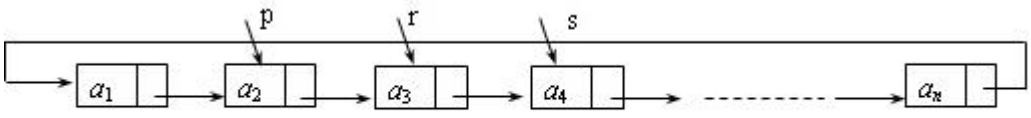


图 2-11　删除结点 s 的前驱结点操作示意图

⑹ 已知一单链表中的数据元素含有三类字符：字母、数字和其他字符。试编写算法，构造三个循环链表，使每个循环链表中只含同一类字符。

【解答】在单链表 A 中依次取元素，若取出的元素是字母，把它插入到字母链表 B 中，若取出的元素是数字，则把它插入到数字链表 D 中，直到链表的尾部，这样表 B，D，A 中分别存放字母、数字和其他字符。具体算法如下：

单链表拆分算法 Adjust

```
template <class T>
void Adjust(Node<T> *A, Node<int> *D, Node<char> *B)
{
  D=new Node<int>; D->next=D;       //创建空循环链表 D，存放数字
  B=new Node<char>; B->next=B;      //创建空循环链表 B，存放字符
  p=A; q=p->next;                   //工作指针 q 初始化
  while (q)
  {
    if (('A'<=q->data)&&(q->data>='Z')||('a' <=q->data)&&(q->data>='z')) {
                    p->next=q->next;
                    q->next=B->next;
                    B->next=q;      //采用头插法插在循环链表 B 的头结点的后面
    }
    else if (('0'<=q->data)&&(q->data>='9')) {
                    p->next=q->next;
                    q->next=D->next;
                    D->next=q;    //采用头插法插在循环链表 D 的头结点的后面
        }
        else p=q;
    q=p->next;
  }
  p->next=A; R=A;    //将链表 A 构造为循环链表，为除字母和数字的其他字符
}
```

⑺ 设单链表以非递减有序排列，设计算法实现在单链表中删去值相同的多余结点。

【解答】从头到尾扫描单链表，若当前结点的元素值与后继结点的元素值不相等，则指针后移；否则删除该后继结点。具体算法如下：

单链表删除相同值算法 Purge

```
void Purge(Node<T> *first)
{
    p=first->next;
    while (p->next)
      if (p->data= =p->next->data) {
                                    q=p->next;
                                    p->next=q->next;
                                    delete q;
      }
      else p=p->next;
}
```

⑻ 判断带头结点的双循环链表是否对称。

【解答】设工作指针 p 和 q 分别指向循环双链表的开始结点和终端结点，若结点 p 和结点 q 的数据域相等，则工作指针 p 后移，工作指针 q 前移，直到指针 p 和指针 q 指向同一结点（循环双链表中结点个数为奇数），或结点 q 成为结点 p 的前驱（循环双链表中结点个数为偶数）。如图 2-12 所示。

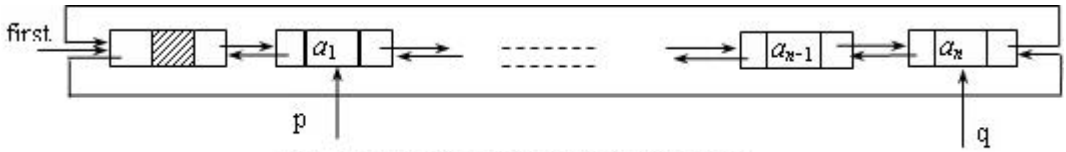


图 2-12　判断循环双链表对称的操作示意图

判断双链表对称算法 Equal

```
template <class T>
struct DulNode
{
   T data;
   DulNode<T> *prior, *next;
};
template <class T>
bool Equal (DulNode<T> *first)
{
   p=first->next; q=first->prior;
   while (p!=q && p->prior!=q)
       if (p->data= =q->data) {
                                  p=p->next;     //工作指针 p 后移
                                  q=q->prior;    //工作指针 q 前移
       }
       else return 0;
   return 1;
}
```

学习自测及答案

1. 已知一维数组 A 采用顺序存储结构，每个元素占用 4 个存储单元，第 9 个元素的地址为 144，则第一个元素的地址是（ ）。

A 108 B 180 C 176 D 112

【解答】D

2. 在长度为 n 的线性表中查找值为 x 的数据元素的时间复杂度为：　（ ）。

A O(0) B O(1) C O(n) D O(n2)

【解答】C

3. 在一个长度为 n 的顺序表的第 i（1≤i≤n+1）个元素之前插入一个元素，需向后移动（ ）个元素，删除第 i（1≤i≤n）个元素时，需向前移动（ ）个元素。

【解答】n-i+1，n-i

4. 在单链表中，除了头结点以外，任一结点的存储位置由（ ）指示。

【解答】其前趋结点的指针域

5. 当线性表采用顺序存储结构时，其主要特点是（ ）。

【解答】逻辑结构中相邻的结点在存储结构中仍相邻

6．在双链表中，每个结点设置了两个指针域，其中一个指向（ ）结点，另一个指向（ ）结点。
【解答】前驱，后继

7．设 A 是一个线性表（a1, a2, …, an），采用顺序存储结构，则在等概率的前提下，平均每插入一个元素需要移动的元素个数为多少？若元素插在 ai 与 ai+1 之间（1≤i≤n）的概率为 $\frac{n-i}{n(n+1)/2}$ ，则平均每插入一个元素所要移动的元素个数又是多少？
【解答】

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

$$\sum_{i=1}^{n}\frac{(n-i)^2}{n(n+1)/2}=(2n+1)/3$$

。

8．线性表存放在整型数组 A[arrsize]的前 elenum 个单元中，且递增有序。编写算法，将元素 x 插入到线性表的适当位置上，以保持线性表的有序性，并且分析算法的时间复杂度。
【解答】本题是在一个递增有序表中插入元素 x，基本思路是从有序表的尾部开始依次取元素与 x 比较，若大于 x，此元素后移一位，再取它前面一个元素重复上述步骤；否则，找到插入位置，将 x 插入。具体算法如下：

有序顺序表插入算法 Insert

```
const int arrsize =100;
class SeqList
{
private:
   int data[arrsize];
   int elenum;
public:
   Insert(int x);
   …
};
void SeqList::Insert(int x)
{
    if (elenum= =arrsize) throw "overflow";
    else {
         i=elenum-1;
         while (i>=1 && x<data[i])
         {
             data[i+1]=data[i];
             i++;
         }
         data[i+1]=x;
    }
}
```

9. 已知单链表中各结点的元素值为整型且递增有序，设计算法删除链表中所有大于 mink 且小于 maxk 的所有元素，并释放被删结点的存储空间。

【解答】因为是在有序单链表上的操作，所以，要充分利用其有序性。在单链表中查找第一个大于 mink 的结点和第一个小于 maxk 的结点，再将二者间的所有结点删除。

有序链表删除算法 DeleteBetween

```
template <class T>
void DeleteBetween(Node<T> *first, int mink, int maxk)
{
    p=first;
    while (p->next && p->next->data<=mink)
        p=p->next;
    if (p->next) {
                q=p->next;
                while (q->data<maxk)
                {
                    u=q->next;
                    p->next=q->next;
                    delete q;
                    q=u;
                }
    }
}
```

10．设单循环链表 L1，对其遍历的结果是：x1, x2, x3,..., xn-1, xn。请将该循环链表拆成两个单循环链表 L1 和 L2，使得 L1 中含有原 L1 表中序号为奇数的结点且遍历结果为：x1, x3,... ；L2 中含有原 L1 表中序号为偶数的结点且遍历结果为：... , x4, x2。

【解答】算法如下：

循环链表拆分算法 DePatch

```
template <class T>
Node<T> *DePatch(Node<T> *L1)
{
     L2=new Node<T>; L2->next=L2;
     q=L1->next; L1->next=L1;
     p=L1; i=1;
     while (q!=L1)
     {
         if (i % 2==1) {    //应用尾插法
                        p->next=q; p=q; p->next=L1;
                        q=q->next; i++;

         }
         else {
              L2->next=q; u=q->next; q->next=L2->next;
              q=u; i++;
         }
     }
}
```

第 3 章 特殊线性表——栈、队列和串

课后习题讲解

1. 填空

⑴ 设有一个空栈，栈顶指针为 1000H，现有输入序列为 1、2、3、4、5， 经过 push，push，pop，push，pop，push，push 后，输出序列是（ ），栈顶指针为（ ）。
【解答】23，1003H

⑵ 栈通常采用的两种存储结构是（ ）；其判定栈空的条件分别是（ ），判定栈满的条件分别是（ ）。
【解答】顺序存储结构和链接存储结构（或顺序栈和链栈），栈顶指针 top= -1 和 top=NULL，栈顶指针 top 等于数组的长度和内存无可用空间

⑶（ ）可作为实现递归函数调用的一种数据结构。
【解答】栈
【分析】递归函数的调用和返回正好符合后进先出性。

⑷ 表达式 a*(b+c)-d 的后缀表达式是（ ）。
【解答】abc+*d-
【分析】将中缀表达式变为后缀表达式有一个技巧：将操作数依次写下来，再将算符插在它的两个操作数的后面。

⑸ 栈和队列是两种特殊的线性表，栈的操作特性是（ ），队列的操作特性是（ ），栈和队列的主要区别在于（ ）。
【解答】后进先出，先进先出，对插入和删除操作限定的位置不同

⑹ 循环队列的引入是为了克服（ ）。
【解答】假溢出

⑺ 数组 Q[n]用来表示一个循环队列，front 为队头元素的前一个位置，rear 为队尾元素的位置，计算队列中元素个数的公式为（ ）。
【解答】（rear-front+n）% n
【分析】也可以是（rear-front）% n，但 rear-front 的结果可能是负整数，而对一个负整数求模，其结果在不同的编译器环境下可能会有所不同。

⑻ 用循环链表表示的队列长度为 n，若只设头指针，则出队和入队的时间复杂度分别是（ ）和（ ）。
【解答】O(1)，O(n)
【分析】在带头指针的循环链表中，出队即是删除开始结点，这只需修改相应指针；入队即是在终端结点的后面插入一个结点，这需要从头指针开始查找终端结点的地址。

⑼ 串是一种特殊的线性表，其特殊性体现在（ ）。
【解答】数据元素的类型是一个字符

⑽ 两个串相等的充分必要条件是（ ）。
【解答】长度相同且对应位置的字符相等
【分析】例如"abc"≠"abc "，"abc"≠"bca"。

2. 选择题

⑴ 若一个栈的输入序列是 1，2，3，…，n，输出序列的第一个元素是 n，则第 i 个输出元素是（ ）。
A 不确定 B n-i C n-i-1 D n-i+1
【解答】D
【分析】此时，输出序列一定是输入序列的逆序。

⑵ 设栈 S 和队列 Q 的初始状态为空，元素 e1、e2、e3、e4、e5、e6 依次通过栈 S，一个元素出栈后即进入队列 Q，若 6 个元素出队的顺序是 e2、e4、e3、e6、e5、e1，则栈 S 的容量至少应该是（ ）。
A 6 B 4 C 3 D 2
【解答】C
【分析】由于队列具有先进先出性，所以，此题中队列形同虚设，即出栈的顺序也是 e2、e4、e3、e6、e5、e1。

⑶ 一个栈的入栈序列是 1，2，3，4，5，则栈的不可能的输出序列是（ ）。
A 54321 B 45321 C 43512 D 12345
【解答】C
【分析】此题有一个技巧：在输出序列中任意元素后面不能出现比该元素小并且是升序（指的是元素的序号）的两个元素。

⑷ 设计一个判别表达式中左右括号是否配对的算法，采用（ ）数据结构最佳
A 顺序表 B 栈 C 队列 D 链表
【解答】B
【分析】每个右括号与它前面的最后一个没有匹配的左括号配对，因此具有后进先出性。

⑸ 在解决计算机主机与打印机之间速度不匹配问题时通常设置一个打印缓冲区，该缓冲区应该是一个（ ）结构。
A 栈 B 队列 C 数组 D 线性表
【解答】B
【分析】先进入打印缓冲区的文件先被打印，因此具有先进先出性。

⑹ 一个队列的入队顺序是 1，2，3，4，则队列的输出顺序是（ ）。
A 4321 B 1234 C 1432 D 3241
【解答】B
【分析】队列的入队顺序和出队顺序总是一致的。

⑺ 栈和队列的主要区别在于（ ）。
A 它们的逻辑结构不一样 B 它们的存储结构不一样
C 所包含的运算不一样 D 插入、删除运算的限定不一样
【解答】D

【分析】栈和队列的逻辑结构都是线性的，都有顺序存储和链接存储，有可能包含的运算不一样，但不是主要区别，任何数据结构在针对具体问题时包含的运算都可能不同。

⑻ 设数组 S[n]作为两个栈 S1 和 S2 的存储空间，对任何一个栈只有当 S[n]全满时才不能进行进栈操作。为这两个栈分配空间的最佳方案是（ ）。

A S1 的栈底位置为 0，S2 的栈底位置为 n-1

B S1 的栈底位置为 0，S2 的栈底位置为 n/2

C S1 的栈底位置为 0，S2 的栈底位置为 n

D S1 的栈底位置为 0，S2 的栈底位置为 1

【解答】A

【分析】两栈共享空间首先两个栈是相向增长的，栈底应该分别指向两个栈中的第一个元素的位置，并注意 C++中的数组下标是从 0 开始的。

⑼ 设有两个串 p 和 q，求 q 在 p 中首次出现的位置的运算称作（ ）。

A 连接 B 模式匹配 C 求子串 D 求串长

【解答】B

3. 判断题

⑴ 有 n 个元素依次进栈，则出栈序列有(n-1)/2 种。

【解答】错。应该有 $\frac{(2n)!}{(n+1)(n!)^2}$ 种。

⑵ 栈可以作为实现过程调用的一种数据结构。

【解答】对。只要操作满足后进先出性，都可以采用栈作为辅助数据结构。

⑶ 在栈满的情况下不能做进栈操作，否则将产生“上溢”。

【解答】对。

⑷ 在循环队列中，front 指向队头元素的前一个位置，rear 指向队尾元素的位置，则队满的条件是 front=rear。

【解答】错。这是队空的判定条件，在循环队列中要将队空和队满的判定条件区别开。

⑸ 空串与空格串是相同的。

【解答】错。空串的长度为零，而空格串的长度不为 0，其长度是串中空格的个数。

4. 设有一个栈，元素进栈的次序为 A，B，C，D，E，能否得到如下出栈序列，若能，请写出操作序列，若不能，请说明原因。

⑴ C，E，A，B，D

⑵ C，B，A，D，E

【解答】⑴不能，因为在 C、E 出栈的情况下，A 一定在栈中，而且在 B 的下面，不可能先于 B 出栈。⑵可以，设 I 为进栈操作，O 为入栈操作，则其操作序列为 IIIOOOIOIO。

5. 举例说明顺序队列的“假溢出”现象。

【解答】假设有一个顺序队列，如图 3-6 所示，队尾指针 rear=4，队头指针 front=1，如果再有元素入队，就会产生“上溢”，此时的“上溢”又称为“假溢出”，因为队列并不是真的溢出了，存储队列的数组中还有 2 个存储单元空闲，其下标分别为 0 和 1。

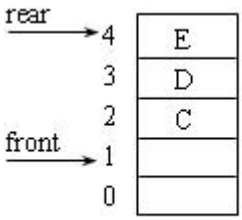


图 3-6　顺序队列的假溢出

6. 在操作序列 push(1)、push(2)、pop、push(5)、push(7)、pop、push(6)之后，栈顶元素和栈底元素分别是什么？（push(k)表示整数 k 入栈，pop 表示栈顶元素出栈。）

【解答】栈顶元素为 6，栈底元素为 1。其执行过程如图 3-7 所示。

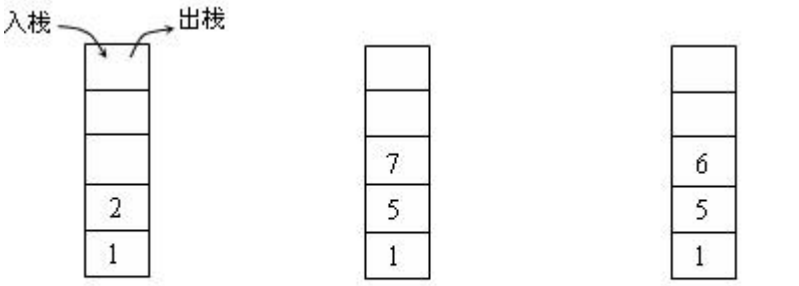


图 3-7　栈的执行过程示意图

7. 在操作序列 EnQueue(1)、EnQueue(3)、DeQueue、EnQueue(5)、EnQueue(7)、DeQueue、EnQueue(9)之后，队头元素和队尾元素分别是什么？（EnQueue(k)表示整数 k 入队，DeQueue 表示队头元素出队）。

【解答】队头元素为 5，队尾元素为 9。其执行过程如图 3-8 所示。

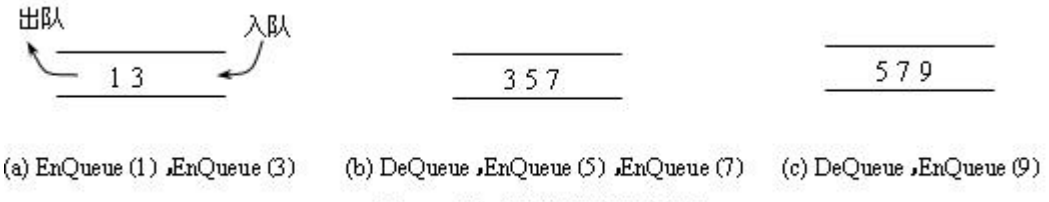


图 3-8　队列的执行过程示意图

8. 空串和空格串有何区别？串中的空格符有何意义？空串在串处理中有何作用？

【解答】不含任何字符的串称为空串，其长度为零。仅含空格的串称为空格串，它的长度为串中空格符的个数。串中的空格符可用来分隔一般的字符，便于人们识别和阅读，但计算串长时应包括这些空格符。空串在串处理中可作为任意串的子串。

9. 算法设计

⑴ 假设以不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结点，但不设头指针。试设计相应的入队和出队的算法。

【解答】出队操作是在循环链表的头部进行，相当于删除开始结点，而入队操作是在循环链表的尾部进行，相当于在终端结点之后插入一个结点。由于循环链表不带头结点，需要处理空表的特殊情况。

入队算法如下：

循环队列入队算法 Enqueue

```
template <class T>
void Enqueue(Node<T> *rear, T x)
{
    s=new Node<T>;
    s->data=x;
    if (rear==NULL) {                    //处理空表的特殊情况
                    rear=s;
                    rear->next=s;
    }
    else {                               //处理除空表以外的一般情况
        s->next=rear->next;
        rear->next=s;
        rear=s;
    }
}
```

出队算法如下：

循环队列出队算法 Dequeue

```
template <class T>
T Dequeue(Node<T> *rear)
{
    if (rear==NULL) throw "underflow";   //判断表空
        else {
            s=rear->next;
            if (s==rear) rear=NULL;      //链表中只有一个结点
                else rear->next=s->next;
            delete s;
        }
}
```

⑵ 设顺序栈 S 中有 2n 个元素，从栈顶到栈底的元素依次为 a2n，a2n-1，…，a1，要求通过一个循环队列重新排列栈中元素，使得从栈顶到栈底的元素依次为 a2n，a2n-2，…，a2，a2n-1，a2n-3，…，a1，请设计算法实现该操作，要求空间复杂度和时间复杂度均为 O(n)。

【解答】操作步骤为：

① 将所有元素出栈并入队；

② 依次将队列元素出队，如果是偶数结点，则再入队，如果是奇数结点，则入栈；

③ 将奇数结点出栈并入队；

④ 将偶数结点出队并入栈；
⑤ 将所有元素出栈并入队；
⑥ 将所有元素出队并入栈即为所求。

⑶ 用顺序存储结构存储串 S，编写算法删除 S 中第 i 个字符开始的连续 j 个字符。
【解答】先判断串 S 中要删除的内容是否存在，若存在，则将第 i+j-1 之后的字符前移 j 个位置。算法如下：

串删除算法 Del

```
void Del(char S[ ], int i, int j)      //数组 0 号单元存放串的长度
{
  if (i+j<S[0]) {
                for (k=i; k<S[0]; k++)
                    S[k]=S[k+j];
                S[0]=S[0]-j;
  }
  else cout<<"参数非法";
}
```

⑷ 对于采用顺序存储结构的串 S，编写一个函数删除其值等于 ch 的所有字符。
【解答】从后向前删除值为 ch 的所有元素，这样所有移动的元素中没有值为 ch 的元素，能减少移动元素的次数，提高算法的效率。算法如下：

串删除算法 Del

```
void Del (char S[ ], char ch )   //数组的 0 号单元存放串的长度
{
    for (i=S[0]; i>0; i--)
      if (S[i]==ch) {
                    for (j=i+1; j<=S[0]; j++)
                      S[j-1]=S[j];
                    S[0]--;
      }
}
```

⑸ 对串的模式匹配 KMP 算法设计求模式滑动位置的 next 函数。
【解答】参见 3.2.5

学习自测及答案

1．在一个具有 n 个单元的顺序栈中，假定以地址低端（即下标为 0 的单元）作为栈底，以 top 作为栈顶指针，当出栈时，top 的变化为（ ）。
A 不变 B top=0; C top=top-1; D top=top+1;
【解答】C

2．一个栈的入栈序列是 a, b, c, d, e，则栈的不可能的出栈序列是（ ）。
A edcba B cdeba C debca D abcde
【解答】C

3．从栈顶指针为 top 的链栈中删除一个结点，用 x 保存被删除结点的值，则执行（ ）。
A x=top; top=top->next; B x=top->data;
C top=top->next; x=top->data; D x=top->data; top=top->next;
【解答】D

4．设元素 1, 2, 3, P, A 依次经过一个栈，进栈次序为 123PA，在栈的输出序列中，有哪些序列可作为 C++ 程序设计语言的变量名。
【解答】PA321, P3A21, P32A1, P321A, AP321

5.设 S="I_ am_ a_ teacther"，其长度为（ ）。
【解答】15

6．对于栈和队列，无论它们采用顺序存储结构还是链接存储结构，进行插入和删除操作的时间复杂度都是（ ）。

【解答】 O(1)

7．如果进栈序列为 A、B、C、D，则可能的出栈序列是什么？
答：共 14 种，分别是：ABCD，ABDC，ACBD，ACDB，ADCB，BACD，BADC，BCAD，BCDA，BDCA，CBAD，CBDA，CDBA，DCBA

8．简述队列和栈这两种数据结构的相同点和不同点。
【解答】相同点：它们都是插入和删除操作的位置受限制的线性表。
不同点：栈是限定仅在表尾进行插入和删除的线性表，是后进先出的线性表，而队列是限定在表的一端进行插入，在另一端进行删除的线性表，是先进先出的线性表。

9. 利用两个栈 S1 和 S2 模拟一个队列，如何利用栈的运算实现队列的插入和删除操作，请简述算法思想。
【解答】利用两个栈 S1 和 S2 模拟一个队列，当需要向队列中插入一个元素时，用 S1 来存放已输入的元素，即通过向栈 S1 执行入栈操作来实现；当需要从队列中删除元素时，则将 S1 中元素全部送入到 S2 中，再从 S2 中删除栈顶元素，最后再将 S2 中元素全部送入到 S1 中；判断队空的条件是：栈 S1 和 S2 同时为空。

10. 设计算法把一个十进制整数转换为二至九进制之间的任一进制数输出。
【解答】算法基于原理：N=(N div d)×d + N mod d （div 为整除运算，mod 为求余运算）。

进制转换算法 Decimaltor

```
void Decimaltor(int num, int r)
{
    top=-1;    //假设采用顺序栈
    while (num!=0)
    {
        k=num % r;
        S[++top]=k;
        num=num/r;
    }
    while (top!=-1)
        cout<<S[top--];
}
```

11．假设一个算术表达式中可以包含三种括号：圆括号“（”和“）”，方括号“[”和“]”以及花括号“{”和“}”，且这三种括号可按任意的次序嵌套使用。编写算法判断给定表达式中所含括号是否配对出现。

【解答】假设表达式已存入字符数组 A[n]中，具体算法如下：

括号匹配算法 Prool

```
void Prool(char A[ ], int n)
{
  top=-1; i=0; flag =1;
  while (i<n && flag)
  {
      if  (A[i]='('||A[i]='['||A[i]='{') S[++top]=A[i++];
      else {
            switch A[i]
            {
              case ')': if (top==-1 || S[top--]!='(') flag =0; break;
              case ']': if (top==-1 || S[top--]!='[') flag=0; break;
              case '}': if (top==-1 || S[top--]!='{') flag=0; break;
            }
      }
      i++;
  }
}
```

第 4 章 广义线性表——多维数组和广义表

课后习题讲解

1. 填空

⑴ 数组通常只有两种运算：（ ）和（ ），这决定了数组通常采用（ ）结构来实现存储。
【解答】存取，修改，顺序存储
【分析】数组是一个具有固定格式和数量的数据集合，在数组上一般不能做插入、删除元素的操作。除了初始化和销毁之外，在数组中通常只有存取和修改两种操作。

⑵ 二维数组 A 中行下标从 10 到 20，列下标从 5 到 10，按行优先存储，每个元素占 4 个存储单元，A[10][5]的存储地址是 1000，则元素 A[15][10]的存储地址是（ ）。
【解答】1140
【分析】数组 A 中每行共有 6 个元素，元素 A[15][10]的前面共存储了(15-10)×6+5 个元素，每个元素占 4 个存储单元，所以，其存储地址是 1000+140=1140。

⑶ 设有一个 10 阶的对称矩阵 A 采用压缩存储，A[0][0]为第一个元素，其存储地址为 d，每个元素占 1 个存储单元，则元素 A[8][5]的存储地址为（ ）。
【解答】d+41
【分析】元素 A[8][5]的前面共存储了(1+2+…+8)+5=41 个元素。

⑷ 稀疏矩阵一般压缩存储方法有两种，分别是（ ）和（ ）。
【解答】三元组顺序表，十字链表

⑸ 广义表((a), (((b),c)),(d))的长度是（ ），深度是（ ），表头是（ ），表尾是（ ）。
【解答】3，4，(a)，((((b),c)),(d))

⑹ 已知广义表 LS=(a，(b，c，d)，e)，用 Head 和 Tail 函数取出 LS 中原子 b 的运算是（ ）。
【解答】Head(Head(Tail(LS)))

2. 选择题

⑴ 二维数组 A 的每个元素是由 6 个字符组成的串，行下标的范围从 0~8，列下标的范围是从 0~9，则存放 A 至少需要（ ）个字节，A 的第 8 列和第 5 行共占（ ）个字节，若 A 按行优先方式存储，元素 A[8][5]的起始地址与当 A 按列优先方式存储时的（ ）元素的起始地址一致。
A 90 B 180 C 240 D 540 E 108 F 114 G 54
H A[8][5] I A[3][10] J A[5][8] K A[4][9]
【解答】D，E，K
【分析】数组 A 为 9 行 10 列，共有 90 个元素，所以，存放 A 至少需要 90×6=540 个存储单元，第 8 列和第 5 行共有 18 个元素（注意行列有一个交叉元素），所以，共占 108 个字节，元素 A[8][5]按行优先存储的起始地址为 d+8×10+5=d+85，设元素 A[i][j]按列优先存储的起始地址与之相同，则 d+j×9+i=d+85，解此方程，得 i=4，j=9。

⑵ 将数组称为随机存取结构是因为（　　）

A 数组元素是随机的 B 对数组任一元素的存取时间是相等的

C 随时可以对数组进行访问 D 数组的存储结构是不定

【解答】B

⑶ 下面的说法中，不正确的是（　　）

A 数组是一种线性结构 B 数组是一种定长的线性结构

C 除了插入与删除操作外，数组的基本操作还有存取、修改、检索和排序等

D 数组的基本操作有存取、修改、检索和排序等，没有插入与删除操

【解答】C

【分析】数组属于广义线性表，数组被创建以后，其维数和每维中的元素个数是确定的，所以，数组通常没有插入和删除操作。

⑷ 对特殊矩阵采用压缩存储的目的主要是为了（　　）

A 表达变得简单 B 对矩阵元素的存取变得简单

C 去掉矩阵中的多余元素 D 减少不必要的存储空间

【解答】D

【分析】在特殊矩阵中，有很多值相同的元素并且他们的分布有规律，没有必要为值相同的元素重复存储。

⑸ 下面（　　）不属于特殊矩阵。

A 对角矩阵 B 三角矩阵 C 稀疏矩阵 D 对称矩阵

【解答】C

⑹ 若广义表 A 满足 Head(A)=Tail(A)，则 A 为（ ）

A ( )　B (( ))　C (( ),( ))　D(( ),( ),( ))

【解答】B

⑺ 下面的说法中，不正确的是（　　）

A 广义表是一种多层次的结构 B 广义表是一种非线性结构

C 广义表是一种共享结构 D 广义表是一种递归

【解答】B

【分析】从各层元素各自具有的线性关系讲，广义表属于线性结构。

⑻ 下面的说法中，不正确的是（　　）

A 对称矩阵只须存放包括主对角线元素在内的下（或上）三角的元素即可。

B 对角矩阵只须存放非零元素即可。

C 稀疏矩阵中值为零的元素较多，因此可以采用三元组表方法存储。

D 稀疏矩阵中大量值为零的元素分布有规律，因此可以采用三元组表方法存储

【解答】D

【分析】稀疏矩阵中大量值为零的元素分布没有规律，因此采用三元组表存储。如果零元素的分布有规律，就没有必要存储非零元素的行号和列号，而需要按其压缩规律找出相应的映象函数。

3. 判断题

⑴ 数组是一种复杂的数据结构，数组元素之间的关系既不是线性的，也不是树形的。

【解答】错。例如二维数组可以看成是数据元素为线性表的线性表。

⑵ 使用三元组表存储稀疏矩阵的元素，有时并不能节省存储空间。

【解答】对。因为三元组表除了存储非零元素值外，还需要存储其行号和列号。

⑶ 稀疏矩阵压缩存储后，必会失去随机存取功能。

【解答】对。因为压缩存储后，非零元素的存储位置和行号、列号之间失去了确定的关系。

⑷ 线性表可以看成是广义表的特例，如果广义表中的每个元素都是单元素，则广义表便成为线性表。

【解答】对。

⑸ 若一个广义表的表头为空表，则此广义表亦为空表。

【解答】错。如广义表 L=(( )，(a，b))的表头为空表，但 L 不是空表。

4．一个稀疏矩阵如图 4-4 所示，写出对应的三元组顺序表和十字链表存储表示。

$$\begin{pmatrix} 0 & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & -1 & 5 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

图 4-4　稀疏矩阵

【解答】对应的三元组顺序表如图 4-5 所示，十字链表如图 4-6 所示。

下标	行号	列号	非零元素
0	1	3	2
1	2	1	3
2	3	3	-1
3	3	4	5
4（行数）			
4（列数）			
4（非零元个数）			

图 4-5　稀疏矩阵的三元组顺序表

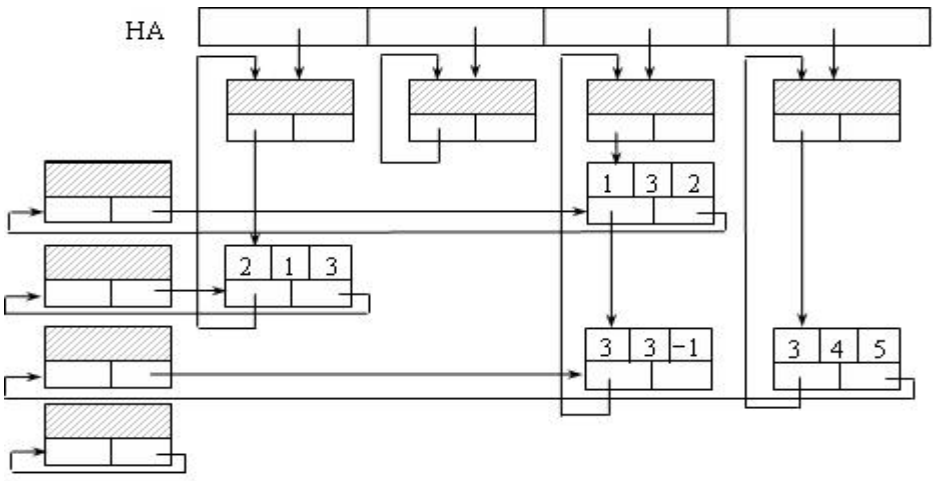


图 4-6　稀疏矩阵的十字链表

5．已知 A 为稀疏矩阵，试从空间和时间角度比较采用二维数组和三元组顺序表两种不同的存储结构完成求 运算的优缺点。

【解答】设稀疏矩阵为 m 行 n 列，如果采用二维数组存储，其空间复杂度为 O(m×n)；因为要将所有的矩

阵元素累加起来，所以，需要用一个两层的嵌套循环，其时间复杂度亦为O(m×n)。如果采用三元组顺序表进行压缩存储，假设矩阵中有t个非零元素，其空间复杂度为O(t)，将所有的矩阵元素累加起来只需将三元组顺序表扫描一遍，其时间复杂度亦为O(t)。当t << m×n时，采用三元组顺序表存储可获得较好的时、空性能。

6．设某单位职工工资表ST由“工资”、“扣除”和“实发金额”三项组成，其中工资项包括“基本工资”、“津贴”和“奖金”，扣除项包括“水”、“电”和“煤气”。

⑴ 请用广义表形式表示所描述的工资表ST，并用表头和表尾求表中的“奖金”项；

⑵ 画出该工资表ST的存储结构。

【解答】⑴ ST=((基本工资，津贴，奖金)，(水，电，煤气)，实发金额)

Head(Tail(Tail(Head(ST))))=奖金

⑵ 工资表ST的头尾表示法如图4-7所示。

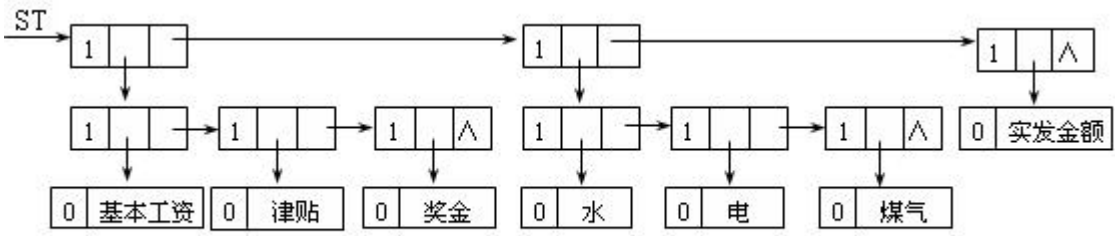


图4-7 工资表的存储示意图

7．若在矩阵A中存在一个元素ai,j（0≤i≤n-1，0≤j≤m-1），该元素是第i行元素中最小值且又是第j列元素中最大值，则称此元素为该矩阵的一个马鞍点。假设以二维数组存储矩阵A，试设计一个求该矩阵所有马鞍点的算法，并分析最坏情况下的时间复杂度。

【解答】在矩阵中逐行寻找该行中的最小值，然后对其所在的列寻找最大值，如果该列上的最大值与该行上的最小值相等，则说明该元素是鞍点，将它所在行号和列号输出。

具体算法如下：

马鞍点算法 Andian

```
void Andian(int a[ ][ ], int m, int n)
{
   for (i=0; i<n; i++)
   {
       d=a[i][0]; k=0;     //d为第i行中的最小值
       for (j=1; j<m; j++)
         if (a[i][j]<d) {
                          d=a[i][j];
                          k=j;
         }     //a[i][k]为第i行最小值
       for (j=0; j<n; j++)
         if (a[j][k]>d) break;
       if (j==n) cout<<"输出鞍点"<<i<<k<< a[i][k];
   }
}
```

分析算法，外层 for 循环共执行 n 次，内层第一个 for 循环执行 m 次，第二个 for 循环最坏情况下执行 n 次，所以，最坏情况下的时间复杂度为 O(mn+n2)。

8．已知两个 n×n 的对称矩阵按压缩存储方法存储在已维数组 A 和 B 中，编写算法计算对称矩阵的乘积。

【解答】对称矩阵采用压缩存储，乘积矩阵也采用压缩存储。注意矩阵元素的表示方法。

矩阵乘积算法 Mul

```
void Mul(int A[ ], int B[ ], int C[ ], int n)
{
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
    {
      mi=max(i, j); mj=min(i, j);
      x=mi*(mi-1)/2+mj-1;    //计算矩阵元素 C[i][j]压缩后的存储地址
      C[x]=0;
      for (k=0; k<n; k++)
      {
        u1=max(i, k); v1=min(i, k);
        u2=max(k, j); v2=min(k, j);
        w1=u1*(u1-1)/2+v1-1;     //计算 A[i][k]的存储地址
        w2=u2*(u2-1)/2+v2-1;     //计算 B[k][j]的存储地址
        c[x]=A[w1]*B[w2];
      }
    }
}
```

第 5 章 树和二叉树

课后习题讲解

1. 填空题

⑴ 树是 n（n≥0）结点的有限集合，在一棵非空树中，有（ ）个根结点，其余的结点分成 m（m＞0）个（ ）的集合，每个集合都是根结点的子树。

【解答】有且仅有一个，互不相交

⑵ 树中某结点的子树的个数称为该结点的（ ），子树的根结点称为该结点的（ ），该结点称为其子树根结点的（ ）。

【解答】度，孩子，双亲

⑶ 一棵二叉树的第 i（i≥1）层最多有（ ）个结点；一棵有 n（n>0）个结点的满二叉树共有（ ）个叶子结点和（ ）个非终端结点。

【解答】2i-1，(n+1)/2，(n-1)/2

【分析】设满二叉树中叶子结点的个数为 n0，度为 2 的结点个数为 n2，由于满二叉树中不存在度为 1 的结点，所以 n=n0+n2；由二叉树的性质 n0=n2+1，得 n0=(n+1)/2，n2=(n-1)/2。

⑷ 设高度为 h 的二叉树上只有度为 0 和度为 2 的结点，该二叉树的结点数可能达到的最大值是（ ），最小值是（ ）。

【解答】2h -1，2h-1

【分析】最小结点个数的情况是第 1 层有 1 个结点，其他层上都只有 2 个结点。

⑸ 深度为 k 的二叉树中，所含叶子的个数最多为（ ）。

【解答】2k-1

【分析】在满二叉树中叶子结点的个数达到最多。

⑹ 具有 100 个结点的完全二叉树的叶子结点数为（ ）。

【解答】50

【分析】100 个结点的完全二叉树中最后一个结点的编号为 100，其双亲即最后一个分支结点的编号为 50，也就是说，从编号 51 开始均为叶子。

⑺ 已知一棵度为 3 的树有 2 个度为 1 的结点，3 个度为 2 的结点，4 个度为 3 的结点。则该树中有（ ）个叶子结点。

【解答】12

【分析】根据二叉树性质 3 的证明过程，有 n0=n2+2n3+1（n0、n2、n3 分别为叶子结点、度为 2 的结点和度为 3 的结点的个数）。

⑻ 某二叉树的前序遍历序列是 ABCDEFG，中序遍历序列是 CBDAFGE，则其后序遍历序列是（ ）。

【解答】CDBGFEA

【分析】根据前序遍历序列和后序遍历序列将该二叉树构造出来。

⑼ 在具有 n 个结点的二叉链表中，共有（ ）个指针域，其中（ ）个指针域用于指向其左右孩子，剩下的（ ）个指针域则是空的。

【解答】2n，n-1，n+1

（10） 在有 n 个叶子的哈夫曼树中，叶子结点总数为（ ），分支结点总数为（ ）。

【解答】n，n-1

【分析】n-1 个分支结点是经过 n-1 次合并后得到的。

2. 选择题

⑴ 如果结点 A 有 3 个兄弟，B 是 A 的双亲，则结点 B 的度是（ ）。

A 1 B 2 C 3 D 4

【解答】D

⑵ 设二叉树有 n 个结点，则其深度为（ ）。

A n-1 B n C $\lfloor \log_2 n \rfloor$+1 D 不能确定

【解答】D

【分析】此题并没有指明是完全二叉树，则其深度最多是 n，最少是$\lfloor \log_2 n \rfloor$ +1。

⑶ 二叉树的前序序列和后序序列正好相反，则该二叉树一定是（ ）的二叉树。

A 空或只有一个结点 B 高度等于其结点数

C 任一结点无左孩子 D 任一结点无右孩子

【解答】B

【分析】此题注意是序列正好相反，则左斜树和右斜树均满足条件。

⑷ 线索二叉树中某结点 R 没有左孩子的充要条件是（ ）。

A R.lchild=NULL B R.ltag=0 C R.ltag=1 D R.rchild=NULL

【解答】C

【分析】线索二叉树中某结点是否有左孩子，不能通过左指针域是否为空来判断，而要判断左标志是否为 1。

⑸ 深度为 k 的完全二叉树至少有（ ）个结点，至多有（ ）个结点，具有 n 个结点的完全二叉树按层序从 1 开始编号，则编号最小的叶子的序号是（ ）。

A 2k-2+1 B 2k-1 C 2k -1 D 2k–1 -1

E 2k+1 F 2k+1 -1 G 2k -1+1 H 2k

【解答】B，C，A

【分析】深度为 k 的完全二叉树最少结点数的情况应是第 k 层上只有 1 个结点，最多的情况是满二叉树，编号最小的叶子应该是在结点数最少的情况下，叶子结点的编号。

⑹ 一个高度为 h 的满二叉树共有 n 个结点，其中有 m 个叶子结点，则有（ ）成立。

A n=h+m B h+m=2n C m=h-1 D n=2m-1

【解答】D

【分析】满二叉树中没有度为 1 的结点，所以有 m 个叶子结点，则度为 2 的结点个数为 m-1。

⑺ 任何一棵二叉树的叶子结点在前序、中序、后序遍历序列中的相对次序（ ）。

A 肯定不发生改变 B 肯定发生改变 C 不能确定 D 有时发生变化

【解答】A

【分析】三种遍历次序均是先左子树后右子树。

⑻ 如果 T' 是由有序树 T 转换而来的二叉树，那么 T 中结点的前序序列就是 T' 中结点的（ ）序列，T 中结点的后序序列就是 T' 中结点的（ ）序列。

A 前序 B 中序 C 后序 D 层序

【解答】A，B

⑼ 设森林中有 4 棵树，树中结点的个数依次为 n1、n2、n3、n4，则把森林转换成二叉树后，其根结点的右子树上有（ ）个结点，根结点的左子树上有（ ）个结点。

A n1-1 B n1 C n1+n2+n3 D n2+n3+n4

【解答】D，A

【分析】由森林转换的二叉树中，根结点即为第一棵树的根结点，根结点的左子树是由第一棵树中除了根结点以外其余结点组成的，根结点的右子树是由森林中除第一棵树外其他树转换来的。

（10）讨论树、森林和二叉树的关系，目的是为了（ ）。

A 借助二叉树上的运算方法去实现对树的一些运算

B 将树、森林按二叉树的存储方式进行存储并利用二叉树的算法解决树的有关问题

C 将树、森林转换成二叉树

D 体现一种技巧，没有什么实际意义

【解答】B

3. 判断题

⑴ 在线索二叉树中，任一结点均有指向其前趋和后继的线索。

【解答】错。某结点是否有前驱或后继的线索，取决于该结点的标志域是否为 1。

⑵ 在二叉树的前序遍历序列中，任意一个结点均处在其子女的前面。

【解答】对。由前序遍历的操作定义可知。

⑶ 二叉树是度为 2 的树。

【解答】错。二叉树和树是两种不同的树结构，例如，左斜树是一棵二叉树，但它的度为 1。

⑷ 由树转换成二叉树，其根结点的右子树总是空的。

【解答】对。因为根结点无兄弟结点。

⑸ 用一维数组存储二叉树时，总是以前序遍历存储结点。

【解答】错。二叉树的顺序存储结构是按层序存储的，一般适合存储完全二叉树。

4. 证明：对任一满二叉树，其分枝数 B=2(n0-1) 。（其中，n0 为终端结点数）

【解答】因为在满二叉树中没有度为 1 的结点，所以有：

n=n0+n2

设 B 为树中分枝数，则

n=B+1

所以

B=n0 +n2-1

再由二叉树性质：

n0=n2+1

代入上式有：

B=n0+n0-1-1=2(n0-1)

5．证明：已知一棵二叉树的前序序列和中序序列，则可唯一确定该二叉树。

【解答】证明采用归纳法。

设二叉树的前序遍历序列为 a1a2a3… an，中序遍历序列为 b1b2b3… bn。

当 n=1 时，前序遍历序列为 a1，中序遍历序列为 b1，二叉树只有一个根结点，所以，a1= b1，可以唯一确定该二叉树；

假设当 n<=k 时，前序遍历序列 a1a2a3… ak 和中序遍历序列 b1b2b3… bk 可唯一确定该二叉树，下面证明当 n=k+1 时，前序遍历序列 a1a2a3… akak+1 和中序遍历序列 b1b2b3… bk bk+1 可唯一确定一棵二叉树。

在前序遍历序列中第一个访问的一定是根结点，即二叉树的根结点是 a1，在中序遍历序列中查找值为 a1 的结点，假设为 bi，则 a1=bi 且 b1b2… bi-1 是对根结点 a1 的左子树进行中序遍历的结果，前序遍历序列 a2a3… ai 是对根结点 a1 的左子树进行前序遍历的结果，由归纳假设，前序遍历序列 a2a3… ai 和中序遍历序列 b1b2… bi-1 唯一确定了根结点的左子树，同样可证前序遍历序列 ai+1ai+2… ak+1 和中序遍历序列 bi+1bi+2… bk+1 唯一确定了根结点的右子树。

6．已知一棵度为 m 的树中有：n1 个度为 1 的结点，n2 个度为 2 的结点，……，nm 个度为 m 的结点，问该树中共有多少个叶子结点？

【解答】设该树的总结点数为 n，则

n=n0+n1+n2+……+nm

又：

n=分枝数+1=0×n0+1×n1+2×n2+……+m×nm+1

由上述两式可得：

n0= n2+2n3+……+(m-1)nm+1

7．已知二叉树的中序和后序序列分别为 CBEDAFIGH 和 CEDBIFHGA，试构造该二叉树。

【解答】二叉树的构造过程如图 5-12 所示。

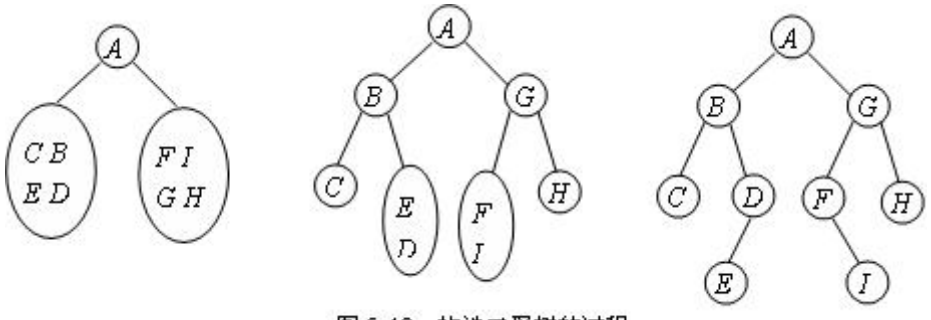


图 5-12　构造二叉树的过程

8．对给定的一组权值 W＝（5，2，9，11，8，3，7），试构造相应的哈夫曼树，并计算它的带权路径长度。

【解答】构造的哈夫曼树如图 5-13 所示。

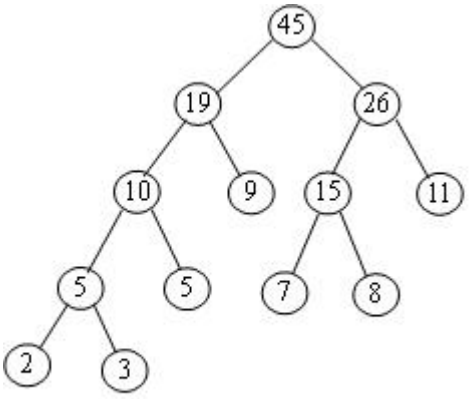


图 5-13　构造的哈夫曼树及带权路径长度

树的带权路径长度为：

WPL=2×4+3×4+5×3+7×3+8×3+9×2+11×2

=120

9．已知某字符串 S 中共有 8 种字符，各种字符分别出现 2 次、1 次、4 次、5 次、7 次、3 次、4 次和 9 次，对该字符串用[0，1]进行前缀编码，问该字符串的编码至少有多少位。

【解答】以各字符出现的次数作为叶子结点的权值构造的哈夫曼编码树如图 5-14 所示。其带权路径长度=2×5+1×5+3×4+5×3+9×2+4×3+4×3+7×2=98，所以，该字符串的编码长度至少为 98 位。

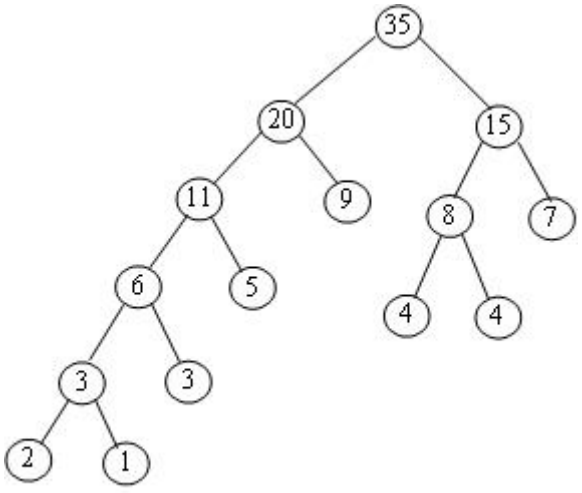


图 5-14　哈夫曼编码树

10．算法设计

⑴ 设计算法求二叉树的结点个数。

【解答】本算法不是要打印每个结点的值，而是求出结点的个数。所以可将遍历算法中的"访问"操作改为"计数操作"，将结点的数目累加到一个全局变量中，每个结点累加一次即完成了结点个数的求解。

具体算法如下：

求二叉树结点个数算法 Count

```
void Count(BiNode *root)    //n 为全局量并已初始化为 0
{
  if (root) {
          Count(root->lchild);
          n++;
          Count(root->rchild);
  }
}
```

⑵ 设计算法按前序次序打印二叉树中的叶子结点。

【解答】本算法的要求与前序遍历算法既有相同之处，又有不同之处。相同之处是打印次序均为前序，不同之处是此处不是打印每个结点的值，而是打印出其中的叶子结点，即为有条件打印。为此，将前序遍历算法中的访问操作改为条件打印即可。算法如下：

打印叶子结点算法 PreOrder

```
void PreOrder(BiNode *root)
{
  if (root) {
          if (!root->lchild && !root->rchild) cout<<root->data;
           PreOrder(root->lchild);
           PreOrder(root->rchild);
  }
}
```

⑶ 设计算法求二叉树的深度。

【解答】当二叉树为空时，深度为 0；若二叉树不为空，深度应是其左右子树深度的最大值加 1，而其左右子树深度的求解又可通过递归调用本算法来完成。具体算法如下：

求二叉树深度算法 Depth

```
int Depth(BiNode *root)
{
    if (!root) return 0;
        else {
            hl= Depth(root->lchild);
            hr= Depth(root->rchild);
            return max(hl, hr)+1;
    }
}
```

⑷ 编写算法，要求输出二叉树后序遍历序列的逆序。

【解答】要想得到后序的逆序，只要按照后序遍历相反的顺序即可，即先访问根结点，再遍历根结点的右子树，最后遍历根结点的左子树。注意和前序遍历的区别，具体算法如下：

⑸ 以二叉链表为存储结构，编写算法求二叉树中结点 x 的双亲。

【解答】对二叉链表进行遍历，在遍历的过程中查找结点 x 并记载其双亲。具体算法如下：

查找某结点的双亲算法 Parent

```
BiNode *Parent(BiNode *root, T x)   //p 是全局量，初值为空
{
  if (root) {
          if (root->data==x) return p;
            else {
                p=root;
                Parent(root->lchild, x);
                Parent(root->rchild, x);
            }
  }
}
```

⑹ 以二叉链表为存储结构，在二叉树中删除以值 x 为根结点的子树。

【解答】对二叉链表进行遍历，在遍历的过程中查找结点 x 并记载其双亲，然后将结点 x 的双亲结点中指向结点 x 的指针置空。具体算法如下：

删除结点 x 算法 Delete

```
void Delete(BiNode *root, T x)   //p 是全局量，初值为空
{
    if (root) {
            if (root->data==x)
                if (!p) root=NULL;
                   else if (p->lchild==root)  p->lchild=NULL;
                           else p->rchild=NULL;
            else {
                 p=root;
                 Delete(root->lchild, x);
                 Delete(root->rchild, x);
                }
    }
}
```

⑺ 一棵具有 n 个结点的二叉树采用顺序存储结构，编写算法对该二叉树进行前序遍历。

【解答】按照题目要求，设置一个工作栈以完成对该树的非递归算法，思路如下：

① 每访问一个结点，将此结点压栈，查看此结点是否有左子树，若有，访问左子树，重复执行该过程直到左子树为空。

② 从栈弹出一个结点，如果此结点有右子树，访问右子树执行步骤①，否则重复执行步骤②。

具体算法如下：

顺序存储的前序遍历算法 Exchange

```
template <class T>
void PreOrder(T A[ ], int n)
{
    top=-1;   //栈初始化，采用顺序栈并假定不会发生溢出
    i=1; cout<<A[i]; S[++top]=i;
    j=2*i;
    while (top!=-1)
    {
        while (j<=n)
        {
            cout<<A[j];
            S[++top]=j;
            i=j; j=2*i;
        }
        i=S[top--];
        i=2*i+1;
    }
}
```

⑻ 编写算法交换二叉树中所有结点的左右子树。

【解答】对二叉树进行后序遍历，在遍历过程中访问某结点时交换该结点的左右子树。

具体算法如下：

交换左右子树算法 Exchange

```
void Exchange(BiNode *root)
{
  if (root) {
          Exchange(root->lchild);
          Exchange(root->rchild);
          root->lchild←→root->rchild;   //交换左右子树
  }
}
```

⑼ 以孩子兄弟表示法做存储结构，求树中结点 x 的第 i 个孩子。

【解答】先在链表中进行遍历，在遍历过程中查找值等于 x 的结点，然后由此结点的最左孩子域 firstchild 找到值为 x 结点的第一个孩子，再沿右兄弟域 rightsib 找到值为 x 结点的第 i 个孩子并返回指向这个孩子的指针。

树的孩子兄弟表示法中的结点结构定义如下：

```
template
struct TNode
{
T data;
TNode *firstchild, *rightsib;
};
```

具体算法如下：

查找第 *i* 个孩子算法 Search

```
template <class T>
TNode *Search(TNode *root, T x, int i)
{
   if (root->data==x) {
                      j=1;
                      p=root->firstchild;
                      while (p!=NULL && j<i)
                      {
                         j++;
                         p=p->rightsib;
                      }
                      if (p) return p;
                        else return NULL;
   }
   Search(root->firstchild, x, i);
   Search(root->rightsib, x, i);
}
```

学习自测及答案

1．前序遍历和中序遍历结果相同的二叉树是（ ）。

A 根结点无左孩子的二叉树

B 根结点无右孩子的二叉树

C 所有结点只有左子树的二叉树

D 所有结点只有右子树的二叉树

【解答】D

2．由权值为{3, 8, 6, 2, 5}的叶子结点生成一棵哈夫曼树，其带权路径长度为（ ）。

A 24　　B 48　　C 53　　D 72

【解答】C

3．用顺序存储的方法将完全二叉树中的所有结点逐层存放在数组 A[1] ~ A[n]中，结点 A[i]若有左子树，则左子树的根结点是（ ）。

A A[2i-1]　　B A[2i+1]　　C A[i/2]　　D A[2i]

【解答】D

4．对任何一棵二叉树 T，如果其终端结点的个数为 n0，度为 2 的结点个数为 n2，则（ ）。

A n0=n2-1　　B n0=n2　　C n0=n2+1　　D 没有规律

【解答】C

5．一棵满二叉树中共有 n 个结点，其中有 m 个叶子结点，深度为 h，则（ ）。

A n=h+m　　B h+m=2n　　C m=h-1　　D n=2h-1

【解答】D

6．对于完全二叉树中的任一结点，若其右分支下的子孙的最大层次为 h，则其左分支下的子孙的最大层次为（ ）。

A h　　B h+1　　C h 或 h+1　　D 任意

【解答】C

7．假定一棵度为 3 的树中结点数为 50，则其最小高度应为 。

A 3　　B 4　　C 5　　D 6

【解答】C

8．在线索二叉树中，一个结点是叶子结点的充要条件为（ ）。

A 左线索标志为 0，右线索标志为 1 B 左线索标志为 1，右线索标志为 0

C 左、右线索标志均为 0 D 左、右线索标志均为 1

【解答】C

9．对于一棵具有 n 个结点的树，其所有结点的度之和为（ ）。

【解答】n-1

10．在顺序存储的二叉树中，编号为 i 和 j 的两个结点处在同一层的条件是（ ）。

【解答】$\lfloor \log_2 i \rfloor = \lfloor \log_2 j \rfloor$

11．现有按前序遍历二叉树的结果 ABC，问有哪几种不同的二叉树可以得到这一结果？

【解答】共有 5 种二叉树可以得到这一结果，如图 5-15 所示。

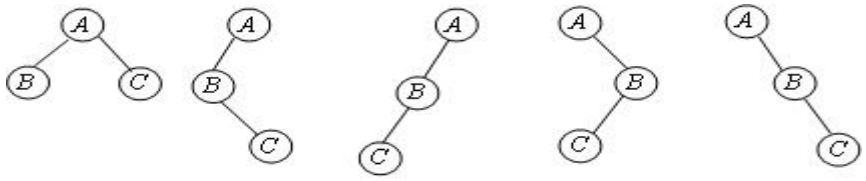


图 5-15　前序序列为 *ABC* 的二叉树

12．试找出分别满足下列条件的所有二叉树：

⑴ 前序序列和中序序列相同。

⑵ 中序序列和后序序列相同。

⑶ 前序序列和后序序列相同。

【解答】　⑴ 空二叉树、只有一个根结点的二叉树和右斜树。

⑵ 空二叉树、只有一个根结点的二叉树和左斜树。

⑶ 空二叉树、只有一个根结点的二叉树

13．将下面图 5-16 所示的树转换为二叉树，图 5-17 所示的二叉树转换为树或森林。

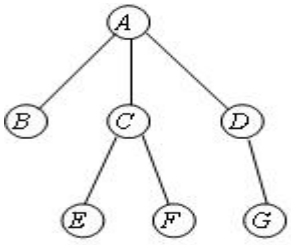


图 5-16　一棵树

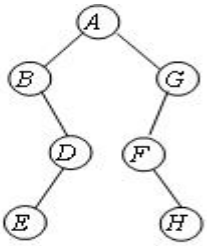


图 5-17　一棵二叉树

【解答】图 5-16 所示树转换的二叉树如图 5-18 所示，图 5-17 所示二叉树转换的森林如图 5-19 所示。

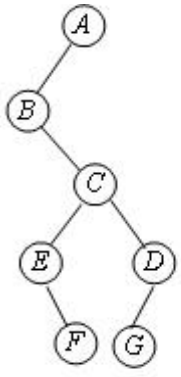


图 5-18　转换后的二叉树

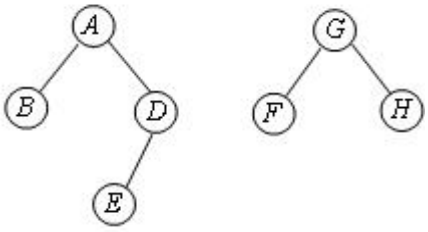


图 5-19　转换后的森林

14．以孩子兄弟表示法作为存储结构，编写算法求树的深度。

【解答】采用递归算法实现。若树为空树，则其深度为 0，否则其深度等于第一棵子树的深度+1 和兄弟子树的深度中的较大者。具体算法如下：

求树的深度算法 Depth

```
int Depth(TNode *root)
{
  if (!root) return 0;
      else {
            h1=Depth(root->firstchild);
            h2=Depth(root->rightsib);
            return max(h1+1, h2);
      }
}
```

15．设计算法，判断一棵二叉树是否为完全二叉树。

【解答】根据完全二叉树的定义可知，对完全二叉树按照从上到下、从左到右的次序（即层序）遍历应该满足：

⑴ 若某结点没有左孩子，则其一定没有右孩子；

⑵ 若某结点无右孩子，则其所有后继结点一定无孩子。

若有一结点不满足其中任意一条，则该二叉树就一定不是完全二叉树。因此可采用按层次遍历二叉树的方法依次对每个结点进行判断是否满足上述两个条件。为此，需设两个标志变量 BJ 和 CM，其中 BJ 表示已扫描过的结点是否均有左右孩子，CM 存放局部判断结果及最后的结果。

具体算法如下：

判断完全二叉树算法 ComBiTree

```
bool ComBiTree(BiNode *root)
{
  front=rear=-1;  //队列初始化，采用顺序队列并假定不会发生溢出
  BJ=1; CM=1;
  if (root) {
            Q[++rear]=root;
            while (front!=rear)
            {
                p=Q[++front];
                if (!p->lchild) {
                                BJ=0;
                                if (p->rchild) CM=0;
                }
                else {
                      CM=BJ;
                      Q[++rear]=p->lchild;
                      if (!p->rchild) BJ=0;
                          else Q[++front]=p->rchild;
                }
            }
  }
  return CM;
}
```

第 6 章 图

课后习题讲解

1. 填空题

⑴ 设无向图 G 中顶点数为 n，则图 G 至少有（ ）条边，至多有（ ）条边；若 G 为有向图，则至少有（ ）条边，至多有（ ）条边。

【解答】0，n(n-1)/2，0，n(n-1)

【分析】图的顶点集合是有穷非空的，而边集可以是空集；边数达到最多的图称为完全图，在完全图中，任意两个顶点之间都存在边。

⑵ 任何连通图的连通分量只有一个，即是（ ）。

【解答】其自身

⑶ 图的存储结构主要有两种，分别是（ ）和（ ）。

【解答】邻接矩阵，邻接表

【分析】这是最常用的两种存储结构，此外，还有十字链表、邻接多重表、边集数组等。

⑷ 已知无向图 G 的顶点数为 n，边数为 e，其邻接表表示的空间复杂度为（ ）。

【解答】O(n+e)

【分析】在无向图的邻接表中，顶点表有 n 个结点，边表有 2e 个结点，共有 n+2e 个结点，其空间复杂度为 O(n+2e)= O(n+e)。

⑸ 已知一个有向图的邻接矩阵表示，计算第 j 个顶点的入度的方法是（ ）。

【解答】求第 j 列的所有元素之和

⑹ 有向图 G 用邻接矩阵 A[n][n]存储，其第 i 行的所有元素之和等于顶点 i 的（ ）。

【解答】出度

⑺ 图的深度优先遍历类似于树的（ ）遍历，它所用到的数据结构是（ ）；图的广度优先遍历类似于树的（ ）遍历，它所用到的数据结构是（ ）。

【解答】前序，栈，层序，队列

⑻ 对于含有 n 个顶点 e 条边的连通图，利用 Prim 算法求最小生成树的时间复杂度为（ ），利用 Kruskal 算法求最小生成树的时间复杂度为（ ）。

【解答】O(n2)，O(elog2e)

【分析】Prim 算法采用邻接矩阵做存储结构，适合于求稠密图的最小生成树；Kruskal 算法采用边集数组做存储结构，适合于求稀疏图的最小生成树。

⑼ 如果一个有向图不存在（ ），则该图的全部顶点可以排列成一个拓扑序列。

【解答】回路

（10） 在一个有向图中，若存在弧、、，则在其拓扑序列中，顶点 vi, vj, vk 的相对次序为（ ）。

【解答】vi, vj, vk

【分析】对由顶点 vi, vj, vk 组成的图进行拓扑排序。

2. 选择题

⑴ 在一个无向图中，所有顶点的度数之和等于所有边数的（ ）倍。

A 1/2　　B 1　　C 2　　D 4

【解答】C

【分析】设无向图中含有 n 个顶点 e 条边，则 $\sum_{i=1}^{n} D(v_i) = 2e$ 。

⑵ n 个顶点的强连通图至少有（　　）条边，其形状是（ ）。

A n　　B n+1　　C n-1　　D n×(n-1)

E 无回路　　F 有回路　　G 环状　　H 树状

【解答】A，G

⑶ 含 n 个顶点的连通图中的任意一条简单路径，其长度不可能超过（ ）。

A 1　　B n/2　　C n-1　　D n

【解答】C

【分析】若超过 n-1，则路径中必存在重复的顶点。

⑷ 对于一个具有 n 个顶点的无向图，若采用邻接矩阵存储，则该矩阵的大小是（ ）。

A n　　B (n-1)2　　C n-1　　D n2

【解答】D

⑸ 图的生成树（　　），n 个顶点的生成树有（ ）条边。

A 唯一　　B 不唯一　　C 唯一性不能确定　　D n　　E n +1　　F n-1

【解答】C，F

⑹ 设无向图 G=(V, E)和 G' =(V', E')，如果 G' 是 G 的生成树，则下面的说法中错误的是（ ）。

A　G' 为 G 的子图　　B　G' 为 G 的连通分量

C　G' 为 G 的极小连通子图且 V = V'　　D　G' 是 G 的一个无环子图

【解答】B

【分析】连通分量是无向图的极大连通子图，其中极大的含义是将依附于连通分量中顶点的所有边都加上，所以，连通分量中可能存在回路。

⑺ 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 6　　B 7　　C 8　　D 9

【解答】D

【分析】n 个顶点的无向图中，边数 e≤n(n-1)/2，将 e=28 代入，有 n≥8，现已知无向图非连通，则 n=9。

⑻ 最小生成树指的是（ ） 。

A 由连通网所得到的边数最少的生成树　　B 由连通网所得到的顶点数相对较少的生成树

C 连通网中所有生成树中权值之和为最小的生成树　　D 连通网的极小连通子图

【解答】C

⑼ 判定一个有向图是否存在回路除了可以利用拓扑排序方法外，还可以用（ ）。

A 求关键路径的方法　　B 求最短路径的方法

C 广度优先遍历算法　　D 深度优先遍历算法

【解答】D

【分析】当有向图中无回路时，从某顶点出发进行深度优先遍历时，出栈的顺序（退出 DFSTraverse 算法）即为逆向的拓扑序列。

⑽ 下面关于工程计划的 AOE 网的叙述中，不正确的是（ ）

A 关键活动不按期完成就会影响整个工程的完成时间

B 任何一个关键活动提前完成，那么整个工程将会提前完成

C 所有的关键活动都提前完成，那么整个工程将会提前完成

D 某些关键活动若提前完成，那么整个工程将会提前完

【解答】B

【分析】AOE 网中的关键路径可能不止一条，如果某一个关键活动提前完成，还不能提前整个工程，而必须同时提高在几条关键路径上的关键活动。

3. 判断题

⑴ 一个有向图的邻接表和逆邻接表中的结点个数一定相等。

【解答】对。邻接表和逆邻接表的区别仅在于出边和入边，边表中的结点个数都等于有向图中边的个数。

⑵ 用邻接矩阵存储图，所占用的存储空间大小只与图中顶点个数有关，而与图的边数无关。

【解答】对。邻接矩阵的空间复杂度为 O(n2)，与边的个数无关。

⑶ 图 G 的生成树是该图的一个极小连通子图

【解答】错。必须包含全部顶点。

⑷ 无向图的邻接矩阵一定是对称的，有向图的邻接矩阵一定是不对称的

【解答】错。有向图的邻接矩阵不一定对称，例如有向完全图的邻接矩阵就是对称的。

⑸ 对任意一个图，从某顶点出发进行一次深度优先或广度优先遍历，可访问图的所有顶点。

【解答】错。只有连通图从某顶点出发进行一次遍历，可访问图的所有顶点。

⑹ 在一个有向图的拓扑序列中，若顶点 a 在顶点 b 之前，则图中必有一条弧。

【解答】错。只能说明从顶点 a 到顶点 b 有一条路径。

⑺ 若一个有向图的邻接矩阵中对角线以下元素均为零，则该图的拓扑序列必定存在。

【解答】对。参见第 11 题的证明。

⑻ 在 AOE 网中一定只有一条关键路径?

【解答】错。AOE 网中可能有不止一条关键路径，他们的路径长度相同。

4. n 个顶点的无向图，采用邻接表存储，回答下列问题?

⑴ 图中有多少条边？

⑵ 任意两个顶点 i 和 j 是否有边相连？
⑶ 任意一个顶点的度是多少?

【解答】
⑴ 边表中的结点个数之和除以 2。
⑵ 第 i 个边表中是否含有结点 j。
⑶ 该顶点所对应的边表中所含结点个数。

5．n 个顶点的无向图，采用邻接矩阵存储，回答下列问题：
⑴ 图中有多少条边？
⑵ 任意两个顶点 i 和 j 是否有边相连？
⑶ 任意一个顶点的度是多少？
【解答】
⑴ 邻接矩阵中非零元素个数的总和除以 2。
⑵ 当邻接矩阵 A 中 A[i][j]=1（或 A[j][i]=1）时，表示两顶点之间有边相连。
⑶ 计算邻接矩阵上该顶点对应的行上非零元素的个数。

6．证明：生成树中最长路径的起点和终点的度均为 1 。
【解答】用反证法证明。
设 v1, v2, …, vk 是生成树的一条最长路径，其中，v1 为起点，vk 为终点。若 vk 的度为 2，取 vk 的另一个邻接点 v，由于生成树中无回路，所以，v 在最长路径上，显然 v1, v2, …, vk , v 的路径最长，与假设矛盾。所以生成树中最长路径的终点的度为 1。
同理可证起点 v1 的度不能大于 1，只能为 1。

7．已知一个连通图如图 6-6 所示，试给出图的邻接矩阵和邻接表存储示意图，若从顶点 v1 出发对该图进行遍历，分别给出一个按深度优先遍历和广度优先遍历的顶点序列。

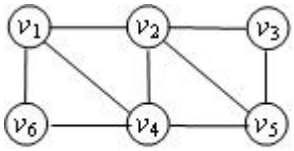
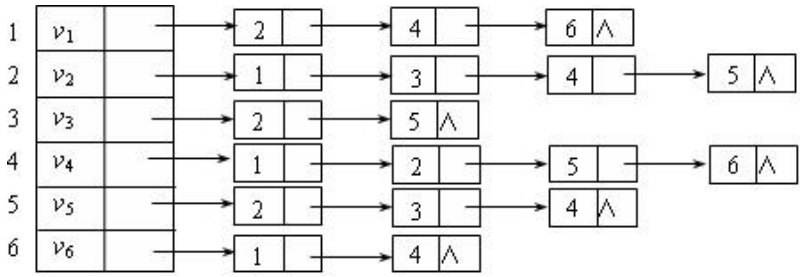


图 6-6　第 7 题图

【解答】邻接矩阵表示如下：

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

深度优先遍历序列为：v1 v2 v3 v5 v4 v6
广度优先遍历序列为：v1 v2 v4 v6 v3 v5
邻接表表示如下：



8．图 6-7 所示是一个无向带权图，请分别按 Prim 算法和 Kruskal 算法求最小生成树。

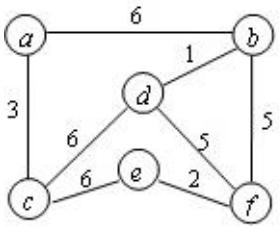
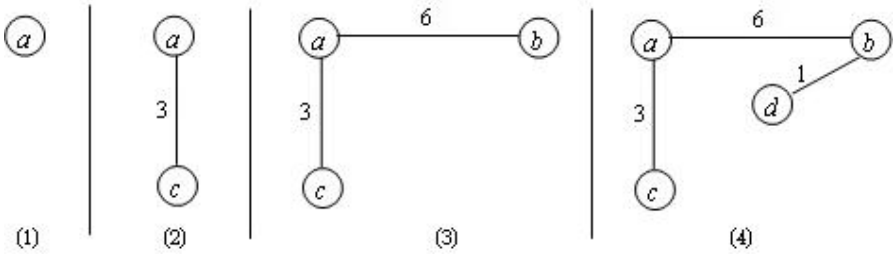
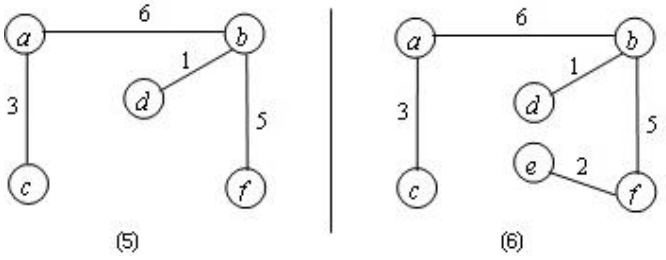


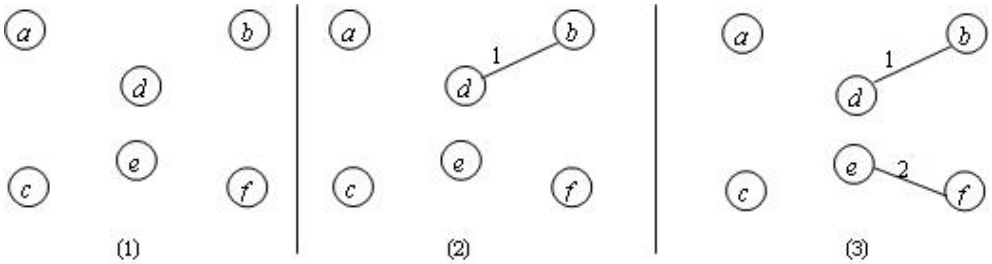
图 6-7　第 8 题图

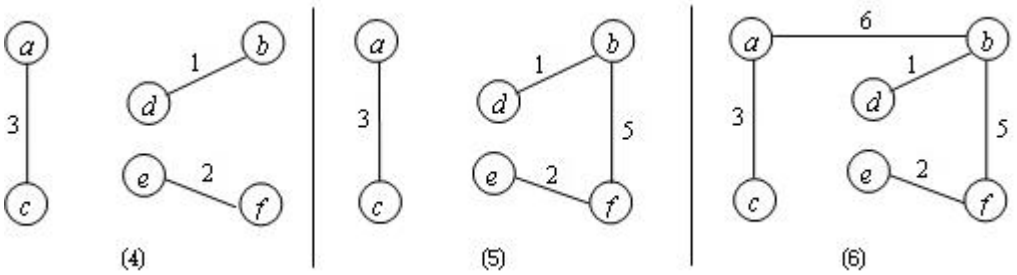
【解答】按 Prim 算法求最小生成树的过程如下：





按 Kruskal 算法求最小生成树的过程如下：





9．对于图 6-8 所示的带权有向图，求从源点 v1 到其他各顶点的最短路径。

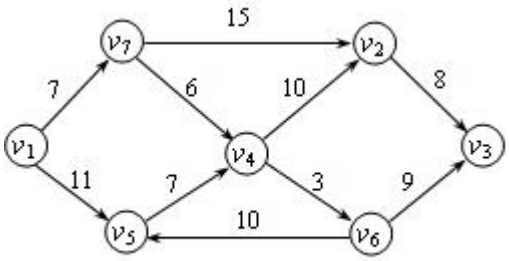


图 6-8　第 9 题图

【解答】从源点 v1 到其他各顶点的最短路径如下表所示。

源点 终点 最短路径 最短路径长度

v1 v7 v1 v7 7
v1 v5 v1 v5 11
v1 v4 v1 v7 v4 13
v1 v6 v1 v7 v4 v6 16
v1 v2 v1 v7 v2 22
v1 v3 v1 v7 v4 v6 v3 25

10．如图 6-9 所示的有向网图，利用 Dijkstra 算法求从顶点 v1 到其他各顶点的最短路径。

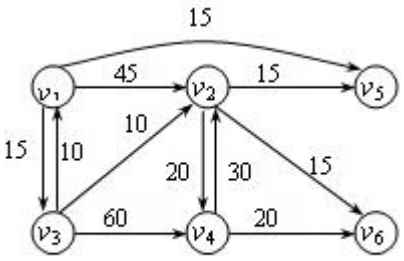


图 6-9　第 10 题图

【解答】从源点 v1 到其他各顶点的最短路径如下表所示。

源点 终点 最短路径 最短路径长度

v1 v3 v1 v3 15
v1 v5 v1 v5 15
v1 v2 v1 v3 v2 25
v1 v6 v1 v3 v2 v6 40
v1 v4 v1 v3 v2 v4 45

11．证明：只要适当地排列顶点的次序，就能使有向无环图的邻接矩阵中主对角线以下的元素全部为 0。
【解答】
任意 n 个结点的有向无环图都可以得到一个拓扑序列。设拓扑序列为 v0v1v2…vn-1，我们来证明此时的邻接矩阵 A 为上三角矩阵。证明采用反证法。
假设此时的邻接矩阵不是上三角矩阵，那么，存在下标 i 和 j（i>j），使得 A[i][j]不等于零，即图中存在从 vi 到 vj 的一条有向边。由拓扑序列的定义可知，在任意拓扑序列中，vi 的位置一定在 vj 之前，而在上述拓扑序列 v0v1v2…vn-1 中，由于 i>j，即 vi 的位置在 vj 之后，导致矛盾。因此命题正确。

12. 算法设计

⑴ 设计算法，将一个无向图的邻接矩阵转换为邻接表。
【解答】先设置一个空的邻接表，然后在邻接矩阵上查找值不为零的元素，找到后在邻接表的对应单链表中插入相应的边表结点。
邻接矩阵存储结构定义如下：

```
const int MaxSize=10;
template
struct AdjMatrix
{
    T vertex[MaxSize]; //存放图中顶点的数组
    int arc[MaxSize][MaxSize]; //存放图中边的数组
    int vertexNum, arcNum; //图的顶点数和边数
};
```

邻接表存储结构定义如下：

```
const int MaxSize=10;
struct ArcNode //定义边表结点
{
int adjvex; //邻接点域
ArcNode *next;
};

template
struct VertexNode //定义顶点表结点
{
T vertex;
ArcNode *firstedge;
};
```

```
struct AdjList
{
VertexNode adjlist[MaxSize];
int vertexNum, arcNum; //图的顶点数和边数
};
```

具体算法如下：

邻接矩阵转为邻接表算法 MatToList

```
void MatToList(AdjMatrix &A, AdjList &B)
{
    B.vertexNum= A.vertexNum;
    B.arcNum= A.arcNum;
    for (i=0; i<A.vertexNum; i++)
      B.adjlist[i].firstedge=NULL;
    for   (i=0; i<A.vertexNum; i++ )
      for   (j=0; j<i; j++)
        if   (A.arc[i][j]!=0) {
                        p=new ArcNode;
                        p->adjvex=j;
                        p->next=B.adjlist[i].firstedge;
                        B.adjlist[i].firstedge=p;
        }
}
```

⑵ 设计算法，将一个无向图的邻接表转换成邻接矩阵。

【解答】在邻接表上顺序地取每个边表中的结点，将邻接矩阵中对应单元的值置为 1。邻接矩阵和邻接表的存储结构定义与上题相同。具体算法如下：

邻接表转为邻接矩阵算法 ListToMat

```
void ListToMat(AdjMatrix &A, AdjList &B)
{
    A.vertexNum=B.vertexNum;
    A.arcNum=B.arcNum;
    for (i=0; i<A.vertexNum; i++)
        for  (j=0; j<A.vertexNum; j++)
            A.arc[i][j]=0;
     for  (i=0; i<A.vertexNum; i++)
     {
         p=B.adjlist[i].firstedge;
         while  (p)
         {
             j= p->adjvex;
             a[i][j]=1;
             p=p->next;
         }
     }
}
```

⑶ 设计算法，计算图中出度为零的顶点个数。

【解答】在有向图的邻接矩阵中，一行对应一个顶点，每行的非零元素的个数等于对应顶点的出度。因此，当某行非零元素的个数为零时，则对应顶点的出度为零。据此，从第一行开始，查找每行的非零元素个数是否为零，若是则计数器加 1。具体算法如下：

统计出度为 0 的算法 SumZero

```
int SumZero(AdjMatrix A)
{
    count=0;
    for (i=0; i<A.vertexNum; i++)
    {
        tag=0;
        for (j=0; j<A.vertexNum; j++)
            if (arcs[i][j] !=0) {
                                tag=1;
                                break;
            }
        if (tag==0) count++;
    }
    return count;
}
```

⑷ 以邻接表作存储结构，设计按深度优先遍历图的非递归算法。

【解答】参见 6.2.1。

⑸ 已知一个有向图的邻接表，编写算法建立其逆邻接表。

【解答】在有向图中，若邻接表中顶点 vi 有邻接点 vj，在逆邻接表中 vj 一定有邻接点 vi，由此得到本题算法思路：首先将逆邻接表的表头结点 firstedge 域置空，然后逐行将表头结点的邻接点进行转化。

建立逆邻接表算法 List

```
void List(AdjList A, AdjList &B)
{
    B.vertexNum= A.vertexNum;
    B.arcNum= A.arcNum;
    for (i=0; i<A.vertexNum; i++)
        B.adjlist[i].firstedge=NULL;
    for (i=0; i<A.vertexNum; i++)
    {
        p1=A.adjlist[i].firstedge;
        while (p1)
        {
            j=p1->adjvex;
            p2=new ArcNode;
            p2->adjvex=i;
            p2->next=B.adjlist[j].firstedge;
            B.adjlist[j].firstedge=p2;
            p1=p1->next;
        }
    }
}
```

⑹ 分别基于深度优先搜索和广度优先搜索编写算法，判断以邻接表存储的有向图中是否存在由顶点 vi 到顶点 vj 的路径（i≠j）。

【解答】⑴ 基于深度优先遍历：

判断路径算法 DFS

```
int DFS(int i, int j)    //visited[]数组已初始化为 0
{
  top=-1;
  visited[i]=1; stack[++top]=i; yes=0;
  while (top!=-1 || yes==0)
  {
      i=stack[top];
      p=adjlist[i].firstedge;
      while (p && yes==0)
      {
          t=p->adjvex;
          if (t==j) yes=1;
            else if (visited[t]==0) {
                                      visited[t]=1;
                                      stack[++top]=t;
                  }
                  else p=p->next;
      }
      if   (!p)   top--;
  }
  return yes;
}
```

⑵ 基于广度优先遍历：

判断路径算法 BFS

```
int   BFS(int i, int j)        //visited[]数组已初始化为 0
{
    front=-1; rear=-1;       //队列首尾指针初始化
    visited[i]=1; queue[++rear]=i; yes=0;
    while (front!=rear || yes==0)
    {
        i=queue[++front];
        p=adjlist[i].firstedge;
        while (p && yes==0)
        {
            t=p->adjvex;
            if (t==j) yes=1;
              else if (visited[t]==0) {
                                        visited[t]=1;
                                        queue[++rear]=t;
                   }
                   else p=p->next;
        }
    }
    return yes;
}
```

学习自测及答案

1．某无向图的邻接矩阵 A=$\begin{bmatrix}0&1&0\\1&0&1\\0&1&0\end{bmatrix}$，可以看出，该图共有（ ）个顶点。

A .3　　B. 6　　C. 9　　D .以上答案均不正确

【解答】A

2．无向图的邻接矩阵是一个（ ），有向图的邻接矩阵是一个（ ）

A 上三角矩阵 B 下三角矩阵 C 对称矩阵 D 无规律

【解答】C，D

3．下列命题正确的是（ ）。

A 一个图的邻接矩阵表示是唯一的，邻接表表示也唯一

B 一个图的邻接矩阵表示是唯一的，邻接表表示不唯一

C 一个图的邻接矩阵表示不唯一的，邻接表表示是唯一

D 一个图的邻接矩阵表示不唯一的，邻接表表示也不唯一

【解答】B

4．十字链表适合存储（ ），邻接多重表适合存储（ ）。

【解答】有向图，无向图

5. 在一个具有 n 个顶点的有向完全图中包含有（ ）条边：

A n(n-1)/2　　B n(n-1)　　C n(n+1)/2　　D n2

【解答】B

6．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。

【解答】2(n-1)

7．表示一个有 100 个顶点，1000 条边的有向图的邻接矩阵有（ ）个非零矩阵元素。

【解答】1000

8．一个具有 n 个顶点 k 条边的无向图是一个森林（n>k），则该森林中必有（ ）棵树。

A k　　B n　　C n - k　　D 1

【解答】C

9. 用深度优先遍历方法遍历一个有向无环图，并在深度优先遍历算法中按退栈次序打印出相应的顶点，则输出的顶点序列是（ ）。

A 逆拓扑有序　　B 拓扑有序　　C 无序　　D 深度优先遍历序列

【解答】A

10. 关键路径是 AOE 网中（ ）。

A 从源点到终点的最长路径　　B　从源点到终点的最长路径

C 最长的回路　　D　最短的回路

【解答】A

11. 已知无向图 G 的邻接表如图 6-10 所示，分别写出从顶点 1 出发的深度遍历和广度遍历序列，并画出相应的生成树。

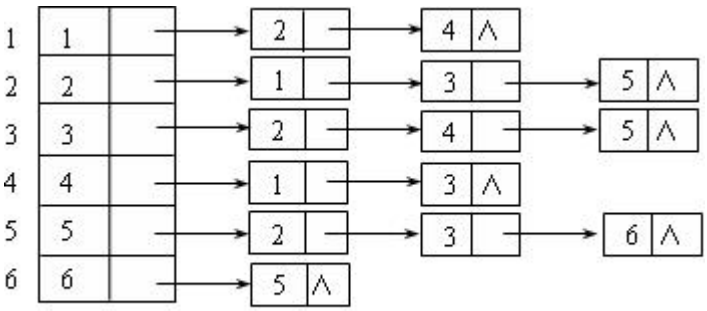
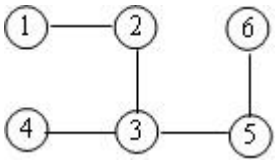


图 6-10　无向图的邻接表

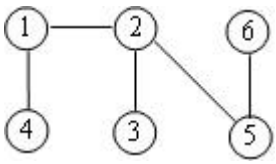
【解答】深度优先遍历序列为：1，2，3，4，5，6

对应的生成树为：



广度优先遍历序列为：1，2，4，3，5，6

对应的生成树为：



12. 已知已个 AOV 网如图 6-11 所示，写出所有拓扑序列。

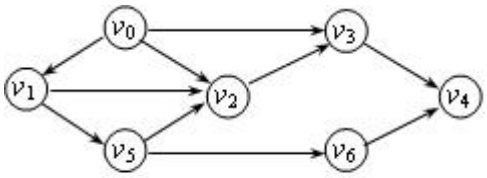


图 6-11　第 12 题图

【解答】拓扑序列为：v0 v1 v5 v2 v3 v6 v4、 v0 v1 v5 v2 v6 v3 v4、 v0 v1 v5 v6 v2 v3 v4。

第 7 章 查找技术

课后习题讲解

1. 填空题

⑴ 顺序查找技术适合于存储结构为（ ）的线性表，而折半查找技术适用于存储结构为（ ）的线性表，并且表中的元素必须是（ ）。

【解答】顺序存储和链接存储，顺序存储，按关键码有序

⑵ 设有一个已按各元素值排好序的线性表，长度为 125，用折半查找与给定值相等的元素，若查找成功，则至少需要比较（ ）次，至多需比较（ ）次。

【解答】1，7

【分析】在折半查找判定树中，查找成功的情况下，和根结点的比较次数最少，为 1 次，最多不超过判定树的深度。

⑶ 对于数列{25，30，8，5，1，27，24，10，20，21，9，28，7，13，15}，假定每个结点的查找概率相同，若用顺序存储结构组织该数列，则查找一个数的平均比较次数为（ ）。若按二叉排序树组织该数列，则查找一个数的平均比较次数为（ ）。

【解答】8，59/15

【分析】根据数列将二叉排序树画出，将二叉排序树中查找每个结点的比较次数之和除以数列中的元素个数，即为二叉排序树的平均查找长度。

⑷ 长度为 20 的有序表采用折半查找，共有（ ）个元素的查找长度为 3。

【解答】4

【分析】在折半查找判定树中，第 3 层共有 4 个结点。

⑸ 假定一个数列{25，43，62，31，48，56}，采用的散列函数为 H(k)=k mod 7，则元素 48 的同义词是（ ）。

【解答】62

【分析】H(48)= H(62)=6

⑹ 在散列技术中，处理冲突的两种主要方法是（ ）和（ ）。

【解答】开放定址法，拉链法

⑺ 在各种查找方法中，平均查找长度与结点个数无关的查找方法是（ ）。

【解答】散列查找

【分析】散列表的平均查找长度是装填因子的函数，而不是记录个数 n 的函数。

⑻ 与其他方法相比，散列查找法的特点是（ ）。

【解答】通过关键码计算记录的存储地址，并进行一定的比较

2. 选择题

⑴ 静态查找与动态查找的根本区别在于（ ）。

A 它们的逻辑结构不一样　　B 施加在其上的操作不同

C 所包含的数据元素的类型不一样　　D 存储实现不一样

【解答】B

【分析】静态查找不涉及插入和删除操作，而动态查找涉及插入和删除操作。

⑵ 有一个按元素值排好序的顺序表（长度大于 2），分别用顺序查找和折半查找与给定值相等的元素，比较次数分别是s和b，在查找成功的情况下，s和b的关系是（ ）；在查找不成功的情况下，s和b的关系是（ ）。

A s=b B s>b C s 【解答】D，D

【分析】此题没有指明是平均性能。例如，在有序表中查找最大元素，则顺序查找比折半查找快，而平均性能折半查找要优于顺序查找，查找不成功的情况也类似。

⑶ 长度为 12 的有序表采用顺序存储结构，采用折半查找技术，在等概率情况下，查找成功时的平均查找长度是（ ），查找失败时的平均查找长度是（ ）。

A 37/12 B 62/13 C 3 9/12 D 49/13

【解答】A，B

【分析】画出长度为 12 的折半查找判定树，判定树中有 12 个内结点和 13 个外结点。

⑷ 用 n 个键值构造一棵二叉排序树，其最低高度为（ ）。

A n/2 B n C log2n D log2n+1

【解答】D

【分析】二叉排序树的最低高度与完全二叉树的高度相同。

⑸ 二叉排序树中，最小值结点的（ ）。

A 左指针一定为空 B 右指针一定为空

C 左、右指针均为空 D 左、右指针均不为空

【解答】A

【分析】在二叉排序树中，值最小的结点一定是中序遍历序列中第一个被访问的结点，即二叉树的最左下结点。

⑹ 散列技术中的冲突指的是（ ）。

A 两个元素具有相同的序号 B 两个元素的键值不同，而其他属性相同

C 数据元素过多 D 不同键值的元素对应于相同的存储地址

【解答】D

⑺ 设散列表表长 m=14，散列函数 H(k)=k mod 11。表中已有 15、38、61、84 四个元素，如果用线性探侧法处理冲突，则元素 49 的存储地址是（ ）。

A 8 B 3 C 5 D 9

【解答】A

【分析】元素 15、38、61、84 分别存储在 4、5、6、7 单元，而元素 49 的散列地址为 5，发生冲突，向后探测 3 个单元，其存储地址为 8。

⑻ 在采用线性探测法处理冲突所构成的闭散列表上进行查找，可能要探测多个位置，在查找成功的情况下，所探测的这些位置的键值（ ）。

A 一定都是同义词 B 一定都不是同义词 C 不一定都是同义词 D 都相同

【解答】C

【分析】采用线性探测法处理冲突会产生堆积，即非同义词争夺同一个后继地址。

3. 判断题

⑴ 二叉排序树的充要条件是任一结点的值均大于其左孩子的值，小于其右孩子的值。

【解答】错。

分析二叉排序树的定义，是左子树上的所有结点的值都小于根结点的值，右子树上的所有结点的值都大于根结点的值。例如图 7-7 所示二叉树满足任一结点的值均大于其左孩子的值，小于其右孩子的值，但不是二叉排序树。

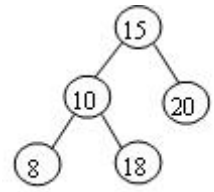


图 7-7 非二叉排序树

⑵ 二叉排序树的查找和折半查找的时间性能相同。

【解答】错。二叉排序树的查找性能在最好情况和折半查找相同。

⑶ 若二叉排序树中关键码互不相同，则其中最小元素和最大元素一定是叶子结点。

【解答】错。在二叉排序树中，最小元素所在结点一定是中序遍历序列中第一个被访问的结点，即是二叉树的最左下结点，但可能有右子树。最大元素所在结点一定是中序遍历序列中最后一个被访问的结点，即是二叉树的最右下结点，但可能有左子树。如图 7-8 所示，5 是最小元素，25 是最大元素，但 5 和 25 都不是叶子结点。

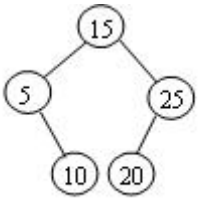


图 7-8 二叉排序树

⑷ 散列技术的查找效率主要取决于散列函数和处理冲突的方法。

【解答】错。更重要的取决于装填因子，散列表的平均查找长度是装填因子的函数。

⑸ 当装填因子小于 1 时，向散列表中存储元素时不会引起冲突。

【解答】错。装填因子越小，只能说明发生冲突的可能性越小。

4．分别画出在线性表（a，b，c，d，e，f，g）中进行折半查找关键码 e 和 g 的过程。

【解答】查找关键码 e 的过程如图 7-9 所示，查找关键码 g 的过程如图 7-10 所示。

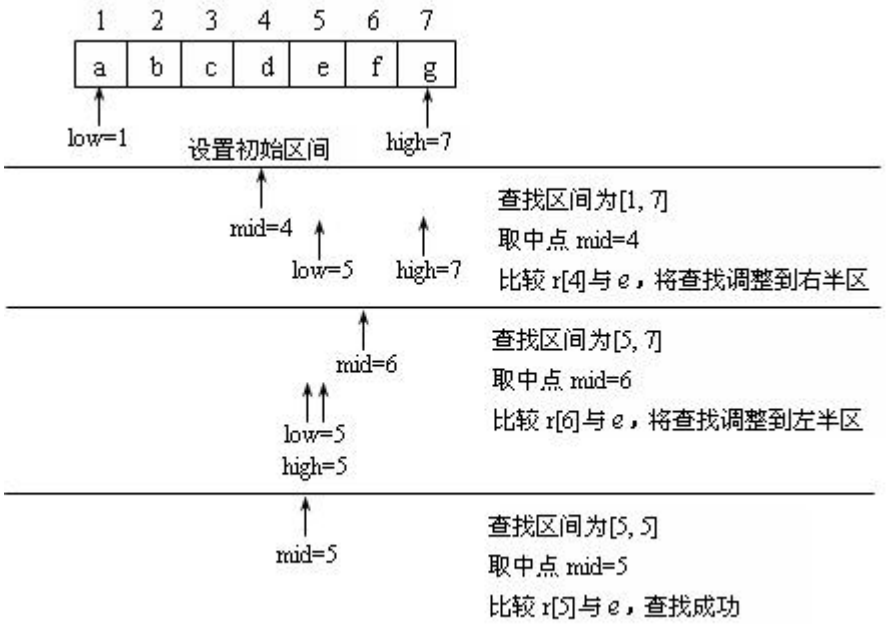


图 7-9　查找关键码 e 的过程

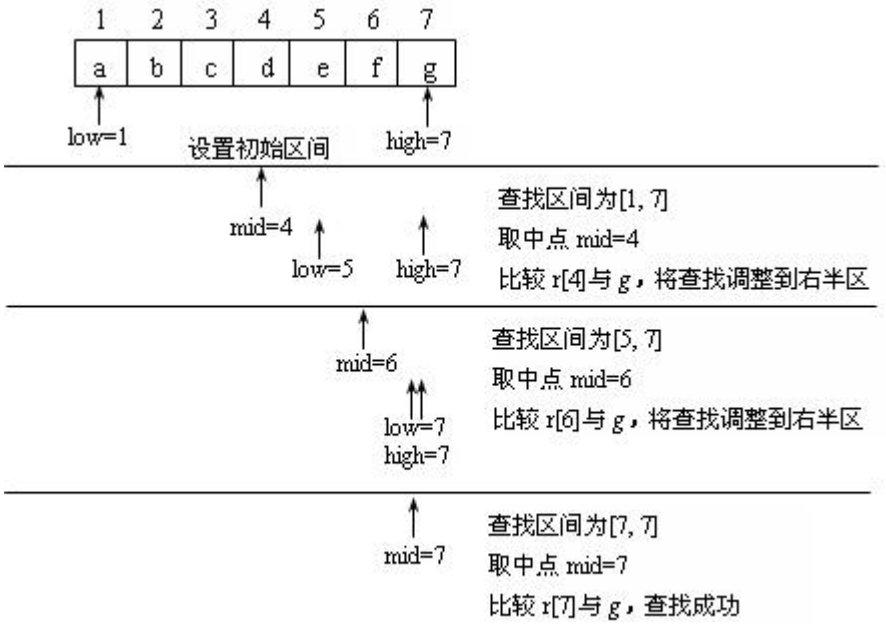


图 7-10　查找关键码 g 的过程

5．画出长度为 10 的折半查找判定树，并求等概率时查找成功和不成功的平均查找长度。

【解答】参见 7.2.1。

6．将数列（24，15，38，27，121，76，130）的各元素依次插入一棵初始为空的二叉排序树中，请画出最后的结果并求等概率情况下查找成功的平均查找长度。

【解答】二叉排序树如图 7-11 所示，其平均查找长度=1+2×2+3×2+4×2=19/7

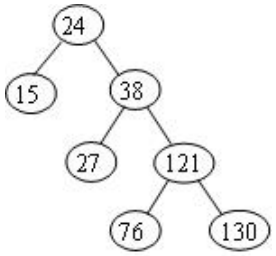


图 7-11　二叉排序树

7．一棵二叉排序树的结构如图 7-12 所示，结点的值为 1～8，请标出各结点的值。

【解答】二叉排序树中各结点的值如图 7-13 所示。

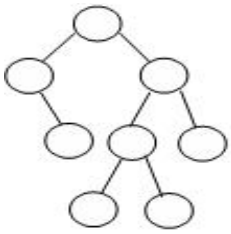


图 7-12　第 7 题图

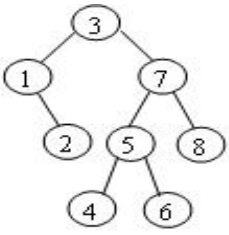


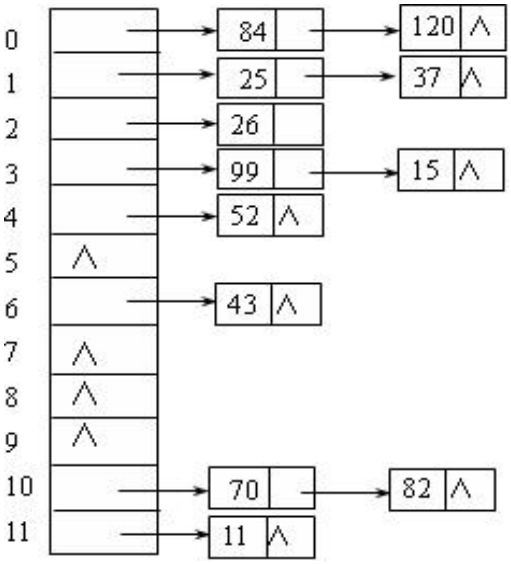
图 7-13　第 7 题答案

8．已知散列函数 H(k)=k mod 12，键值序列为(25, 37, 52, 43, 84, 99, 120, 15, 26, 11, 70, 82)，采用拉链法处理冲突，试构造开散列表，并计算查找成功的平均查找长度。

【解答】H(25)=1,　H(37)=1,　H(52)=4,　H(43)=7,　H(84)=0,　H(99)=3,

H(120)=0,　H(15)=3,　H(26)=2,　H(11)=11,　H(70)=10,　H(82)=10

构造的开散列表如下：



平均查找长度 ASL=(8×1+4×2)/12=16/12

9．算法设计

⑴ 设计顺序查找算法，将哨兵设在下标高端。

【解答】将哨兵设置在下标高端，表示从数组的低端开始查找，在查找不成功的情况下，算法自动在哨兵处终止。具体算法如下：

顺序查找算法 Search

```
int Search(int r[ ], int n, int k)
{
    i=1; r[n+1]=k;
    while (r[i]!=k)
        i++;
    return  i % (n+1);
}
```

⑵ 编写算法求给定结点在二叉排序树中所在的层数。

【解答】根据题目要求采用递归方法，从根结点开始查找结点 p，若待查结点是根结点，则深度为 1，否则到左子树（或右子树）上去找，查找深度加 1。

具体算法如下：

结点在二叉排序树的层数算法 Level

```
int Level (BiNode *root, BiNode *p)  //BiNode 请参见二查排序树的结点结构
{
   if (!p) return 0;
   if (p==root) return 1;
       else if (p->data<root->data) return Level(root->lchild, p)+1;
                else return Level (root->rchild, p)+1;
}
```

⑶ 编写算法，在二叉排序树上找出任意两个不同结点的最近公共祖先。

【解答】设两个结点分别为 A 和 B，根据题目要求分下面情况讨论：

⑴ 若 A 为根结点，则 A 为公共祖先；

⑵ 若 A->datadata 且 root->datadata，root 为公共祖先；

⑶ 若 A->datadata 且 B->datadata，则到左子树查找；

⑷ 若 A->data>root->data 且 B->data>root->data，则到右子树查找。

具体算法如下：

求公共祖先算法 Ancestor

```
BiNode *Ancestor(BiNode *A, BiNode *B, BiNode *root)
{
  if(!root) return NULL;
  else if ((A->data<root->data)&&(root->data<B->data)||(A->data==root->data))
            return root;
     else if ((A->data>root->data)&&(B->data>root->data))
               return Ancestor (A, B, root->rchild);
        else return Ancestor (A, B, root->lchild);
}
```

⑷ 设计算法判定一棵二叉树是否为二叉排序树。

【解答】对二叉排序树来讲，其中序遍历序列为一个递增序列。因此，对给定二叉树进行中序遍历，如果始终能够保证前一个值比后一个值小，则说明该二叉树是二叉排序树。

具体算法如下：

判断是否为二叉排序树算法 SortBiTree

```
int SortBiTree(BiNode *root)   //pre 记录当前结点的前驱结点值，初值为-∞
{
    if (!root) return 1;
    else {
        b1= SortBiTree (root->lchild);
        if (!b1 || pre>=root->data) return 0;
        pre=root->data;
        b2= SortBiTree(root->rchild);
        return b2;
    }
}
```

学习自测及答案

1．已知一个有序表为（12，18，24，35，47，50，62，83，90，115，134），当折半查找值为 90 的元素时，经过（ ）次比较后查找成功。

A 2　　B 3　　C 4　　D 5

【解答】A

2．已知 10 个元素（54，28，16，73，62，95，60，26，43），按照依次插入的方法生成一棵二叉排序树，查找值为 62 的结点所需比较次数为（ ）。

A 2　　B 3　　C 4　　D 5

【解答】B

3．已知数据元素为（34，76，45，18，26，54，92，65），按照依次插入结点的方法生成一棵二叉排序树，则该树的深度为（ ）。

A 4　　B 5　　C 6　　D 7

【解答】B

4．按（ ）遍历二叉排序树得到的序列是一个有序序列。

A 前序 B 中序 C 后序 D 层次

【解答】B

5．将二叉排序树 T 按前序遍历序列依次插入初始为空的二叉排序树 T '中，则 T 与 T'是相同的，这种说法是否正确？

【解答】正确

6. 一棵高度为 h 的平衡二叉树，最少含有 个结点。

A 2h B 2 h -1 C 2 h +1 D 2 h -1

【解答】D

7．在散列函数 H(k)= k mod m 中，一般来讲，m 应取（ ）。

A 奇数 B 偶数 C 素数 D 充分大的数

【解答】C

8．已知关键码序列为（Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec），散列表的地址空间为 0~16，设散列函数为 H(x)=$\lfloor i/2 \rfloor$，其中 i 为关键码中第一个字母在字母表中的序号，采用线性探测法和链地址法处理冲突，试分别构造散列表，并求等概率情况下查找成功的平均查找长度。

【解答】H(Jan)=10/2=5, H(Feb)=6/2=3, H(Mar)=13/2=6, H(Apr)=1/2=0

H(May)=13/2=6, H(Jun)=10/25, H(Jul)=10/25, H(Aug)=1/2=0

H(Sep)=19/2=8, H(Oct) =15/2=7, H(Nov) =14/2=7, H(Dec) =4/2=2

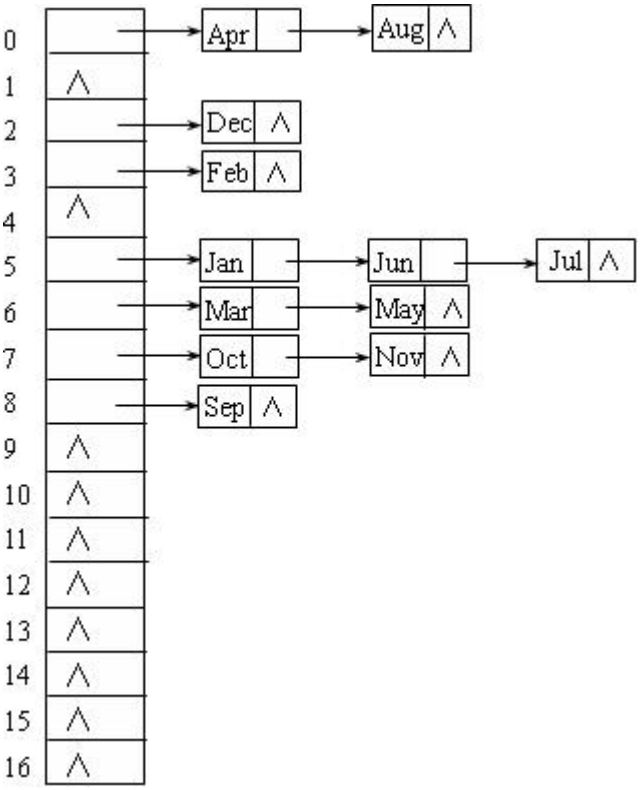
采用线性探测法处理冲突，得到的闭散列表如下：

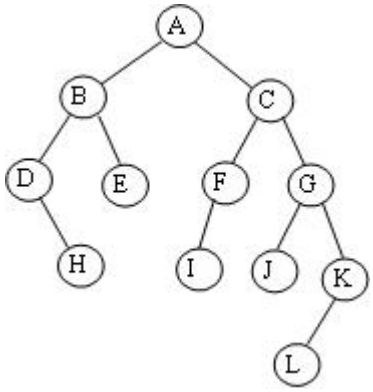
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Apr	Aug	Dec	Feb		Jan	Mar	May	Jun	Jul	Sep	Oct	Nov				

平均查找长度=(1+1+1+1+2+4+5+2+3+5+6+1)/12=32/12

采用链地址法处理冲突，得到的开散列表如下：

平均查找长度=(1×7+2×4+3×1)/12=18/12





第九题

9. 试推导含有 12 个结点的平衡二叉树的最大深度，并画出以棵这样的树。

【解答】令 Fk 表示含有最少结点的深度为 k 的平衡二叉树的结点树目，则：

F1=1，F2=2，…，Fn= Fn-2+Fn-1+1。含有 12 个结点的平衡二叉树的最大深度为 5，例如：

第 8 章 排序技术

课后习题讲解

1. 填空题

⑴ 排序的主要目的是为了以后对已排序的数据元素进行（ ）。
【解答】查找
【分析】对已排序的记录序列进行查找通常能提高查找效率。

⑵ 对 n 个元素进行起泡排序，在（ ）情况下比较的次数最少，其比较次数为（ ）。在（ ）情况下比较次数最多，其比较次数为（ ）。
【解答】正序，n-1，反序，n(n-1)/2

⑶ 对一组记录（54, 38, 96, 23, 15, 72, 60, 45, 83）进行直接插入排序，当把第 7 个记录 60 插入到有序表时，为寻找插入位置需比较（ ）次。
【解答】3
【分析】当把第 7 个记录 60 插入到有序表时，该有序表中有 2 个记录大于 60。

⑷ 对一组记录（54, 38, 96, 23, 15, 72, 60, 45, 83）进行快速排序，在递归调用中使用的栈所能达到的最大深度为（ ）。
【解答】3

⑸ 对 n 个待排序记录序列进行快速排序，所需要的最好时间是（ ），最坏时间是（ ）。
【解答】O(nlog2n)，O(n2)

⑹ 利用简单选择排序对 n 个记录进行排序，最坏情况下，记录交换的次数为（ ）。
【解答】n-1

⑺ 如果要将序列（50，16，23，68，94，70，73）建成堆，只需把 16 与（ ）交换。
【解答】50

⑻ 对于键值序列（12，13，11，18，60，15，7，18，25，100），用筛选法建堆，必须从键值为（ ）的结点开始。
【解答】60
【分析】60 是该键值序列对应的完全二叉树中最后一个分支结点。

2. 选择题

⑴ 下述排序方法中，比较次数与待排序记录的初始状态无关的是（ ）。
A 插入排序和快速排序 B 归并排序和快速排序
C 选择排序和归并排序 D 插入排序和归并排序
【解答】C
【分析】选择排序在最好、最坏、平均情况下的时间性能均为 O(n2)，归并排序在最好、最坏、平均情况下的时间性能均为 O(nlog2n)。

⑵ 下列序列中，（ ）是执行第一趟快速排序的结果。
A [da，ax，eb，de，bb] ff [ha，gc] B [cd，eb，ax，da] ff [ha，gc，bb]

C [gc，ax，eb，cd，bb] ff [da，ha] D [ax，bb，cd，da] ff [eb，gc，ha]

【解答】A

【分析】此题需要按字典序比较，前半区间中的所有元素都应小于 ff，后半区间中的所有元素都应大于 ff。

⑶ 对初始状态为递增有序的序列进行排序，最省时间的是（ ），最费时间的是（ ）。已知待排序序列中每个元素距其最终位置不远，则采用（ ）方法最节省时间。

A 堆排序 B 插入排序 C 快速排序 D 直接选择排序

【解答】B，C，B

【分析】待排序序列中每个元素距其最终位置不远意味着该序列基本有序。

⑷ 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 判定树

【解答】C

【分析】从逻辑结构的角度来看，堆实际上是一种完全二叉树的结构。

⑸ 当待排序序列基本有序或个数较小的情况下，最佳的内部排序方法是（ ），就平均时间而言，（ ）最佳。

A 直接插入排序 B 起泡排序 C 简单选择排序 D 快速排序

【解答】A，D

⑹ 设有 5000 个元素，希望用最快的速度挑选出前 10 个最大的，采用（ ）方法最好。

A 快速排序 B 堆排序 C 希尔排序 D 归并排序

【解答】B

【分析】堆排序不必将整个序列排序即可确定前若干个最大（或最小）元素。

⑺ 设要将序列（Q，H，C，Y，P，A，M，S，R，D，F，X）中的关键码按升序排列，则（ ）是起泡排序一趟扫描的结果，（ ）是增量为 4 的希尔排序一趟扫描的结果，（ ）二路归并排序一趟扫描的结果，（ ）是以第一个元素为轴值的快速排序一趟扫描的结果，（ ）是堆排序初始建堆的结果。

A （F，H，C，D，P，A，M，Q，R，S，Y，X）

B （P，A，C，S，Q，D，F，X，R，H，M，Y）

C （A，D，C，R，F，Q，M，S，Y，P，H，X）

D （H，C，Q，P，A，M，S，R，D，F，X，Y）

E （H，Q，C，Y，A，P，M，S，D，R，F，X）

【解答】D，B，E，A，C

【分析】此题需要按字典序比较，并且需要掌握各种排序方法的执行过程。

⑻ 排序的方法有很多种，（ ）法从未排序序列中依次取出元素，与已排序序列中的元素作比较，将其放入已排序序列的正确位置上。（ ）法从未排序序列中挑选元素，并将其依次放入已排序序列的一端。交换排序是对序列中元素进行一系列比较，当被比较的两元素为逆序时，进行交换；（ ）和（ ）是基于这类方法的两种排序方法，而（ ）是比（ ）效率更高的方法；（ ）法是基于选择排序的一种方法，是完全二叉树结构的一个重要应用。

A 选择排序 B 快速排序 C 插入排序

D 起泡排序 E 归并排序 F 堆排序

【解答】C，A，D，B，B，D，F

⑼ 快速排序在（ ）情况下最不利于发挥其长处。

A 待排序的数据量太大 B 待排序的数据中含有多个相同值

C 待排序的数据已基本有序 D 待排序的数据数量为奇数

【解答】C

【分析】快速排序等改进的排序方法均适用于待排序数据量较大的情况，各种排序方法对待排序的数据中是否含有多个相同值，待排序的数据数量为奇数或偶数都没有影响。

⑽ （ ）方法是从未排序序列中挑选元素，并将其放入已排序序列的一端。

A 归并排序 B 插入排序 C 快速排序 D 选择排序

【解答】D

3. 判断题

⑴ 如果某种排序算法是不稳定的，则该排序方法没有实际应用价值。

【解答】错。一种排序算法适合于某种特定的数据环境，有时对排序的稳定性没有要求。

⑵ 当待排序的元素很大时，为了交换元素的位置，移动元素要占用较多的时间，这是影响时间复杂性的主要因素。

【解答】对。此时着重考虑元素的移动次数。

⑶ 对 n 个记录的集合进行快速排序，所需要的附加空间是 O(n)。

【解答】错。最坏情况下是 O(n)。

⑷ 堆排序所需的时间与待排序的记录个数无关。

【解答】错。堆排序最好、最坏及平均时间均为 O(nlog2n)，是待排序的记录个数 n 的函数。一般来说，待排序的记录个数越多，排序所消耗的时间也就越多。

⑸ 设有键值序列（k1, k2, …, kn），当 i>n/2 时，任何一个子序列（ki, ki+1,… , kn）一定是堆。

【解答】对。当 i>n/2 时，ki, ki+1,… , kn 均是叶子结点，所以一定是堆。

4. 已知数据序列为(12，5，9，20，6，31，24)，对该数据序列进行排序，写出插入排序、起泡排序、快速排序、简单选择排序、堆排序以及二路归并排序每趟的结果。

【解答】用上述排序方法的每趟结果如下：

插入排序：

初始键值序列	[12]	5	9	20	6	31	24
第一趟结果	[5	12]	9	20	8	31	24
第二趟结果	[5	9	12]	20	6	31	24
第三趟结果	[5	9	12	20]	6	31	24
第四趟结果	[5	6	9	12	20]	31	24
第五趟结果	[5	6	9	12	20	31]	24
第六趟结果	[5	6	9	12	20	24	31]

简单选择排序：

初始键值序列	[12	5	9	20	6	31	24]
第一趟结果	5	[12	9	20	8	31	24]
第二趟结果	5	8	[9	20	12	31	24]
第三趟结果	5	8	9	[20	12	31	24]
第四趟结果	5	8	9	12	[20	31	24]
第五趟结果	5	8	9	12	20	[31	24]
第六趟结果	5	8	9	12	20	24	31

快速排序：

初始键值序列	12	5	9	20	6	31	24
第一趟结果	[6	5	9]	12	[20	31	24]
第二趟结果	[5]	6	[9]	12	20	[31	24]
第三趟结果	5	6	9	12	20	[24]	31
第四趟结果	5	6	9	12	20	24	31

起泡排序：

初始键值序列	[12	5	9	20	6	31	24]
第一趟结果	[5	9	12	6	20	24]	31
第二趟结果	[5	9	6]	12	20	24	31
第三趟结果	[5	6]	9	12	20	24	31
第四趟结果	5	6	9	12	20	24	31

堆排序：

初始键值序列	[12	5	9	20	6	31	24]
初始建堆结果	[31	20	24	5	6	9	12]
第一趟结果	[24	20	12	5	6	9]	31
第二趟结果	[20	9	12	5	6]	24	31
第三趟结果	[12	9	6	5]	20	24	31
第四趟结果	[9	5	6]	12	20	24	31
第五趟结果	[6	5]	9	12	20	24	31
第六趟结果	5	6	9	12	20	24	31

二路归并排序：

初始键值序列	12	5	9	20	6	31	24
第一趟结果	[5	12]	[9	20]	[6	31]	[24]
第二趟结果	[5	9	12	20]	[6	24	31]
第三趟结果	[5	6	9	12	20	24	31]

5．对 n=7，给出快速排序一个最好情况和最坏情况的初始排列的实例。

【解答】最好情况：4，7，5，6，3，1，2

最坏情况：7，6，5，4，3，2，1

6．判别下列序列是否为堆，如不是，按照堆排序思想把它调整为堆，用图表示建堆的过程。

⑴（1，5，7，25，21，8，8，42）　　⑵（3，9，5，8，4，17，21，6）

【解答】序列⑴是堆，序列⑵不是堆，调整为堆（假设为大根堆）的过程如图 8-5 所示。

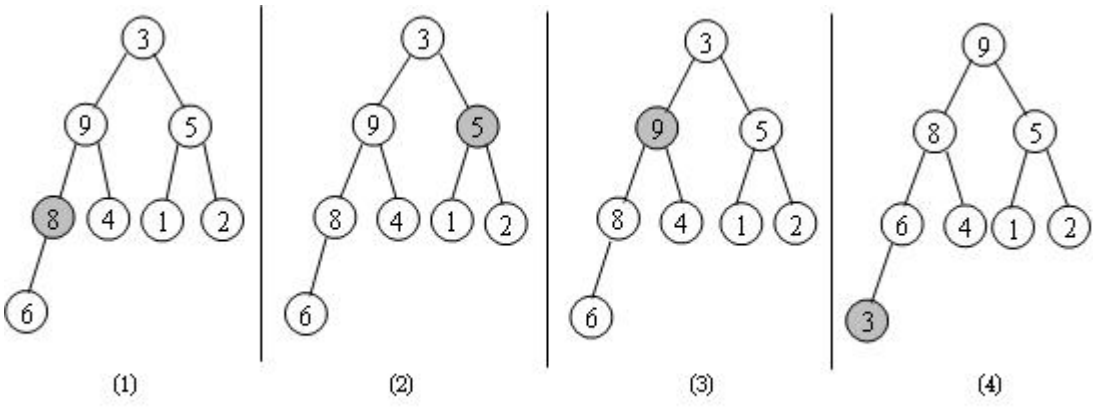


图 8-5　堆调整的过程

7．已知下列各种初始状态（长度为 n）的元素，试问当利用直接插入排序进行排序时，至少需要进行多少次比较（要求排序后的记录由小到大顺序排列）？

⑴ 关键码从小到大有序（key1< key2< … < keyn）。

⑵ 关键码从大到小有序（key1> key2> … > keyn）。

⑶ 奇数关键码顺序有序，偶数关键码顺序有序（key1< key3<…，key2 key4…）。br />⑷ 前半部分元素按关键码顺序有序，后半部分元素按关键码顺序有序，即：

（key1< key2< … < keym，keym+1< keym+2< …

【解答】依题意，最好情况下的比较次数即为最少比较次数。

⑴ 插入第 i（2≤i≤n）个元素的比较次数为 1，因此总的比较次数为： 1+1+……+1=n-1

⑵ 插入第 i（2≤i≤n）个元素的比较次数为 i，因此总的比较次数为： 2+3+……+n=(n-1)(n+2)/2

⑶ 比较次数最少的情况是所有记录关键码按升序排列，总的比较次数为： n-1

⑷ 在后半部分元素的关键码均大于前半部分元素的关键码时需要的比较次数最少，总的比较次数为： n-1

8．算法设计

⑴ 直接插入排序中寻找插入位置的操作可以通过折半查找来实现。据此写一个改进的插入排序的算法。

【解答】插入排序的基本思想是：每趟从无序区中取出一个元素，再按键值大小插入到有序区中。对于有序区，当然可以采用折半查找来确定插入位置。具体算法如下：

mosimage}

⑵ 设待排序的记录序列用单链表作存储结构，试写出直接插入排序算法。

【解答】本算法采用的存储结构是带头结点的单链表。首先找到元素的插入位置，然后把元素从链表中原位置删除，再插入到相应的位置处。具体算法如下：

插入排序算法 StraightSort

```
void  StraightSort(int r[ ], int n)
{
   for (i=2; i<=n; i++)
   {
       r[0]=r[i];
       low=1; high=i-1; flag=1;
       while (low<=high && flag)
       {
            mid=(low+high)/2;
            if (r[0]<r[mid]) high=mid-1;
                else if (r[0]>r[mid]) low=mid+1;
                          else flag=0;
       }
       for (j=i-1; j>=mid; j--)
           r[j+1]=r[j];
       r[mid]=r[0];
   }
}
```

⑶ 设待排序的记录序列用单链表作存储结构，试写出简单选择排序算法。

【解答】参见 8.2.2。

⑷ 对给定的序号 j（1＜j＜n），要求在无序记录 A[1]~A[n]中找到按关键码从小到大排在第 j 位上的记录，试利用快速排序的划分思想设计算法实现上述查找。

【解答】本算法不要求将整个记录进行排序，而只进行查找第 j 个记录。

插入排序算法 StraightSort

```
void  StraightSort(Node *first)   //Node 请参见第二章单链表的结点结构
{
  pre=first; p=first->next; q=p->next;
  while (p)
  {
     while (q!=p)
     {
        while (p->data<q->data)
        {
            pre=p;
            p=p->next;
        }
        if (p!=q) {
                  u=q->next; pre->next=q;
                  q->next=p; q=u;
        }
        else q=q->next;
     }
     pre=first; p=first->next;
  }
}
```

⑸ 写出快速排序的非递归调用算法。

【解答】先调用划分函数 Quickpass（划分函数同教材），以确定中间位置，然后再借助栈分别对中间元

素的左、右两边的区域进行快速排序。

查找第 j 小算法 Search

```
int   Search(int r[ ], int n, int j)
{
     s=1; t=n;
     k=Devo(r, s, t);
     while (k!=j)
          if (k<j)   k=Devo(r, k+1, t);
             else   k=Devo(r, s, k-1);
     return r[j];
}
int Devo(int r[ ], int low, int high)
{
    i=low; j=high; x=r[low];
    while (i<j)
    {
        while   (r[j]>=x && i<j)   j--;
        if (i<j)   {r[i]=r[j]; i++; }
        while   (r[i]<x && i<j)   i++;
        if (i<j)   {r[j]=r[i]; j--;}
    }
    r[i]=x;
    return i;
}
```

⑹ 一个线性表中的元素为正整数或负整数。设计算法将正整数和负整数分开，使线性表的前一半为负整数，后一半为正整数。不要求对这些元素排序，但要求尽量减少比较次数。

【解答】本题的基本思想是：先设置好上、下界和轴值，然后分别从线性表两端查找正数和负数，找到后进行交换，直到上下界相遇。算法如下：

快速排序非递归算法 Quicksort

```
void Quicksort(int r[ ], int n)
{
   top=-1;   //采用顺序栈并假定不会发生溢出
   low=1; high=n;
   while (low<high||top!=-1)
   {
       while (low<high)
       {
           i=Quickpass(r, low, high);
           S[++top]=i;   high=i-1;
       }
       if (top!=-1) {
                    i=S[--top];
                    low=i+1;
       }
   }
}
```

⑺ 已知（k1, k2, …, kn）是堆，试写一算法将（k1, k2, …, kn, kn+1）调整为堆。

【解答】增加一个元素应从叶子向根方向调整，假设调整为小根堆。

正负整数分开算法 Devot

```
void Devot(int r[ ], int n)
{
  i=1; j=n;
  while (i<j)
  {
     while (r[j]>0 && i<j) j--;
     while (r[i]<0 && i<j) i++;
     if (i<j) {
              r[i]←→r[j];     //交换元素
              i++;
              j--;
     }
  }
}
```

⑻ 给定 n 个记录的有序序列 A[n]和 m 个记录的有序序列 B[m]，将它们归并为一个有序序列，存放在 C[m+n]中，试写出这一算法。

【解答】采用二路归并排序中一次归并的思想，设三个参数 i、j 和 k 分别指向两个待归并的有序序列和最终有序序列的当前记录，初始时 i、j 分别指向两个有序序列的第一个记录，即 i=1，j=1，k 指向存放归并结果的位置，即 k=1。然后，比较 i 和 j 所指记录的关键码，取出较小者作为归并结果存入 k 所指位置，直至两个有序序列之一的所有记录都取完，再将另一个有序序列的剩余记录顺序送到归并后的有序序列中。

插入法建堆算法 InsertHeap

```
void InsertHeap(int r[ ], int k)    //r[1]~r[k]为堆，将 r[1]~r[k+1]调整为堆
{
   x=r[k+1];
   i=k+1;
   while (i/2>0 && r[i/2]>x)
   {
      r[i]=r[i/2];
      i=i/2;
   }
   r[i]=x;
}
```

学习自测及答案

1. 评价基于比较的排序算法的时间性能，主要标准是（ ）和（ ）。

【解答】关键码的比较次数，记录的移动次数

2. 对 n 个记录组成的任意序列进行简单选择排序，所需进行的关键码间的比较次数总共为（ ）。

【解答】比较次数=(n-1)+(n-2)+ … +2+1=n×(n-1)/2

3. 对于一个堆，按二叉树的层序遍历可以得到一个有序序列，这种说法是否正确？

【解答】错误。堆的定义只规定了结点与其左右孩子结点之间的大小关系，而同一层上的结点之间并无明确的大小关系。

4. 一组记录的关键码为{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 {40, 38, 46, 56, 79, 84} B {40, 38, 46, 79, 56, 84}

C {40, 38, 46, 84, 56, 79}　D {84, 79, 56, 46, 40, 38}

【解答】A

5. 排序趟数与序列的原始状态有关的排序方法是（ ）。

A 直接插入排序 B 简单选择排序 C 快速排序 D 归并排序

【解答】C

6. 用直接插入排序对下面四个序列进行由小到大排序，元素比较次数最少的是（ ）。

A 94, 32, 40, 90, 80, 46, 21, 69 B 21, 32, 46, 40, 80, 69, 90, 94

C 32, 40, 21, 46, 69, 94, 90, 80 D 90, 69, 80, 46, 21, 32, 94, 40

【解答】B

7. 对数列（25, 84, 21, 47, 15, 27, 68, 35, 20）进行排序，元素序列的变化情况如下：

⑴ 25, 84, 21, 47, 15, 27, 68, 35, 20 ⑵ 20, 15, 21, 25, 47, 27, 68, 35, 84

⑶ 15, 20, 21, 25, 35, 27, 47, 68, 84 ⑷ 15, 20, 21, 25, 27, 35, 47, 68, 84

则采用的排序方法是（ ）。

A 希尔排序 B 简单选择排序 C 快速排序 D 归并排序

【解答】C

8. 如果只想得到一个序列中第 k 个最小元素之前的部分排序序列，最好采用什么排序方法？为什么？对于序列{57, 40, 38, 11, 13, 34, 48, 75, 25, 6, 19, 9, 7}，得到其第 4 个最小元素之前的部分序列{6,7,9,11}，使用所选择的排序算法时，要执行多少次比较？

【解答】采用堆排序最合适，依题意可知只需取得第 k 个最小元素之前的排序序列时，堆排序的时间复杂度 O(n+klog2n)，若 k≤nlog2n，则得到的时间复杂性是 O(n)。

对于上述序列得到其前 4 个最小元素，使用堆排序实现时，执行的比较次数如下：

初始建堆：比较 20 次，得到 6；

第一次调整：比较 5 次，得到 7；

第二次调整：比较 4 次，得到 9；

第三次调整：比较 5 次，得到 11。

9. 荷兰国旗问题。要求重新排列一个由字符 R, W, B（R 代表红色，W 代表白色，B 代表兰色，这都是荷兰国旗的颜色）构成的数组，使得所有的 R 都排在最前面，W 排在其次，B 排在最后。为荷兰国旗问题设计一个算法，其时间性能是 O(n)。

【解答】设立三个参数 i、j、k，其中 i 以前的元素全部为红色；j 表示当前元素；k 以后的元素全部为蓝色。这样，就可以根据 j 的颜色，把其交换到序列的前部或后部。

具体算法如下：

一次归并算法 Union

```
void Union(int A[ ], int n, int B[ ], int n, int C[ ])
{
   i=1; j=1; k=1;
   while (i<=n && j<=m)
   {
       if (A[i]<=B[j]) C[k++]=A[i++];
           else C[k++]=B[j++];
   }
   while (i<=n) C[k++]=A[i++];
   while (j<=m) C[k++]=B[j++];
}
```

10. 已知记录序列 A[1] ~ A[n]中的关键码各不相同，可按如下方法实现计数排序：另设一个数组 C[1] ~ C[n]，对每个记录 A[i]，统计序列中关键码比它小的记录个数 C[i]，则 C[i]=0 的记录必为关键码最小的记录，C[i]=1 的记录必为关键码次小的记录，依此类推，即按 C[i]值的大小对 A 中记录进行重新排列。试编写算法实现上述计数排序。

11. 对于记录序列 A[1] ~ A[n]可按如下如下方法实现奇偶交换排序：第一趟对所有的奇数 i，将 A[i]和 A[i+1]进行比较，第二趟对所有的偶数 i，将 A[i]和 A[i+1]进行比较，每次比较时若 A[i]>A[i+1]，则将二者交换，然后重复上述排序过程，直至整个数组有序。编写算法实现上述奇偶交换排序。

【解答】具体算法如下：

计数排序算法 CountSort

```
void CountSort(int A[ ], int B[ ], int n)   //对数组 A 排序，结果存于数组 B 中
{
  for (i=1; i<=n; i++)   //计数数组 C 初始化
     C[i]=0;
  for (i=1; i<=n; i++)
  {
    for (j=1; j<=n; j++)
       if (A[j]<A[i]) C[i]++;   //统计每个记录按关键码大小的次序
  }
  for (i=1; i<=n; i++)
    B[C[i]+1]=A[i];      //重排记录
}
```

第 9 章 索引技术

课后习题讲解

1. 填空题

⑴ 在索引表中，每个索引项至少包含（ ）和（ ）等信息
【解答】关键码，关键码对应的记录在存储器中的位置

⑵ 在线性索引中，（ ）称为稠密索引
【解答】若文件中的每个记录对应一个索引项

⑶ 分块有序是指将文件划分为若干块，（ ）无序，（ ）有序。
【解答】块内，块间

⑷ 在分块查找方法中，首先查找（ ），然后查找相应的（ ）。
【解答】索引表，块

⑸ 在 10 阶 B—树中根结点所包含的关键码个数最多为（ ），最少为（ ）。
【解答】9，1
【分析】m 阶的 B－树中每个结点至多有 m 棵子树，若根结点不是终端结点，则至少有两棵子树，每个结点中关键码的个数为子树的个数减 1。

⑹ 一棵 5 阶 B—树中，除根结点外，每个结点的子树树目最少为（ ），最多为（ ）。
【解答】3，5
【分析】m 阶的 B－树中每个结点至多有 m 棵子树，除根结点之外的所有非终端结点至少有?m/2? 棵子树。

⑺ 对于包含 n 个关键码的 m 阶 B—树，其最小高度是（ ），最大高度是（ ）。
【解答】[logm(n+1)]， [logm/2(n+1)/2]

⑻ 在一棵 B—树中删除关键码，若最终引起树根结点的合并，则新树比原树的高度（ ）。
【解答】减少 1 层

⑼ 在一棵高度为 h 的 B—树中，叶子结点处于第（ ）层，当向该 B—树中插入一个新关键码时，为查找插入位置需读取（ ）个结点。
【解答】h+1，h
【分析】B－树的叶子结点可以看作是外部结点（即查找失败）的结点，通常称为外结点。实际上这些结点不存在，指向这些结点的指针为空，B－树将记录插入在终端结点中。

⑽ 对于长度为 n 的线性表，若采用分块查找（假定总块数和每块长度均接近 ，用顺序查找确定所在块），则时间复杂性为（ ）。

【解答】O($\sqrt{n}$)

2. 判断题

⑴ 在索引顺序表上采用分块查找，在等概率情况下，其平均查找长度不仅与子表个数有关，而且与每一个子表中的对象个数有关。

【解答】对。分块查找的平均查找长度不仅和文件中记录的个数 n 有关，而且和每一块中的记录个数 t 有关，当 t 取 时，ASL 取最小值 +1。

⑵ B—树是一种动态索引结构，它既适用于随机查找，也适用于顺序查找。
【解答】错。B—树不能进行顺序查找。

⑶ 对于 B—树中任何一个非叶结点中的某个关键码 k 来说，比 k 大的最小关键码和比 k 小的最大关键码一定都在叶结点中。
【解答】对。

⑷ 在索引顺序表的查找中，对索引表既可以采取顺序查找，也可以采用折半查找。
【解答】对。因为索引表有序。

⑸ m 阶 B—树中每个结点的子树个数都大于或等于?m/2?。
【解答】错。m 阶的 B－树中除根结点之外的所有非终端结点至少有[m/2] 棵子树。若根结点不是终端结点，则至少有两棵子树。

⑹ m 阶 B—树中任何一个结点的左右子树的高度都相等。
【解答】对。B 树都是树高平衡的。

3．对图 9-2 所示的 3 阶 B—树，分别给出插入关键码为 2，12，16，17 和 18 之后的结果。

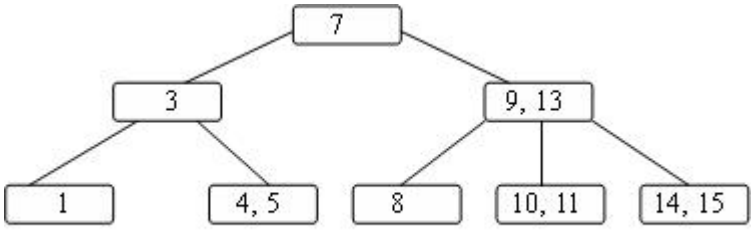
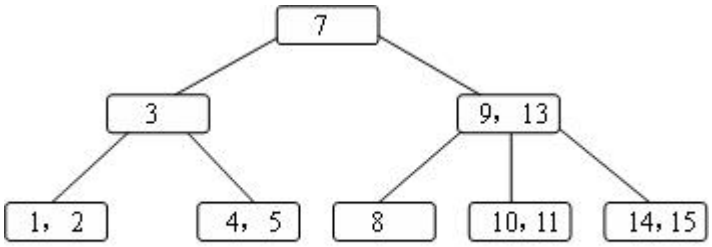
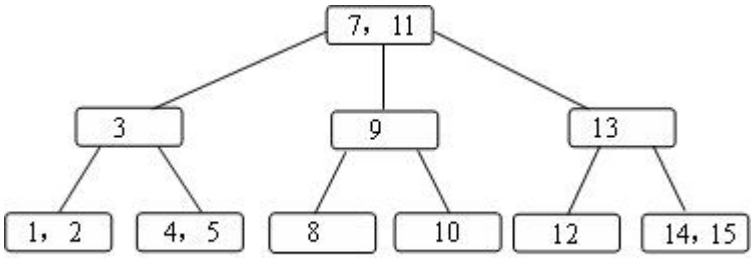


图 9-2　3 阶 B-树

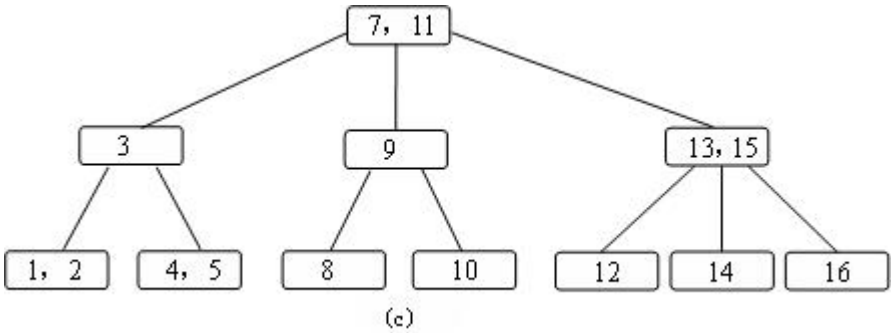
【解答】插入关键码为 2，12，16，17，18 之后的结果分别如图 9-3 中(a)、(b)、(c)、(d)、(e)所示。



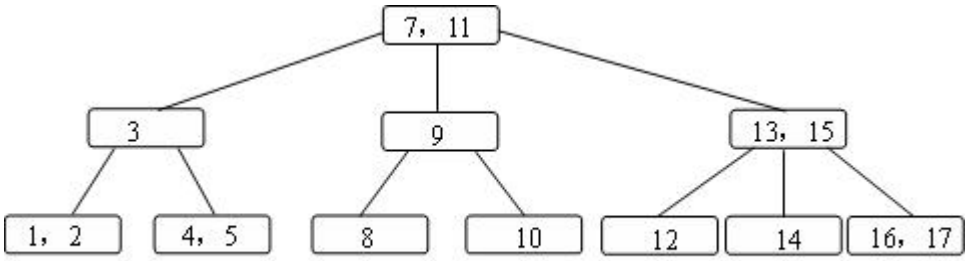
(a)



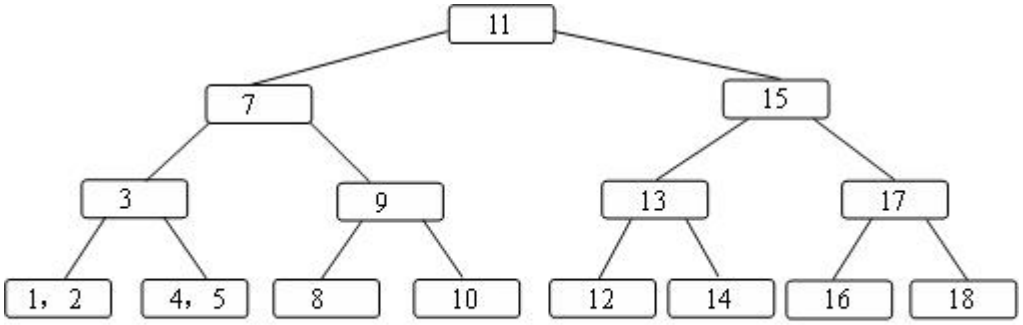
(b)



(c)



(d)

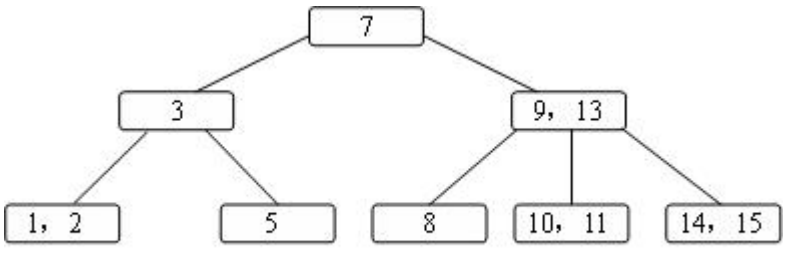


(e)

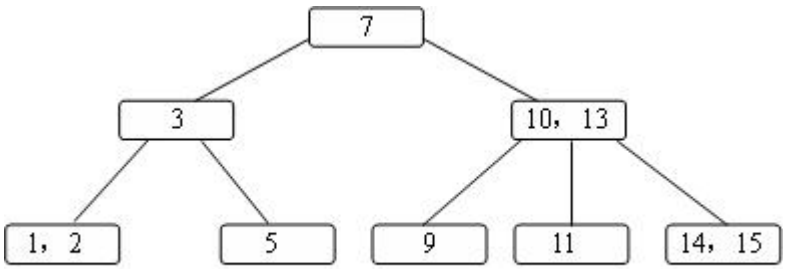
图 9-3　B-树的插入过程

4．对上题所示的 3 阶 B—树，分别给出删除关键码为 4，8，9 之后的结果。

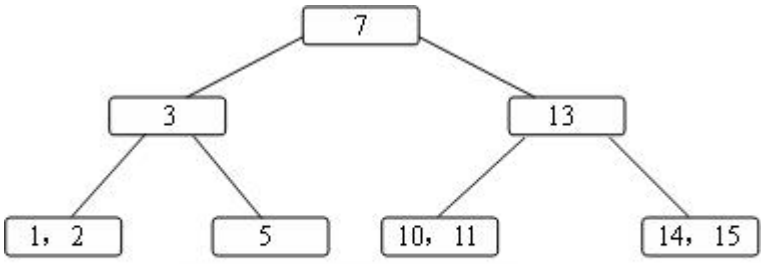
【解答】删除关键码为 4，8，10 之后的结果如图 9-4(a)，(b)，(c)所示：



(a)



(b)



(c)

图 9-4　B_树的删除过程

5．为什么在内存中使用的 B—树通常是 3 阶的，而不使用更高阶的 B—树？

【解答】作为外存上的动态查找，B—树比平衡二叉树的性能要好，但若要作为内存中的查找表，B—树却不一定比平衡二叉树性能好，因为查找等操作的时间性能在 m 阶 B—树上是 O(mlogtn)=O(log2n*(m/log2t))（n 为记录个数），而 m/log2t>1，故 m 较大时，O(mlog2n)比平衡的二叉排序树上相应操作的时间 O(log2n)大得多。因此，仅在内存中使用的 B—树必须取较小的 m，通常取最小值 m=3。

6．设有 10000 个记录，通过分块划分为若干子表并建立索引，那么为了提高查找效率，每一个子表的大小应设计为多大？

【解答】每个子表的大小应为 $\sqrt{10000}=100$。

学习自测及答案

1．在索引顺序表中，首先查找（ ），然后再查找相应的（ ），其平均查找长度等于（ ）。

【解答】索引表，块，查找索引表的平均长度与检索相应块的平均查找长度的和

2．既希望较快的查找又便于线性表动态变化的查找方法是（ ）。

A 顺序查找 B 折半查找 C 散列查找 D 索引顺序查找

【解答】D

3．在一个 3 阶的 B—树上，每个结点所含的子树数目最多为（ ）。

【解答】3

4．在一棵 m 阶的 B—树中，当将一个关键码插入某结点而引起该结点分裂时，此结点原有（ ）个关键码；若删去某结点中的一个关键码，而导致结点合并时，该结点原有（ ）个关键码。

【解答】m-1， ?m/2? -1

5．当向 B—树中插入关键码时，可能引起结点的（ ），最终可能导致整个 B-树的高度（ ），当从 B—树中删除关键码时，可能引起结点（ ），最终可能导致整个 B—树的高度（ ）。

【解答】分裂，增加 1，合并，减少 1

6．在 9 阶 B—树中，除根结点以外其他非叶子结点中的关键码个数不少于（ ）。

【解答】4

7．当向一棵 m 阶的 B—树做插入操作时，若一个结点中的关键字个数等于（ ），则必须分裂为两个结点。

A m B m-1 C m+1 D m/2

【解答】A

8．在一个 5 阶的 B—树上，每个非终端结点所含的子树数最少为（ ）。

A 2 B 3 C 4 D 5

【解答】B

9. 给定一组记录，其关键码为字母。记录按照下面的顺序插入一棵空的 B—树中：C，S，D，T，A，M，P，I，B，W，N，G，V，R，K，E，H，O，L，J。请画出插入这些记录后的 3 阶 B—树。

【解答】最后的 B—树如图 9-5 所示。

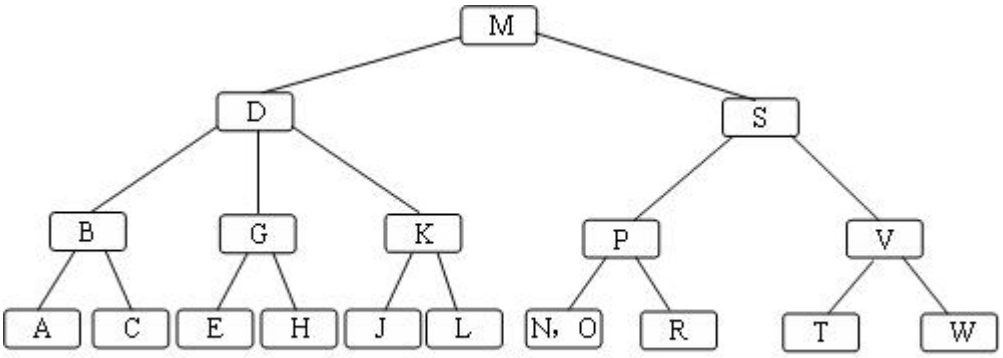


图 9-5　3 阶 B_树的最后结果

10．已知一个 B+树有 5 个叶子结点，每个叶子结点中的关键码如图 9-6 所示，请画出这棵 3 阶 B+树，然后在此 3 阶 B+树中插入关键码 65，再画出插入后的 B+树。

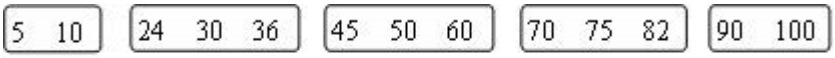


图 9-6　3 阶 B+树的叶子结点

【解答】该 B+树如图 9-7 所示，插入关键码 65 后，B+树如图 9-8 所示。

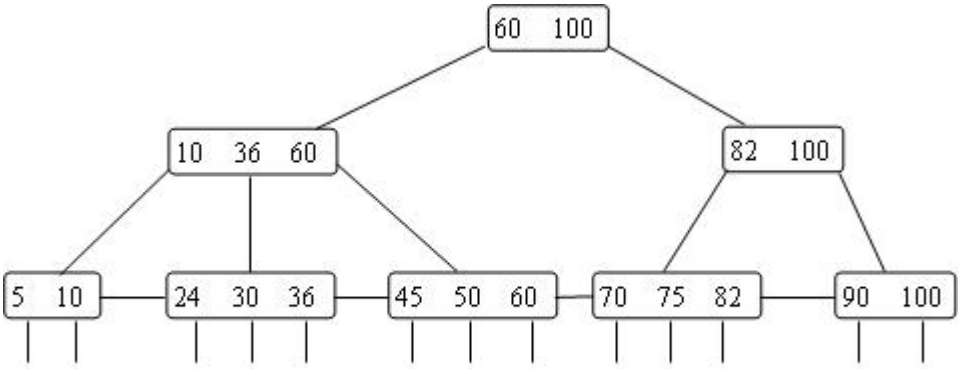


图 9-7　3 阶 B^+树的最后结果

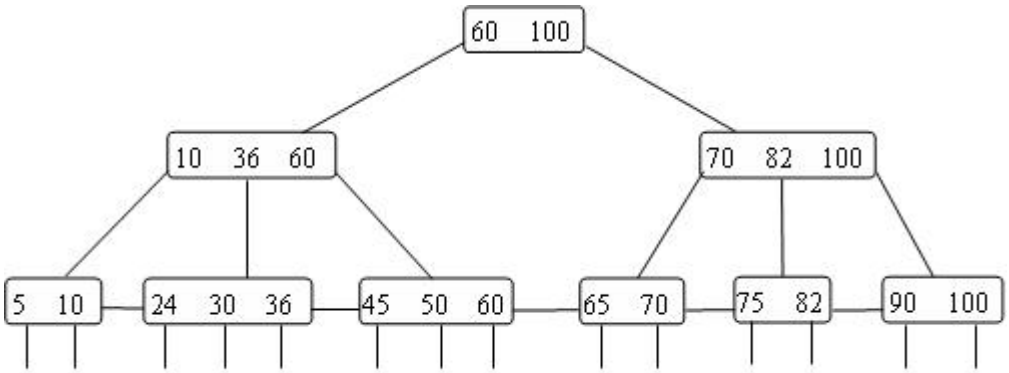


图 9-8　插入 65 后的 B^+树