

# 数据结构试题库及答案

## 第一章 概论

### 一、选择题

- 1、研究数据结构就是研究 ( D )。  
A. 数据的逻辑结构  
B. 数据的存储结构  
C. 数据的逻辑结构和存储结构  
D. 数据的逻辑结构、存储结构及其基本操作
- 2、算法分析的两个主要方面是 ( A )。  
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
- 3、具有线性结构的数据结构是 ( D )。  
A. 图  
B. 树  
C. 广义表  
D. 栈
- 4、计算机中的算法指的是解决某一个问题的有限运算序列，它必须具备输入、输出、 ( B ) 等5个特性。  
A. 可执行性、可移植性和可扩充性  
B. 可执行性、有穷性和确定性  
C. 确定性、有穷性和稳定性  
D. 易读性、稳定性和确定性
- 5、下面程序段的时间复杂度是 ( C )。  

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

  
A.  $O(m^2)$   
B.  $O(n^2)$   
C.  $O(m*n)$   
D.  $O(m+n)$
- 6、算法是 ( D )。  
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
- 7、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 ( C )。  
A.  $O(n)$   
B.  $O(n\log_2 n)$   
C.  $O(n^2)$   
D.  $O(\log_2 n)$
- 8、下面程序段的时间复杂度为 ( C )。  

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

  
A.  $O(n)$   
B.  $O(3n)$   
C.  $O(\log_3 n)$   
D.  $O(n^3)$
- 9、数据结构是一门研究非数值计算的程序设计问题中计算机的数据元素以及它们之间的 ( ) 和运算等的学科。  
A. 结构  
B. 关系  
C. 运算  
D. 算法
- 10、下面程序段的时间复杂度是 ( )。  

```
i=s=0;  
while(s<n){  
    i++;s+=i;  
}
```

  
A.  $O(n)$   
B.  $O(n^2)$   
C.  $O(\log_2 n)$   
D.  $O(n^3)$
- 11、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
- 12、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
- 13、下列程序段的时间复杂度为 ( B )。  

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

y=y+1;

A.  $O(n)$

B.  $O(\sqrt{n})$

C.  $O(1)$

D.  $O(n^2)$

## 二、填空题

- 1、程序段 “  $i=1; \text{while}(i \leq n) i=i*2;$  ” 的时间复杂度为 \_\_\_\_\_。
- 2、数据结构的四种基本类型中， \_\_\_\_\_ 树形结构 \_\_\_\_\_ 的元素是一对多关系。

## 三、综合题

- 1、将数量级  $O(1), O(N), O(N^2), O(N^3), O(N \log_2 N), O(\log_2 N), O(2^N)$  按增长率由小到大排序。
- 答案：  $O(1)$   $O(\log_2 N)$   $O(N)$   $O(N \log_2 N)$   $O(N^2)$   $O(N^3)$   $O(2^N)$

## 一、填空题

1. 数据结构被形式地定义为 (  $D, R$  ), 其中  $D$  是 数据元素 的有限集合,  $R$  是  $D$  上的 关系 有限集合。
2. 数据结构包括数据的 逻辑结构、数据的 存储结构 和数据的 运算 这三个方面的内容。
3. 数据结构按逻辑结构可分为两大类, 它们分别是 线性结构 和 非线性结构。
4. 线性结构中元素之间存在 一对一 关系, 树形结构中元素之间存在 一对多 关系, 图形结构中元素之间存在多对多关系。
5. 在线性结构中, 第一个结点 没有 前驱结点, 其余每个结点有且只有 1 个前驱结点; 最后一个结点 没有 后续结点, 其余每个结点有且只有 1 个后续结点。
6. 在树形结构中, 树根结点没有前驱结点, 其余每个结点有且只有 1 个前驱结点; 叶子结点没有 后续结点, 其余每个结点的后续结点数可以 任意多个。
7. 在图形结构中, 每个结点的前驱结点数和后续结点数可以 任意多个。
8. 数据的存储结构可用四种基本的存储方法表示, 它们分别是 顺序、链式、索引、散列。
9. 数据的运算最常用的有 5 种, 它们分别是 插入、删除、修改、查找、排序。
10. 一个算法的效率可分为 时间 效率和 空间 效率。
11. 任何一个 C 程序都由 一个主函数 和若干个被调用的其它函数组成。

## 二、单项选择题

- ( B ) 1. 非线性结构是数据元素之间存在一种 :  
A) 一对多关系      B) 多对多关系      C) 多对一关系      D) 一对一关系
- ( C ) 2. 数据结构中, 与所使用的计算机无关的是数据的 \_\_\_\_\_ 结构 ;  
A) 存储      B) 物理      C) 逻辑      D) 物理和存储
- ( C ) 3. 算法分析的目的是 :  
A) 找出数据结构的合理性      B) 研究算法中的输入和输出的关系  
C) 分析算法的效率以求改进      D) 分析算法的易懂性和文档性

- ( A ) 4. 算法分析的两个主要方面 5 是：
- A) 空间复杂性和时间复杂性                      B) 正确性和简明性  
C) 可读性和文档性                                  D) 数据复杂性和程序复杂性
- ( C ) 5. 计算机算法指的是：
- A) 计算方法                      B) 排序方法                      C) 解决问题的有限运算序列                      D) 调度方法
- ( B ) 6. 计算机算法必须具备输入、输出和 \_\_\_\_\_ 等 5 个特性。
- A) 可行性、可移植性和可扩充性                      B) 可行性、确定性和有穷性  
C) 确定性、有穷性和稳定性                              D) 易读性、稳定性和安全性

### 三、简答题

1. 数据结构和数据类型两个概念之间有区别吗？

答：简单地说，数据结构定义了一组按某些关系结合在一起的数组元素。数据类型不仅定义了一组带结构的数据元素，而且还在其上定义了一组操作。

2. 简述线性结构与非线性结构的不同点。

答：线性结构反映结点间的逻辑关系是一对一的，非线性结构反映结点间的逻辑关系是多对多的。

### 四、分析下面各程序段的时间复杂度

1.   for (i=0; i<n; i++)  
      for (j=0; j<m; j++)  
          A[i][j]=0;

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

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

4.   i=1;  
      while(i<=n)  
          i=i\*3;

五、设有数据逻辑结构  $S = (D, R)$ ，试按各小题所给条件画出这些逻辑结构的图示，并确定其是哪种逻辑结构。

1.    $D = \{d_1, d_2, d_3, d_4\}$                        $R = \{(d_1, d_2), (d_2, d_3), (d_3, d_4)\}$

2.    $D = \{d_1, d_2, \dots, d_9\}$   
       $R = \{(d_1, d_2), (d_1, d_3), (d_3, d_4), (d_3, d_6), (d_6, d_8), (d_4, d_5), (d_6, d_7), (d_8, d_9)\}$

3.    $D = \{d_1, d_2, \dots, d_9\}$   
       $R = \{(d_1, d_3), (d_1, d_8), (d_2, d_3), (d_2, d_4), (d_2, d_5), (d_3, d_9), (d_5, d_6), (d_8, d_9), (d_9, d_7), (d_4, d_7), (d_4, d_6)\}$

## 第二章 线性表

### 一、选择题

- 1、若长度为  $n$  的线性表采用顺序存储结构，在其第  $i$  个位置插入一个新元素算法的时间复杂度 ( )。  
A.  $O(\log_2 n)$  B.  $O(1)$  C.  $O(n)$  D.  $O(n^2)$
- 2、若一个线性表中最常用的操作是取第  $i$  个元素和找第  $i$  个元素的前趋元素，则采用 ( ) 存储方式最节省时间。  
A. 顺序表 B. 单链表 C. 双链表 D. 单循环链表
- 3、具有线性结构的数据结构是 ( )。  
A. 图 B. 树 C. 广义表 D. 栈
- 4、在一个长度为  $n$  的顺序表中，在第  $i$  个元素之前插入一个新元素时，需向后移动 ( ) 个元素。  
A.  $n-i$  B.  $n-i+1$  C.  $n-i-1$  D.  $i$
- 5、非空的循环单链表  $head$  的尾结点  $p$  满足 ( )。  
A.  $p \rightarrow next == head$  B.  $p \rightarrow next == NULL$   
C.  $p == NULL$  D.  $p == head$
- 6、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
- 7、在双向循环链表中，在  $p$  指针所指的结点后插入一个指针  $q$  所指向的新结点，修改指针的操作是 ( )。  
A.  $p \rightarrow next = q; q \rightarrow prior = p; p \rightarrow next \rightarrow prior = q; q \rightarrow next = q;$   
B.  $p \rightarrow next = q; p \rightarrow next \rightarrow prior = q; q \rightarrow prior = p; q \rightarrow next = p \rightarrow next;$   
C.  $q \rightarrow prior = p; q \rightarrow next = p \rightarrow next; p \rightarrow next \rightarrow prior = q; p \rightarrow next = q;$   
D.  $q \rightarrow next = p \rightarrow next; q \rightarrow prior = p; p \rightarrow next = q; p \rightarrow next = q;$
- 8、线性表采用链式存储时，结点的存储地址 ( )。  
A. 必须是连续的 B. 必须是不连续的  
C. 连续与否均可 D. 和头结点的存储地址相连续
- 9、在一个长度为  $n$  的顺序表中删除第  $i$  个元素，需要向前移动 ( ) 个元素。  
A.  $n-i$  B.  $n-i+1$  C.  $n-i-1$  D.  $i+1$
- 10、线性表是  $n$  个 ( ) 的有限序列。  
A. 表元素 B. 字符 C. 数据元素 D. 数据项
- 11、从表中任一结点出发，都能扫描整个表的是 ( )。  
A. 单链表 B. 顺序表 C. 循环链表 D. 静态链表
- 12、在具有  $n$  个结点的单链表上查找值为  $x$  的元素时，其时间复杂度为 ( )。  
A.  $O(n)$  B.  $O(1)$  C.  $O(n^2)$  D.  $O(n-1)$
- 13、线性表  $L=(a_1, a_2, \dots, a_n)$ ，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. 线性表中至少要有有一个元素  
C. 表中诸元素的排列顺序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，其余每个元素都由一个且仅有一个直接前驱和直接后继
- 14、一个顺序表的第一个元素的存储地址是 90，每个元素的长度为 2，则第 6 个元素的存储地址是 ( )。  
A. 98 B. 100 C. 102 D. 106
- 15、在线性表的下列存储结构中，读取元素花费的时间最少的是 ( )。  
A. 单链表 B. 双链表 C. 循环链表 D. 顺序表
- 16、在一个单链表中，若删除  $p$  所指向结点的后续结点，则执行 ( )。  
A.  $p \rightarrow next = p \rightarrow next \rightarrow next;$   
B.  $p = p \rightarrow next; p \rightarrow next = p \rightarrow next \rightarrow next;$   
C.  $p = p \rightarrow next;$   
D.  $p = p \rightarrow next \rightarrow next;$
- 17、将长度为  $n$  的单链表连接在长度为  $m$  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.  $O(m+n)$
- 18、线性表的顺序存储结构是一种 ( ) 存储结构。  
A. 随机存取 B. 顺序存取 C. 索引存取 D. 散列存取

- 19、顺序表中，插入一个元素所需移动的元素平均数是 ( )。
- A.  $(n-1)/2$                       B.  $n$                       C.  $n+1$                       D.  $(n+1)/2$
- 10、循环链表的主要优点是 ( )。
- A. 不再需要头指针                      B. 已知某结点位置后能容易找到其直接前驱
- C. 在进行插入、删除运算时能保证链表不断开
- D. 在表中任一结点出发都能扫描整个链表
- 11、不带头结点的单链表 head 为空的判定条件是 ( )。
- A.  $head == NULL$                       B.  $head \rightarrow next == NULL$
- C.  $head \rightarrow next == head$                       D.  $head != NULL$
- 12、在下列对顺序表进行的操作中，算法时间复杂度为  $O(1)$  的是 ( )。
- A. 访问第  $i$  个元素的前驱 ( $1 < i \leq n$ )                      B. 在第  $i$  个元素之后插入一个新元素 ( $1 \leq i \leq n$ )
- C. 删除第  $i$  个元素 ( $1 \leq i \leq n$ )                      D. 对顺序表中元素进行排序
- 13、已知指针 p 和 q 分别指向某单链表中第一个结点和最后一个结点。假设指针 s 指向另一个单链表中某个结点，则在 s 所指结点之后插入上述链表应执行的语句为 ( )。
- A.  $q \rightarrow next = s \rightarrow next$  ;  $s \rightarrow next = p$  ;
- B.  $s \rightarrow next = p$  ;  $q \rightarrow next = s \rightarrow next$  ;
- C.  $p \rightarrow next = s \rightarrow next$  ;  $s \rightarrow next = q$  ;
- D.  $s \rightarrow next = q$  ;  $p \rightarrow next = s \rightarrow next$  ;
- 14、在以下的叙述中，正确的是 ( )。
- A. 线性表的顺序存储结构优于链表存储结构
- B. 线性表的顺序存储结构适用于频繁插入 / 删除数据元素的情况
- C. 线性表的链表存储结构适用于频繁插入 / 删除数据元素的情况
- D. 线性表的链表存储结构优于顺序存储结构
- 15、在表长为 n 的顺序表中，当在任何位置删除一个元素的概率相同时，删除一个元素所需移动的平均个数为 ( )。
- A.  $(n-1)/2$                       B.  $n/2$                       C.  $(n+1)/2$
- D.  $n$
- 16、在一个单链表中，已知 q 所指结点是 p 所指结点的前驱结点，若在 q 和 p 之间插入一个结点 s，则执行 ( )。
- A.  $s \rightarrow next = p \rightarrow next$ ;  $p \rightarrow next = s$ ;
- B.  $p \rightarrow next = s \rightarrow next$ ;  $s \rightarrow next = p$ ;
- C.  $q \rightarrow next = s$ ;  $s \rightarrow next = p$ ;
- D.  $p \rightarrow next = s$ ;  $s \rightarrow next = q$ ;
- 17、在单链表中，指针 p 指向元素为 x 的结点，实现删除 x 的后继的语句是 ( )。
- A.  $p = p \rightarrow next$ ;                      B.  $p \rightarrow next = p \rightarrow next \rightarrow next$ ;
- C.  $p \rightarrow next = p$ ;                      D.  $p = p \rightarrow next \rightarrow next$ ;
- 18、在头指针为 head 且表长大于 1 的单循环链表中，指针 p 指向表中某个结点，若  $p \rightarrow next \rightarrow next == head$ ，则 ( )。
- A. p 指向头结点                      B. p 指向尾结点                      C. p 的直接后继是头结点
- D. p 的直接后继是尾结点

## 二、填空题

1、设单链表的结点结构为 ( data, next )。已知指针 p 指向单链表中的结点，q 指向新结点，欲将 q 插入到 p 结点之后，则需要执行的语句：\_\_\_\_\_；\_\_\_\_\_。

答案：  $q \rightarrow next = p \rightarrow next$                        $p \rightarrow next = q$

2、线性表的逻辑结构是 \_\_\_\_\_，其所含元素的个数称为线性表的 \_\_\_\_\_。

答案： 线性结构      长度

3、写出带头结点的双向循环链表 L 为空表的条件 \_\_\_\_\_。

答案：  $L \rightarrow prior == L \rightarrow next == L$

4、带头结点的单链表 head 为空的条件是 \_\_\_\_\_。

答案： head->next==NULL

5、在一个单链表中删除 p所指结点的后继结点时，应执行以下操作：

```
q = p->next;
p->next=_____;
```

答案： q->next

### 三、判断题

- 1、单链表不是一种随机存储结构。
- 2、在具有头结点的单链表中，头指针指向链表的第一个数据结点。
- 3、用循环单链表表示的链队列中，可以不设队头指针，仅在队尾设置队尾指针。
- 4、顺序存储方式只能用于存储线性结构。
- 5、在线性表的顺序存储结构中，逻辑上相邻的两个元素但是在物理位置上不一定是相邻的。
- 6、链式存储的线性表可以随机存取。

### 四、程序分析填空题

1、函数 GetElem 实现返回单链表的第 i 个元素，请在空格处将算法补充完整。

```
int GetElem(LinkList L,int i,Elemtype *e){
    LinkList p      ; int j      ;
    p=L->next;j=1;
    while(p&&j<i){
        (1)      ;++j;
    }
    if(!p||j>i) return ERROR;
    *e= (2)      ;
    return OK;
}
```

答案： (1) p=p->next (2)p->data

2、函数实现单链表的插入算法，请在空格处将算法补充完整。

```
int ListInsert(LinkList L,int i,ElemType e){
    LNode *p,*s;int j;
    p=L;j=0;
    while((p!=NULL)&&(j<i-1)){
        >next;j++;
    }
    if(p==NULL||j>i-1) return ERROR;
    s=(LNode *)malloc(sizeof(LNode));
    s->data=e;
    (1)      ;
    (2)      ;
    return OK;
}/*ListInsert*/
```

p=p-

答案： (1) s->next=p->next (2)p->next=s

3、函数 ListDelete\_sq 实现顺序表删除算法，请在空格处将算法补充完整。

```
int ListDelete_sq(SqList *L,int i){
    int k;
    if(i<1||i>L->length) return ERROR;
    for(k=i-1;k<L->length-1;k++)
        L->slist[k]= ( 1 )      ;
    ( 2 )      ;
    return OK;
}
```

答案： ( 1 ) L->slist[k+1] ( 2 ) --L->Length

4、函数实现单链表的删除算法，请在空格处将算法补充完整。

```
int ListDelete(LinkList L,int i,ElemType *s){
    LNode *p,*q;
    int j;
```

```

p=L;j=0;
while((      ( 1 )      )&&(j<i-1)){
    p=p->next;j++;
}
if(p->next==NULL||j>i-1) return ERROR;
q=p->next;
      ( 2 )      ;
*s=q->data;
free(q);
return OK;
}/*listDelete*/
答案： (1)p->next!=NULL (2)p->next=q->next

```

5、写出算法的功能。

```

int L(head){
    node * head;
    int n=0;
    node *p;
    p=head;
    while(p!=NULL)
    { p=p->next;
      n++;
    }
    return(n);
}

```

答案：求单链表 head 的长度

## 五、综合题

1、编写算法，实现带头结点单链表的逆置算法。

```

答案： void invert(Lnode *head)
{
    Lnode *p,*q;
    if(!head->next) return ERROR;
    p=head->next; q=p->next; p->next =NULL;
    while(q)
        {p=q; q=q->next; p->next=head->next; head->next=p;}
}

```

2、有两个循环链表，链头指针分别为 L1 和 L2，要求写出算法将 L2 链表链到 L1 链表之后，且连接后仍保持循环链表形式。

```

答案： void merge(Lnode *L1, Lnode *L2)
{
    Lnode *p,*q ;
    while(p->next!=L1)
        p=p->next;
    while(q->next!=L2)
        q=q->next;
    q->next=L1; p->next =L2;
}

```

3、设一个带头结点的单向链表的头指针为 head，设计算法，将链表的记录，按照 data 域的值递增排序。

```

答案： void assending(Lnode *head)
{
    Lnode *p,*q , *r, *s;
    p=head->next; q=p->next; p->next=NULL;
    while(q)
        {r=q; q=q->next;
          if(r->data<=p->data)
              {r->next=p; head->next=r; p=r; }
          else
              {while(!p && r->data>p->data)

```



```

        {s=p; p=p->next; }
        r->next=p; s->next=r;}
    p=head->next; }
}

```

4、编写算法，将一个头指针为 head 不带头结点的单链表改造为一个单向循环链表，并分析算法的时间复杂度。

答案：

```

void linklist_c(Lnode *head)
{
    Lnode *p; p=head;
    if(!p) return ERROR;
    while(p->next!=NULL)
        p=p->next;
    p->next=head;
}

```

设单链表的长度（数据结点数）为 N，则该算法的时间主要花费在查找链表最后一个结点上（算法中的 while 循环），所以该算法的时间复杂度为  $O(N)$ 。

5、已知 head 为带头结点的单循环链表的头指针，链表中的数据元素依次为（a1, a2, a3, a4, ..., an），A 为指向空的顺序表的指针。阅读以下程序段，并回答问题：

（1）写出执行下列程序段后的顺序表 A 中的数据元素；

（2）简要叙述该程序段的功能。

```

if(head->next!=head)
{
    p=head->next;
    A->length=0;
    while(p->next!=head)
    {
        p=p->next;
        A->data[A->length++] = p->data;
        if(p->next!=head) p=p->next;
    }
}

```

答案：

（1）（a2, a4, ..., an） （2）将循环单链表中偶数结点位置的元素值写入顺序表 A

6、设顺序表 va 中的数据元素递增有序。试写一算法，将 x 插入到顺序表的适当位置上，以保持该表的有序性。

答案：

```

void Insert_sq(SqList va[], ElemType x)
{
    int i, j, n;
    n=length(va);
    if(x>=va[n])
        va[n]=x;
    else
    {
        i=0;
        while(x>va[i]) i++;
        for(j=n-1; j>=i; j--)
            va[j+1]=va[j];
        va[i]=x;
    }
    n++;
}

```

7、假设线性表采用顺序存储结构，表中元素值为整型。阅读算法 f2，设顺序表 L=(3,7,3,2,1,1,8,7,3)，写出执行算法 f2 后的线性表 L 的数据元素，并描述该算法的功能。

```

void f2(SeqList *L){
    int i,j,k;
    k=0;
    for(i=0;i<L->length;i++){
        for(j=0;j<k && L->data[i]!=L->data[j];j++);
        if(j==k){
            if(k!=i)L->data[k]=L->data[i];

```



```

        k++;
    }
}
L->length=k;
}

```

答案：

(3,7,2,1,8) 删除顺序表中重复的元素

8、已知线性表中的元素以值递增有序排列，并以单链表作存储结构。试写一算法，删除表中所有大于  $x$  且小于  $y$  的元素（若表中存在这样的元素）同时释放被删除结点空间。

答案：

```

void Delete_list(Lnode *head, ElemType x, ElemType y)
{
    Lnode *p, *q;
    if(!head) return ERROR;
    p=head; q=p;
    while(p)
    {
        if(p->data>x) && (p->data<y))i++;
        if(p==head)
        {
            head=p->next; free(p);
            p=head; q=p;
        }
        else
        {
            q->next=p->next; free(p);
            p=q->next;
        }
        else
        {
            q=p; p=p->next;
        }
    }
}

```

9、在带头结点的循环链表  $L$  中，结点的数据元素为整型，且按值递增有序存放。给定两个整数  $a$  和  $b$ ，且  $a < b$ ，编写算法删除链表  $L$  中元素值大于  $a$  且小于  $b$  的所有结点。

### 第三章 栈和队列

#### 一、选择题

- 一个栈的输入序列为： $a, b, c, d, e$ ，则栈的不可能输出的序列是（ ）。  
 A.  $a, b, c, d, e$       B.  $d, e, c, b, a$   
 C.  $d, c, e, a, b$       D.  $e, d, c, b, a$
- 判断一个循环队列  $Q$ （最多  $n$  个元素）为满的条件是（ ）。  
 A.  $Q \rightarrow rear == Q \rightarrow front$       B.  $Q \rightarrow rear == Q \rightarrow front + 1$   
 C.  $Q \rightarrow front == (Q \rightarrow rear + 1) \% n$       D.  $Q \rightarrow front == (Q \rightarrow rear - 1) \% n$
- 设计一个判别表达式中括号是否配对的算法，采用（ ）数据结构最佳。  
 A. 顺序表      B. 链表      C. 队列      D. 栈
- 带头结点的单链表  $head$  为空的判定条件是（ ）。  
 A.  $head == NULL$       B.  $head \rightarrow next == NULL$   
 C.  $head \rightarrow next != NULL$       D.  $head != NULL$
- 一个栈的输入序列为： $1, 2, 3, 4$ ，则栈的不可能输出的序列是（ ）。  
 A. 1243      B. 2134      C. 1432      D. 4312      E. 3214
- 若用一个大小为 6 的数组来实现循环队列，且当  $rear$  和  $front$  的值分别为 0，3。当从队列中删除一个元素，再加入两个元素后， $rear$  和  $front$  的值分别为（ ）。  
 A. 1 和 5      B. 2 和 4      C. 4 和 2      D. 5 和 1
- 队列的插入操作是在（ ）。  
 A. 队尾      B. 队头      C. 队列任意位置

- 8、循环队列的队头和队尾指针分别为 front 和 rear , 则判断循环队列为空的条件是 ( )。
- A. front==rear  
B. front==0  
C. rear==0  
D. front=rear+1

- 9、一个顺序栈 S，其栈顶指针为 top，则将元素 e入栈的操作是（ ）。
- A. \*S->top=e;S->top++;                      B. S->top++;\*S->top=e;
- C. \*S->top=e                                      D. S->top=e;

- 10、表达式  $a*(b+c)-d$  的后缀表达式是 ( )。
- A.  $abcd+ -$                       B.  $abc+*d -$                       C.  $abc*+d -$                       D.  $-+*abcd$

- 11、将递归算法转换成对应的非递归算法时，通常需要使用（ ）来保存中间结果。
- A. 队列                      B. 栈                      C. 链表                      D. 树

- 12、栈的插入和删除操作在( )。
- A. 栈底                      B. 栈顶                      C. 任意位置                      D. 指定位置

- 13、五节车厢以编号 1 , 2 , 3 , 4 , 5 顺序进入铁路调度站(栈) , 可以得到( )的编组。
- A. 3 , 4 , 5 , 1 , 2                      B. 2 , 4 , 1 , 3 , 5
- C. 3 , 5 , 4 , 2 , 1                        D. 1 , 3 , 5 , 2 , 4

- 14、判定一个顺序栈 S（栈空间大小为 n）为空的条件是（ ）。
- A. S->top==0                      B. S->top!=0
- C. S->top==n                        D. S->top!=n

- 15、在一个链队列中，front 和 rear 分别为头指针和尾指针，则插入一个结点 s 的操作为 ( )。
- A. front=front->next  
B. s->next=rear;rear=s  
C. rear->next=s;rear=s;  
D. s->next=front;front=s;

- 16、一个队列的入队序列是 1，2，3，4，则队列的出队序列是（ ）。
- A. 1，2，3，4                      B. 4，3，2，1
- C. 1，4，3，2                      D. 3，4，1，2

- 17、依次在初始为空的队列中插入元素 a,b,c,d 以后，紧接着做了两次删除操作，此时的队头元素是（ ）。  
A. a B. b C. c D. d

- 18、正常情况下，删除非空的顺序存储结构的堆栈的栈顶元素，栈顶指针 top 的变化是 ( )。
- A. top 不变                      B. top=0                      C. top=top+1                      D. top=top-1

- 19、判断一个循环队列 Q(空间大小为 M)为空的条件是( )。
- A.  $Q \rightarrow front == Q \rightarrow rear$  B.  $Q \rightarrow rear - Q \rightarrow front - 1 == M$

- C.  $Q \rightarrow \text{front} + 1 = Q \rightarrow \text{rear}$                       D.  $Q \rightarrow \text{rear} + 1 = Q \rightarrow \text{front}$

- 20、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. 栈                      D. 线性表的链式存储结构

- 21、当用大小为 N 的数组存储顺序循环队列时，该队列的最大长度为 ( )。
- A. N                                      B. N+1                                      C. N-1                                      D. N-2

- 22、队列的删除操作是在 ( )。
- A. 队首                      B. 队尾                      C. 队前                      D. 队后

- 23、若让元素 1, 2, 3依次进栈，则出栈次序不可能是( )。
- A. 3, 2, 1                  B. 2, 1, 3                  C. 3, 1, 2                  D. 1, 3, 2

- 24、循环队列用数组 A[0 , m-1] 存放其元素值，已知其头尾指针分别是 front 和 rear ，则当前队列中的元素个数是（ ）。
- A. (rear-front+m)%m                      B. rear-front+1
- C. rear-front-1                                D. rear-front

- [illegible]

- 26、栈和队列都是（ ）。

- ## 二、填空题

答案：3

答案：  $(\text{rear-front}+M)\%M$

答案：  $n-1$

答案：61

### 三、判断题

- #### 四、程序分析填空题

int InitStack(SqStack *S); //	构造空栈
int StackEmpty(SqStack *S); //	判断栈空
int Push(SqStack *S, ElemType e); //	入栈
int Pop(SqStack *S, ElemType *e); //	出栈

```
void conversion(){
    InitStack(S);
    scanf(    "%d ",&N);
    while(N){
        _____ ( 1 ) _____ ;
        N=N/8;
    }
    while( _____ ( 2 ) _____ ){
        Pop(S,&e);
        printf(    "%d",e);
    }
}
//conversion
```

2、写出算法的功能。

11

```

    Q->front=(Q->front+1)%MAXSIZE;
    return OK;
}

```

3、阅读算法 f2，并回答下列问题：

(1) 设队列 Q=(1, 3, 5, 2, 4, 6)。写出执行算法 f2 后的队列 Q;

(2) 简述算法 f2 的功能。

```

void f2(Queue *Q){
    DataType e;
    if (!QueueEmpty(Q)){
        e=DeQueue(Q);
        f2(Q);
        EnQueue(Q,e);
    }
}

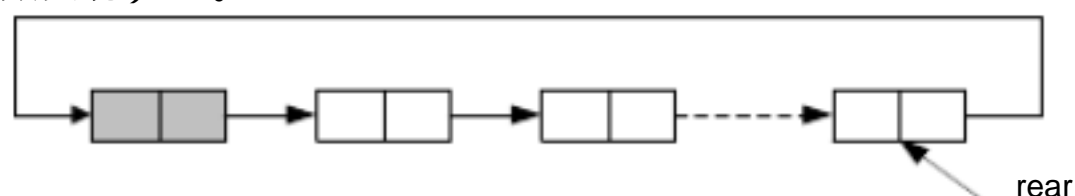
```

答案：(1) 6,4,2,5,3,1

(2) 将队列倒置

## 五、综合题

1、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结点，但不设头指针，请写出相应的入队列算法（用函数实现）。



答案：void EnQueue(Lnode \*rear, ElemType e)

```

{
    Lnode *new;
    New=(Lnode *)malloc(sizeof(Lnode));
    If(!new) return ERROR;
    new->data=e; new->next=rear->next;
    rear->next=new; rear =new;
}

```

2、已知 Q 是一个非空队列，S 是一个空栈。编写算法，仅用队列和栈的 ADT 函数和少量工作变量，将队列 Q 的所有元素逆置。

栈的 ADT 函数有：

void makeEmpty(SqStack s);	置空栈
void push(SqStack s,ElemType e);	元素 e 入栈
ElemType pop(SqStack s);	出栈，返回栈顶元素
int isEmpty(SqStack s);	判断栈空

队列的 ADT 函数有：

void enQueue(Queue q,ElemType e);	元素 e 入队
ElemType deQueue(Queue q);	出队，返回队头元素
int isEmpty(Queue q);	判断队空

答案：void QueueInvert(Queue q)

```

{
    ElemType x;
    makeEmpty(SqStack s);
    while(!isEmpty(Queue q))
        {x=deQueue(Queue q);
        push(SqStack s, ElemTypex);}
    while(!isEmpty(SqStack s))
        {x=pop(SqStack s);
        enQueue(Queue q, ElemType x);}
}

```

3、对于一个栈，给出输入项 A,B,C,D，如果输入项序列为 A,B,C,D，试给出全部可能的输出序列。

答案：出栈的可能序列：

ABCD ABDC ACDB ACBD ADCB BACD BADC BCAD BCDA  
 CBDA CBAD CDBA DCBA

## 第四章 串

### 一、选择题

- 1、设有两个串 S1和 S2，求串 S2在S1中首次出现位置的运算称作（ C ）。  
A. 连接 B. 求子串 C. 模式匹配 D. 判断子串
- 2、已知串 S=?aaab ?，则 next 数组值为（ A ）。  
A. 0123 B. 1123 C. 1231 D. 1211
- 3、串与普通的线性表相比较，它的特殊性体现在（ C ）。  
A. 顺序的存储结构 B. 链式存储结构  
C. 数据元素是一个字符 D. 数据元素任意
- 4、设串长为 n，模式串长为 m，则 KMP 算法所需的附加空间为（ A ）。  
A. O(m) B. O(n) C. O(m\*n) D. O(nlog<sub>2</sub>m)
- 5、空串和空格串（ B ）。  
A. 相同 B. 不相同 C. 可能相同 D. 无法确定
- 6、与线性表相比，串的插入和删除操作的特点是（ ）。  
A. 通常以串整体作为操作对象 B. 需要更多的辅助空间  
C. 算法的时间复杂度较高 D. 涉及移动的元素更多
- 7、设 SUBSTR(S,i,k) 是求 S中从第 i 个字符开始的连续 k 个字符组成的子串的操作，则对于 S='Beijing&Nanjing'，SUBSTR(S,4,5)=（ B ）。  
A. ,ijing? B. ,jing&? C. ,ingNa?  
D. ,ing&N?

### 二、判断题

- ( ) 1、造成简单模式匹配算法 BF算法执行效率低的原因是有回溯存在。
- ( ) 2、KMP算法的最大特点是指主串的指针不需要回溯。
- ( ) 3、完全二叉树某结点有右子树，则必然有左子树。

### 三、填空题

- 1、求子串在主串中首次出现的位置的运算称为 模式匹配 。
- 2、设 s=?I AM A TEACHER?，其长度是 。
- 3、两个串相等的充分必要条件是 两个串的长度相等且 对应位置字符相同 。

### 四、程序填空题

- 1、函数 kmp实现串的模式匹配，请在空格处将算法补充完整。

```
int kmp(sqstring *s,sqstring *t,int start,int next[]){
    int i=start-1,j=0;
    while(i<s->len&& j<t->len)
        if(j== -1 || s->data[i]==t->data[j]){
            i++;j++;
        }
        else j= _____;
    if(j>=t->len)
        return( _____);
    else
        return(-1);
}
```

- 2、函数实现串的模式匹配算法，请在空格处将算法补充完整。

```
int index_bf(sqstring*s,sqstring *t,int start){
    int i=start-1,j=0;
    while(i<s->len&& j<t->len)
        if(s->data[i]==t->data[j]){
            i++;j++;
        }else{
            i= _____;j=0;
        }
    if(j>=t->len)
        return _____;
}
```

```

else
    return -1;
}/*listDelete*/
3、写出下面算法的功能。
int function(SqString *s1,SqString *s2){
    int i;
    for(i=0;i<s1->length&& i<s1->length;i++)
        if(s->data[i]!=s2->data[i])
            return s1->data[i]-s2->data[i];
    return s1->length-s2->length;
}

```

答案：.串比较算法

4、写出算法的功能。

```

int fun(sqstring *s,sqstring *t,int start){
    int i=start-1,j=0;
    while(i<s->len&&j<t->len)
        if(s->data[i]==t->data[j]){
            i++;j++;
        }else{
            i=i-j+1;j=0;
        }
    if(j>=t->len)
        return i-t->len+1;
    else
        return -1;
}

```

答案：串的模式匹配算法

## 第五章 数组和广义表

一、选择题

- 1、设广义表  $L=((a, b, c))$ ，则  $L$  的长度和深度分别为 ( C )。  
 A. 1 和 1                      B. 1 和 3                      C. 1 和 2                      D. 2 和 3
- 2、广义表  $((a),a)$  的表尾是 ( B )。  
 A. a                      B. (a)                      C. ()                      D. ((a))
- 3、稀疏矩阵的常见压缩存储方法有 ( C ) 两种。  
 A. 二维数组和三维数组                      B. 三元组和散列表                      C. 三元组和十字链表                      D. 散列表和十字链表
- 4、一个非空广义表的表头 ( D )。  
 A. 不可能是子表                      B. 只能是子表                      C. 只能是原子                      D. 可以是子表或原子
- 5、数组  $A[0..5,0..6]$  的每个元素占 5 个字节，将其按列优先次序存储在起始地址为 1000 的内存单元中，则元素  $A[5][5]$  的地址是 ( A )。  
 A. 1175                      B. 1180                      C. 1205                      D. 1210
- 6、广义表  $G=(a,b(c,d,(e,f)),g)$  的长度是 ( A )。  
 A. 3                      B. 4                      C. 7                      D. 8
- 7、采用稀疏矩阵的三元组表形式进行压缩存储，若要完成对三元组表进行转置，只要将行和列对换，这种说法 ( B )。  
 A. 正确                      B. 错误                      C. 无法确定                      D. 以上均不对
- 8、广义表  $(a,b,c)$  的表尾是 ( B )。  
 A. b,c                      B. (b,c)                      C. c                      D. (c)
- 9、常对数组进行两种基本操作是 ( C )。  
 A. 建立和删除                      B. 索引和修改                      C. 查找和修改

# D. 查找与索引

10、对一些特殊矩阵采用压缩存储的目的主要是为了( D )。

- 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开销

11、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<sub>11</sub> 为第一个元素，其存储地址为 1，每元素占 1 个地址空间，则 a<sub>85</sub> 的地址为( )。

- A. 13
- B. 33
- C. 18
- D. 40

12、设矩阵 A 是一个对称矩阵，为了节省存储，将其下三角部分按行序存放在一维数组 B[1,n(n-1)/2] 中，对下三角部分中任一元素 a<sub>ij</sub>(i>=j)，在一维数组 B 的下标位置 k 的值是( B )。

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

13、广义表 A=((a),a) 的表头是( B )。

- A. a
- B. (a)
- C. b
- D. ((a))

14、稀疏矩阵一般的压缩存储方法有两种，即( C )。

- A. 二维数组和三维数组
- B. 三元组和散列
- C. 三元组和十字链表
- D. 散列和十字链表

15、假设以三元组表表示稀疏矩阵，则与如图所示三元组表对应的 4×5 的稀疏矩阵是(注：矩阵的行列下标均从 1 开始)( B )。

A. 
$$\begin{pmatrix} 0 & -8 & 0 & 6 & 0 \\ 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -5 & 0 & 4 & 0 & 0 \end{pmatrix}$$

B. 
$$\begin{pmatrix} 0 & -8 & 0 & 6 & 0 \\ 7 & 0 & 0 & 0 & 3 \\ -5 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

C. 
$$\begin{pmatrix} 0 & -8 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 7 & 0 & 0 & 0 & 0 \\ -5 & 0 & 4 & 0 & 0 \end{pmatrix}$$

D. 
$$\begin{pmatrix} 0 & -8 & 0 & 6 & 0 \\ 7 & 0 & 0 & 0 & 0 \\ -5 & 0 & 4 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

0	1	2	-8
1	1	4	6
2	2	1	7
3	2	5	3
4	3	1	-5
5	3	3	4

16、以下有关广义表的表述中，正确的是( A )。

- A. 由 0 个或多个原子或子表构成的有限序列
- B. 至少有一个元素是子表
- C. 不能递归定义
- D. 不能为空

表

17、对广义表 L=((a,b),((c,d),(e,f)))

执行 head(tail(head(tail(L))))

操作的结果是

- ( )。
- A. 的
- B. e
- C. (e)
- D. (e,f)

## 二、判断题

- ( ) 1、广义表中原子个数即为广义表的长度。
- ( ) 2、一个稀疏矩阵采用三元组表示，若把三元组中有关行下标与列下标的值互换，并把 mu 和 nu 的值进行互换，则完成了矩阵转置。
- ( ) 3、稀疏矩阵压缩存储后，必会失去随机存取功能。
- ( ) 4、广义表的长度是指广义表中括号嵌套的层数。
- ( ) 5、广义表是一种多层次的数据结构，其元素可以是单原子也可以是子表。

## 三、填空题

1、已知二维数组 A[m][n] 采用行序为主方式存储，每个元素占 k 个存储单元，并且第一个元素的存储地址是 LOC(A[0][0])，则 A[i][j] 的地址是 LOC(A[0][0])+(i\*N+j)\*k。

2、广义表运算式 HEAD(TAIL((a,b,c),(x,y,z))) 的结果是： (x,y,z)。

3、二维数组，可以按照 行序为主 两种不同的存储方式。

4、稀疏矩阵的压缩存储方式有： 三元组表 和 十字链表。

## 四、综合题

1、现有一个稀疏矩阵，请给出它的三元组表。



0	3	1	0
1	0	0	0
0	2	1	0
0	0	-2	0

答案：

i	j	v
1	2	3
1	3	1
2	1	1
3	2	2
3	3	1
4	3	-2

第六章 树

一、选择题

- 1、二叉树的深度为 k，则二叉树最多有（ C ）个结点。  
A. 2k                                      B.  $2^{k-1}$                                       C.  $2^k-1$                                       D. 2k-1
- 2、用顺序存储的方法，将完全二叉树中所有结点按层逐个从左到右的顺序存放在一维数组 R[1..N] 中，若结点 R[i] 有右孩子，则其右孩子是（ B ）。  
A. R[2i-1]                                      B. R[2i+1]                                      C. R[2i]                                      D. R[2/i]
- 3、设 a,b 为一棵二叉树上的两个结点，在中序遍历时， a在b前面的条件是（ B ）。  
A. a 在b的右方                                      B. a 在b的左方                                      C. a 是b的祖先                                      D. a 是b的子孙
- 4、设一棵二叉树的中序遍历序列： badce ，后序遍历序列： bdeca ，则二叉树先序遍历序列为（ ）。  
A. adbce                                      B. decab                                      C. debac                                      D. abcde
- 5、在一棵具有 5 层的满二叉树中结点总数为（ A ）。  
A. 31                                      B. 32                                      C. 33                                      D. 16
- 6、由二叉树的前序和后序遍历序列（ B ）惟一确定这棵二叉树。  
A. 能                                      B. 不能
- 7、某二叉树的中序序列为 ABCDEFG，后序序列为 BDCAFGE，则其左子树中结点数目为（ C ）。  
A. 3                                      B. 2                                      C. 4                                      D. 5
- 8、若以 {4,5,6,7,8} 作为权值构造哈夫曼树，则该树的带权路径长度为（ C ）。  
A. 67                                      B. 68                                      C. 69                                      D. 70
- 9、将一棵有 100 个结点的完全二叉树从根这一层开始，每一层上从左到右依次对结点进行编号，根结点的编号为 1，则编号为 49 的结点的左孩子编号为（ A ）。  
A. 98                                      B. 99                                      C. 50                                      D. 48
- 10、表达式 a\*(b+c)-d 的后缀表达式是（ B ）。  
A. abcd+-                                      B. abc+\*d-                                      C. abc\*+d-                                      D. -+\*abcd
- 11、对某二叉树进行先序遍历的结果为 ABDEFC，中序遍历的结果为 DBFEAC，则后序遍历的结果是（ B ）。  
A. DBFEAC                                      B. DFEBCA                                      C. BDFECA                                      D. BDEFAC
- 12、树最适合用来表示（ C ）。  
A. 有序数据元素                                      B. 无序数据元素                                      C. 元素之间具有分支层次关系的数据                                      D. 元素之间无联系的数据
- 13、表达式 A\*(B+C)/(D-E+F) 的后缀表达式是（ C ）。  
A. A\*B+C/D-E+F                                      B. AB\*C+D/E-F+                                      C. ABC+\*DE-F+/                                      D. ABCDED\*+/-+
- 14、在线索二叉树中， t 所指结点没有左子树的充要条件是（ ）。  
A. t->left==NULL                                      B. t->ltag==1                                      C.                                      D. t->

>lt;tag==1&&t->left==NULL

D. 以上都不对

15、任何一棵二叉树的叶结点在先序、中序和后序遍历序列中的相对次序 ( )。

A. 不发生改变

B. 发生改变

C. 不能确定

D. 以上都不对

16、假定在一棵二叉树中，度为 2 的结点数为 15，度为 1 的结点数为 30，则叶子结点数为 ( ) 个。

A. 15

B. 16

C. 17

D. 47

17、在下列情况中，可称为二叉树的是 ( B )。

A. 每个结点至多有两棵子树的树

B. 哈夫曼树

C. 每个结点至多有两棵子树的有序树

D. 每个结点只有一棵子树

18、用顺序存储的方法，将完全二叉树中所有结点按层逐个从左到右的顺序存放在一维数组 R[1..n] 中，若结点 R[i] 有左孩子，则其左孩子是 ( )。

A. R[2i-1]

B. R[2i+1]

C. R[2i]

D. R[2/i]

19、下面说法中正确的是 ( )。

A. 度为 2 的树是二叉树

B. 度为 2 的有序树是

二叉树

C. 子树有严格左右之分的树是二叉树

D. 子树有严格左右之分，且

度不超过 2 的树是二叉树

20、树的先根序列等同于与该树对应的二叉树的 ( )。

A. 先序序列

B. 中序序列

C. 后序序列

D.

层序序列

21、按照二叉树的定义，具有 3 个结点的二叉树有 ( C ) 种。

A. 3

B. 4

C. 5

D. 6

22、由权值为 3, 6, 7, 2, 5 的叶子结点生成一棵哈夫曼树，它的带权路径长度为 ( A )。

A. 51

B. 23

C. 53

D. 74

## 二、判断题

( ) 1、存在这样的二叉树，对它采用任何次序的遍历，结果相同。

( ) 2、中序遍历一棵二叉排序树的结点，可得到排好序的结点序列。

( ) 3、对于任意非空二叉树，要设计其后序遍历的非递归算法而不使用堆栈结构，最适合的方法是对该二叉树采用三叉链表。

( ) 4、在哈夫曼编码中，当两个字符出现的频率相同时，其编码也相同，对于这种情况应做特殊处理。

( ) 5、一个含有 n 个结点的完全二叉树，它的高度是  $\lfloor \log_2 n \rfloor + 1$ 。

( ) 6、完全二叉树的某结点若无左孩子，则它必是叶结点。

## 三、填空题

1、具有 n 个结点的完全二叉树的深度是  $\lfloor \log_2 n \rfloor + 1$ 。

2、哈夫曼树是其树的带权路径长度 最小 的二叉树。

3、在一棵二叉树中，度为 0 的结点的个数是  $n_0$ ，度为 2 的结点的个数为  $n_2$ ，则有  $n_0 = n_2 + 1$ 。

4、树内各结点度的 最大值 称为树的度。

## 四、代码填空题

1、函数 InOrderTraverse(Bitree bt) 实现二叉树的中序遍历，请在空格处将算法补充完整。

```
void InOrderTraverse(BiTree bt){
    if( ) {
        InOrderTraverse(bt->lchild);
        printf( " %c", bt->data);
        ;
    }
}
```

2、函数 depth 实现返回二叉树的高度，请在空格处将算法补充完整。

```
int depth(Bitree *t){
    if(t==NULL)
```

```

        return 0;
    else{
        hl=depth(t->lchild);
        hr=      depth(t->rchild)      ;
        if(      hl>hr      )
            return hl+1;
        else
            return hr+1;
    }
}

```

3、写出下面算法的功能。

```

Bitree *function(Bitree *bt){
    Bitree *t,*t1,*t2;
    if(bt==NULL)
        t=NULL;
    else{
        t=(Bitree *)malloc(sizeof(Bitree));
        t->data=bt->data;
        t1=function(bt->left);
        t2=function(bt->right);
        t->left=t2;
        t->right=t1;
    }
    return(t);
}

```

答案：交换二叉树结点左右子树的递归算法

4、写出下面算法的功能。

```

void function(Bitree *t){
    if(p!=NULL){
        function(p->lchild);
        function(p->rchild);
        printf(    "%d",p->data);
    }
}

```

答案：二叉树后序遍历递归算法

## 五、综合题

1、假设以有序对  $\langle p,c \rangle$  表示从双亲结点到孩子结点的一条边，若已知树中边的集合为

$\{\langle a,b \rangle, \langle a,d \rangle, \langle a,c \rangle, \langle c,e \rangle, \langle c,f \rangle, \langle c,g \rangle, \langle c,h \rangle, \langle e,i \rangle, \langle e,j \rangle, \langle g,k \rangle\}$ ,

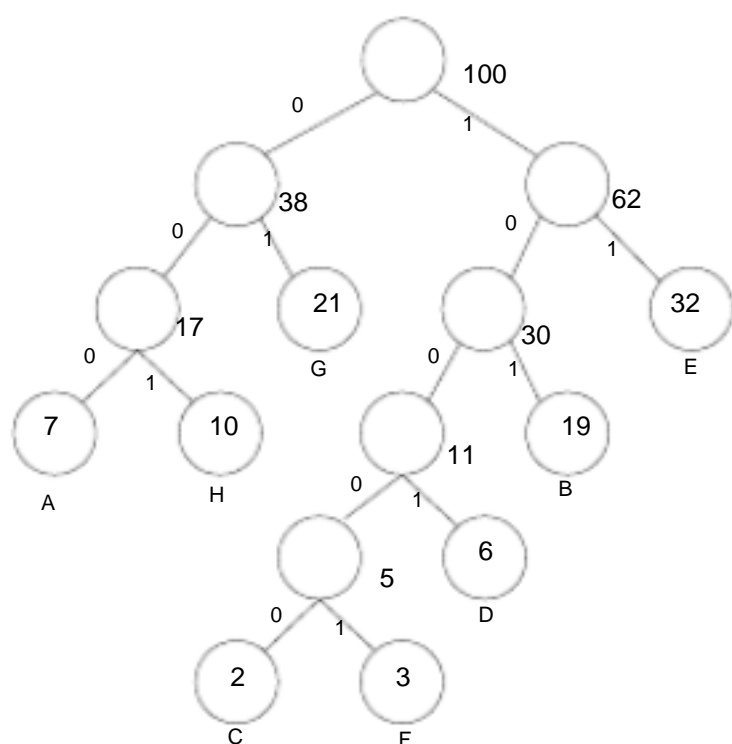
请回答下列问

题：

- (1) 哪个结点是根结点？
- (2) 哪些结点是叶子结点？
- (3) 哪些结点是 k 的祖先？
- (4) 哪些结点是 j 的兄弟？
- (5) 树的深度是多少？

2、假设一棵二叉树的先序序列为 EBADCFHGIKJ，中序序列为 ABCDEFGHIJK，请画出该二叉树。

3、假设用于通讯的电文仅由 8 个字母 A B、C、D E、F、G H 组成，字母在电文中出现的频率分别为：0.07，0.19，0.02，0.06，0.32，0.03，0.21，0.10。请为这 8 个字母设计哈夫曼编码。

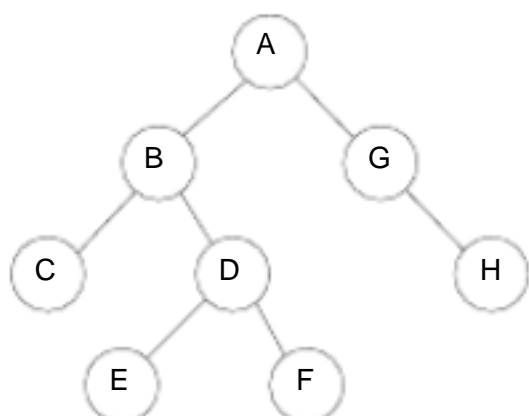


A	000
B	101
C	10000
D	1001
E	11
F	10001
G	01
H	001

答案：

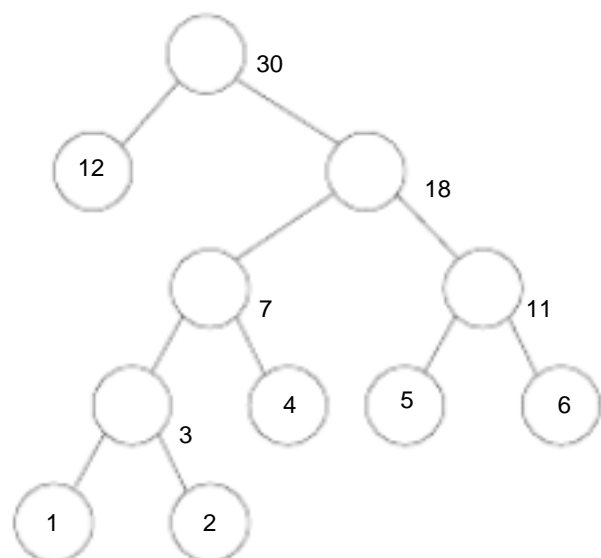
4、已知二叉树的先序遍历序列为 ABCDEFGH, 中序遍历序列为 CBEDFAGH, 画出二叉树。

答案：二叉树形态



5、试用权集合 {12,4,5,6,1,2} 构造哈夫曼树，并计算哈夫曼树的带权路径长度。

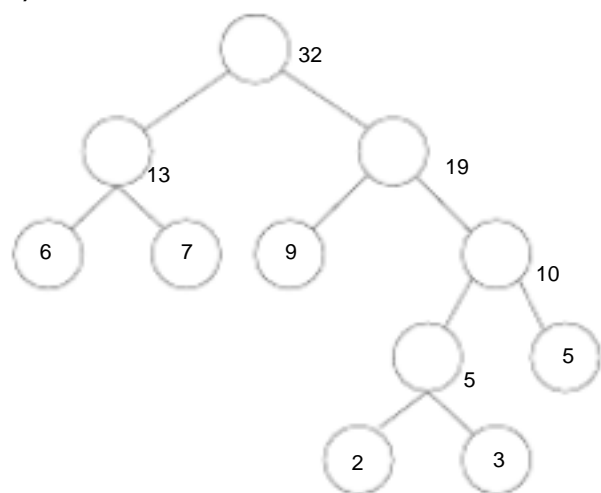
答案：



$$WPL=12*1+(4+5+6)*3+(1+2)*4=12+45+12=69$$

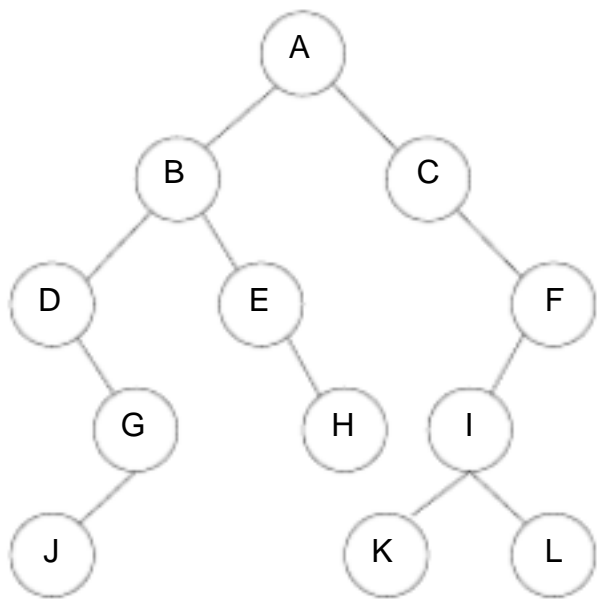
6、已知权值集合为 {5,7,2,3,6,9}，要求给出哈夫曼树，并计算带权路径长度 WPL。

答案：(1) 树形态：



(2) 带权路径长度：  $WPL=(6+7+9)*2+5*3+(2+3)*4=44+15+20=79$

7、已知一棵二叉树的先序序列： ABDGJEHCFIKL；中序序列： DJGBEHACKILF。画出二叉树的形态。



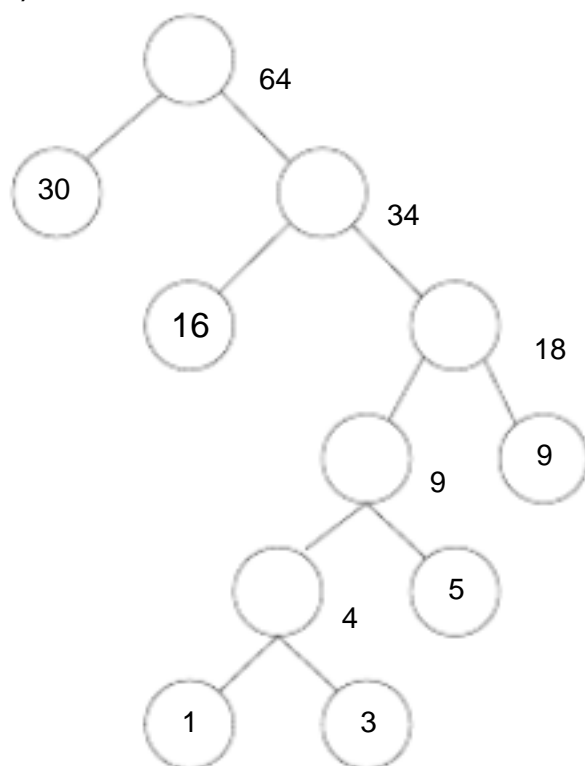
答案：

8、一份电文中有 6 种字符：A,B,C,D,E,F，它们的出现频率依次为 16, 5, 9, 3, 30, 1，完成问题：

(1) 设计一棵哈夫曼树；（画出其树结构）

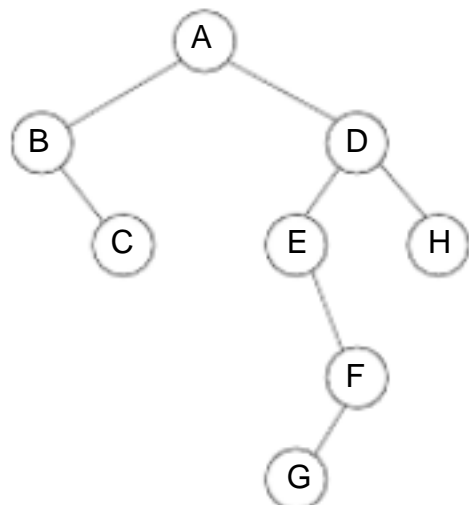
(2) 计算其带权路径长度 WPL；

答案：(1) 树形态：

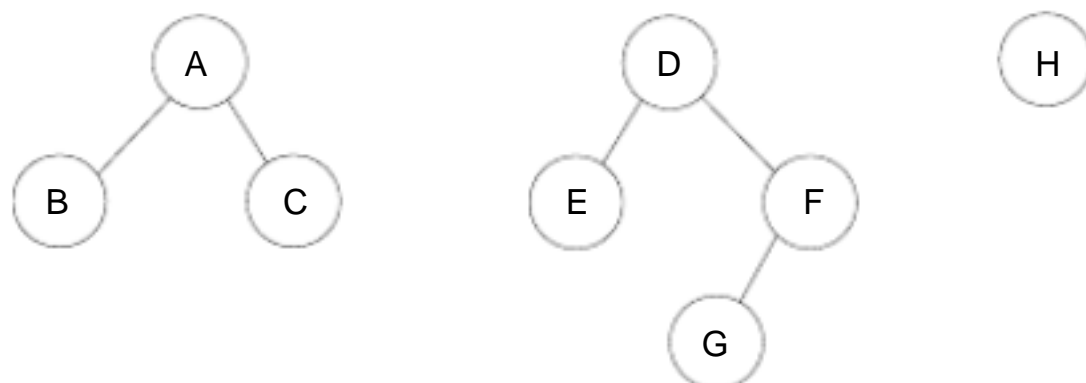


(2) 带权路径长度：  $WPL = 30 \times 1 + 16 \times 2 + 9 \times 3 + 5 \times 4 + (1+3) \times 5 = 30 + 32 + 27 + 20 + 20 = 129$

9、已知某森林的二叉树如下所示，试画出它所表示的森林。

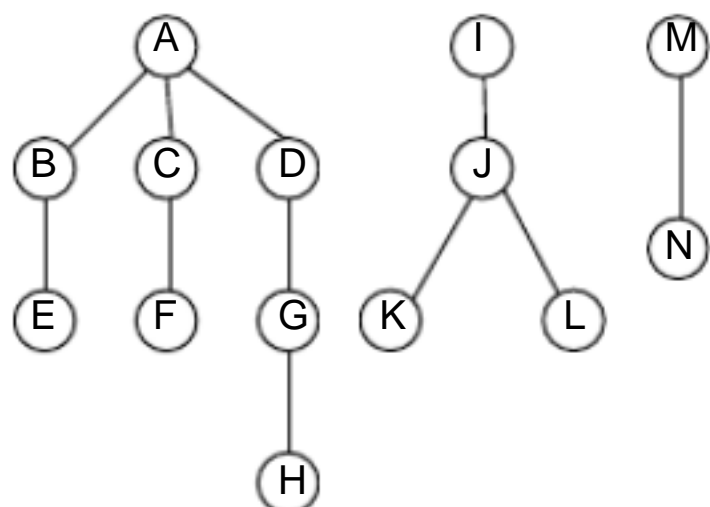


答案：

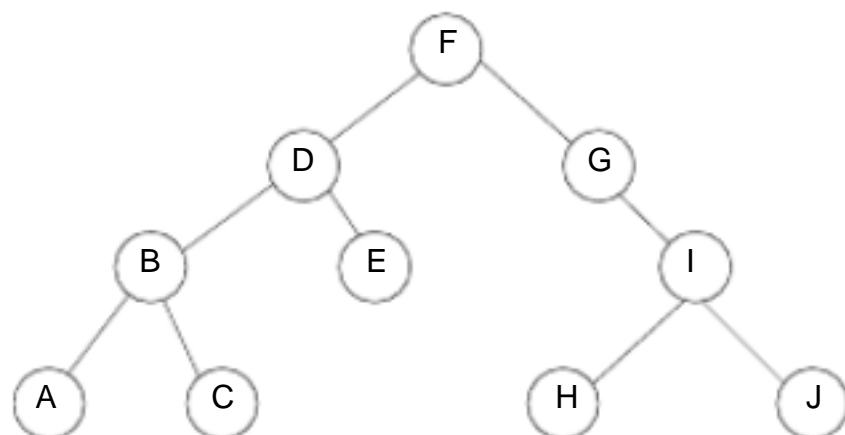


10、有一分电文共使用 5 个字符 ;a,b,c,d,e，它们的出现频率依次为 4、7、5、2、9，试构造哈夫曼树，并给出每个字符的哈夫曼编码。

11、画出与下图所示的森林相对应的二叉树，并指出森林中的叶子结点在二叉树中具有什么特点。



12、如下所示的二叉树，请写出先序、中序、后序遍历的序列。



答案：先序： FDBACEGIHJ

中序： ABCDEFGHIJ

后序： ACBEDHJIGF

## 六、编程题

1、编写求一棵二叉树中结点总数的算法。

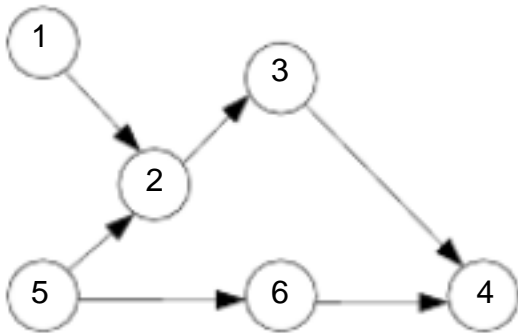
答案：（以先序遍历的方法为例）

```
void count_preorder(Bitree *t, int *n)
{
    if(t!=NULL)
    {
        *n++;
        count_preorder(t->lchild);
        count_preorder(t->rchild);
    }
}
```

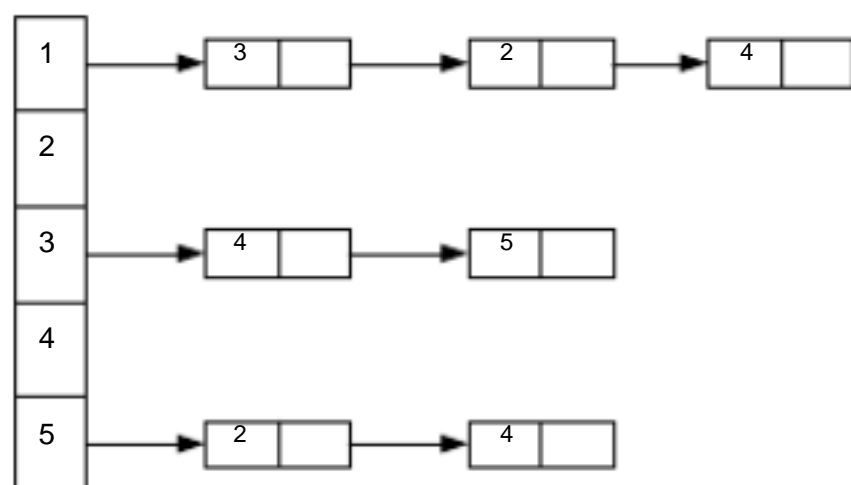
## 第七章 图

### 一、选择题

- 12、对于具有  $n$  个顶点的图，若采用邻接矩阵表示，则该矩阵的大小为（ ）。  
A.  $n^2$  B.  $n^2$  C.  $n-1$  D.  $(n-1)^2$
- 如果从无向图的任一顶点出发进行一次深度优先搜索即可访问所有顶点，则该图一定是（ ）。  
A. 完全图 B. 连通图 C. 有回路 D. 一棵树
- 关键路径是事件结点网络中（ ）。  
A. 从源点到汇点的最长路径 B. 从源点到汇点的最短路径  
C. 最长的回路 D. 最短的回路
- 下面（ ）可以判断出一个有向图中是否有环（回路） 。  
A. 广度优先遍历 B. 拓扑排序  
C. 求最短路径 D. 求关键路径
- 带权有向图  $G$  用邻接矩阵  $A$  存储，则顶点  $i$  的入度等于  $A$  中（ ）。  
A. 第  $i$  行非无穷的元素之和 B. 第  $i$  列非无穷的元素个数之和

- C. 第*i*行非无穷且非 0 的元素个数      D. 第*i*行与第*i*列非无穷且非 0 的元素之和
- 6、采用邻接表存储的图，其深度优先遍历类似于二叉树的（      ）。
- A. 中序遍历      B. 先序遍历      C. 后序遍历      D. 按层次遍历
- 7、无向图的邻接矩阵是一个（      ）。
- A. 对称矩阵      B. 零矩阵      C. 上三角矩阵      D. 对角矩阵
- 8、当利用大小为 *N* 的数组存储循环队列时，该队列的最大长度是（      ）。
- A. *N*-2      B. *N*-1      C. *N*      D. *N*+1
- 9、邻接表是图的一种（      ）。
- A. 顺序存储结构      B. 链式存储结构      C. 索引存储结构      D. 散列存储结构
- 10、下面有向图所示的拓扑排序的结果序列是（      ）。
- A. 125634      B. 516234      C. 123456      D. 521643
- 
- 11、在无向图中定义顶点  $v_i$  与  $v_j$  之间的路径为从  $v_i$  到  $v_j$  的一个（      ）。
- A. 顶点序列      B. 边序列      C. 权值总和      D. 边的条数
- 12、在有向图的逆邻接表中，每个顶点邻接表链接着该顶点所有（      ）邻接点。
- A. 入边      B. 出边      C. 入边和出边      D. 不是出边也不是入边
- 13、设  $G_1=(V_1,E_1)$  和  $G_2=(V_2,E_2)$  为两个图，如果  $V_1 \subseteq V_2, E_1 \subseteq E_2$  则称（      ）。
- A.  $G_1$  是  $G_2$  的子图      B.  $G_2$  是  $G_1$  的子图      C.  $G_1$  是  $G_2$  的连通分量      D.  $G_2$  是  $G_1$  的连通分量
- 14、已知一个有向图的邻接矩阵表示，要删除所有从第  $i$  个结点发出的边，应（      ）。
- A. 将邻接矩阵的第  $i$  行删除      B. 将邻接矩阵的第  $i$  行元素全部置为 0      C. 将邻接矩阵的第  $i$  列删除      D. 将邻接矩阵的第  $i$  列元素全部置为 0
- 15、任一个有向图的拓扑序列（      ）。
- A. 不存在      B. 有一个      C. 一定有多      D. 有一个或多个
- 16、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
- 17、下列关于图遍历的说法不正确的是（      ）。
- A. 连通图的深度优先搜索是一个递归过程      B. 图的广度优先搜索中邻接点的寻找具有“先进先出”的特征      C. 非连通图不能用深度优先搜索法      D. 图的遍历要求每一顶点仅被访问一次
- 18、带权有向图  $G$  用邻接矩阵  $A$  存储，则顶点  $i$  的入度为  $A$  中：（      ）。
- A. 第  $i$  行非  $\infty$  的元素之和      B. 第  $i$  列非  $\infty$  的元素之和      C. 第  $i$  行非  $\infty$  且非 0 的元素个数      D. 第  $i$  列非  $\infty$  且非 0 的元素个数
- 19、采用邻接表存储的图的广度优先遍历算法类似于二叉树的（      ）。
- A. 先序遍历      B. 中序遍历      C. 后序遍历      D. 按层次遍历
- 20、一个具有  $n$  个顶点的有向图最多有（      ）条边。
- A.  $n \times (n-1)/2$       B.  $n \times (n-1)$       C.  $n \times (n+1)/2$       D.  $n^2$
- 21、已知一个有向图的邻接表存储结构如图所示，根据深度优先遍历算法，从顶点  $v_1$  出发，所得到的顶点序列是（      ）。





- A.  $v_1, v_2, v_3, v_5, v_4$       B.  $v_1, v_2, v_3, v_4, v_5$   
 C.  $v_1, v_3, v_4, v_5, v_2$       D.  $v_1, v_4, v_3, v_5, v_2$
- 22、关键路径是事件结点网络中 ( )。
- A. 从源点到汇点的最长路径      B. 从源点到汇点的最短路径  
 C. 最长的回路      D. 最短的回路
- 23、以下说法正确的是 ( )。
- A. 连通分量是无向图中的极小连通子图  
 B. 强连通分量是有向图中的极大强连通子图  
 C. 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  $\langle a, b \rangle$   
 D. 对有向图  $G$ ，如果以任一顶点出发进行一次深度优先或广度优先搜索能访问到每个顶点，则该图一定是完全图
- 24、假设有向图含  $n$  个顶点及  $e$  条弧，则表示该图的邻接表中包含的弧结点个数为 ( )。
- A.  $n$       B.  $e$       C.  $2e$       D.  $n \cdot e$
- 25、设图的邻接矩阵为  $\begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ ，则该图为 ( )。
- A. 有向图      B. 无向图      C. 强连通图  
 D. 完全图
- 26、为便于判别有向图中是否存在回路，可借助于 ( )。
- A. 广度优先搜索算法      B. 最小生成树算法  
 C. 最短路径算法      D. 拓扑排序算法
- 27、任何一个无向连通图的最小生成树 ( ) 种。
- A. 只有一棵      B. 有一棵或多棵      C. 一定有多棵      D. 可能不存在
- 28、已知一有向图的邻接表存储结构如图所示，根据有向图的广度优先遍历算法，从顶点  $v_1$  出发，所得到的顶点序列是 ( )。
- |   |   |   |   |
|---|---|---|---|
| 1 | 3 | 2 | Λ |
| 2 | Λ |   |   |
| 3 | 4 | 5 | Λ |
| 4 | Λ |   |   |
| 5 | 2 | 4 | Λ |
- A.  $v_1, v_2, v_3, v_4, v_5$       B.  $v_1, v_3, v_2, v_4, v_5$       C.  $v_1, v_2, v_3, v_5, v_4$       D.  $v_1, v_4, v_3, v_5, v_2$
- 29、对于一个有向图，若一个顶点的入度为  $k_1$ 、出度为  $k_2$ ，则对应邻接表中该顶点单链表中的结点数为 ( )。
- A.  $k_1$       B.  $k_2$       C.  $k_1 + k_2$       D.  $k_1 - k_2$
- 30、一个具有 8 个顶点的有向图中，所有顶点的入度之和与所有顶点的出度之和的差等于 ( )。
- A. 16      B. 4      C. 0      D. 2
- 31、无向图中的一个顶点的度是指图中 ( )。

A. 通过该顶点的简单路径数

B. 与该顶点相邻接的顶点数

C. 与该顶点连通的顶点数

D. 通过该顶点的回路数

## 二、填空题

1、n个顶点的连通图至少有 \_\_\_\_\_ 边。

答案：n-1 条

2、一个连通图的生成树是一个 \_\_\_\_\_，它包含图中所有顶点，但只有足以构成一棵树的n-1 条边。

答案：极小连通子图

3、一个图的 \_\_\_\_\_ 表示法是惟一的。

答案：邻接矩阵

4、遍历图的基本方法有深度优先搜索和广度优先搜索，其中 \_\_\_\_\_ 是一个递归过程。

答案：深度优先搜索

5、在无向图 G的邻接矩阵 A中，若  $A[i][j]$  等于 1，则  $A[j][i]$  等于 \_\_\_\_\_。

答案：1

6、判定一个有向图是否存在回路，可以利用 \_\_\_\_\_。

答案：拓扑排序

7、已知一个图的邻接矩阵表示，计算第 i 个结点的入度的方法是 \_\_\_\_\_。

8、n个顶点的无向图最多有 \_\_\_\_\_ 边。

9、已知一个图的邻接矩阵表示，删除所有从第 i 个结点出发的边的方法是 \_\_\_\_\_。

10、若以邻接矩阵表示有向图，则邻接矩阵上第 i 行中非零元素的个数即为顶点  $v_i$  的 \_\_\_\_\_。

## 三、判断题

1、图的连通分量是无向图的极小连通子图。

2、一个图的广度优先搜索树是惟一的。

3、图的深度优先搜索序列和广度优先搜索序列不是惟一的。

4、邻接表只能用于存储有向图，而邻接矩阵则可存储有向图和无向图。

5、存储图的邻接矩阵中，邻接矩阵的大小不但与图的顶点个数有关，而且与图的边数也有关。

6、AOV网是一个带权的有向图。

7、从源点到终点的最短路径是唯一的。

8、邻接表只能用于存储有向图，而邻接矩阵则可存储有向图和无向图。

9、图的生成树是惟一的。

## 四、程序分析题

1、写出下面算法的功能。

```
typedef struct{
    int vexnum,arcnum;
    char vexts[N];
    int arcs[N][N];
}graph;
void funtion(int i,graph *g){
    int j;
    printf("node:%c\n",g->vexts[i]);
    visited[i]=TRUE;
    for(j=0;j<g->vexnum;j++)
        if((g->arcs[i][j]==1)&&(!visited[j]))
            function(j,g);
}
```

答案：实现图的深度优先遍历算法

## 五、综合题

1、已知图 G 的邻接矩阵如下所示：

(1) 求从顶点 1 出发的广度优先搜索序列；

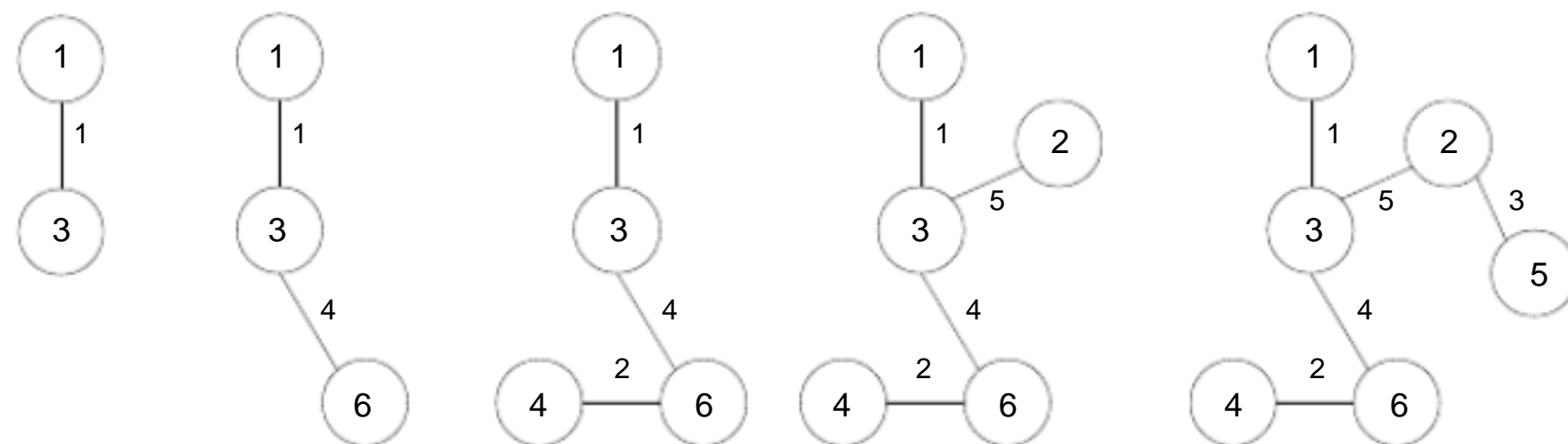
(2) 根据 prim 算法，求图 G 从顶点 1 出发的最小生成树，要求表示出其每一步生成过程。 (用图

或者表的方式均可)。

$\infty$	6	1	5	$\infty$	$\infty$
6	$\infty$	5	$\infty$	3	$\infty$
1	5	$\infty$	5	6	4
5	$\infty$	5	$\infty$	$\infty$	2
$\infty$	3	6	$\infty$	$\infty$	6
$\infty$	$\infty$	4	2	6	$\infty$

答案：(1) 广度优先遍历序列：1; 2, 3, 4; 5; 6

(2) 最小生成树 (prim 算法)



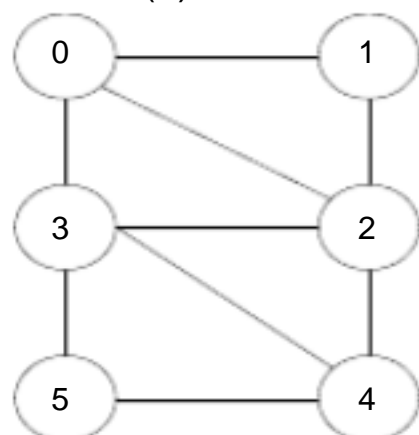
2、设一个无向图的邻接矩阵如下图所示：

(1) 画出该图；

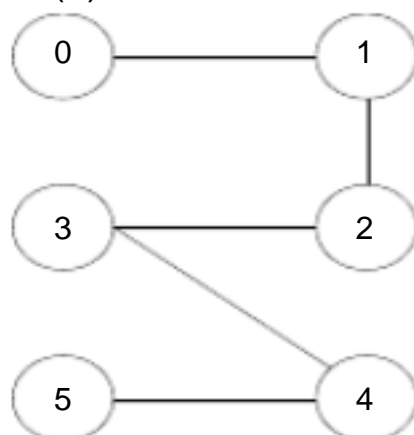
(2) 画出从顶点 0 出发的深度优先生成树；

0	1	1	1	0	0
1	0	1	0	0	0
1	1	0	1	1	0
1	0	1	0	1	1
0	0	1	1	0	1
0	0	0	1	1	0

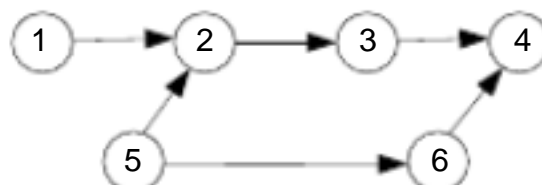
答案：(1) 图形态



(2) 深度优先搜索树



3、写出下图中全部可能的拓扑排序序列。



答案：1, 5, 2, 3, 6, 4

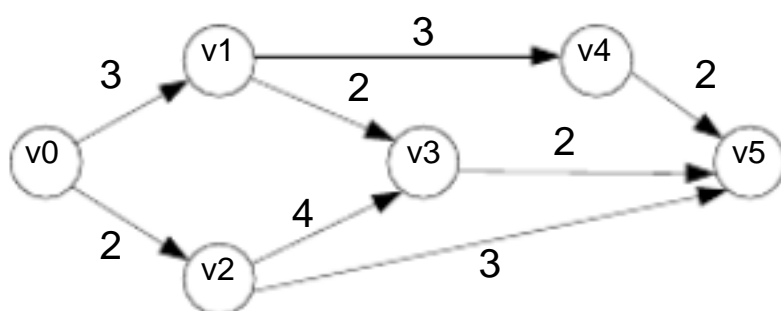
1, 5, 6, 2, 3, 4

5, 1, 2, 3, 6, 4

5, 1, 6, 2, 3, 4

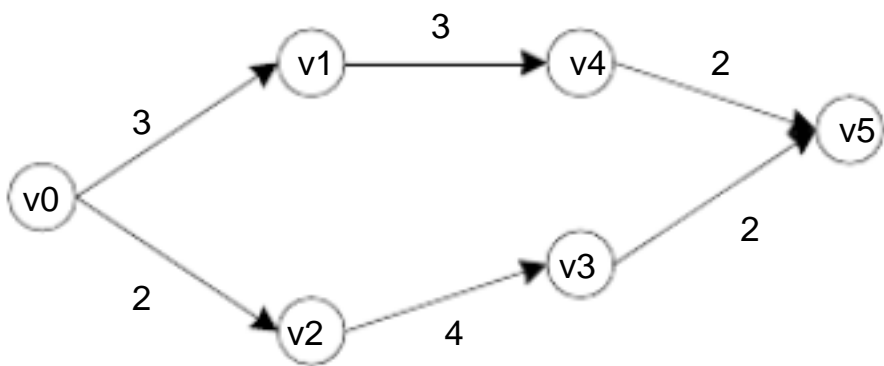
5, 6, 1, 2, 3, 4

4、AOE网 G 如下所示，求关键路径。（要求标明每个顶点的最早发生时间和最迟发生时间，并画出关键路径）

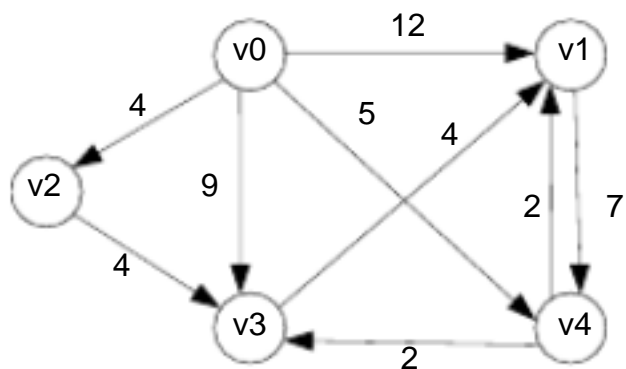


答案： (1)最早发生时间和最迟发生时间： (2) 关键路径：

顶点	ve	vl
v0	0	0
v1	3	3
v2	2	2
v3	6	6
v4	6	6
v5	8	8



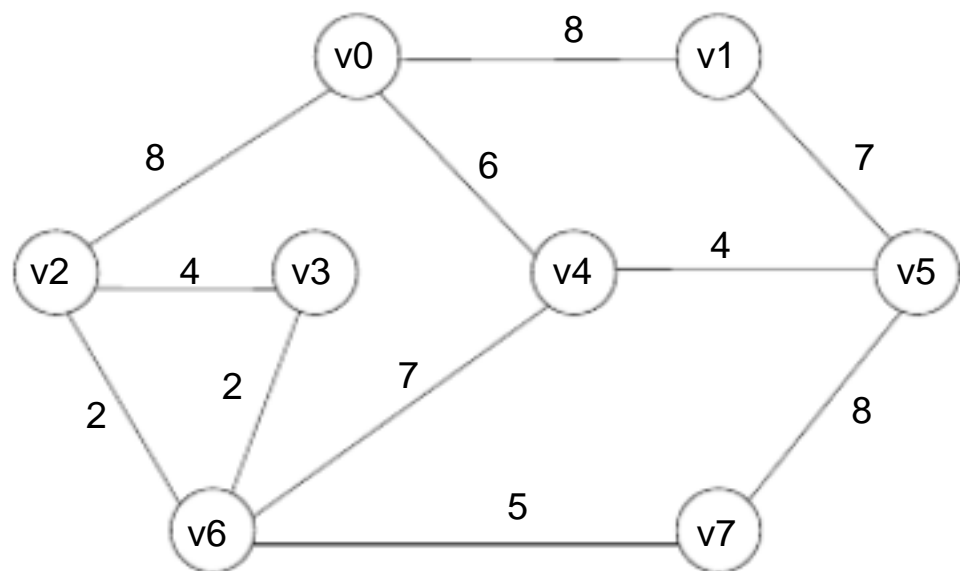
5、已知有向图 G 如下所示，根据迪杰斯特拉算法求顶点 v0 到其他顶点的最短距离。（给出求解过程）



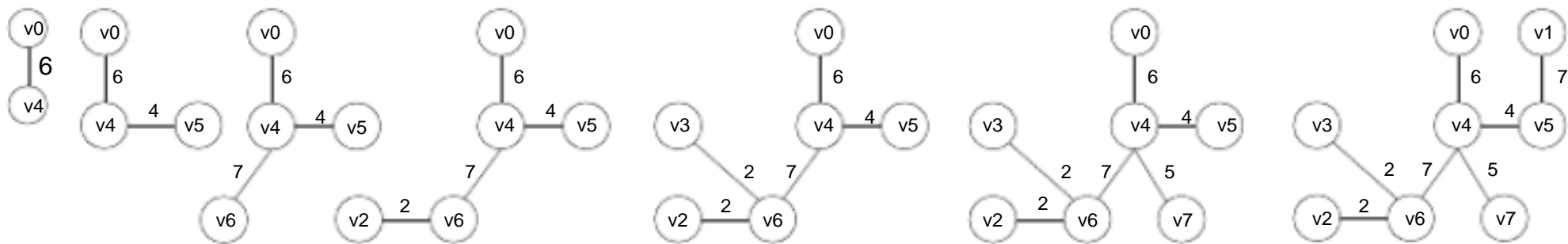
答案：

终点	从 v0 到各终点的 d 值和最短路径的求解过程			
	i=1	i=2	i=3	i=4
v1	12 (v0,v1)	12 (v0,v1)	<b>7 (v0,v4,v1)</b>	
v2	<b>4 (v0,v2)</b>			
v3	9 (v0,v3)	8 (v0,v2,v3)	7 (v0,v4,v3)	<b>7 (v0,v4,v3)</b>
v4	5 (v0,v4)	<b>5 (v0,v4)</b>		
vj	v2	v4	v1	v3
s	{v0,v2}	{v0,v4}	{v0,v4,v1}	{v0,v4,v3}

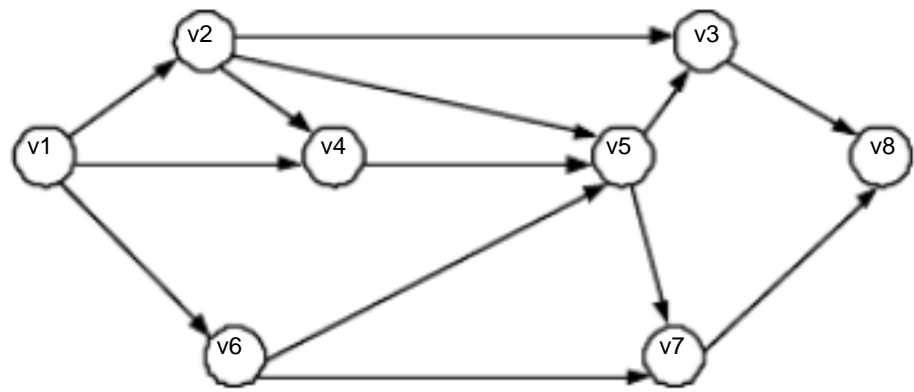
6、已知图 G 如下所示，根据 Prim 算法，构造最小生成树。（要求给出生成过程）



答案： prim算法求最小生成树如下：



7、已知有向图如下所示，请写出该图所有的拓扑序列。



答案：拓扑排序如下：

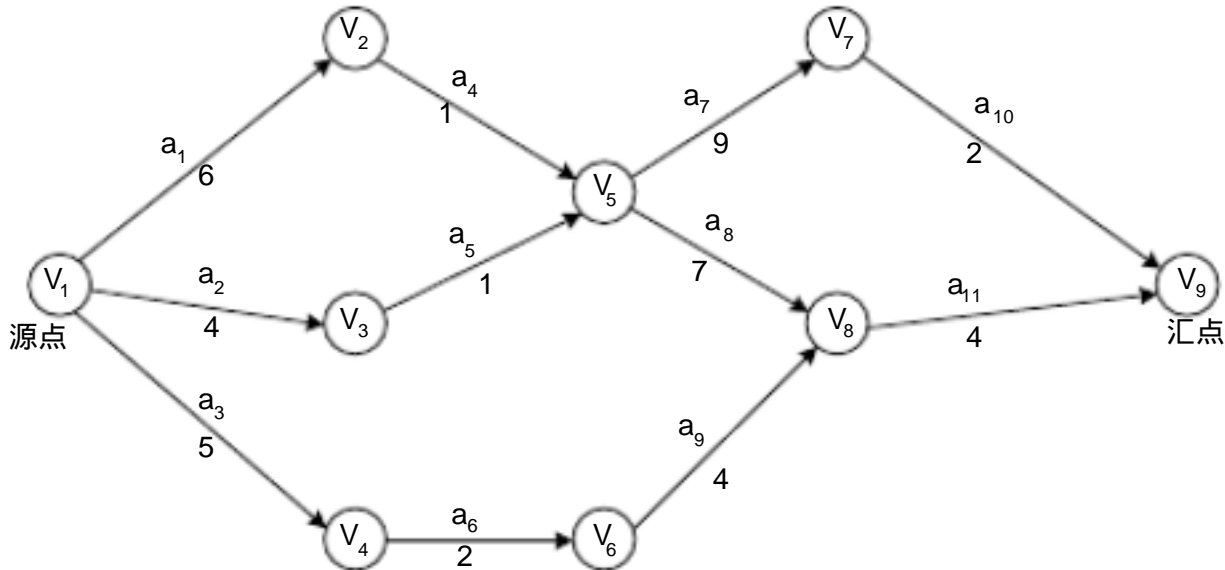
- v1, v2, v4, v6, v5, v3, v7, v8
v1, v2, v6, v4, v5, v3, v7, v8
v1, v6, v2, v4, v5, v3, v7, v8
- v1, v2, v4, v6, v5, v7, v3, v8
v1, v2, v6, v4, v5, v7, v3, v8
v1, v6, v2, v4, v5, v7, v3, v8

8、如下图所示的 AOE 网，求：

（1）求事件的最早开始时间  $ve$  和最迟开始时间  $vl$  ；

事件	1	2	3	4	5	6	7	8	9
$Ve$									
$Vl$									

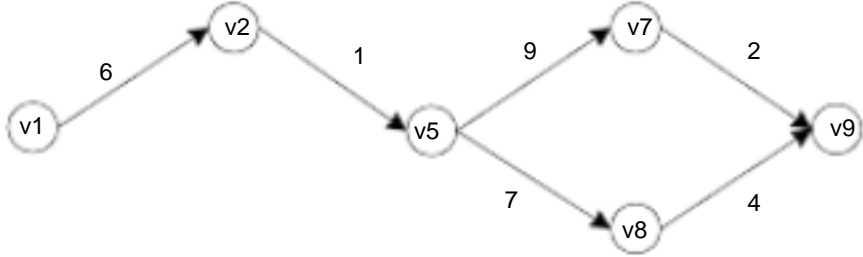
（2）求出关键路径；



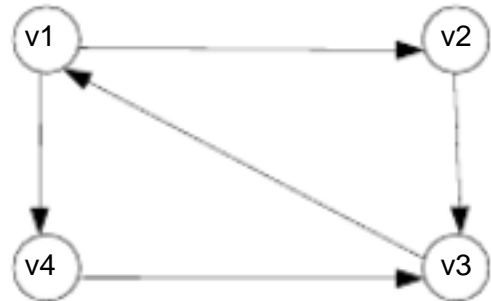
答案：(1)求  $ve$  和  $vl$

事件	1	2	3	4	5	6	7	8	9
$ve$	0	6	4	5	7	7	16	14	18
$vl$	0	6	6	8	7	10	16	14	18
	*	*			*		*	*	*

(2)关键路径



如下所示的有向图，回答下面问题：



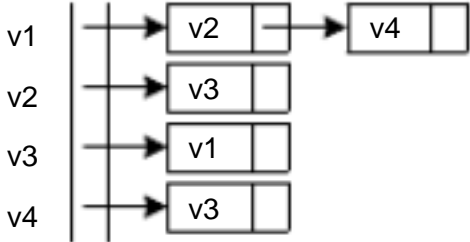
（1）该图是强连通的吗？若不是，给出强连通分量。

（2）请给出图的邻接矩阵和邻接表表示。

答案：(1) 是强连通图

(2) 邻接矩阵和邻接表为：

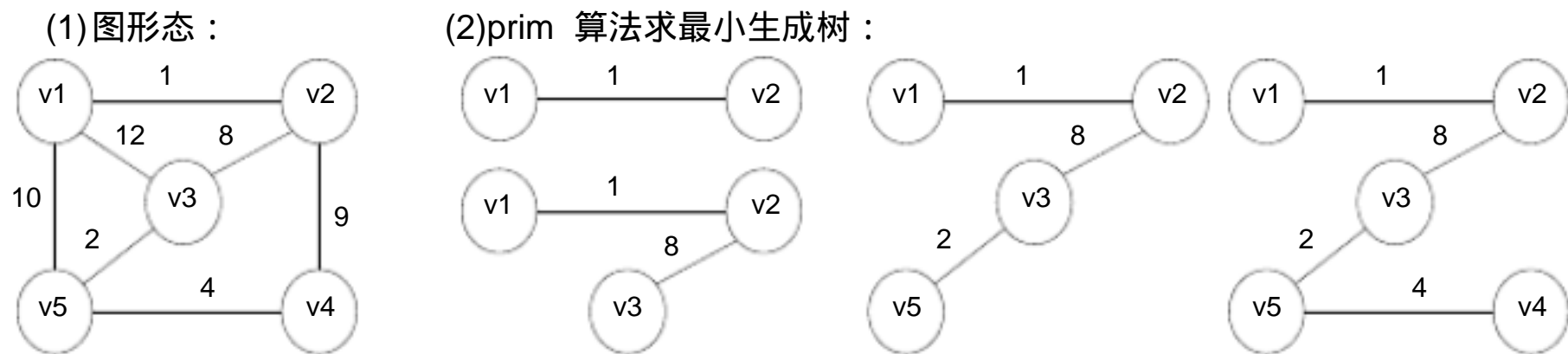
0	1	0	1
0	0	1	0
1	0	0	0
0	0	1	0



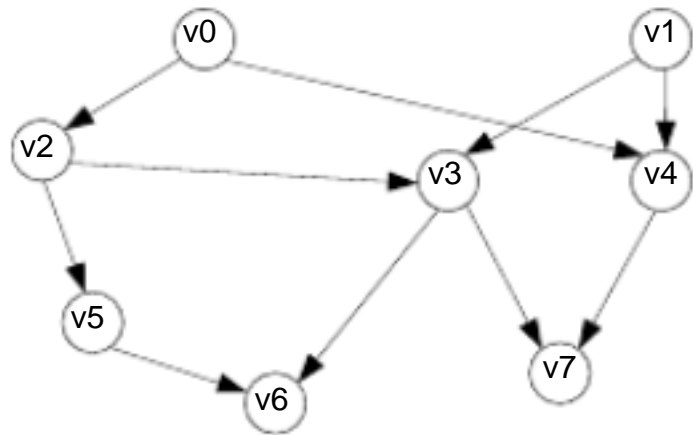
9、已知图 G 的邻接矩阵  $A = \begin{bmatrix} \infty & 1 & 12 & 6 & 10 \\ 1 & \infty & 8 & 9 & \infty \\ 12 & 8 & \infty & \infty & 2 \\ 6 & 9 & \infty & \infty & 4 \\ 10 & \infty & 2 & 4 & \infty \end{bmatrix}$ ，试画出它所表示的图 G，并根据 Prim 算法求出

图的最小生成树（给出生成过程）。

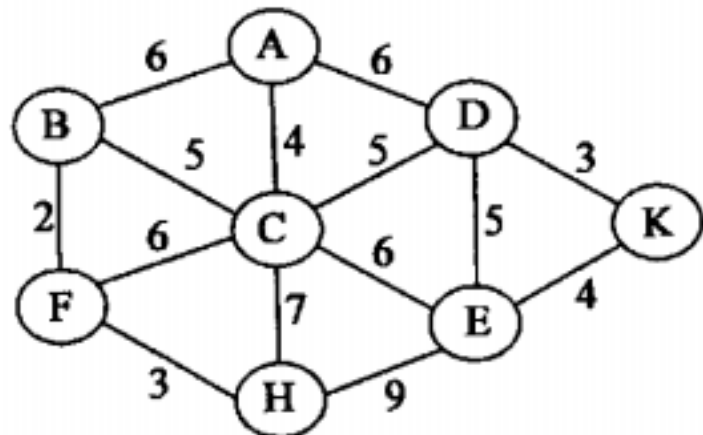
答案：



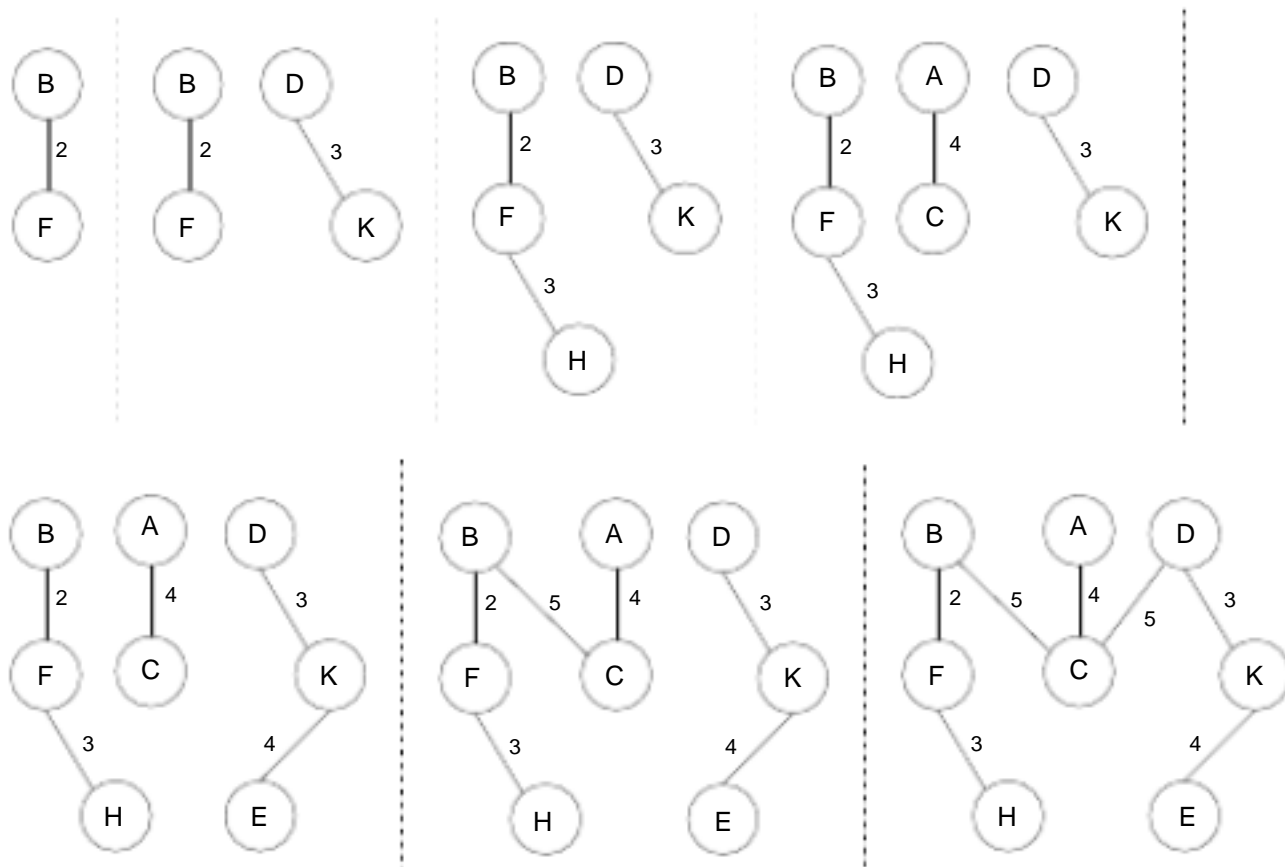
10、如下图所示的 AOV 网，写出其中三种拓扑排序序列。



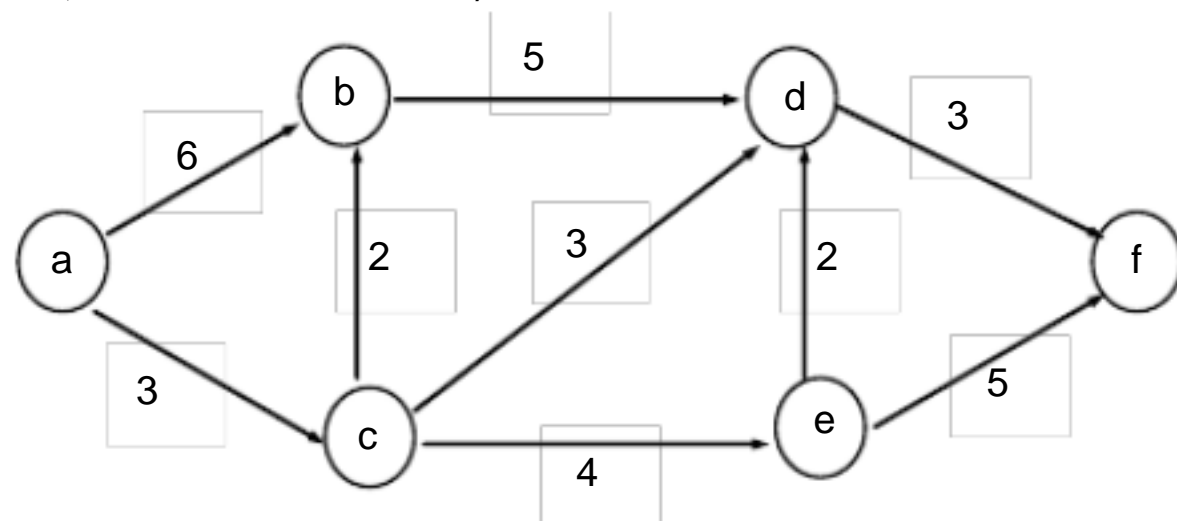
11、已知图 G 如下，根据克鲁斯卡尔算法求图 G 的一棵最小生成树。（要求给出构造过程）



答案：kruskal算法的最小生成树



12、已知图 G 如下所示，求从顶点 a 到其余各顶点的最短路径。（给出求解过程）



答案：

终点	最短路径求解过程				
b	6 (a,b)	5 (a,c,b)			
c	3 (a,c)				
d	$\infty$	6 (a,c,d)	6 (a,c,d)		
e	$\infty$	7 (a,c,e)	7 (a,c,e)	7 (a,c,e)	
f	$\infty$	$\infty$	$\infty$	9 (a,c,d,f)	9 (a,c,d,f)
vj	c	b	d	e	f
S	{a,c}	{a,c,b}	{a,c,d}	{a,c,e}	{a,c,d,f}

## 第九章 查找

### 一、选择题

- 已知一个有序表为 ( 11 , 22 , 33 , 44 , 55 , 66 , 77 , 88 , 99 ) , 则折半查找 55 需要比较 ( A ) 次。  
A. 1 B. 2 C. 3 D. 4
- 有一组关键字序列 {13,16,6,34,32,98,73,1,27} , 哈希表的表长为 13 , 哈希函数为  $H(key)=key \text{ MOD } 13$  , 冲突解决的办法为链地址法 , 请构造哈希表 ( 用图表示 ) 。
- 解决哈希冲突的主要方法有 ( )。  
A. 数字分析法、除余法、平方取中法 B. 数字分析法、除余法、线性探测法  
C. 数字分析法、线性探测法、再哈希法 D. 线性探测法、再哈希法、链地址法
- 在一棵深度为 h 的具有 n 个元素的二叉排序树中 , 查找所有元素的最长查找长度为 ( )。  
A. n B.  $\log_2 n$  C.  $(h+1)/2$  D. h
- 已知表长为 25 的哈希表 , 用除留取余法 , 按公式  $H(key)=key \text{ MOD } p$  建立哈希表 , 则 p 应取 ( ) 为宜。  
A. 23 B. 24 C. 25 D. 26
- 设哈希表长  $m=14$  , 哈希函数  $H(key)=key \text{ MOD } 11$  。表中已有 4 个结点 :  
 $addr(15)=4, addr(38)=5, addr(61)=6, addr(84)=7$  其余地址为空 , 如用二次探测再散列处理冲突 , 则关键字为 49 的地址为 ( A )。  
A. 8 B. 3 C. 5 D. 9
- 在散列查找中 , 平均查找长度主要与 ( C ) 有关。  
A. 散列表长度 B. 散列元素个数 C. 装填因子 D. 处理冲突方法
- 根据一组记录 ( 56 , 42 , 50 , 64 , 48 ) 依次插入结点生成一棵 AVL 树 , 当插入到值为 48 的结点时需要进行旋转调整。
- m 阶 B- 树中的 m 是指 ( )。  
A. 每个结点至少具有 m 棵子树 B. 每个结点最多具有 m 棵子树  
C. 分支结点中包含的关键字的个数 D. m 阶 B- 树的深度



10、一个待散列的线性表为  $k=\{18,25,63,50,42,32,9\}$  , 散列函数为  $H(k)=k \text{ MOD } 9$  , 与 18 发生冲突的元素有 ( ) 个。

- A. 1 B. 2 C. 3 D. 4

11、在对查找表的查找过程中,若被查找的数据元素不存在,则把该数据元素插到集合中,这种方式主要适合于 ( )。

- A. 静态查找表 B. 动态查找表 C. 静态查找表和动态查找表 D. 两种表都不适合

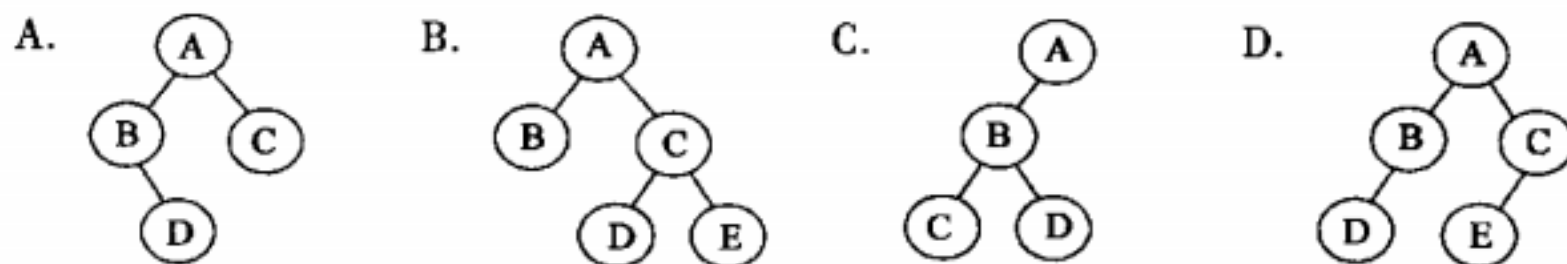
12、有一个有序表为  $\{1,3,9,12,32,41,45,62,75,77,82,95,100\}$ , 当折半查找值为 82 的结点时, ( B ) 次比较后查找成功。

- A. 1 B. 4 C. 2 D. 8

13、在各种查找方法中,平均查找承担与结点个数  $n$  无关的查找方法是 ( C )。

- A. 顺序查找 B. 折半查找 C. 哈希查找 D. 分块查找

14、下列二叉树中,不 平衡的二叉树是 ( C )。



15、对一棵二叉排序树按 ( B ) 遍历,可得到结点值从小到大的排列序列。

- A. 先序 B. 中序 C. 后序 D. 层次

16、解决散列法中出现的冲突问题常采用的方法是 ( D )。

- A. 数字分析法、除余法、平方取中法 B. 数字分析法、除余法、线性探测法  
C. 数字分析法、线性探测法、多重散列法 D. 线性探测法、多重散列法、链地址法

17、对线性表进行折半查找时,要求线性表必须 ( C )。

- A. 以顺序方式存储 B. 以链接方式存储  
C. 以顺序方式存储,且结点按关键字有序排序 D. 以链接方式存储,且结点按关键字有序排序

## 二、填空题

1、在散列函数  $H(\text{key})=\text{key} \% p$  中,  $p$  应取 \_\_\_\_\_。

2、已知有序表为 ( 12, 18, 24, 35, 47, 50, 62, 83, 90, 115, 134 ), 当用折半查找 90 时,需进行 \_\_\_\_\_ 2 \_\_\_\_\_ 次查找可确定成功。

3、具有相同函数值的关键字对哈希函数来说称为 \_\_\_\_\_。

4、在一棵二叉排序树上实施 \_\_\_\_\_ 遍历后,其关键字序列是一个有序表。

5、在散列存储中,装填因子 \_\_\_\_\_ 的值越大,则存取元素时发生冲突的可能性就越大; 值越小,则存取元素发生冲突的可能性就越小。

## 三、判断题

( × ) 1、折半查找只适用于有序表,包括有序的顺序表和链表。

( ) 2、二叉排序树的任意一棵子树中,关键字最小的结点必无左孩子,关键字最大的结点必无右孩子。

( ) 3、哈希表的查找效率主要取决于哈希表造表时所选取的哈希函数和处理冲突的方法。

( ) 4、平衡二叉树是指左右子树的高度差的绝对值不大于 1 的二叉树。

( ) 5、AVL 是一棵二叉树,其树上任一结点的平衡因子的绝对值不大于 1。

## 四、综合题

1、选取哈希函数  $H(k) = (k) \text{ MOD } 11$  用二次探测再散列处理冲突,试在 0-10 的散列地址空间中对关键字序列 ( 22,41,53,46,30,13,01,67 ) 造哈希表,并求等概率情况下查找成功时的平均查找长度。

答案: (1)表形态:

0	1	2	3	4	5	6	7	8	9	10
22	01	46	13				30	41	53	
1	1	1	2				3	1	1	

(2)ASL :  $ASL(7)=(1*5+2*1+3*1)/7=(5+2+3)/7=10/7$

2、设哈希表 HT 表长 m 为 13，哈希函数为  $H(k)=k \text{ MOD } m$ ，给定的关键值序列为 {19,14,23,10,68,20,84,27,55,11}。试求出用线性探测法解决冲突时所构造的哈希表，并求出在等概率的情况下查找成功的平均查找长度 ASL。

答案：(1)表形态：

0	1	2	3	4	5	6	7	8	9	10	11	12
	14	27	68	55		19	20	84		23	10	11
	1	2	1	2		1	1	3		1	2	2

(2)平均查找长度：  $ASL(10)=(1*5+2*4+3*1)/10=1.6$

3、设散列表容量为 7（散列地址空间 0..6），给定表（30，36，47，52，34），散列函数  $H(K)=K \text{ mod } 6$ ，采用线性探测法解决冲突，要求：

（1）构造散列表；

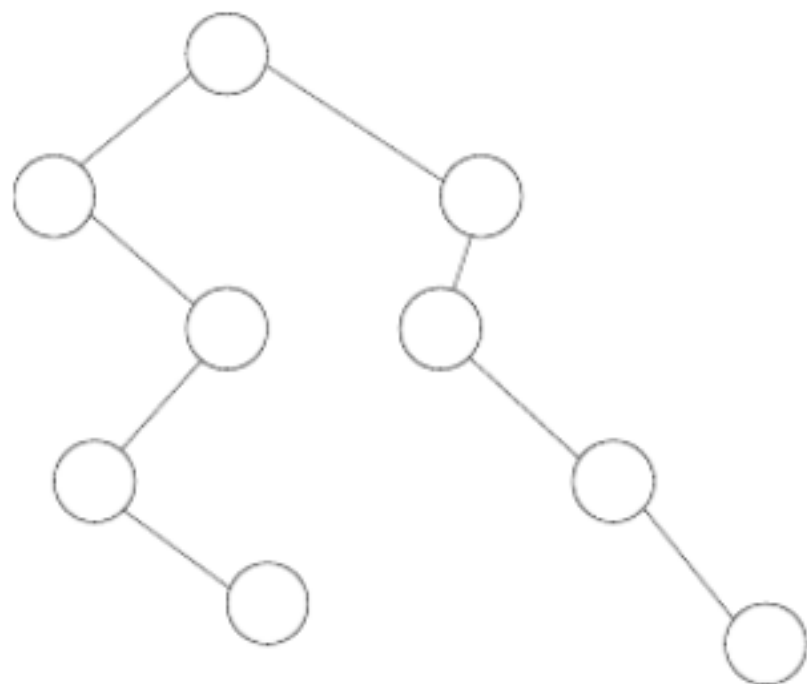
（2）求查找数 34 需要比较的次数。

答案：(1)表形态：

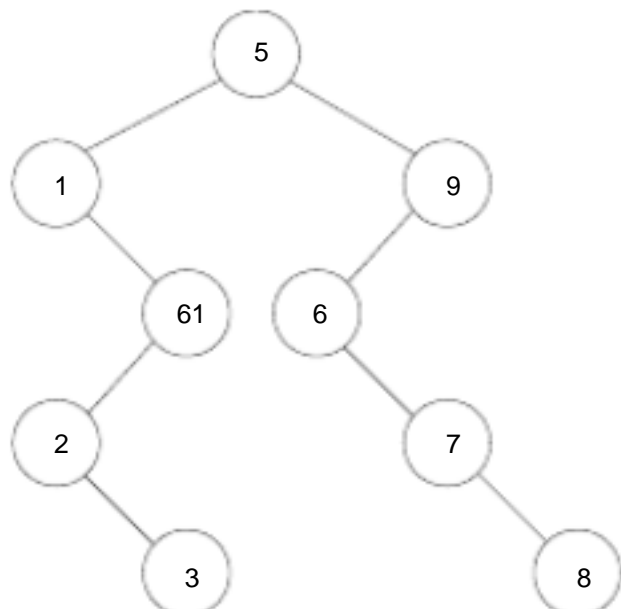
0	1	2	3	4	5	6
30	26			52	47	34
1	2			1	1	3

(2)查找 34 的比较次数： 3

4、已知下面二叉排序树的各结点的值依次为 1 - 9，请标出各结点的值。



答案：



5、若依次输入序列 {62,68,30,61,25,14,53,47,90,84} 中的元素，生成一棵二叉排序树。画出生成后的二叉排序树（不需画出生成过程）。

6、设有一组关键字 {19,1,23,14,55,20,84,27,68,11,10,77}, 采用哈希函数  $H(key)=key \text{ MOD } 13$ , 采用开放地址法的二次探测再散列方法解决冲突, 试在 0 - 18 的散列空间中对关键字序列构造哈希表, 画出哈希表, 并求其查找成功时的平均查找长度。

7、已知关键字序列 {11,2,13,26,5,18,4,9}, 设哈希表表长为 16, 哈希函数  $H(key)=key \text{ MOD } 13$ , 处理冲突的方法为线性探测法, 请给出哈希表, 并计算在等概率的条件下的平均查找长度。

8、设散列表的长度为  $m=13$ , 散列函数为  $H(k)=k \text{ MOD } m$ , 给定的关键码序列为 19, 14, 23, 1, 68, 20, 84, 27, 55, 11, 13, 7, 试写出用线性探查法解决冲突时所构造的散列表。

答案：表形态：

0	1	2	3	4	5	6	7	8	9	10	11	12
13	14	1	68	27	55	19	20	84	7	23	11	
1	1	2	1	4	3	1	1	3	3	1	1	

9、依次读入给定的整数序列 {7,16,4,8,20,9,6,18,5}, 构造一棵二叉排序树, 并计算在等概率情况下该二叉排序树的平均查找长度 ASL。(要求给出构造过程)

10、设有一组关键字 (19, 1, 23, 14, 55, 20, 84, 27, 68, 11, 10, 77), 采用哈希函数  $H(key)=key \% 13$ , 采用二次探测再散列的方法解决冲突, 试在 0-18 的散列地址空间中对关键字序列构造哈希表。

答案：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
27	1	14	55	68	84	19	20		10	23	11	77						
3	1	2	1	2	3	1	1		3	1	1	1						

## 第十章 内部排序

### 一、选择题

- 若需要在  $O(n \log_2 n)$  的时间内完成对数组的排序, 且要求排序是稳定的, 则可选的排序方法是 ( )。  
A. 快速排序 B. 堆排序 C. 归并排序 D. 直接插入排序
- 下列排序方法中 ( ) 方法是不稳定的。  
A. 冒泡排序 B. 选择排序 C. 堆排序 D. 直接插入排序
- 一个序列中有 10000 个元素, 若只想得到其中前 10 个最小元素, 则最好采用 ( ) 方法。  
A. 快速排序 B. 堆排序 C. 插入排序 D. 归并排序
- 一组待排序序列为 (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, 80 B. 84, 79, 56, 38, 40, 46  
C. 84, 79, 56, 46, 40, 38 D. 84, 56, 79, 40, 46, 38
- 快速排序方法在 ( ) 情况下最不利于发挥其长处。  
A. 要排序的数据量太大 B. 要排序的数据中有多个相同值  
C. 要排序的数据已基本有序 D. 要排序的数据个数为奇数
- 排序时扫描待排序记录序列, 顺次比较相邻的两个元素的大小, 逆序时就交换位置, 这是 ( ) 排序的基本思想。  
A. 堆排序 B. 直接插入排序 C. 快速排序 D. 冒泡排序
- 在任何情况下, 时间复杂度均为  $O(n \log n)$  的不稳定的排序方法是 ( )。  
A. 直接插入 B. 快速排序 C. 堆排序 D. 归并排序
- 如果将所有中国人按照生日来排序, 则使用 ( ) 算法最快。  
A. 归并排序 B. 希尔排序 C. 快速排序 D. 基

## 数排序

- 9、在对  $n$  个元素的序列进行排序时，堆排序所需要的附加存储空间是 ( )。
- A.  $O(\log_2 n)$       B.  $O(1)$       C.  $O(n)$       D.  $O(n \log_2 n)$
- 10、排序方法中，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，称为 ( )。
- A. 希尔排序      B. 冒泡排序      C. 插入排序      D. 选择排序
- 11、一组记录的序列为 ( 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 , 80      B. 84 , 79 , 56 , 38 , 40 , 46  
C. 84 , 79 , 56 , 46 , 40 , 38      D. 84 , 56 , 79 , 40 , 46 , 38
- 12、用某种排序方法对线性表 ( 25 , 84 , 21 , 47 , 15 , 27 , 68 , 35 , 20 ) 进行排序时，元素序列的变化情况如下：
- ? 25 , 84 , 21 , 47 , 15 , 27 , 68 , 35 , 20  
? 20 , 15 , 21 , 25 , 47 , 27 , 68 , 35 , 84  
? 15 , 20 , 21 , 25 , 35 , 27 , 47 , 68 , 84  
? 15 , 20 , 21 , 25 , 27 , 35 , 47 , 68 , 84
- 则所采用的排序方法是 ( )。
- A. 选择排序      B. 希尔排序      C. 归并排序      D. 快速排序
- 13、设有 1024 个无序的元素，希望用最快的速度挑选出其中前 5 个最大的元素，最好选用 ( )。
- A. 冒泡排序      B. 选择排序      C. 快速排序      D. 堆排序
- 14、下列排序方法中，平均时间性能为  $O(n \log n)$  且空间性能最好的是 ( )。
- A. 快速排序      B. 堆排序      C. 归并排序      D. 基数排序
- 15、希尔排序的增量序列必须是 ( )。
- A. 递增的      B. 递减的      C. 随机的      D. 非递减的

## 二、填空题

- 1、在插入和选择排序中，若初始数据基本正序，则选用 \_\_\_\_\_，若初始数据基本反序，则选用 \_\_\_\_\_。

答案：递增排列      递减排列

- 2、在插入排序、希尔排序、选择排序、快速排序、堆排序、归并排序和基数排序中，排序是不稳定的有 \_\_\_\_\_。

## 三、判断题

- 1、直接选择排序是一种稳定的排序方法。  
2、快速排序在所有排序方法中最高，而且所需附加空间也最少。  
3、直接插入排序是不稳定的排序方法。  
4、选择排序是一种不稳定的排序方法。

## 四、程序分析题

## 五、综合题

- 1、写出用直接插入排序将关键字序列 {54,23,89,48,64,50,25,90,34} 排序过程的每一趟结果。

答案：初始： 54 , 23 , 89 , 48 , 64 , 50 , 25 , 90 , 34  
1 : ( 23 , 54 ) , 89 , 48 , 64 , 50 , 25 , 90 , 34  
2 : ( 23 , 54 , 89 ) , 48 , 64 , 50 , 25 , 90 , 34  
3 : ( 23 , 48 , 54 , 89 ) , 64 , 50 , 25 , 90 , 34  
4 : ( 23 , 48 , 54 , 64 , 89 ) , 50 , 25 , 90 , 34  
5 : ( 23 , 48 , 50 , 54 , 64 , 89 ) , 25 , 90 , 34  
6 : ( 23 , 25 , 48 , 50 , 54 , 64 , 89 ) , 90 , 34

7 : ( 23 , 25 , 48 , 50 , 54 , 64 , 89 , 90 ) , 34

8 : ( 23 , 25 , 48 , 50 , 54 , 64 , 89 , 90 , 34 )

2、设待排序序列为 {10,18,4,3,6,12,1,9,15,8}

请写出希尔排序每一趟的结果。增量序列为

5 , 3 , 2 , 1。

答案：初始： 10 , 18 , 4 , 3 , 6 , 12 , 1 , 9 , 15 , 8

d=5 : 10 , 1 , 4 , 3 , 6 , 12 , 18 , 9 , 15 , 8

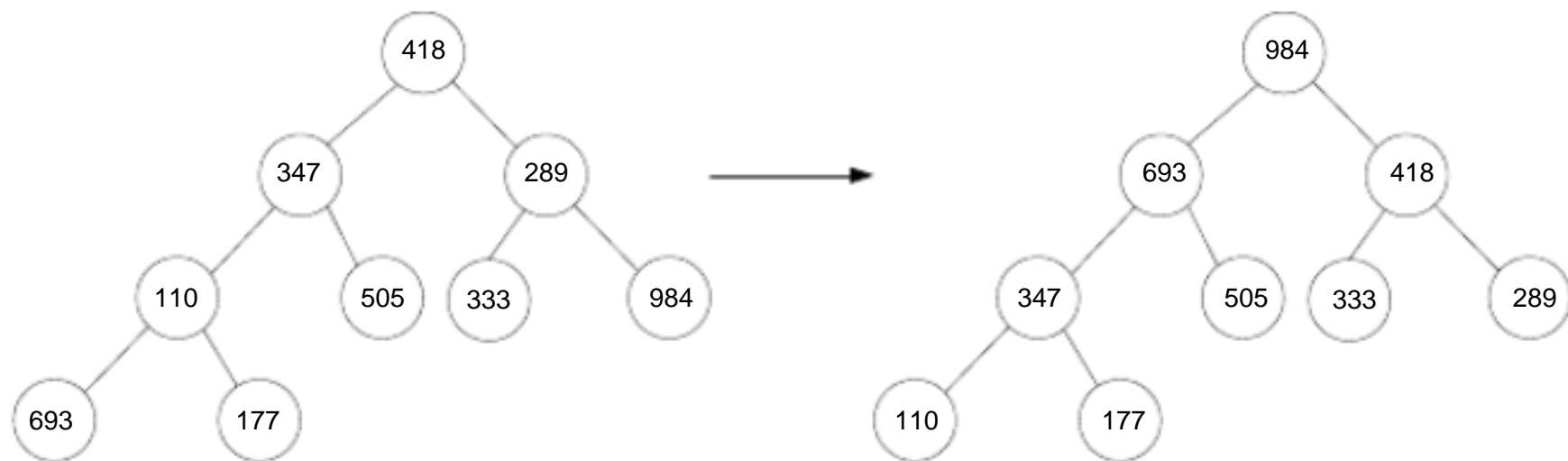
d=3 : 3 , 1 , 4 , 8 , 6 , 12 , 10 , 9 , 15 , 18

d=2 : 3 , 1 , 4 , 8 , 6 , 9 , 10 , 12 , 15 , 18

d=1 : 1 , 3 , 4 , 6 , 8 , 9 , 10 , 12 , 15 , 18

3、已知关键字序列 {418 , 347 , 289 , 110 , 505 , 333 , 984 , 693 , 177} , 按递增排序, 求初始堆 (画出初始堆的状态) 。

答案： 418 , 347 , 289 , 110 , 505 , 333 , 984 , 693 , 177



4、有一关键字序列 ( 265 , 301 , 751 , 129 , 937 , 863 , 742 , 694 , 076 , 438 ) , 写出希尔排序的每趟排序结果。(取增量为 5 , 3 , 1)

答案：

初始： 265 , 301 , 751 , 129 , 937 , 863 , 742 , 694 , 076 , 438

d=5 : 265 , 301 , 694 , 076 , 438 , 863 , 742 , 751 , 129 , 937

d=3 : 076 , 301 , 129 , 265 , 438 , 694 , 742 , 751 , 863 , 937

d=1 : 076 , 129 , 265 , 301 , 438 , 694 , 742 , 751 , 863 , 937

5、对于直接插入排序, 希尔排序, 冒泡排序, 快速排序, 直接选择排序, 堆排序和归并排序等排序方法, 分别写出：

(1) 平均时间复杂度低于  $O(n^2)$  的排序方法；

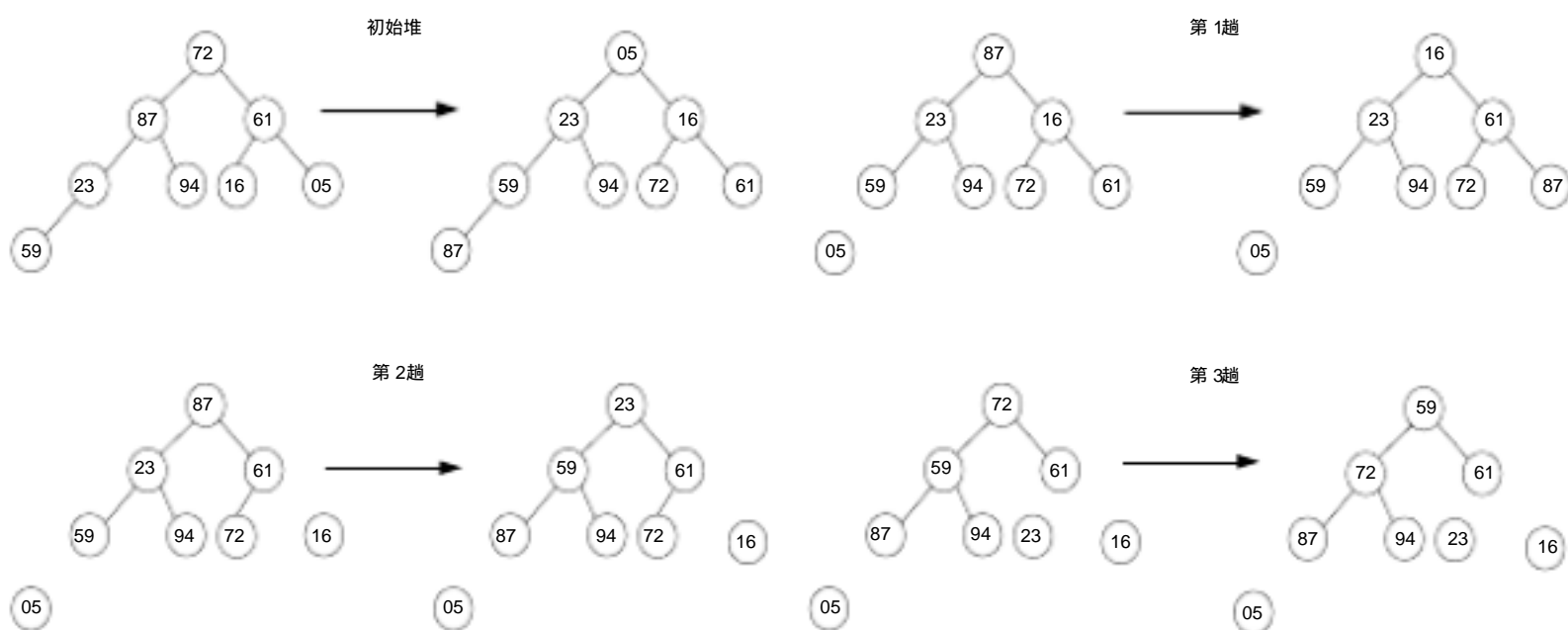
(2) 所需辅助空间最多的排序方法；

答案：(1) 希尔、快速、堆、归并

(2) 归并

6、对关键字序列 ( 72 , 87 , 61 , 23 , 94 , 16 , 05 , 58 ) 进行堆排序, 使之按关键字递减次序排列 (最小堆), 请写出排序过程中得到的初始堆和前三趟的序列状态。

答案：



## 一、填空题

1. 数据结构被形式地定义为 ( D, R ), 其中 D 是 数据元素 的有限集合, R 是 D 上的 关系 有限集合。
2. 数据结构包括数据的 逻辑结构、数据的 存储结构 和数据的 运算 这三个方面的内容。
3. 数据结构按逻辑结构可分为两大类, 它们分别是 线性结构 和 非线性结构。
4. 线性结构中元素之间存在 一对一 关系, 树形结构中元素之间存在 一对多 关系, 图形结构中元素之间存在 多对多 关系。
5. 在线性结构中, 第一个结点 没有 前驱结点, 其余每个结点有且只有 1 个前驱结点; 最后一个结点 没有 后续结点, 其余每个结点有且只有 1 个后续结点。
6. 在树形结构中, 树根结点没有 前驱 结点, 其余每个结点有且只有 1 个前驱结点; 叶子结点没有 后续 结点, 其余每个结点的后续结点数可以 任意多个。
7. 在图形结构中, 每个结点的前驱结点数和后续结点数可以 任意多个。
8. 数据的存储结构可用四种基本的存储方法表示, 它们分别是 顺序、链式、索引 和 散列。
9. 数据的运算最常用的有 5 种, 它们分别是 插入、删除、修改、查找、排序。
10. 一个算法的效率可分为 时间 效率和 空间 效率。
11. 任何一个 C 程序都由 一个主函数 和若干个被调用的其它函数组成。

## 二、单项选择题

- ( B ) 1. 非线性结构是数据元素之间存在一种 :  
A) 一对多关系      B) 多对多关系      C) 多对一关系      D) 一对一关系
- ( C ) 2. 数据结构中, 与所使用的计算机无关的是数据的          结构 ;  
A) 存储      B) 物理      C) 逻辑      D) 物理和存储
- ( C ) 3. 算法分析的目的是 :  
A) 找出数据结构的合理性      B) 研究算法中的输入和输出的关系  
C) 分析算法的效率以求改进      D) 分析算法的易懂性和文档性
- ( A ) 4. 算法分析的两个主要方面是 :  
A) 空间复杂性和时间复杂性      B) 正确性和简明性  
C) 可读性和文档性      D) 数据复杂性和程序复杂性
- ( C ) 5. 计算机算法指的是 :  
A) 计算方法      B) 排序方法      C) 解决问题的有限运算序列      D) 调度方法
- ( B ) 6. 计算机算法必须具备输入、输出和          等 5 个特性。  
A) 可行性、可移植性和可扩充性      B) 可行性、确定性和有穷性  
C) 确定性、有穷性和稳定性      D) 易读性、稳定性和安全性

## 三、简答题

1. 数据结构和数据类型两个概念之间有区别吗?

答: 简单地说, 数据结构定义了一组按某些关系结合在一起的数组元素。数据类型不仅定义了一组带结构的数据元素, 而且还在其上定义了一组操作。

2. 简述线性结构与非线性结构的不同点。

答: 线性结构反映结点间的逻辑关系是 一对一 的, 非线性结构反映结点间的逻辑关系是多对多的。

## 四、分析下面各程序段的时间复杂度

```
1.  for (i=0; i<n; i++)
      for (j=0; j<m; j++)
          A[i][j]=0;
```

答:  $O(m \cdot n)$

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

解: 因为  $x++$  共执行了  $n-1+n-2+\dots+1=n(n-1)/2$ , 所以执行时间为  $O(n^2)$

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

答:  $O(n^2)$

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

答:  $O(\log_3 n)$



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

## 第 2 章 自测卷答案

### 一、填空

1. 在顺序表中插入或删除一个元素，需要平均移动 表中一半 元素，具体移动的元素个数与 表长和该元素在表中的位置 有关。
2. 线性表中结点的集合是 有限 的，结点间的关系是 一对一 的。
3. 向一个长度为 n 的向量的第 i 个元素 (1 ≤ i ≤ n+1) 之前插入一个元素时，需向后移动 n-i+1 个元素。
4. 向一个长度为 n 的向量中删除第 i 个元素 (1 ≤ i ≤ n) 时，需向前移动 n-i 个元素。
5. 在顺序表中访问任意一结点的时间复杂度均为 O(1)，因此，顺序表也称为 随机存取 的数据结构。
6. 顺序表中逻辑上相邻的元素的物理位置 必定 相邻。单链表中逻辑上相邻的元素的物理位置 不一定 相邻。
7. 在单链表中，除了首元结点外，任一结点的存储位置由 其直接前驱结点的链域的值 指示。
8. 在 n 个结点的单链表中要删除已知结点 \*p，需找到它的 前驱结点的地址，其时间复杂度为 O(n)。

### 二、判断正误

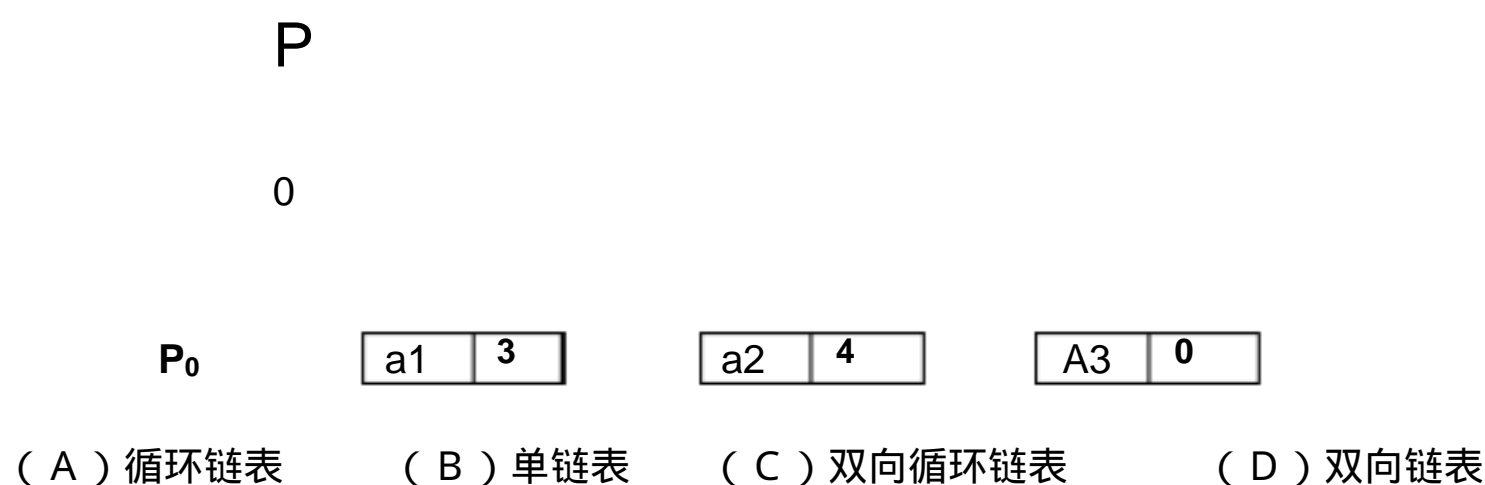
- ( × ) 1. 链表的每个结点中都恰好包含一个指针。  
答：错误。链表中的结点可含多个指针域，分别存放多个指针。例如，双向链表中的结点可以含有两个指针域，分别存放指向其直接前趋和直接后继结点的指针。
- ( × ) 2. 链表的物理存储结构具有同链表一样的顺序。 错，链表的存储结构特点是无序，而链表的示意图有序。
- ( × ) 3. 链表的删除算法很简单，因为当删除链中某个结点后，计算机会自动地将后续的各个单元向前移动。 错，链表的结点不会移动，只是指针内容改变。
- ( × ) 4. 线性表的每个结点只能是一个简单类型，而链表的每个结点可以是一个复杂类型。  
错，混淆了逻辑结构与物理结构，链表也是线性表！且即使是顺序表，也能存放记录型数据。
- ( × ) 5. 顺序表结构适宜于进行顺序存取，而链表适宜于进行随机存取。  
错，正好说反了。顺序表才适合随机存取，链表恰恰适于“顺藤摸瓜”
- ( × ) 6. 顺序存储方式的优点是存储密度大，且插入、删除运算效率高。  
错，前半正确，但后半说法错误，那是链式存储的优点。顺序存储方式插入、删除运算效率较低，在表长为 n 的顺序表中，插入和删除一个数据元素，平均需移动表长一半个数的数据元素。
- ( × ) 7. 线性表在物理存储空间中也一定是连续的。  
错，线性表有两种存储方式，顺序存储和链式存储。后者不要求连续存放。
- ( × ) 8. 线性表在顺序存储时，逻辑上相邻的元素未必在存储的物理位置次序上相邻。  
错误。线性表有两种存储方式，在顺序存储时，逻辑上相邻的元素在存储的物理位置次序上也相邻。
- ( × ) 9. 顺序存储方式只能用于存储线性结构。  
错误。顺序存储方式不仅能用于存储线性结构，还可以用来存放非线性结构，例如完全二叉树是属于非线性结构，但其最佳存储方式是顺序存储方式。（后一节介绍）
- ( × ) 10. 线性表的逻辑顺序与存储顺序总是一致的。  
错，理由同 7。链式存储就无需一致。

### 三、单项选择题

- ( C ) 1. 数据在计算机存储器内表示时，物理地址与逻辑地址相同并且是连续的，称之为：  
( A ) 存储结构 ( B ) 逻辑结构 ( C ) 顺序存储结构 ( D ) 链式存储结构
- ( B ) 2. 一个向量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



- ( A ) 3. 在  $n$  个结点的顺序表中，算法的时间复杂度是  $O(1)$  的操作是：
- ( A ) 访问第  $i$  个结点 ( $1 \leq i \leq n$ ) 和求第  $i$  个结点的直接前驱 ( $2 \leq i \leq n$ )
- ( B ) 在第  $i$  个结点后插入一个新结点 ( $1 \leq i \leq n$ )
- ( C ) 删除第  $i$  个结点 ( $1 \leq i \leq n$ )
- ( D ) 将  $n$  个结点从小到大排序
- ( B ) 4. 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   7   个元素
- ( A ) 8      ( B ) 63.5      ( C ) 63      ( D ) 7
- ( A ) 5. 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
- ( B ) 6. 链表是一种采用   链式   存储结构存储的线性表；
- ( A ) 顺序      ( B ) 链式      ( C ) 星式      ( D ) 网状
- ( D ) 7. 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   连续或不连续都可以  ：
- ( 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
- ( B ) 8. 线性表  $L$  在   需不断对  $L$  进行删除插入   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
- ( C ) 9. 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
- ( B ) 10. 设  $a_1$ 、 $a_2$ 、 $a_3$  为 3 个结点，整数  $P_0$ ，3，4 代表地址，则如下的链式存储结构称为



#### 四、简答题

1. 试比较顺序存储结构和链式存储结构的优缺点。在什么情况下用顺序表比链表好？

答：顺序存储时，相邻数据元素的存放地址也相邻（逻辑与物理统一）；要求内存中可用存储单元的地址必须是连续的。

优点：存储密度大（= 1），存储空间利用率高。缺点：插入或删除元素时不方便。

链式存储时，相邻数据元素可随意存放，但所占存储空间分两部分，一部分存放结点值，另一部分存放表示结点间关系的指针

优点：插入或删除元素时很方便，使用灵活。缺点：存储密度小（ $<1$ ），存储空间利用率低。

顺序表适宜于做查找这样的静态操作；链表宜于做插入、删除这样的动态操作。

若线性表的长度变化不大，且其主要操作是查找，则采用顺序表；

若线性表的长度变化较大，且其主要操作是插入、删除操作，则采用链表。

2. 描述以下三个概念的区别：头指针、头结点、首元结点（第一个元素结点）。在单链表中设置头结点的作用是什么？

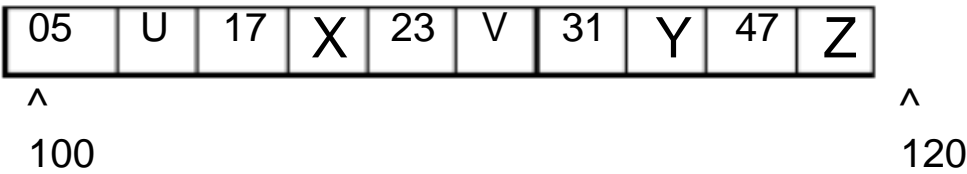
答：首元结点是指链表中存储线性表中第一个数据元素  $a_1$  的结点。为了操作方便，通常在链表的首元结点之前附设一个结点，称为头结点，该结点的 数据域 中不存储线性表的数据元素，其作用是为了对链表进行操作时，可以对空表、非空表的情况以及对首元结点进行统一处理。头指针是指向链表中第一个结点（或为头结点或为首元结点）的指针。若链表中附设头结点，则不管线性表是否为空表，头指针均不为空。否则表示空表的链表的头指针为空。这三个概念对单链表、双向链表和循环链表均适用。是否设置头结点，是不同的存储结构表示同一逻辑结构的问题。



简而言之，

头指针是指向链表中第一个结点（或为头结点或为首元结点）的指针；  
头结点是在链表的首元结点之前附设的一个结点；数据域内只放空表标志和表长等信息（内放头指针？那还得另配一个头指针！！）  
首元素结点是指链表中存储线性表中第一个数据元素  $a_1$  的结点。

五、线性表具有两种存储方式，即顺序方式和链接方式。现有一个具有五个元素的线性表  $L=\{23, 17, 47, 05, 31\}$ ，若它以链接方式存储在下列 100 ~ 119 号地址空间中，每个结点由数据（占 2 个字节）和指针（占 2 个字节）组成，如下所示：



其中指针 X，Y，Z 的值分别为多少？该线性表的首结点起始地址为多少？末结点的起始地址为多少？（10 分）

答：X= 116      Y= 0      Z= 100      首址 = 108      末址 = 112

六、编程题

1. 写出在顺序存储结构下将线性表逆转的算法，要求使用最少的附加空间。

解：输入：长度为 n 的线性表数组 A(1:n)

输出：逆转后的长度为 n 的线性表数组 A(1:n)。

C 语言描述如下（其中 ET 为数据元素的类型）：

```
invsl(n,a)
int n;
ET a[];
{int k;
ET t;
for (k=1; k<=n/2; k++)
{t=a[k-1]; a[k-1]=a[n-k]; a[n-k]=t;}
return;
}
```

2. 编写程序，将若干整数从键盘输入，以单链表形式存储起来，然后计算单链表中结点的个数（其中指针 P 指向该链表的第一个结点）。

解：编写 C 程序如下（已上机通过）：

全局变量及函数提前说明：

```
-----
#include<stdio.h>
#include<stdlib.h>
typedef struct liuyu{int data;struct liuyu*link;}test;
liuyu *p,*q,*r,*head;
int m=sizeof(test);

void main () /* 第一步，从键盘输入整数，不断添加到链表 */
{int i;
head=(test*)malloc(m); /*m=sizeof(test);*/
p=head; i=0;
while (i!=-9999)
{ printf("/ninput an integer [stop by '-9999'] :");
scanf("%d",&i);
p->data=i; /* input data is saved */
p->link=(test*)malloc(m); /*m=sizeof(test);*/
q=p;
p=p->link;
}
q->link=NULL; /* 原先用 p->link=NULL 似乎太晚！ */

p=head; i=0; /* 统计链表结点的个数并打印出来 */
while (p->link!=NULL)
```

```

{printf("%d",p->data);
p=p->link;
i++;
}
printf("\n node number=%d\n", i-1); /* 结点的个数不包括 -9999*/
}

```

### 第 3 章 栈和队列 自测卷答案

#### 一、填空题

1. 向量、栈和队列都是 线性 结构，可以在向量的 任何 位置插入和删除元素；对于栈只能在 栈顶 插入和删除元素；对于队列只能在 队尾 插入和 队首 删除元素。
2. 栈是一种特殊的线性表，允许插入和删除运算的一端称为 栈顶。不允许插入和删除运算的一端称为 栈底。
3. 队列 是被限定为只能在表的一端进行插入运算，在表的另一端进行删除运算的线性表。
4. 在一个循环队列中，队首指针指向队首元素的 前一个 位置。
5. 在具有  $n$  个单元的循环队列中，队满时共有  $n-1$  个元素。
6. 向栈中压入元素的操作是 先 移动栈顶指针，后 存入元素。
7. 从循环队列中删除一个元素时，其操作是 先 移动队首指针，后 取出元素。
8. 带表头结点的空循环双向链表的长度等于 0。

解：

head | **L=hea** | 头 结 | R=head

#### 二、判断正误（判断下列概念

的正确性，并作出简要的

说明。）

- ( ☒ ) 1. 线性表的每个结点只能是一个简单类型，而链表的每个结点可以是一个复杂类型。  
错，线性表是逻辑结构概念，可以顺序存储或链式存储，与元素数据类型无关。
- ( ☒ ) 2. 在表结构中最常用的是线性表，栈和队列不太常用。  
错，不一定吧？调用子程序或函数常用，CPU 中也用队列。
- ( ☐ ) 3. 栈是一种对所有插入、删除操作限于在表的一端进行的线性表，是一种后进先出型结构。
- ( ☐ ) 4. 对于不同的使用者，一个表结构既可以是栈，也可以是队列，也可以是线性表。  
正确，都是线性逻辑结构，栈和队列其实是特殊的线性表，对运算的定义略有不同而已。
- ( ☒ ) 5. 栈和链表是两种不同的数据结构。  
错，栈是逻辑结构的概念，是特殊线性表，而链表是存储结构概念，二者不是同类项。
- ( ☒ ) 6. 栈和队列是一种非线性数据结构。  
错，他们都是线性逻辑结构，栈和队列其实是特殊的线性表，对运算的定义略有不同而已。
- ( ☐ ) 7. 栈和队列的存储方式既可是顺序方式，也可是链接方式。
- ( ☐ ) 8. 两个栈共享一片连续内存空间时，为提高内存利用率，减少溢出机会，应把两个栈的栈底分别设在这片内存空间的两端。
- ( ☒ ) 9. 队是一种插入与删除操作分别在表的两端进行的线性表，是一种先进后出型结构。  
错，后半句不对。
- ( ☒ ) 10. 一个栈的输入序列是 12345，则栈的输出序列不可能是 12345。  
错，有可能。

#### 三、单项选择题

- ( ☐ B ) 1. 栈中元素的进出原则是  
A . 先进先出      B . 后进先出      C . 栈空则进      D . 栈满则出
- ( ☐ C ) 2. 若已知一个栈的入栈序列是 1, 2, 3, ..., n, 其输出序列为  $p_1, p_2, p_3, \dots, p_n$ , 若  $p_1=n$ , 则  $p_i$  为  
A .  $i$       B .  $n-i$       C .  $n-i+1$       D . 不确定  
解释：当  $p_1=n$ , 即  $n$  是最先出栈的，根据栈的原理， $n$  必定是最后入栈的（事实上题目已经表明了），那么输入顺序必定是 1, 2, 3, ...,  $n$ , 则出栈的序列是  $n, \dots, 3, 2, 1$ 。  
（若不要求顺序出栈，则输出序列不确定）
- ( ☐ B ) 3. 判定一个栈 ST（最多元素为  $m_0$ ）为空的条件是  
A .  $ST \rightarrow top \neq 0$       B .  $ST \rightarrow top = 0$       C .  $ST \rightarrow top \neq m_0$       D .  $ST \rightarrow$



>top=m0

( A ) 4.判定一个队列 QU ( 最多元素为 m0 ) 为满队列的条件是

- A . QU->rear - QU->front == m0      B . QU->rear - QU->front - 1 == m0  
C . QU->front == QU->rear      D . QU->front == QU->rear+1

解：队满条件是元素个数为 m0。由于约定满队时队首指针与队尾指针相差 1，所以不必再减 1 了，应当选 A。当然，更正确的答案应该取模，即：QU->front == (QU->rear+1)% m0

( D ) 5 . 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，计算队列中元素的公式为

- ( A ) r - f;      ( B ) ( n + f - r ) % n;      ( C ) n + r - f;      ( D ) ( n + r - f ) % n

6. 从供选择的答案中，选出应填入下面叙述 \_\_\_\_\_ 内的最确切的解答，把相应编号写在答卷的对应栏内。

设有 4 个数据元素 a1、a2、a3 和 a4，对他们分别进行栈操作或队操作。在进栈或进队操作时，按 a1、a2、a3、a4 次序每次进入一个元素。假设栈或队的初始状态都是空。现要进行的栈操作是进栈两次，出栈一次，再进栈两次，出栈一次；这时，第一次出栈得到的元素是 A \_\_\_\_\_，第二次出栈得到的元素是 B \_\_\_\_\_是；类似地，考虑对这四个数据元素进行的队操作是进队两次，出队一次，再进队两次，出队一次；这时，第一次出队得到的元素是 C \_\_\_\_\_，第二次出队得到的元素是 D \_\_\_\_\_。经操作后，最后在栈中或队中的元素还有 E \_\_\_\_\_个。

供选择的答案：

A ~ D :    a1      a2      a3      a4  
E :        1        2        3        0

答：ABCDE = 2, 4, 1, 2, 2

7. 从供选择的答案中，选出应填入下面叙述 \_\_\_\_\_ 内的最确切的解答，把相应编号写在答卷的对应栏内。

栈是一种线性表，它的特点是 A \_\_\_\_\_。设用一维数组 A[1,...,n] 来表示一个栈，A[n] 为栈底，用整型变量 T 指示当前栈顶位置，A[T] 为栈顶元素。往栈中推入 ( PUSH ) 一个新元素时，变量 T 的值 B \_\_\_\_\_；从栈中弹出 ( POP ) 一个元素时，变量 T 的值 C \_\_\_\_\_。设栈空时，有输入序列 a, b, c，经过 PUSH, POP, PUSH, PUSH, POP 操作后，从栈中弹出的元素的序列是 D \_\_\_\_\_，变量 T 的值是 E \_\_\_\_\_。

供选择的答案：

A :        先进先出      后进先出      进优于出      出优于进      随机进出  
B, C :     加 1        减 1        不变            清 0        加 2        减 2  
D :        a,b        b,c        c,a            b,a        c,b        a,c  
E :        n+1        n+2        n            n-1        n-2

答案：ABCDE=2, 2, 1, 6, 4

注意，向地址的高端生长，称为向上生成堆栈；向地址低端生长叫向下生成堆栈，本题中底部为 n，向地址的低端递减生成，称为向下生成堆栈。

8. 从供选择的答案中，选出应填入下面叙述 \_\_\_\_\_ 内的最确切的解答，把相应编号写在答卷的对应栏内。

在做进栈运算时，应先判别栈是否 A \_\_\_\_\_；在做退栈运算时，应先判别栈是否 B \_\_\_\_\_。当栈中元素为 n 个，做进栈运算时发生上溢，则说明该栈的最大容量为 C \_\_\_\_\_。为了增加内存空间的利用率和减少溢出的可能性，由两个栈共享一片连续的内存空间时，应将两栈的 D \_\_\_\_\_分别设在这片内存空间的两端，这样，只有当 E \_\_\_\_\_时，才产生上溢。

供选择的答案：

A, B :    空            满            上溢            下溢  
C :        n-1        n            n+1            n/2  
D :        长度        深度        栈顶            栈底  
E :    两个栈的栈顶同时到达栈空间的中心点      其中一个栈的栈顶到达栈空间的中心点  
      两个栈的栈顶在达栈空间的某一位置相遇      两个栈均不空，且一个栈的栈顶到达另一个栈的

栈底

答案：ABCDE = 2, 1, 2, 4, 3

#### 四、简答题

1.说明线性表、栈与队的异同点。

刘答：相同点：都是线性结构，都是逻辑结构的概念。都可以用顺序存储或链表存储；栈和队列是两种特殊的线性表，即受限的线性表，只是对插入、删除运算加以限制。

不同点：运算规则不同，线性表为随机存取，而栈是只允许在一端进行插入、删除运算，因而是后进先出表

**LIFO**；队列是只允许在一端进行插入、另一端进行删除运算，因而是先进先出表 **FIFO**。

用途不同，堆栈用于子程调用和保护现场，队列用于多道作业处理、指令寄存及其他运算等等。

2.设有编号为 1, 2, 3, 4 的四辆列车，顺序进入一个栈式结构的车站，具体写出这四辆列车开出车站的所有可能的顺序。

刘答：至少有 14 种。

全进之后再出情况，只有 1 种：4, 3, 2, 1

进 3 个之后再出的情况，有 3 种：3,4,2,1 3,2,4,1 3,2,1,4

进 2 个之后再出的情况，有 5 种：2,4,3,1 2,3,4,1 2,1,3,4 2,1,4,3 2,1,3,4

进 1 个之后再出的情况，有 5 种：1,4,3,2 1,3,2,4 1,3,4,2 1,2,3,4 1,2,4,3

3.顺序队的“假溢出”是怎样产生的？如何知道循环队列是空还是满？

答：一般的一维数组队列的尾指针已经到了数组的上界，不能再有入队操作，但其实数组中还有空位置，这就叫“假溢出”。

采用循环队列是解决假溢出的途径。

另外，解决队满队空的办法有三：

设置一个布尔变量以区别队满还是队空；

浪费一个元素的空间，用于区别队满还是队空。

使用一个计数器记录队列中元素个数（即队列长度）。

我们常采用法，即队头指针、队尾指针中有一个指向实元素，而另一个指向空闲元素。

判断循环队列队空标志是： $f = rear$  队满标志是： $f = (r+1) \% N$

4.设循环队列的容量为 40（序号从 0 到 39），现经过一系列的入队和出队运算后，有

$front = 11$ ， $rear = 19$ ； $front = 19$ ， $rear = 11$ ；问在这两种情况下，循环队列中各有元素多少个？

答：用 队列长度计算公式： $(N + r - f) \% N$

$L = (40 + 19 - 11) \% 40 = 8$

$L = (40 + 11 - 19) \% 40 = 32$

## 第 4~5 章 串和数组 自测卷答案

### 一、填空题（每空 1 分，共 20 分）

1. 不包含任何字符（长度为 0）的串 称为空串；由一个或多个空格（仅由空格符）组成的串 称为空白串。

（对应严题集 4.1，简答题：简述空串和空格串的区别）

2. 设  $S = "A:/document/Mary.doc"$ ，则  $strlen(s) = 20$ ，“的字符定位的位置为 3”。

4. 子串的定位运算称为串的模式匹配；被匹配的主串 称为目标串，子串 称为模式。

5. 设目标  $T = "abccdcddccbaa"$ ，模式  $P = "cdcc"$ ，则第 6 次匹配成功。

6. 若  $n$  为主串长， $m$  为子串长，则串的古典（朴素）匹配算法最坏的情况下需要比较字符的总次数为  $(n-m+1)*m$ 。

7. 假设有二维数组  $A_{6 \times 8}$ ，每个元素用相邻的 6 个字节存储，存储器按字节编址。已知  $A$  的起始存储位置（基地址）为 1000，则数组  $A$  的体积（存储量）为 288 B；末尾元素  $A_{57}$  的第一个字节地址为 1282；若按行存储时，元素  $A_{14}$  的第一个字节地址为  $(8+4) \times 6 + 1000 = 1072$ ；若按列存储时，元素  $A_{47}$  的第一个字节地址为  $(6 \times 7 + 4) \times 6 + 1000 = 1276$ 。

（注：数组是从 0 行 0 列还是从 1 行 1 列计算起呢？由末单元为  $A_{57}$  可知，是从 0 行 0 列开始！）

8. 设数组  $a[1, 60, 1, 70]$  的基地址为 2048，每个元素占 2 个存储单元，若以列序为主序顺序存储，则元素  $a[32, 58]$  的存储地址为 8950。

答：不考虑 0 行 0 列，利用列优先公式： $LOC(a_{ij}) = LOC(a_{c_1, c_2}) + [(j - c_2) * (d_1 - c_1 + 1) + i - c_1] * L$

得： $LOC(a_{32, 58}) = 2048 + [(58 - 1) * (60 - 1 + 1) + 32 - 1] * 2 = 8950$

9. 三元素组表中的每个结点对应于稀疏矩阵的一个非零元素，它包含有三个数据项，分别表示该元素的 行下标、列下标 和 元素值。

10. 求下列广义表操作的结果：

- (1) GetHead【((a,b),(c,d))】=== (a, b) ; // 头元素不必加括号  
 (2) GetHead【GetTail【((a,b),(c,d))】】=== (c,d) ;  
 (3) GetHead【GetTail【GetHead【((a,b),(c,d))】】】=== b ;  
 (4) GetTail【GetHead【GetTail【((a,b),(c,d))】】】=== (d) ;

## 二、单选题（每小题 1 分，共 15 分）

( B ) 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

( B ) 2. 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

( D ) 3. 设串 s1=?ABCDEFG?，s2=?PQRST?，函数 con(x,y) 返回 x 和 y 串的连接串，subs(s, i, j) 返回串 s 的从序号 i 开始的 j 个字符组成的子串，len(s) 返回串 s 的长度，则 con(subs(s1, 2, len(s2)), subs(s1, len(s2), 2)) 的结果串是：

- A . BCDEF                      B . BCDEFG                      C . BCPQRST                      D . BCDEFEF

解：con(x,y) 返回 x 和 y 串的连接串，即 con(x,y) = ' ABCDEFGPQRST '；

subs(s, i, j) 返回串 s 的从序号 i 开始的 j 个字符组成的子串，则

subs(s1, 2, len(s2)) = subs(s1, 2, 5)=? BCDEF?； subs(s1, len(s2), 2) = subs(s1, 5, 2)=? EF?；

所以 con(subs(s1, 2, len(s2)), subs(s1, len(s2), 2)) = con(? BCDEF?, ? EF?)之连接，即 BCDEFEF

( A ) 4. 假设有 60 行 70 列的二维数组 a[1, 60, 1, 70] 以列序为主序顺序存储，其基地址为 10000，每个元素占 2 个存储单元，那么第 32 行第 58 列的元素 a[32,58] 的存储地址为         。（无第 0 行第 0 列元素）

- A . 16902                      B . 16904                      C . 14454                      D . 答案 A, B, C 均不对

答：此题与填空题第 8 小题相似。（57 列 × 60 行 + 31 行）× 2 字节 + 10000=16902

( B ) 5. 设矩阵 A 是一个对称矩阵，为了节省存储，将其下三角部分（如下图所示）按行序存放在一维数组 B[ 1, n(n-1)/2 ] 中，对下三角部分中任一元素 a<sub>ij</sub>(i ≤ j)，在一维数组 B 中下标 k 的值是：

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

解：注意 B 的下标要求从 1 开始。  
 先用第一个元素去套用，可能有 B 和 C；  
 再用第二个元素去套用 B 和 C，B=2 而 C=3（不符）；  
 所以选 B

$$A = \begin{bmatrix} a_{1,1} & & & & \\ a_{2,1} & a_{2,2} & & & \\ \dots & & \dots & & \\ a_{n,1} & a_{n,2} & \dots & \dots & a_{n,n} \end{bmatrix}$$

6. 从供选择的答案中，选出应填入下面叙述          ？          内的最确切的解答，把相应编号写在答卷的对应栏内。

有一个二维数组 A，行下标的范围是 0 到 8，列下标的范围是 1 到 5，每个数组元素用相邻的 4 个字节存储。存储器按字节编址。假设存储数组元素 A[0,1] 的第一个字节的地址是 0。  
 存储数组 A 的最后一个元素的第一个字节的地址是     A    。若按行存储，则 A[3,5] 和 A[5,3] 的第一个字节的地址分别是     B     和     C    。若按列存储，则 A[7,1] 和 A[2,4] 的第一个字节的地址分别是     D     和     E    。

供选择的答案：

A ~ E:    28                      44                      76                      92                      108  
           116                      132                      176                      184                      188

答案：ABCDE=8, 3, 5, 1, 6



7. 有一个二维数组 A，行下标的范围是 1 到 6，列下标的范围是 0 到 7，每个数组元素用相邻的 6 个字节存储，存储器按字节编址。那么，这个数组的体积是 A 个字节。假设存储数组元素 A[1,0] 的第一个字节的地址是 0，则存储数组 A 的最后一个元素的第一个字节的地址是 B。若按行存储，则 A[2,4] 的第一个字节的地址是 C。若按列存储，则 A[5,7] 的第一个字节的地址是 D。

供选择的答案

A ~ D : 12      66      72      96      114      120  
156      234      276      282      ( 11 ) 283      ( 12 ) 288

答案：ABCD=12, 10, 3, 9

## 第 6 章 树和二叉树 自测卷解答

一、下面是有关二叉树的叙述，请判断正误（每小题 1 分，共 10 分）

- ( ) 1. 若二叉树用二叉链表作存储结构，则在  $n$  个结点的二叉树链表中只有  $n-1$  个非空指针域。
- ( × ) 2. 二叉树中每个结点的两棵子树的高度差等于 1。
- ( ) 3. 二叉树中每个结点的两棵子树是有序的。
- ( × ) 4. 二叉树中每个结点有两棵非空子树或有两棵空子树。
- ( × ) 5. 二叉树中每个结点的关键字值大于其左非空子树（若存在的话）所有结点的关键字值，且小于其右非空子树（若存在的话）所有结点的关键字值。（应当是二叉排序树的特点）
- ( × ) 6. 二叉树中所有结点个数是  $2^{k-1}-1$ ，其中  $k$  是树的深度。（应  $2^k-1$ ）
- ( × ) 7. 二叉树中所有结点，如果不存在非空左子树，则不存在非空右子树。
- ( × ) 8. 对于一棵非空二叉树，它的根结点作为第一层，则它的第  $i$  层上最多能有  $2^i-1$  个结点。（应  $2^{i-1}$ ）
- ( ) 9. 用二叉链表法（link-rlink）存储包含  $n$  个结点的二叉树，结点的  $2n$  个指针区域中有  $n+1$  个为空指针。  
（正确。用二叉链表存储包含  $n$  个结点的二叉树，结点共有  $2n$  个链域。由于二叉树中，除根结点外，每一个结点有且仅有一个双亲，所以只有  $n-1$  个结点的链域存放指向非空子女结点的指针，还有  $n+1$  个空指针。）即有后继链接的指针仅  $n-1$  个。
- ( ) 10. 具有 12 个结点的完全二叉树有 5 个度为 2 的结点。  
最快方法：用叶子数 =  $\lceil n/2 \rceil = 6$ ，再求  $n_2 = n_0 - 1 = 5$

## 二、填空（每空 1 分，共 15 分）

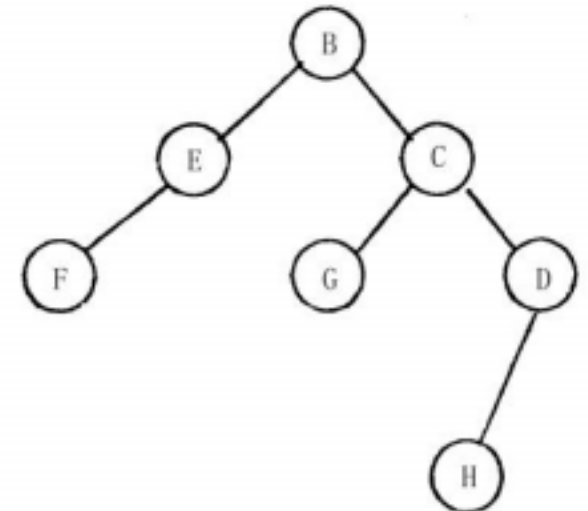
1. 由 3 个结点所构成的二叉树有 5 种形态。
2. 一棵深度为 6 的满二叉树有  $n_1+n_2=0+n_2=n_0-1=31$  个分支结点和  $2^{6-1}=32$  个叶子。  
注：满二叉树没有度为 1 的结点，所以分支结点数就是二度结点数。
3. 一棵具有 2 5 7 个结点的完全二叉树，它的深度为 9。  
（注：用  $\lceil \log_2(n) \rceil + 1 = \lceil 8.xx \rceil + 1 = 9$ ）
4. 设一棵完全二叉树有 700 个结点，则共有 350 个叶子结点。  
答：最快方法：用叶子数 =  $\lceil n/2 \rceil = 350$
5. 设一棵完全二叉树具有 1000 个结点，则此完全二叉树有 500 个叶子结点，有 499 个度为 2 的结点，有 1 个结点只有非空左子树，有 0 个结点只有非空右子树。  
答：最快方法：用叶子数 =  $\lceil n/2 \rceil = 500$ ， $n_2 = n_0 - 1 = 499$ 。另外，最后一结点为  $2i$  属于左叶子，右叶子是空的，所以有 1 个非空左子树。完全二叉树的特点决定不可能有左空右不空的情况，所以非空右子树数 = 0。
6. 一棵含有  $n$  个结点的  $k$  叉树，可能达到的最大深度为  $n$ ，最小深度为 2。  
答：当  $k=1$  (单叉树) 时应该最深，深度 =  $n$  (层)；当  $k=n-1$  ( $n-1$  叉树) 时应该最浅，深度 = 2 (层)，但不包括  $n=0$  或 1 时的特例情况。教材答案是“完全  $k$  叉树”，未定量。)

7. 二叉树的基本组成部分是：根（ D ）、左子树（ L ）和右子树（ R ）。因而二叉树的遍历次序有六种。最常用的是三种：前序法（即按 NLR 次序），后序法（即按 LRD 次序）和中序法（也称对称序法，即按 LNR 次序）。这三种方法相互之间有关联。若已知一棵二叉树的前序序列是 **BEFCGDH**，中序序列是 **FEBGCHD**，则它的后序序列必是 **FEGHDCB**。

解：法 1：先由已知条件画图，再后序遍历得到结果；

法 2：不画图也能快速得出后序序列，只要找到根的位置特征。由前序先确定 root，由中序先确定左子树。例如，前序遍历 **BEFCGDH** 中，根结点在最前面，是 **B**；则后序遍历中 **B** 一定在最后面。

法 3：递归计算。如 **B** 在前序序列中第一，中序中在中间（可知左右子树上有哪些元素），则在后序中必为最后。如法对 **B** 的左右子树同样处理，则问题得解。

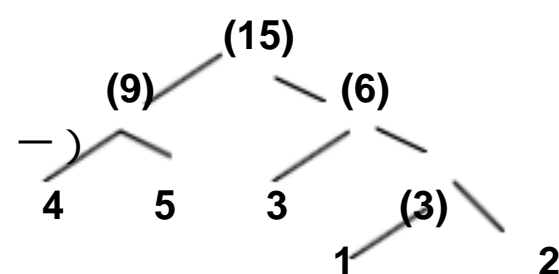


8. 中序遍历的递归算法平均空间复杂度为  $O(n)$ 。

答：即递归最大嵌套层数，即栈的占用单元数。精确值应为树的深度  $k+1$ ，包括叶子的空域也递归了一次。

9. 用 5 个权值 {3, 2, 4, 5, 1} 构造的哈夫曼（ Huffman ）树的带权路径长度是 33。

解：先构造哈夫曼树，得到各叶子的路径长度之后便可求出  $WPL = (4 + 5 + 3) \times 2 + (1 + 2) \times 3 = 33$



（注：两个合并值先后不同会导致编码不同，即哈夫曼编码不唯一）

（注：合并值应排在叶子值之后）

（注：原题为选择题：A . 32                      B . 33                      C . 34                      D . 15）

### 三、单项选择题（每小题 1 分，共 11 分）

（ C ） 1. 不含任何结点的空树           。

（ A ）是一棵树；

（ B ）是一棵二叉树；

（ C ）是一棵树也是一棵二叉树；

（ D ）既不是树也不是二叉树

答：以前的标答是 B，因为那时树的定义是  $n \geq 1$

（ C ） 2. 二叉树是非线性数据结构，所以           。

（ A ）它不能用顺序存储结构存储；

（ B ）它不能用链式存储结构存储；

（ C ）顺序存储结构和链式存储结构都能存储

；（ D ）顺序存储结构和链式存储结构都不能使用

（ C ） 3. 具有  $n(n>0)$  个结点的完全二叉树的深度为           。

（ A ）  $\lceil \log_2(n) \rceil$

（ B ）  $\lfloor \log_2(n) \rfloor$

（ C ）  $\lfloor \log_2(n) \rfloor + 1$

（ D ）  $\lceil \log_2(n) + 1 \rceil$

注 1：  $\lceil x \rceil$  表示不小于  $x$  的最小整数；  $\lfloor x \rfloor$  表示不大于  $x$  的最大整数，它们与  $[]$  含义不同！

注 2：选（ A ）是错误的。例如当  $n$  为 2 的整数幂时就会少算一层。似乎  $\lfloor \log_2(n) \rfloor + 1$  是对的？

（ A ） 4. 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

5. 从供选择的答案中，选出应填入下面叙述        ？        内的最确切的解答，把相应编号写在答卷的对应栏内。

树是结点的有限集合，它 A 根结点，记为 T。其余的结点分成为 m (m ≥ 0) 个 B 的集合 T<sub>1</sub>, T<sub>2</sub>, ..., T<sub>m</sub>，每个集合又都是树，此时结点 T 称为 T<sub>i</sub> 的父结点， T<sub>i</sub> 称为 T 的子结点（ $1 \leq i \leq m$ ）。一个结点的子结点个数为该结点的 C。

供选择的答案



A : 有 0 个或 1 个      有 0 个或多个      有且只有 1 个      有 1 个或 1 个以上  
 B: 互不相交      允许相交      允许叶结点相交      允许树枝结点相交  
 C : 权      维数      次数 ( 或度 )      序  
 答案 : **ABC = 1, 1, 3**

6. 从供选择的答案中, 选出应填入下面叙述 \_\_\_\_\_ ? \_\_\_\_\_ 内的最确切的解答, 把相应编号写在答卷的对应栏内。

二叉树 A。在完全的二叉树中, 若一个结点没有 B, 则它必定是叶结点。每棵树都能惟一地转换成与它对应的二叉树。由树转换成的二叉树里, 一个结点 N 的左子女是 N 在原树里对应结点的 C, 而 N 的右子女是它在原树里对应结点的 D。

供选择的答案

A : 是特殊的树      不是树的特殊形式      是两棵树的总称      有是只有二个根结点的树形结构  
 B: 左子结点      右子结点      左子结点或者没有右子结点      兄弟  
 C ~ D : 最左子结点      最右子结点      最邻近的右兄弟      最邻近的左兄弟

答案 : A= 最左的兄弟      B= 最右的兄弟      C= 最左的兄弟      D = 最右的兄弟

答案 : ABCDE = 2, 1, 1, 3

#### 四、简答题 ( 每小题 4 分, 共 20 分 )

1. 一棵度为 2 的树与一棵二叉树有何区别 ?

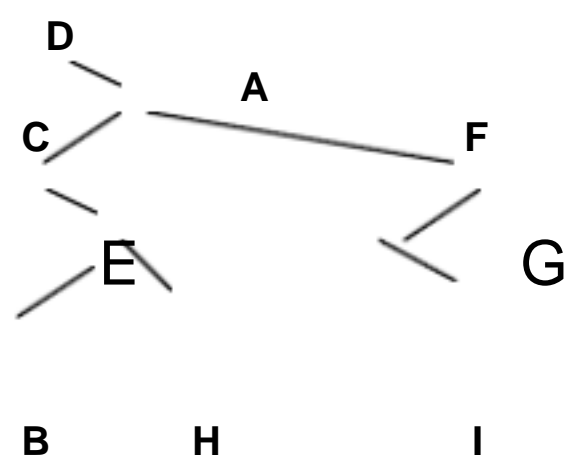
答 : 度为 2 的树从形式上看与二叉树很相似, 但它的子树是无序的, 而二叉树是有序的。即, 在一般树中若某结点只有一个孩子, 就无需区分其左右次序, 而在二叉树中即使是一个孩子也有左右之分。

2. 给定二叉树的两种遍历序列, 分别是 :

前序遍历序列 : D, A, C, E, B, H, F, G, I ;      中序遍历序列 : D, C, B, E, H, A, G, I, F,

试画出二叉树 B, 并简述由任意二叉树 B 的前序遍历序列和中序遍历序列求二叉树 B 的思想方法。

解 : 方法是 : 由前序先确定 root, 由中序可确定 root 的左、右子树。然后由其左子树的元素集合和右子树的集合对应前序遍历序列中的元素集合, 可继续确定 root 的左右孩子。将他们分别作为新的 root, 不断递归, 则所有元素都将被唯一确定, 问题得解。



下面是按先序遍历的思路建立二叉树的两种方法

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct node {
    char data;
    struct node *lchild, *rchild;
} Bitree;
void creatree1( Bitree *&bt)
{
    char ch;
    if (ch== ? ,) bt=NULL
    else
```

```

    { bt=(Bitree*)malloc(sizeof(Bitree));
      bt->data=ch;
      creatree1(bt->lchild);
      creatree2(bt->rchild);
    }
}

```

**Bitree \*creatree2( )**

```

{  char ch; Bitree *&bt
   if (ch== ? ,)  bt=NULL
   else
       { bt=(Bitree*)malloc(sizeof(Bitree));
         bt->data=ch;
         bt->lchild= creatree2();
         bt->rchild= creatree2();
       }
   return bt;
}

```

## 第 6 章 树和二叉树 自测卷解答

一、下面是有关二叉树的叙述，请判断正误（每小题 1 分，共 10 分）

- ( ) 1. 若二叉树用二叉链表作存储结构，则在  $n$  个结点的二叉树链表中只有  $n-1$  个非空指针域。
- ( × ) 2. 二叉树中每个结点的两棵子树的高度差等于 1。
- ( ) 3. 二叉树中每个结点的两棵子树是有序的。
- ( × ) 4. 二叉树中每个结点有两棵非空子树或有两棵空子树。
- ( × ) 5. 二叉树中每个结点的关键字值大于其左非空子树（若存在的话）所有结点的关键字值，且小于其右非空子树（若存在的话）所有结点的关键字值。（应当是二叉排序树的特征点）
- ( × ) 6. 二叉树中所有结点个数是  $2^{k-1}-1$ ，其中  $k$  是树的深度。（应  $2^k-1$ ）
- ( × ) 7. 二叉树中所有结点，如果不存在非空左子树，则不存在非空右子树。
- ( × ) 8. 对于一棵非空二叉树，它的根结点作为第一层，则它的第  $i$  层上最多能有  $2^i-1$  个结点。（应  $2^{i-1}$ ）
- ( ) 9. 用二叉链表法（link-rlink）存储包含  $n$  个结点的二叉树，结点的  $2n$  个指针区域中有  $n+1$  个为空指针。  
（正确。用二叉链表存储包含  $n$  个结点的二叉树，结点共有  $2n$  个链域。由于二叉树中，除根结点外，每一个结点有且仅有一个双亲，所以只有  $n-1$  个结点的链域存放指向非空子女结点的指针，还有  $n+1$  个空指针。）即有后继链接的指针仅  $n-1$  个。
- ( ) 10. 具有 12 个结点的完全二叉树有 5 个度为 2 的结点。  
最快方法：用叶子数 =  $\lfloor n/2 \rfloor = 6$ ，再求  $n_2 = n_0 - 1 = 5$

二、填空（每空 1 分，共 15 分）

1. 由 3 个结点所构成的二叉树有 5 种形态。
2. 一棵深度为 6 的满二叉树有  $n_1+n_2=0+n_2=n_0-1=31$  个分支结点和  $2^{6-1}=32$  个叶子。  
注：满二叉树没有度为 1 的结点，所以分支结点数就是二度结点数。
3. 一棵具有 2 5 7 个结点的完全二叉树，它的深度为 9。  
（注：用  $\lfloor \log_2(n) \rfloor + 1 = \lfloor 8.xx \rfloor + 1 = 9$ ）
4. 设一棵完全二叉树有 700 个结点，则共有 350 个叶子结点。  
答：最快方法：用叶子数 =  $\lfloor n/2 \rfloor = 350$
5. 设一棵完全二叉树具有 1000 个结点，则此完全二叉树有 500 个叶子结点，有 499 个度为 2 的结点，有 1 个结点只有非空左子树，有 0 个结点只有非空右子树。

答：最快方法：用叶子数 =  $\lfloor n/2 \rfloor = 500$ ， $n_2 = n_0 - 1 = 499$ 。另外，最后一结点为  $2i$  属于左叶子，右叶子是空的，所以有 1 个非空左子树。完全二叉树的特点决定不可能有左空右不空的情况，所以非空右子树数 = 0。

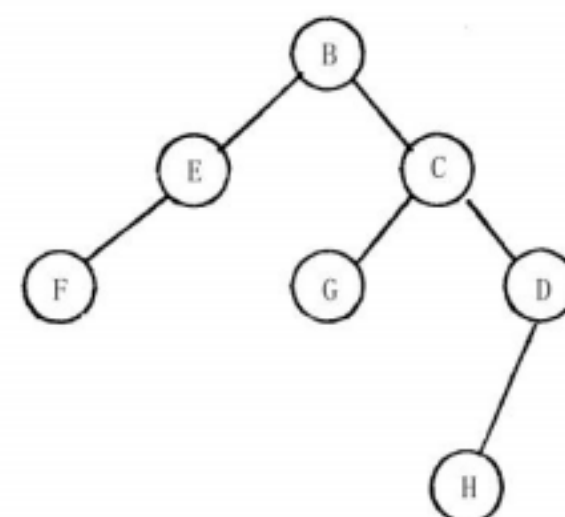
6. 一棵含有  $n$  个结点的  $k$  叉树，可能达到的最大深度为  $n$ ，最小深度为 2。  
 答：当  $k=1$  (单叉树) 时应该最深，深度 =  $n$  (层)；当  $k=n-1$  ( $n-1$  叉树) 时应该最浅，深度 = 2 (层)，但不包括  $n=0$  或 1 时的特例情况。教材答案是“完全  $k$  叉树”，未定量。)

7. 二叉树的基本组成部分是：根 (N)、左子树 (L) 和右子树 (R)。因而二叉树的遍历次序有六种。最常用的是三种：前序法 (即按 NLR 次序)，后序法 (即按 LRN 次序) 和中序法 (也称对称序法，即按 LNR 次序)。这三种方法相互之间有关联。若已知一棵二叉树的前序序列是 BEFCGDH，中序序列是 FEBGCHD，则它的后序序列必是 FEHDCB。

解：法 1：先由已知条件画图，再后序遍历得到结果；

法 2：不画图也能快速得出后序序列，只要找到根的位置特征。由前序先确定 root，由中序先确定左子树。例如，前序遍历 BEFCGDH 中，根结点在最前面，是 B；则后序遍历中 B 一定在最后面。

法 3：递归计算。如 B 在前序序列中第一，中序中在中间 (可知左右子树上有哪些元素)，则在后序中必为最后。如法对 B 的左右子树同样处理，则问题得解。

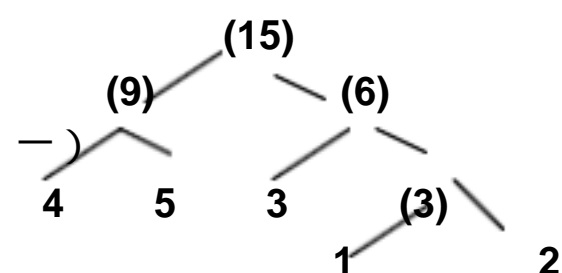


8. 中序遍历的递归算法平均空间复杂度为  $O(n)$ 。

答：即递归最大嵌套层数，即栈的占用单元数。精确值应为树的深度  $k+1$ ，包括叶子的空域也递归了一次。

9. 用 5 个权值 {3, 2, 4, 5, 1} 构造的哈夫曼 (Huffman) 树的带权路径长度是 33。

解：先构造哈夫曼树，得到各叶子的路径长度之后便可求出  $WPL = (4 + 5 + 3) \times 2 + (1 + 2) \times 3 = 33$



(注：两个合并值先后不同会导致编码不同，即哈夫曼编码不唯一)

(注：合并值应排在叶子值之后)

(注：原题为选择题：A. 32 B. 33 C. 34 D. 15)

1. 深度为  $k$  的完全二叉树至少有  $(2^{k-1})$  个结点，至多有  $(2^k - 1)$  个结点，若按自上而下，从左到右次序给结点编号 (从 1 开始)，则编号最小的叶子结点的编号是  $(2^{k-1} - 1 + 1 = 2^{k-1})$

2. 一棵二叉树的第  $i (i \geq 1)$  层最多有  $(2^{i-1})$  个结点，一棵有  $n (n > 0)$  个结点的满二叉树共有  $(2^{\log_2(n+1)-1})$  个叶子结点和  $(2^{\log_2(n+1)-1} - 1)$  个非叶子结点。

3. 现有按中序遍历二叉树的结果是 abc，问有 (5) 种不同形态的二叉树可以得到这一遍历结果，这些二叉树分别是 ( )。

4. 以数据集 {4, 5, 6, 7, 10, 12, 18} 为结点权值所构造的哈夫曼树为 ( )，其带权路径长度为 ( )。

5. 具有 10 个叶子结点的哈夫曼树，最大高度为 (10)，最小高度为 (5)。

$n$  个叶子结点的哈夫曼树，最大高度为  $n$  (除最上一层和最下一层每层一个叶子点)，最小高度为  $\log_2 n + 1$

### 三、单项选择题 (每小题 1 分，共 11 分)

1. 某二叉树的先序遍历序列和后序遍历序列正好相反，则该二叉树一定是 (D)

A. 空或只有一个结点 B. 完全二叉树 C. 二叉排序树 D. 高度等于其结点数

2. 设高度为  $h$  的二叉树上只有度为 0 和度为 2 的结点，则此类二叉树中所包含的结点数至少为 (B)

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

除根结点层只有 1 个结点外，其余  $h-1$  层均有两个结点，结点总数  $=2(h-1)+1=2h-1$ 。

3. 对一个满二叉树， $m$  个树叶， $n$  个结点，深度为  $h$ ，则 ( D )

A.  $n=h+m$  B.  $h+m=2n$  C.  $m=h-1$  D.  $n=2^{h-1}$

4. 根据使用频率为 5 的字符设计的哈夫曼编码不可能是 ( C )

A. 111, 110, 10, 01, 00 B. 000, 001, 010, 011, 1

C. 100, 11, 10, 1, 0 D. 001, 000, 01, 11, 10

C 中，100 和 10 冲突，即一个结点既是叶子结点又是内部结点，哈夫曼树中不可能出现这种情况。

5. 根据使用频率为 5 的字符设计的哈夫曼编码不可能是 ( D )

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

C. 000, 001, 01, 10, 11 D. 00, 100, 101, 110, 111

哈夫曼树中只有度为 0 或度为 2 的结点，D 不满足这种条件。

( C ) 1. 不含任何结点的空树 \_\_\_\_\_。

( A ) 是一棵树；

( B ) 是一棵二叉树；

( C ) 是一棵树也是一棵二叉树；

( D ) 既不是树也不是二叉树

答：以前的标签是 B，因为那时树的定义是  $n \geq 1$

( C ) 2. 二叉树是非线性数据结构，所以 \_\_\_\_\_。

( A ) 它不能用顺序存储结构存储；

( B ) 它不能用链式存储结构存储；

( C ) 顺序存储结构和链式存储结构都能存储；

( D ) 顺序存储结构和链式存储结构都不能使用

( C ) 3. 具有  $n(n>0)$  个结点的完全二叉树的深度为 \_\_\_\_\_。

( A )  $\lceil \log_2(n) \rceil$

( B )  $\lfloor \log_2(n) \rfloor$

( C )  $\lfloor \log_2(n) \rfloor + 1$

( D )  $\lceil \log_2(n) + 1 \rceil$

注 1： $\lceil x \rceil$  表示不小于  $x$  的最小整数； $\lfloor x \rfloor$  表示不大于  $x$  的最大整数，它们与  $[ ]$  含义不同！

注 2：选 ( A ) 是错误的。例如当  $n$  为 2 的整数幂时就会少算一层。似乎  $\lfloor \log_2(n) \rfloor + 1$  是对的？

( A ) 4. 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

5. 从供选择的答案中，选出应填入下面叙述 \_\_\_\_\_ ？ 内的最确切的解答，把相应编号写在答卷的对应栏内。

树是结点的有限集合，它 A 根结点，记为  $T$ 。其余的结点分成为  $m(m \geq 0)$  个 B 的集合  $T_1, T_2, \dots, T_m$ ，每个集合又都是树，此时结点  $T$  称为  $T_i$  的父结点， $T_i$  称为  $T$  的子结点 ( $1 \leq i \leq m$ )。一个结点的子结点个数为该结点的 C。

供选择的答案

A：有 0 个或 1 个

有 0 个或多个

有且只有 1 个

有 1 个或 1 个以上

B：互不相交

允许相交

允许叶结点相交

允许树枝结点相交

C：权

维数

次数 (或度)

序

答案：ABC = 1, 1, 3

6. 从供选择的答案中，选出应填入下面叙述 \_\_\_\_\_ ？ 内的最确切的解答，把相应编号写在答卷的对应栏内。

二叉树 A。在完全的二叉树中，若一个结点没有 B，则它必定是叶结点。每棵树都能惟一地转换成与它对应的二叉树。由树转换成的二叉树里，一个结点  $N$  的左子女是  $N$  在原树里对应结点的 C，而  $N$  的右子女是它在原树里对应结点的 D。

供选择的答案

A：是特殊的树

不是树的特殊形式

是两棵树的总称

有是只有二个根结点的树形结



构

B: 左子结点 右子结点 左子结点或者没有右子结点 兄弟

C ~ D: 最左子结点 最右子结点 最邻近的右兄弟 最邻近的左兄弟

答案: A= 最左的兄弟 B= 最右的兄弟 C= D =

答案: ABCDE = 2, 1, 1, 3

#### 四、简答题（每小题 4 分，共 20 分）

1. 一棵度为 2 的树与一棵二叉树有何区别？

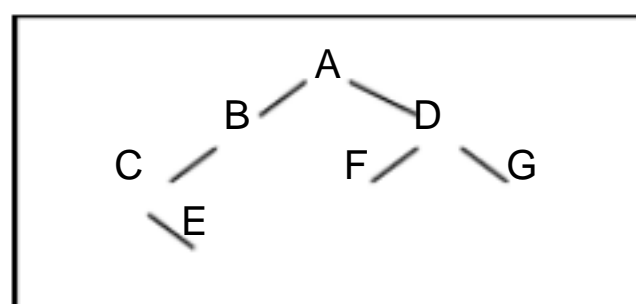
答：度为 2 的树从形式上看与二叉树很相似，但它的子树是无序的，而二叉树是有序的。即，在一般树中若某结点只有一个孩子，就无需区分其左右次序，而在二叉树中 即使是一个孩子也有左右之分。

2. 【01 年计算机研题】 设如下图所示的二叉树 B 的存储结构为二叉链表， root 为根指针，结点结构为：( lchild,data,rchild )。其中 lchild , rchild 分别为指向左右孩子的指针， data 为字符型， root 为根指针，试回答下列问题：

1. 对下列二叉树 B，执行下列算法 traversal(root)，试指出其输出结果；

2. 假定二叉树 B 共有 n 个结点，试分析算法 traversal(root) 的时间复杂度。（共 8 分）

二叉树 B



C 的结点类型定义如下：

```
struct node
{char data;
struct node * lchild, rchild;
};
```

C 算法如下：

```
void traversal(struct node *root)
{if (root)
{printf( " %c" ,root->data);
traversal(root->lchild);
printf( " %c" ,root->data);
traversal(root->rchild);
}
}
```

解：这是“先根再左再根再右”，比前序遍历多打印各结点一次，输出结果为： A B C C E E B A D F F D G G

特点： 每个结点肯定都会被打印两次； 但出现的顺序不同，其规律是：凡是有左子树的结点，必间隔左子树的全部结点后再重复出现；如 A , B , D 等结点。反之马上就会重复出现。如 C , E , F , G 等结点。

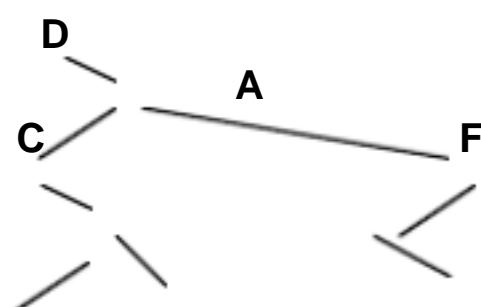
时间复杂度以访问结点的次数为主，精确值为  $2*n$ ，时间渐近度为  $O(n)$ 。

3. 【01 年计算机研题】 【严题集 6.27】 给定二叉树的两种遍历序列，分别是：

前序遍历序列： D , A , C , E , B , H , F , G , I ; 中序遍历序列： D , C , B , E , H , A , G , I , F ,

试画出二叉树 B，并简述由任意二叉树 B 的前序遍历序列和中序遍历序列求二叉树 B 的思想方法。

解：方法是：由前序先确定 root，由中序可确定 root 的左、右子树。然后由其左子树的元素集合和右子树的集合对应前序遍历序列中的元素集合，可继续确定 root 的左右孩子。将他们分别作为新的 root，不断递归，则所有元素都将被唯一确定，问题得解。



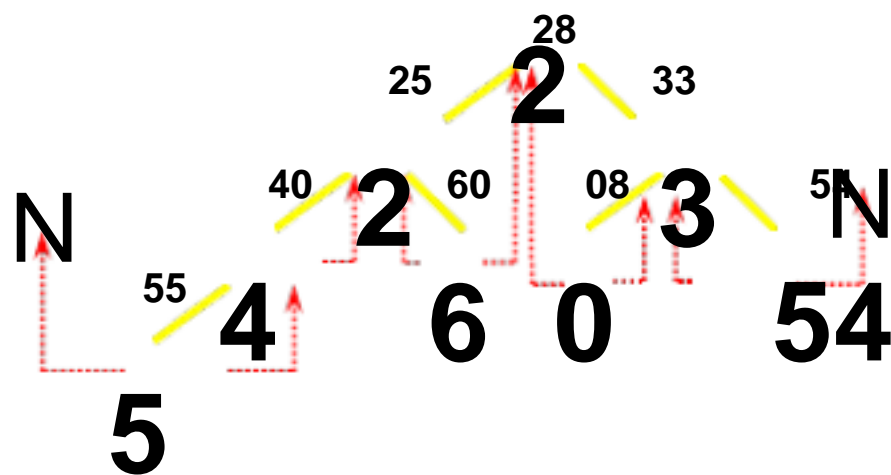
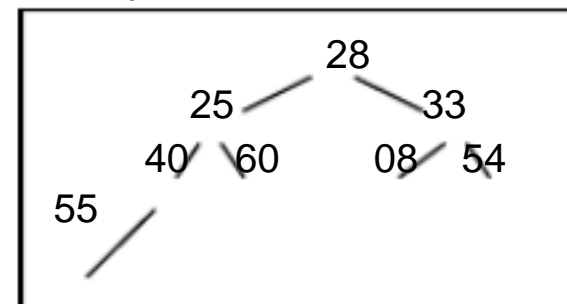
E                      G

B            H                      I

4. 【计算机研 2000】 给定如图所示二叉树 T，请画出与其对应的中序线索二叉树。

解：要遵循中序遍历的轨迹来画出每个前驱和后继。

中序遍历序列： 55 40 25 60 28 08 33 54



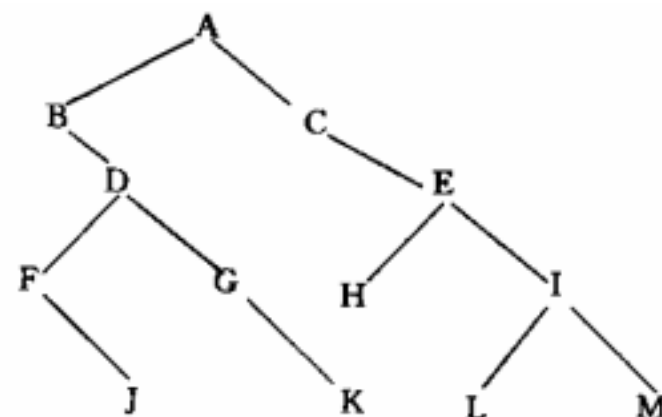
五、阅读分析题（每题 5 分，共 20 分）

1. （P60 4-26）试写出如图所示的二叉树分别按先序、中序、后序遍历时得到的结点序列。

答：DLR：A B D F J G K C E H I L M

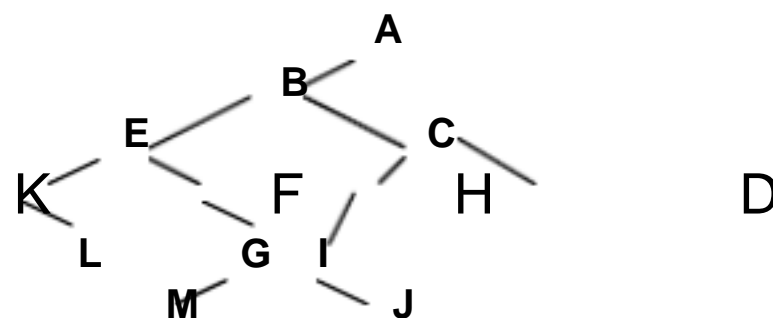
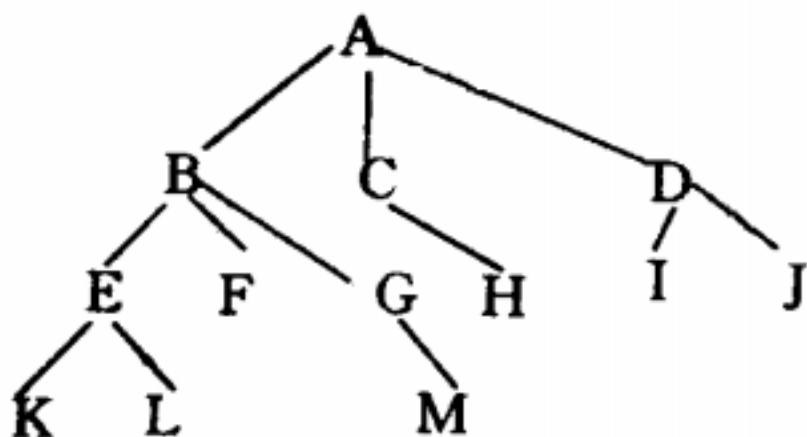
LDR: B F J D G K A C H E L I M

LRD：J F K G D B H L M I E C A



2. （P60 4-27）把如图所示的树转化成二叉树。

答：注意全部兄弟之间都要连线（包括度为 2 的兄弟），并注意原有连线结点一律归入左子树，新添连线结点一律归入右子树。



答：这是找结点后继的程序。

共有 3 处错误。

注：当 rtag = 1 时说明内装后继指针，可直接返回，第一句无错。

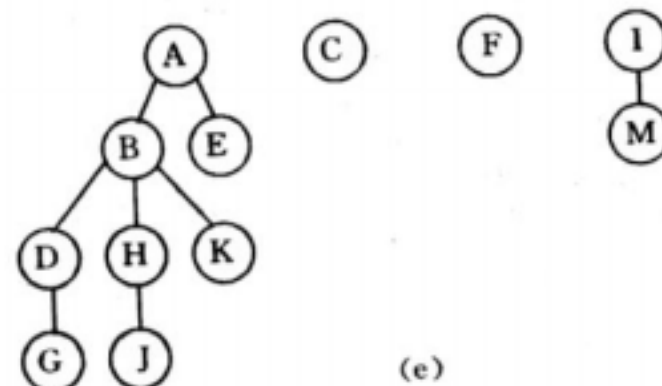
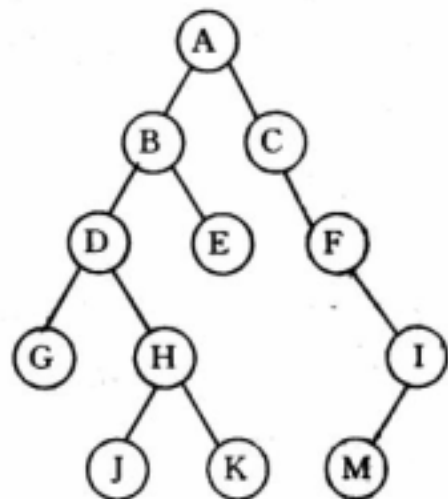
当 rtag = 0 时说明内装右孩子指针，但孩子未必是后继，需要计算。中序遍历应当先左再根再右，所以应当找左子树直到叶子处。 r=r->lchild; 直到 LTag=1; 应改为： while(!r-> Ltag)r=r-> Lchild;

```
BiTree InSucc(BiTree q){
// 已知 q 是指向中序线索二叉树上某个结点的指针，
// 本函数返回指向 *q 的后继的指针。
r=q->rchild;           //应改为 r=q ;
if(!r->rtag)
    while(!r->rtag)r=r->rchild; //应改为 while(!r-
    >Ltag) r=r->Lchild;
    return r; //应改为 return r->rchild ;
} //InSucc
```

3. 【严题集 6.17】 阅读下列算法，若有错，改正之。

4.【严题集 6.21】画出和下列二叉树相应的森林。

答：注意根右边的子树肯定是森林，而孩子结点的右子树均为兄弟。



## 六、算法设计题（前 5 题中任选 2 题，第 6 题必做，每题 8 分，共 24 分）

1.【严题集 6.42】编写递归算法，计算二叉树中叶子结点的数目。

解：思路：输出叶子结点比较简单，用任何一种遍历递归算法，凡是左右指针均空者，则为叶子，将其打印出来。

法一：核心部分为：

```
DLR(liuyu *root)      /* 中序遍历    递归函数 */
{if(root!=NULL)
  {if((root->lchild==NULL)&&(root->rchild==NULL)){sum++; printf("%d\n",root->data);}
   DLR(root->lchild);
   DLR(root->rchild); }
  return(0);
}
```

法二：

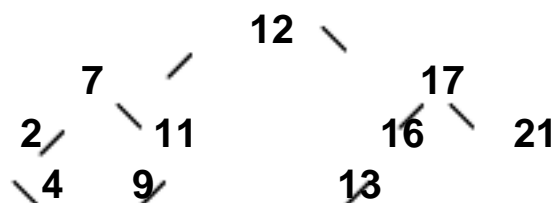
```
int LeafCount_BiTree(Bitree T)// 求二叉树中叶子结点的数目
{
  if(!T) return 0; // 空树没有叶子
  else if(!T->lchild&&!T->rchild) return 1; // 叶子结点
  else return Leaf_Count(T->lchild)+Leaf_Count(T->rchild);// 左子树的叶子数加
  上右子树的叶子数
} //LeafCount_BiTree
```

注：上机时要先建树！例如实验二的方案一。

打印叶子结点值（并求总数）

思路：先建树，再从遍历过程中打印结点值并统计。

步骤 1 键盘输入 序列 12, 8, 17, 11, 16, 2, 13, 9, 21, 4, 构成一棵二叉排序树。叶子结点值应该是 4, 9, 13, 21, 总数应该是 4。



编程：生成二叉树排序树之后，再中序遍历排序查找结点的完整程序如下：

说明部分为：

```
#include <stdio.h>
#include <stdlib.h>
typedef struct liuyu{int data;struct liuyu *lchild,*rchild;}test;
liuyu *root;
```



```

int sum=0;int m=sizeof(test);

void insert_data(int x)          /* 如何生成二叉排序树？参见教材    P43C 程序 */
{ liuyu *p,*q,*s;
s=(test*)malloc(m);
s->data=x;
s->lchild=NULL;
s->rchild=NULL;

if(!root){root=s; return;}
p=root;
while(p)                      /* 如何接入二叉排序树的适当位置    */
{q=p;
if(p->data==x){printf("data already exist! \n");return;}
else if(x<p->data)p=p->lchild;    else p=p->rchild;
}
if(x<q->data)q->lchild=s;
else q->rchild=s;
}

DLR(liuyu *root)              /* 中序遍历    递归函数 */
{if(root!=NULL)
{if((root->lchild==NULL)&&(root->rchild==NULL)){sum++; printf("%d\n",root->data);}
DLR(root->lchild);
DLR(root->rchild); }
return(0);
}

main()                        /* 先生成二叉排序树，再调用中序遍历递归函数进行排序输出    */
{int i,x;
i=1;
root=NULL;                    /* 千万别忘了赋初值给    root!*/
do{printf("please input data%d:",i);
i++;
scanf("%d",&x);              /* 从键盘采集数据，以    -9999 表示输入结束    */
if(x== -9999){
DLR(root);
printf("\nNow output count value:%d\n",sum);
return(0); }
else insert_data(x);          /* 调用插入数据元素的函数    */
}while(x!= -9999);
return(0);}
执行结果：

```

```

(Inactive x)
please input data1:12
please input data2:8
please input data3:17
please input data4:11
please input data5:16
please input data6:2
please input data7:13
please input data8:9
please input data9:21
please input data10:4
please input data11:-9999
4
9
13
21
Now output count value:4

```

若一开始运行就输入 -9999，则无叶子输出，sum=0。

2. 【全国专升本统考题】 写出求二叉树深度的算法，先定义二叉树的抽象数据类型。 (10分)

或【严题集 6.44】编写递归算法，求二叉树中以元素值为  $x$  的结点为根的子树的深度。

答；设计思路：只查后继链表指针，若左或右孩子的左或右指针非空，则层次数加 1；否则函数返回。

但注意，递归时应当从叶子开始向上计数，否则不易确定层数。

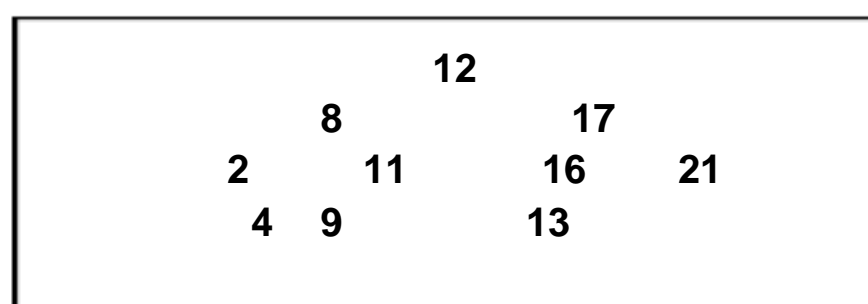
```
int depth(liuyu*root) /* 统计层数 */
{int d,p; /* 注意每一层的局部变量 d,p 都是各自独立的 */
p=0;
if(root==NULL)return(p); /* 找到叶子之后才开始统计 */
else{
d=depth(root->lchild);
if(d>p) p=d; /* 向上回溯时，要挑出左右子树中的相对大的那个深度值 */
d=depth(root->rchild);
if(d>p)p=d;
}
p=p+1;
return(p);
}
```

法二：

```
int Get_Sub_Depth(Bitree T,int x)// 求二叉树中以值为 x 的结点为根的子树深度
{
if(T->data==x)
{
printf("%d\n",Get_Depth(T)); // 找到了值为 x 的结点,求其深度
exit 1;
}
}
else
{
if(T->lchild) Get_Sub_Depth(T->lchild,x);
if(T->rchild) Get_Sub_Depth(T->rchild,x); // 在左右子树中继续寻找
}
}
}
//Get_Sub_Depth
int Get_Depth(Bitree T)// 求子树深度的递归算法
{
if(!T) return 0;
else
{
m=Get_Depth(T->lchild);
n=Get_Depth(T->rchild);
return (m>n?m:n)+1;
}
}
//Get_Depth
```

附：上机调试过程

步骤 1 键盘输入 序列 12，8，17，11，16，2，13，9，21，4，构成一棵二叉排序树。层数应当为 4



步骤 2： 执行求深度的函数，并打印统计出来的深度值。

完整程序如下：

```
#include <stdio.h>
#include <stdlib.h>
typedef struct liuyu{int data;struct liuyu *lchild,*rchild;}test;
liuyu *root;
int sum=0;int m=sizeof(test);

void insert_data(int x)          /* 如何生成二叉排序树？参见教材    P43C 程序 */
{ liuyu *p,*q,*s;
s=(test*)malloc(m);
s->data=x;
s->lchild=NULL;
s->rchild=NULL;

if(!root){root=s; return;}
p=root;
while(p)          /* 如何接入二叉排序树的适当位置    */
{q=p;
if(p->data==x){printf("data already exist! \n");return;}
else if(x<p->data)p=p->lchild;    else p=p->rchild;
}
if(x<q->data)q->lchild=s;
else q->rchild=s;
}

int depth(liuyu*root)          /* 统计层数 */
{int d,p;          /* 注意每一层的局部变量    d,p 都是各自独立的 */
p=0;
if(root==NULL)return(p);          /* 找到叶子之后才开始统计    */
else{
d=depth(root->lchild);
if(d>p) p=d;          /* 向上回溯时，要挑出左右子树中的相对大的那个深度值    */
d=depth(root->rchild);
if(d>p)p=d;
}
p=p+1;
return(p);
}

void main()          /* 先生成二叉排序树，再调用深度遍历递归函数进行统计并输出    */
{int i,x;
i=1;
root=NULL;          /* 千万别忘了赋初值给    root! */
do{printf("please input data%d:",i);
i++;
scanf("%d",&x);          /* 从键盘采集数据，以    -9999 表示输入结束    */
if(x== -9999){
printf("\nNow output depth value=%d\n", depth (root)); return; }
else insert_data(x);          /* 调用插入数据元素的函数    */
}while(x!= -9999);
return;}
执行结果：
```

```

(Inactive x)
please input data1:      1 2
please input data2:      8
please input data3:      1 7
please input data4:      1 1
please input data5:      1 6
please input data6:      2
please input data7:      1 3
please input data8:      9
please input data9:      2 1
please input data10:     4
please input data11:     - 9 9 9 9

Now output depth value=4

```

3. 【严题集 6.47】编写按层次顺序（同一层自左至右）遍历二叉树的算法。

或：按层次输出二叉树中所有结点；

解：思路：既然要求从上到下，从左到右，则利用队列存放各子树结点的指针是个好办法。

这是一个循环算法，用 while 语句不断循环，直到队空之后自然退出该函数。

技巧之处：当根结点入队后，会自然使得左、右孩子结点入队，而左孩子出队时又会立即使得它的左右孩子结点入队，，，以此产生了按层次输出的效果。

```

level(liuyu*T)
/* liuyu *T,*p,*q[100];      假设 max 已知 */
{int f,r;
f=0; r=0;          /* 置空队 */
r=(r+1)%max;
q[r]=T;            /* 根结点进队 */
while(f!=r)        /* 队列不空 */
{f=(f+1)%max);
p=q[f];            /* 出队 */
printf("%d",p->data);      /* 打印根结点 */
if(p->lchild){r=(r+1)%max; q[r]=p->lchild;}      /* 若左子树不空，则左子树进队 */
if(p->rchild){r=(r+1)%max; q[r]=p->rchild;}      /* 若右子树不空，则右子树进队 */
}
return(0);
}

```

法二：

```

void LayerOrder(Bitree T)// 层序遍历二叉树
{
    InitQueue(Q); // 建立工作队列

    EnQueue(Q,T);
    while(!QueueEmpty(Q))
    {
        DeQueue(Q,p);
        visit(p);
        if(p->lchild) EnQueue(Q,p->lchild);
        if(p->rchild) EnQueue(Q,p->rchild);
    }
}
//LayerOrder

```

可以用前面的函数建树，然后调用这个函数来输出。

完整程序如下（已上机通过）

```

#include <stdio.h>
#include <stdlib.h>
#define max 50

```

```

typedef struct liuyu{int data;struct liuyu *lchild,*rchild;}test;
liuyu *root,*p,*q[max];
int sum=0;int m=sizeof(test);

void insert_data(int x)          /* 如何生成二叉排序树？参见教材    P43C 程序 */
{ liuyu *p,*q,*s;
s=(test*)malloc(m);
s->data=x;
s->lchild=NULL;
s->rchild=NULL;

if(!root){root=s; return;}
p=root;
while(p)          /* 如何接入二叉排序树的适当位置    */
{q=p;
if(p->data==x){printf("data already exist! \n");return;}
else if(x<p->data)p=p->lchild;    else p=p->rchild;
}
if(x<q->data)q->lchild=s;
else q->rchild=s;
}

level(liuyu*T)
/* liuyu *T,*p,*q[100];    假设 max 已知 */
{int f,r;
f=0; r=0;          /* 置空队 */
r=(r+1)%max;
q[r]=T;          /* 根结点进队 */
while(f!=r)      /* 队列不空 */
{f=(f+1)%max;
p=q[f];          /* 出队 */
printf("%d",p->data);          /* 打印根结点 */
if(p->lchild){r=(r+1)%max; q[r]=p->lchild;}          /* 若左子树不空，则左子树进队 */
if(p->rchild){r=(r+1)%max; q[r]=p->rchild;}          /* 若右子树不空，则右子树进队 */
}
return(0);
}

void main()          /* 先生成二叉排序树，再调用深度遍历递归函数进行统计并输出    */
{int i,x;
i=1;
root=NULL;          /* 千万别忘了赋初值给 root! */
do{printf("please input data%d:",i);
i++;
scanf("%d",&x);          /* 从键盘采集数据，以 -9999 表示输入结束 */
if(x==-9999){
printf("\nNow output data value:\n", level(root)); return; }
else insert_data(x);}          /* 调用插入数据元素的函数 */
while(x!=-9999);
return;}

```

4. 已知一棵具有  $n$  个结点的完全二叉树被顺序存储于一维数组  $A$  中，试编写一个算法打印出编号为  $i$  的结点的双亲和所有的孩子。

答：首先，由于是完全二叉树，不必担心中途会出现孩子为 **null** 的情况。

其次分析：结点  $i$  的左孩子为  $2i$ ，右孩子为  $2i+1$ ；直接打印即可。

```
Printf(    " Left_child=    ", %d, v[2*i].data;
          " Right_child=    ", %d, v[2*i+1].data);
```

但其双亲是  $i/2$  , 需先判断  $i$  为奇数还是偶数。若  $i$  为奇数 , 则应当先  $i--$  , 然后再除以  $2$ 。  
If( $i/2 \neq 0$ ) $i--$ ;  
Printf( " Parents= " , %d, v[ $i/2$ ].data);

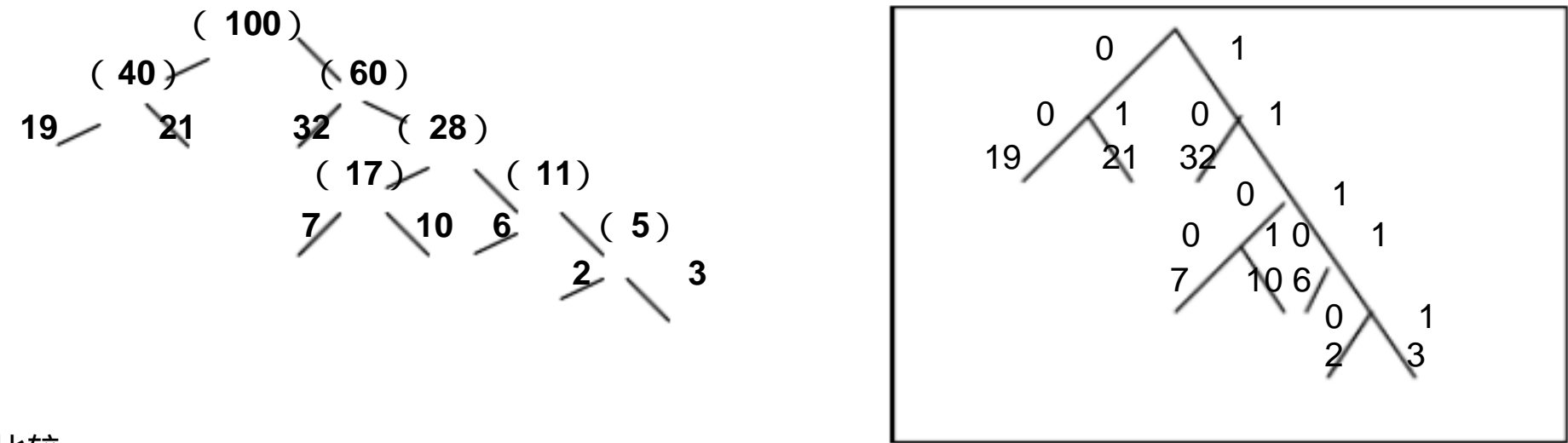
```
5. 【严题集 6.49 】编写算法判别给定二叉树是否为完全二叉树。  
答：int IsFull_Bitree(Bitree T)// 判断二叉树是否完全二叉树 , 是则返回 1, 否则返回 0  
{  
    InitQueue(Q);  
    flag=0;  
    EnQueue(Q,T); // 建立工作队列  
    while(!QueueEmpty(Q))  
    {  
        {  
            DeQueue(Q,p);  
            if(!p) flag=1;  
            else if(flag) return 0;  
            else  
            {  
                EnQueue(Q,p->lchild);  
                EnQueue(Q,p->rchild); // 不管孩子是否为空 , 都入队列  
            }  
        }  
    }  
    return 1;  
}  
//IsFull_Bitree
```

分析 : 该问题可以通过层序遍历的方法来解决 . 与 6.47 相比 , 作了一个修改 , 不管当前结点是否有左右孩子 , 都入队列 . 这样当树为完全二叉树时 , 遍历得到的是一个连续的不包含空指针的序列 . 反之 , 则序列中会含有空指针 .

6. 【严题集 6.26 】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 , 0.10。试为这 8 个字母设计哈夫曼编码。使用 0~ 7 的二进制表示形式是另一种编码方案。对于上述实例 , 比较两种方案的优缺点。

解：方案 1；哈夫曼编码  
先将概率放大 100 倍 , 以方便构造哈夫曼树。

w={7,19,2,6,32,3,21,10} , 按哈夫曼规则：【 [( 2,3) , 6], (7,10) 】 , , , 19, 21, 32



方案比较：

字母编号	对应编码	出现频率
1	1100	0.07
2	00	0.19
3	11110	0.02
4	1110	0.06
5	10	0.32
6	11111	0.03
7	01	0.21
8	1101	0.10

字母编号	对应编码	出现频率
1	000	0.07
2	001	0.19
3	010	0.02
4	011	0.06
5	100	0.32
6	101	0.03
7	110	0.21
8	111	0.10

方案 1 的 WPL = 2(0.19+0.32+0.21)+4(0.07+0.06+0.10)+5(0.02+0.03)=1.44+0.92+0.25=2.61  
方案 2 的 WPL = 3(0.19+0.32+0.21+0.07+0.06+0.10+0.02+0.03)=3

结论：哈夫曼编码优于等长二进制编码

1. 二叉树采用链接存储结构，设计算法计算一棵给定二叉树的度为 2 的结点数。

方法 1:

```
int f(bitree T)
{ int num1,num2;
  if (T==NULL) return 0;
  else
    { num1=f(T->lchild); nmu2=f(T->rchild);
      If (T->lchild && T->rchild) return (num1+num2+1);
      Else return(num1+num2);
    }
}
```

方法 2:

```
void f(bitree T, int &n)
{ if (T)
  { If (T->lchild && T->rchild) n++;
    f(T->lchild,n);
    f(T->rchild,n);
  }
}
```

2. 采用二叉链表的形式存储二叉树，编写一个算法判定两棵二叉树是否相似。

```
int like(bitree S, bitree T)
{ if (S==NULL && T=NULL) return (1);
  else if (S && T)
    return (like(S->lchild,T->lchild) && like(S->rchild,T->rchild));
    else return 0;
}
```

3. 给定一棵用链表表示的二叉树，编写算法判定一棵二叉树 T 是否是完全二叉树。

采用层次遍历的思想解决这一问题。设置一个标志 flag 一旦某结点的左或右分支为空，则将 flag 置 1。若此后遍历的结点的左、右分支都为空，则二叉树是完全二叉树，否则不是完全二叉树。

```
Int iscompletebit(bitree T)
{ queue Q[maxsize];
  Int Flag=0;
  Bitree p=T;
  If (T==NULL) return 1;
  Initqueue(Q);
  Enqueue(Q,p);
  While (!queueempty(Q))
  { dequeue(Q,p);
    If (p->lchild && !flag) enqueue(Q,p->lchild);
    Else
      { if (p->lchild ) return 0;
        Else flag=1; }
    If (T->rchild && !flag) enqueue(Q,p->rchild);
    Else
      { if (p->rchild ) return 0;
        Else flag=1; }
  } Return 1;
}
```

4. 已知一棵二叉树的中序遍历序列为 CDBAEGF 前序遍历序列为 ABCDEFG 试问能不能惟一确定一棵二叉树，若能请画出该二叉树。

## 第 7 章 图 自测卷解答



## 一、单选题（每题 1 分，共 16 分）

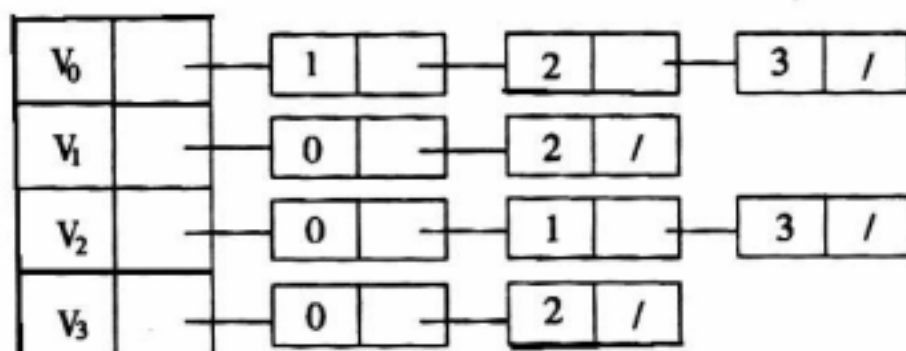
- ( **C** ) 1. 在一个图中，所有顶点的度数之和等于图的边数的 \_\_\_\_\_ 倍。  
A . 1/2                      B . 1                      C . 2                      D . 4
- ( **B** ) 2. 在一个有向图中，所有顶点的入度之和等于所有顶点的出度之和的 \_\_\_\_\_ 倍。  
A . 1/2                      B . 1                      C . 2                      D . 4
- ( **B** ) 3. 有 8 个结点的无向图最多有 \_\_\_\_\_ 条边。  
A . 14                      B . 28                      C . 56                      D . 112
- ( **C** ) 4. 有 8 个结点的无向连通图最少有 \_\_\_\_\_ 条边。  
A . 5                      B . 6                      C . 7                      D . 8
- ( **C** ) 5. 有 8 个结点的有向完全图有 \_\_\_\_\_ 条边。  
A . 14                      B . 28                      C . 56                      D . 112
- ( **B** ) 6. 用邻接表表示图进行广度优先遍历时，通常是采用 \_\_\_\_\_ 来实现算法的。  
A . 栈                      B . 队列                      C . 树                      D . 图
- ( **A** ) 7. 用邻接表表示图进行深度优先遍历时，通常是采用 \_\_\_\_\_ 来实现算法的。  
A . 栈                      B . 队列                      C . 树                      D . 图
- ( **C** ) 8. 已知图的邻接矩阵，根据算法思想，则从顶点 0 出发按深度优先遍历的结点序列是

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

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

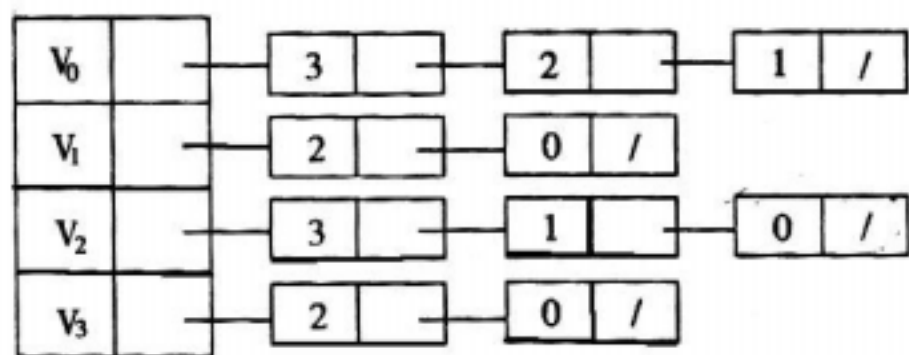
建议：0 1 3 4 2 5 6

- ( **D** ) 9. 已知图的邻接矩阵同上题 8，根据算法，则从顶点 0 出发，按深度优先遍历的结点序列是  
A . 0 2 4 3 1 5 6      B . 0 1 3 5 6 4 2      C . 0 4 2 3 1 6 5      D . 0 1 3 4 2 5 6
- ( **B** ) 10. 已知图的邻接矩阵同上题 8，根据算法，则从顶点 0 出发，按广度优先遍历的结点序列是  
A . 0 2 4 3 6 5 1      B . 0 1 3 6 4 2 5      C . 0 4 2 3 1 5 6      D . 0 1 3 4 2 5 6  
(建议：0 1 2 3 4 5 6)
- ( **C** ) 11. 已知图的邻接矩阵同上题 8，根据算法，则从顶点 0 出发，按广度优先遍历的结点序列是  
A . 0 2 4 3 1 6 5      B . 0 1 3 5 6 4 2      C . 0 1 2 3 4 6 5      D . 0 1 2 3 4 5 6
- ( **D** ) 12. 已知图的邻接表如下所示，根据算法，则从顶点 0 出发按深度优先遍历的结点序列是



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

- ( **A** ) 13. 已知图的邻接表如下所示，根据算法，则从顶点 0 出发按广度优先遍历的结点序列是



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

- ( A ) 14. 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
- ( D ) 15. 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
- ( A ) 16. 任何一个无向连通图的最小生成树  
A. 只有一棵      B. 一棵或多棵      C. 一定有多棵      D. 可能不存在  
(注, 生成树不唯一, 但最小生成树唯一, 即边权之和或树权最小的情况唯一)
17. 下面关于 AOE网的叙述中, 不正确的是 ( ) B  
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. 所有的关键活动提前完成, 那么整个工程将会提前完成  
D. 某个关键活动提前完成, 那么整个工程将会提前完成
18. 若邻接表中有奇数个表结点, 则一定 ( ) D  
A. 图中有奇数个顶点      B. 图中有偶数个顶点      C. 图为无向图      D. 图为有向图
19. 在一个无向图中, 所有顶点的度数之和等于所有边数的 ( B ) 倍, 在一个有向图中, 所有顶点的入度之和等于所有顶点出度之和的 ( C ).  
A. 1/2      B. 2      C. 1      D. 4
20. 若邻接表中的有奇数个表结点, 则一定 ( D )  
A. 图中有奇数个顶点      B. 图中有偶数个顶点      C. 图为无向图      D. 图为有向图
21. 下面关于 AOE 网的叙述中, 不正确的是 ( B )  
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. 所有的关键活动提前完成, 那么整个工程将会提前完成  
D. 某个关键活动提前完成, 那么整个工程将会提前完成

## 二、填空题

- 29 条边的无向连通图, 至少有 ( 9 ) 个顶点, 至多有 ( 30 ) 个顶点, 有 29 条边的无向非连通图, 至少有 ( 10 ) 个顶点。
- $n$  个顶点的强连通有向图  $G$ , 最多有 (  $n(n-1)$  ) 条边, 最少有 (  $n$  ) 边。  
强连通图即是任何两个顶点之间有路径相通, 当所有结点在一个环上时, 必定是强连通图。
- 29 条边的有向连通图, 至少有 ( 6 ) 个顶点, 至多有 ( 29 ) 个顶点, 有 29 条边的有向非连通图, 至少有 ( 7 ) 个顶点。
- 已知一个图的邻接矩阵表示, 删除所有从第  $i$  个结点出发的边的方法是 ( 将矩阵第  $i$  行全部置为 0 )
- 图有 邻接矩阵、邻接表 等存储结构, 遍历图有 深度优先遍历、广度优先遍历 等方法。
- 有向图  $G$  用邻接表矩阵存储, 其第  $i$  行的所有元素之和等于顶点  $i$  的 出度。
- 如果  $n$  个顶点的图是一个环, 则它有  $n$  棵生成树。 ( 以任意一顶点为起点, 得到  $n-1$  条边 )
- $n$  个顶点  $e$  条边的图, 若采用邻接矩阵存储, 则空间复杂度为  $O(n^2)$ 。
- $n$  个顶点  $e$  条边的图, 若采用邻接表存储, 则空间复杂度为  $O(n+e)$ 。

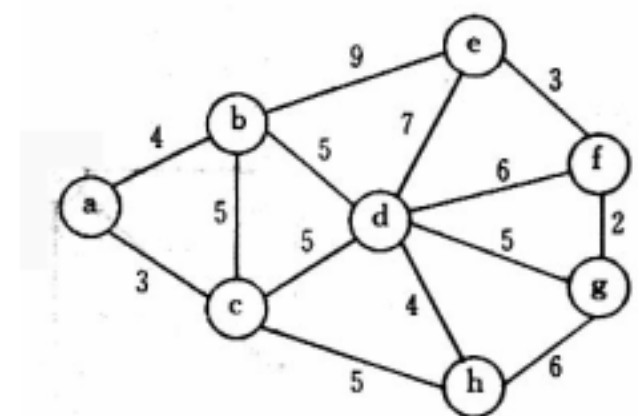
6. 设有一稀疏图  $G$ ，则  $G$  采用 邻接表 存储较省空间。
7. 设有一稠密图  $G$ ，则  $G$  采用 邻接矩阵 存储较省空间。
8. 图的逆邻接表存储结构只适用于 有向 图。
9. 已知一个有向图的邻接矩阵表示，删除所有从第  $i$  个顶点出发的方法是 将邻接矩阵的第  $i$  行全部置 0。
10. 图的深度优先遍历序列 不是 惟一的。
11.  $n$  个顶点  $e$  条边的图采用邻接矩阵存储，深度优先遍历算法的时间复杂度为  $O(n^2)$ ；若采用邻接表存储时，该算法的时间复杂度为  $O(n+e)$ 。
12.  $n$  个顶点  $e$  条边的图采用邻接矩阵存储，广度优先遍历算法的时间复杂度为  $O(n^2)$ ；若采用邻接表存储，该算法的时间复杂度为  $O(n+e)$ 。
13. 若要求一个稀疏图  $G$  的最小生成树，最好用 克鲁斯卡尔 (Kruskal) 算法来求解。
14. 若要求一个稠密图  $G$  的最小生成树，最好用 普里姆 (Prim) 算法来求解。
15. 用 **Dijkstra** 算法求某一顶点到其余各顶点间的最短路径是按路径长度 递增 的次序来得到最短路径的。

16. 拓扑排序算法是通过重复选择具有 0 个前驱顶点的过程来完成的。

### 三、简答题（每题 6 分，共 24 分）

1. 请对下图的无向带权图：

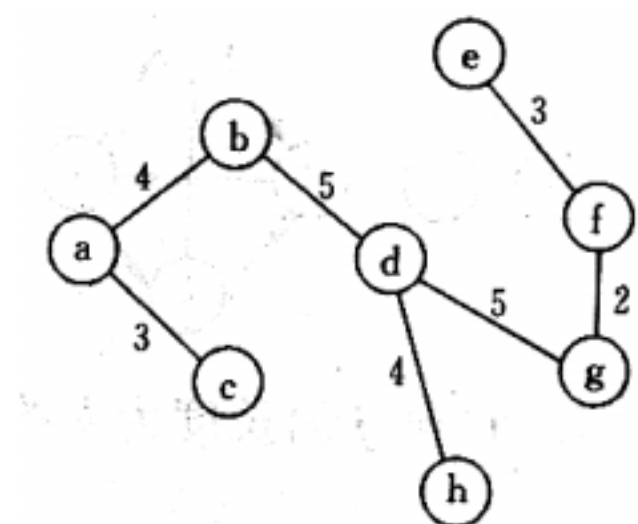
- (1) 写出它的邻接矩阵，并按普里姆算法求其最小生成树；
- (2) 写出它的邻接表，并按克鲁斯卡尔算法求其最小生成树。



解：设起点为  $a$ 。

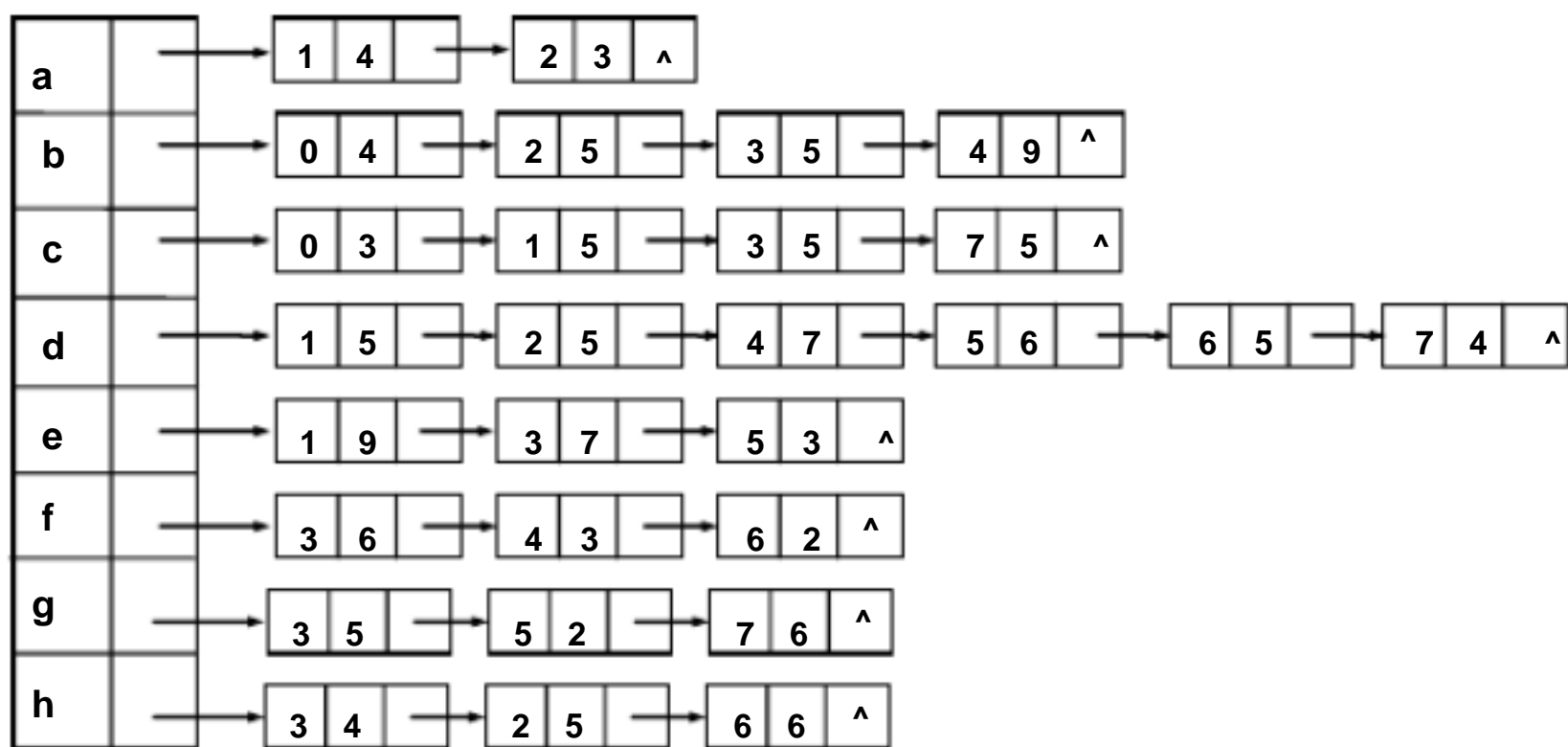
0	4	3	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
4	0	5	5	9	$\infty$	$\infty$	$\infty$
3	5	0	5	$\infty$	$\infty$	$\infty$	5
$\infty$	5	5	0	7	6	5	4
$\infty$	9	$\infty$	7	0	3	$\infty$	$\infty$
$\infty$	$\infty$	$\infty$	6	3	0	2	$\infty$
$\infty$	$\infty$	$\infty$	5	$\infty$	2	0	6
$\infty$	$\infty$	5	4	$\infty$	$\infty$	6	0

最小生成树



**PRIM** 算法是将顶点归并：  
 设初始顶点为  $a$ ，依次归并的  
 顶点集为： $\{a\}, \{a,c\}, \{a,c,b\}, \{a,c,b,d\}, \{a,c,b,d,h\},$   
 $\{a,c,b,d,h,g\}, \{a,c,b,d,h,g,f\}, \{a,c,b,d,h,g,f,e\}$ 。所得最小生成树如上。

邻接表为：



克鲁斯卡尔算法 是将连归并：依次归并的边集为：

$\{(f,g)\}$

$\{(f,g), (e,f)\}$ ,

$\{(f,g), (e,f), (a,c)\}$ ,

$\{(f,g), (e,f), (a,c), (a,b)\}$ ,

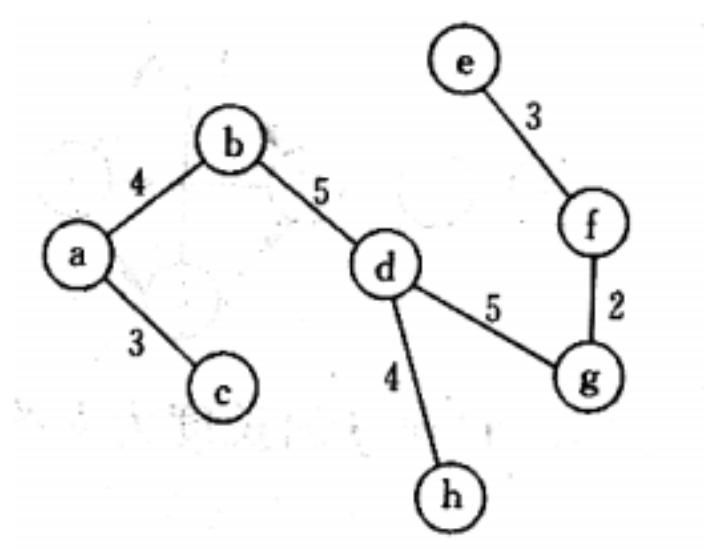
$\{(f,g), (e,f), (a,c), (a,b), (d,h)\}$ ,

$\{(f,g), (e,f), (a,c), (a,b), (d,h), (d,g)\}$ ,

$\{(f,g), (e,f), (a,c), (a,b), (d,h), (d,g), (b,d)\}$ ,

$\{(f,g), (e,f), (a,c), (a,b), (d,h), (d,g), (b,d)\}$ , 所有的顶点同在一个连通分量上。

所得最小生成树如下。



## 第一章 绪论

1. 下面是几种数据的逻辑结构  $S=(D, R)$ ，分别画出对应的数据逻辑结构，并指出它们分别属于何种结构。

$D=\{a, b, c, d, e, f\}$   $R=\{r\}$

(a)  $r=\{<a, b>, <b, c>, <c, d>, <d, e>, <e, f>\}$

(b)  $r=\{<a, b>, <b, c>, <b, d>, <d, e>, <d, f>\}$

(c)  $r=\{<a, b>, <b, c>, <d, a>, <d, b>, <d, e>\}$

2. 分析下列程序段的时间复杂度

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

(b)  $s=0$   
for( $i=0$  ;  $i<n$  ;  $i++$ )  
for( $j=0$  ;  $j<n$  ;  $j++$ )  
 $s+=b[i][j]$  ;

(c)  $i=1$   
While( $i<n$ )  
 $i*=2$  ;

3. 在数据结构中, 与所使用的计算机无关的是 \_\_\_\_\_。
- A. 存储结构    B. 物理结构    C. 物理和存储结构    D. 逻辑结构
4. 非线性结构中每个结点 \_\_\_\_\_。
- A. 无直接前驱结点    B. 只有一个直接前驱和直接后继结点
- C. 无直接后继结点    D. 可能有多个直接前驱和多个直接后继结点
5. 可以把数据的逻辑结构划分成 \_\_\_\_\_。
- A. 内部结构和外部结构    B. 动态结构和静态结构
- C. 紧凑结构和非紧凑结构    D. 线性结构和非线性结构

## 第二章 线性表

### 一、单项选择题

1. 下面关于线性表叙述中, 错误的是 (1)。
- (1) : A. 顺序表必须占用一片地址连续的存储单元  
B. 链表不必占用一片地址连续的存储单元  
C. 顺序表可以随机存取任一元素  
D. 链表可以随机存取任一元素
2. 在表长为  $n$  的单链表中, 算法时间复杂度为  $O(n)$  的操作是 (2)。
- (2) : A. 查找单链表中第  $i$  个结点    B. 在  $p$  结点之后插入一个结点  
C. 删除表中第一个结点    D. 删除  $p$  结点的直接后继结点
3. 单链表的存储密度 (3)。
- (3) : A. 大于 1    B. 等于 1    C. 小于 1    D. 不能确定
4. 在表长为  $n$  的顺序表中, 算法的时间复杂度为  $O(1)$  的操作是 (4)。
- (4) : A. 在第  $n$  个结点之后插入一个结点    B. 在第  $i$  个结点前插入一个新结点  
C. 删除第  $i$  个结点    D. 求表长
5. 在下列链表中不能从当前结点出发访问到其余各结点的是 ( 5 )。
- (5) : A. 单链表    B. 单循环链表    C. 双向链表    D. 双向循环链表
6. 在设头、尾指针的单链表中, 与长度  $n$  有关的操作是 ( 6 )。
- (6) : A. 删除第一个结点    B. 删除最后一个结点  
C. 在第一个结点之前插入一个结点    D. 在表尾结点后插入一个结点
7. 设  $p$  结点是带表头结点的双循环链表中的结点, 则
- (1) 在  $p$  结点后插入  $s$  结点的语句序列中正确的是 (7)。
- (7) : A.  $s \rightarrow \text{next} = p \rightarrow \text{next}$  ;  $p \rightarrow \text{next} \rightarrow \text{prior} = s$  ;  
 $p \rightarrow \text{next} = s$  ;  $s \rightarrow \text{next} = p \rightarrow \text{next}$  ;  
B.  $p \rightarrow \text{next} = s$  ;  $s \rightarrow \text{next} = p \rightarrow \text{next}$  ;  
 $p \rightarrow \text{next} \rightarrow \text{prior} = s$  ;  $s \rightarrow \text{next} = p$  ;  
C.  $p \rightarrow \text{next} = s$  ;  $p \rightarrow \text{next} \rightarrow \text{prior} = s$  ;  
 $s \rightarrow \text{next} = p \rightarrow \text{next}$  ;  $s \rightarrow \text{next} = p$  ;  
D.  $p \rightarrow \text{next} \rightarrow \text{prior} = s$  ;  $p \rightarrow \text{next} = s$  ;  
 $s \rightarrow \text{next} = p \rightarrow \text{next}$  ;  $s \rightarrow \text{next} = p$  ;
- (2) 在  $p$  结点之前插入  $s$  结点的语句序列中正确的是 (8)。
- (8) : A.  $s \rightarrow \text{prior} = p \rightarrow \text{prior}$  ;  $p \rightarrow \text{prior} \rightarrow \text{next} = s$  ;  
 $p \rightarrow \text{prior} = s$  ;  $s \rightarrow \text{next} = p$  ;  
B.  $p \rightarrow \text{prior} = s$  ;  $p \rightarrow \text{prior} \rightarrow \text{next} = s$  ;  
 $s \rightarrow \text{prior} = p \rightarrow \text{prior}$  ;  $s \rightarrow \text{next} = p$  ;  
C.  $p \rightarrow \text{prior} \rightarrow \text{next} = s$  ;  $p \rightarrow \text{prior} = s$  ;  
 $s \rightarrow \text{prior} = p \rightarrow \text{prior}$  ;  $s \rightarrow \text{next} = p$  ;  
D.  $p \rightarrow \text{prior} = s$  ;  $s \rightarrow \text{next} = p$  ;  
 $p \rightarrow \text{prior} \rightarrow \text{next} = s$  ;  $s \rightarrow \text{prior} = p \rightarrow \text{prior}$  ;
8. 下列关于链表说法错误的是 (9)。
- (9) : A. 静态链表中存取每一个元素的时间相同  
B. 动态链表中存取每一个元素的时间不同  
C. 静态链表上插入或删除一个元素不必移动其他元素  
D. 动态链表上插入或删除一个元素不必移动其他元素



9. 在链表中最常用的操作是在最后一个数据元素之后插入一个数据元素和删除第一个数据元素, 则最节省运算时间的存储方式是 (10) 。

- (10) : A. 仅有头指针的单链表      B. 仅有头指针的单循环链表  
C. 仅有头指针的双向链表      D. 仅有尾指针的单循环链表

## 二、填空题

1. 单链表中每个结点需要两个域, 一个是数据域, 另一个是 (1) 。

2. 链表相对于顺序表的优点是 (2) 和 (3) 操作方便。

3. 表长为  $n$  的顺序表中, 若在第  $j$  个数据元素 ( $1 \leq j \leq n+1$ ) 之前插入一个数据元素, 需要向后移动 (4) 个数据元素; 删除第  $j$  个数据元素需要向前移动 (5) 个数据元素; 在等概率的情况下, 插入一个数据元素平均需要移动 (6) 个数据元素, 删除一个数据元素平均需要移动 (7) 个数据元素。

4. 单链表  $h$  为空表的条件是 (8) 。

5. 带表头结点的单链表  $h$  为空表的条件是 (9) 。

6. 在非空的单循环链表  $h$  中, 某个  $p$  结点为尾结点的条件是 (10) 。

7. 在非空的双循环链表中, 已知  $p$  结点是表中任一结点, 则

(1) 在  $p$  结点后插入  $s$  结点的语句序列是:

$s \rightarrow \text{next} = p \rightarrow \text{next}$  ;  $s \rightarrow \text{prior} = p$  ; (11) ; (12)

(2) 在  $p$  结点前插入  $s$  结点的语句序列是:

$s \rightarrow \text{prior} = p \rightarrow \text{prior}$  ;  $s \rightarrow \text{next} = p$  ; (13) ; (14)

(3) 删除  $p$  结点的直接后继结点的语句序列是:

$q = p \rightarrow \text{next}$  ;  $p \rightarrow \text{next} = q \rightarrow \text{next}$  ; (15) ;  $\text{free}(q)$  ;

(4) 删除  $p$  结点的直接前驱结点的语句序列是:

$q = p \rightarrow \text{prior}$  ;  $p \rightarrow \text{prior} = q \rightarrow \text{prior}$  ; (16) ;  $\text{free}(q)$  ;

(5) 删除  $p$  结点的语句序列是:

$p \rightarrow \text{prior} \rightarrow \text{next} = p \rightarrow \text{next}$  ; (17) ;  $\text{free}(q)$  ;

8. 在带有尾指针  $r$  的单循环链表中, 在尾结点后插入  $p$  结点的语句序列是 (18) ; (19) ; 删除第一个结点的语句序列是  $q = r \rightarrow \text{next}$  ; (20) ;  $\text{free}(q)$  。

## 三、应用题

1. 简述顺序表和链表各自的优点。

2. 简述头指针和头结点的作用。

## 四、算法设计题

1. 请编写一个算法, 实现将  $x$  插入到已按数据域值从小到大排列的有序表中。

2. 设计一个算法, 计算单链表中数据域值为  $x$  的结点个数。

3. 设计一个用前插法建立带表头结点的单链表的算法。

4. 请编写一个建立单循环链表的算法。

5. 设计一个算法, 实现将带头结点的单链表进行逆置。

6. 编写一个算法, 实现以较高的效率从有序顺序表  $A$  中删除其值在  $x$  和  $y$  之间 ( $x \leq A[i] \leq y$ ) 的所有元素。

## 第三章 栈和队列

### 一、选择题

1. 插入和删除只能在表的一端进行的线性表, 称为 (1) 。

(1) : A. 队列    B. 循环队列    C. 栈    D. 双栈

2. 队列操作应遵循的原则是 (2) 。

(2) : A. 先进先出    B. 后进先出    C. 先进后出    D. 随意进出

3. 栈操作应遵循的原则是 (3) 。

(3) : A. 先进先出    B. 后进后出    C. 后进先出    D. 随意进出

4. 设队长为  $n$  的队列用单循环链表表示且仅设头指针, 则进队操作的时间复杂度为 (4) 。

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

5. 设栈  $s$  和队列  $q$  均为空, 先将  $a, b, c, d$  依次进队列  $q$ , 再将队列  $q$  中顺次出队的元素进栈  $s$ , 直至队空。再将栈  $s$  中的元素逐个出栈, 并将出栈元素顺次进队列  $q$ , 则队列  $q$  的状态是 (5) 。

- (5) : A. abcd B. dcba C. bcad D. dbca
6. 若用一个大小为 6 的数组来实现循环队列, 且当前 front 和 rear 的值分别为 3 和 0, 当从队列中删除一个元素, 再加入两个元素后, front 和 rear 的值分别为 (6)。
- (6) : A. 5 和 1 B. 4 和 2 C. 2 和 4 D. 1 和 5
7. 一个栈的入栈序列是 a,b,c,d,e, 则栈的不可能的输出序列是 (7)。
- (7) : A.edcba B.decba C.dceab D.abcde

## 二、填空题

1. 在栈结构中, 允许插入、删除的一端称为 (1), 另一端称为 (2)。
2. 在队列结构中, 允许插入的一端称为 (3), 允许删除的一端称为 (4)。
3. 设长度为 n 的链队列用单循环链表表示, 若只设头指针, 则进队和出队操作的时间复杂度分别是 (5) 和 (6); 若只设尾指针, 则进队和出队操作的时间复杂度分别为 (7) 和 (8)。
4. 设用少用一个元素空间的数组 A[m] 存放循环队列, front 指向实际队首, rear 指向新元素应存放的位置, 则判断队空的条件是 (9), 判断队满的条件是 (10), 当队未满时, 循环队列的长度是 (11)。
5. 两个栈共享一个向量空间时, 可将两个栈底分别设在 (12)。

## 三、应用题

1. 简述线性表、栈和队列有什么异同?
2. 循环队列的优点是什么? 设用数组 A[m] 来存放循环队列, 如何判断队满和队空。
3. 若进栈序列为 abcd, 请给出全部可能的出栈序列和不可能的出栈序列。
4. 设栈 s 和队列 q 初始状态为空, 元素 a, b, c, d, e 和 f, 依次通过栈 s, 一个元素出栈后即进入队列, 若 6 个元素出队的序列是 bdcfea, 则栈 s 的容量至少应该存多少个元素?
5. 已知一个中缀算术表达式为  $3 + 4 / (25 - (6 + 15)) * 8$  写出对应的后缀算术表达式 (逆波兰表达式)。

## 四、算法设计题

1. 已知 q 是一个非空顺序队列, s 是一个顺序栈, 请设计一个算法, 实现将队列 q 中所有元素逆置。

2. 已知递归函数:

$$F(n) = \begin{cases} 1 & \text{当 } n=0 \text{ 时} \\ n \cdot F(n/2) & \text{当 } n>0 \text{ 时} \end{cases}$$

(1) 写出求 F(n) 递归算法;

(2) 写出求 F(n) 的非递归算法。

3. 假设以带头结点的循环链表表示队列, 并且仅设一个指针指向队尾元素结点 (注意不设头指针), 试编写相应的队列初始化、入队列和出队列的算法。

## 第四章 串、数组和广义表

### 一、选择题

1. 串的模式匹配是指 (1)。
- (1) : A. 判断两个串是否相等  
B. 对两个串进行大小比较  
C. 找某字符在主串中第一次出现的位置  
D. 找某子串在主串中第一次出现的第一个字符位置
2. 设二维数组 A[m][n], 每个数组元素占用 d 个存储单元, 第一个数组元素的存储地址是如 Loc(a[0][0]), 求按行优先顺序存放的数组元素 a[i][j] (0 ≤ i ≤ m-1, 0 ≤ j ≤ n-1) 的存储地址 (2)。
- (2) : A. Loc(a[0][0]) + [(i-1)\*n+j-1]\*d  
B. Loc(a[0][0]) + [i\*n+j]\*d



- C .  $\text{Loc}(a[0][0]+[j*m+i]*d)$   
D .  $\text{Loc}(a[0][0]+[(j-1)*m+i-1]*d)$
- 3 . 设二维数组  $A[m][n]$  , 每个数组元素占  $d$  个存储单元, 第 1 个数组元素的存储地址是  $\text{Loc}(a[0][0])$  , 求按列优先顺序存放的数组元素  $a[j][i](0 \leq i \leq m-1, 0 \leq j \leq n-1)$  的存储地址 (3) 。
- (3) : A .  $\text{Loc}(a[0][0]+[(i-1)*n+j-1]*d)$   
B .  $\text{Loc}(a[0][0]+[i*n+j]*d)$   
C .  $\text{Loc}(a[0][0]+[(j-1)*m+i]*d)$   
D .  $\text{Loc}(a[0][0]+[j*m+i]*d)$
- 4 . 已知二维数组  $A[6][10]$  , 每个数组元素占 4 个存储单元, 若按行优先顺序存放数组元素  $a[3][5]$  的存储地址是 1000, 则  $a[0][0]$  的存储地址是 (4) 。
- (4) : A . 872 B . 860 C . 868 D . 864
- 5 . 若将  $n$  阶上三角矩阵  $A$ , 按列优先顺序压缩存放在一维数组  $F[n(n+1)/2]$  中, 第 1 个非零元素  $a_{11}$  存于  $F[0]$  中, 则应存放到  $F[K]$  中的非零元素  $a_{ij}(1 \leq i \leq n, 1 \leq j \leq i)$  的下标  $i, j$  与  $K$  的对应关系是 (5) 。
- (5) : A .  $i(i+1)/2+j$  B .  $i(i-1)/2+j-1$   
C .  $j(j+1)/2+j$  D .  $j(j-1)/2+i-1$
- 6 . 若将  $n$  阶下三角矩阵  $A$ , 按列优先顺序压缩存放在一维数组  $F[n(n+1)/2]$  中, 第一个非零元素  $a_{11}$  存于  $F[0]$  中, 则应存放到  $F[K]$  中的非零元素  $a_{ij}(1 \leq j \leq n, 1 \leq j \leq i)$  的下标  $i, j$  与  $K$  的对应关系是 (6) 。
- (6): A.  $(2n-j+1)j/2+i-j$  B.  $(2n-j+2)(j-1)/2+i-j$   
C .  $(2n-i+1)i/2+j-i$  D .  $(2n-i+2)i/2+j-i$
- 7 . 设有 10 阶矩阵  $A$ , 其对角线以上的元素  $a_{ij}(1 \leq j \leq 10, 1 \leq i < j)$  均取值为 -3, 其他矩阵元素为正整数, 现将矩阵  $A$  压缩存放在一维数组  $F[m]$  中, 则  $m$  为 (7) 。
- (7) : A . 45 B . 46 C . 55 D . 56
- 8 . 设广义表  $L=(a, b, L)$  其深度是 (8) 。
- (8) : A . 3 B .  $\infty$  C . 2 D . 都不对
- 9 . 广义表  $B: (d)$  , 则其表尾是 (9) , 表头是 (10) 。
- (9) —(10) : A .  $d$  B .  $()$  C .  $(d)$  D .  $((d))$
- 10 . 下列广义表是线性表的有 (11) 。
- (11) : A .  $Ls=(a, (b, c))$  B .  $Ls=(a, Ls)$   
C .  $Ls=(a, b)$  D .  $Ls=(a, ((d)))$
- 11 . 一个非空广义表的表尾 (12) 。
- (12) : A . 只能是子表 B . 不能是子表  
C . 只能是原子元素 D . 可以是原子元素或子表
- 12 . 已知广义表  $A=((a, (b, c)), (a, (b, c), d))$  , 则运算  $\text{head}(\text{head}(\text{tail}(A)))$  的结果是 (13) 。
- (13) : A .  $a$  B .  $(b, c)$  C .  $(a, (b, c))$  D .  $d$

## 二、填空题

- 1 . 两个串相等的充分必要条件是 (1) 。
- 2 . 空串是 (2) , 其长度等于 (3) 。
- 3 . 设有串  $S="good"$  ,  $T="morning"$  , 求:
- (1) $\text{concat}(S, T)=$  (4) ;  
(2) $\text{substr}(T, 4, 3)=$  (5) ;  
(3) $\text{index}(T, "n")=$  (6) ;  
(4) $\text{replace}(S, 3, 2, "to")=$  (7) 。
- 4 . 若  $n$  为主串长,  $m$  为子串长, 则用简单模式匹配算法最好情况下, 需要比较字符总数是 (8) , 最坏情况下, 需要比较字符总数是 (9) 。
- 5 . 设二维数组  $A[8][10]$  中, 每个数组元素占 4 个存储单元, 数组元素  $a[2][2]$  按行优先顺序存放的存储地址是 1000, 则数组元素  $a[0][0]$  的存储地址是 (10) 。
- 6 . 设有矩阵

$$A = \begin{bmatrix} 8 & -3 & -3 & -3 \\ 4 & 2 & -3 & -3 \\ 6 & 5 & 7 & -3 \\ 1 & 3 & 9 & 11 \end{bmatrix}$$

压缩存储到一维数组  $F[m]$  中, 则  $m$  为 (11),  $-3$  应存放到  $F[k_1]$  中,  $k_1$  为 (12), 元素  $a_{ij}(1 \leq i \leq 4, 1 \leq j \leq i)$  按行优先顺序存放到  $F[k_2]$  中,  $k_2$  为 (13), 按列优先顺序存放到  $F[k_3]$  中,  $k_3$  为 (14)。

7. 广义表  $LS=(a, (b), ((c, (d))))$  的长度是 (15), 深度是 (16), 表头是 (17), 表尾是 (18)。

8. 稀疏矩阵的压缩存储方法通常有两种, 分别是 (19) 和 (20)。

9. 任意一个非空广义表的表头可以是原子元素, 也可以是 (21), 而表尾必定是 (22)。

### 三、应用题

1. 已知  $S="xyz)+*"$  试利用联接 ( $\text{concat}(S, T)$ ), 取子串 ( $\text{substr}(S, i, j)$ ) 和置换 ( $\text{replace}(S, i, j, T)$ ) 基本操作将  $S$  转化为  $T="(x+2)*y"$ 。

2. 设串  $S$  的长度为  $n$ , 其中的字符各不相同, 求  $S$  中互异的非平凡子串 (非空且不同于  $S$  本身) 的个数。

3. 设模式串  $T="abcaaccbaca"$ , 请给出它的  $\text{next}$  函数及  $\text{next}$  函数的修正值  $\text{nextval}$  之值。

4. 特殊矩阵和稀疏矩阵哪一种压缩存储会失去随机存储功能?

5. 设  $n$  阶对称矩阵  $A$  压缩存储于一维数组  $F[m]$  中, 矩阵元素  $a_{ij}(1 \leq i \leq n, 1 \leq j \leq n)$ , 存于  $F[k](0 \leq k < m)$  中, 当用行优先顺序转换时的对应关系是什么? 用列优先顺序转换时的对应关系是什么?

## 第五章 树和二叉树

### 一、单项选择题

1. 关于二叉树的下列说法正确的是 (1)。

(1) : A. 二叉树的度为 2 B. 二叉树的度可以小于 2

C. 每一个结点的度都为 2 D. 至少有一个结点的度为

2. 设深度为  $h(h>0)$  的二叉树中只有度为 0 和度为 2 的结点, 则此二叉树中所含的结点总数至少为 (2)。

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

3. 在树中, 若结点  $A$  有 4 个兄弟, 而且  $B$  是  $A$  的双亲, 则  $B$  的度为 (3)。

(3) : A. 3 B. 4 C. 5 D. 6

4. 若一棵完全二叉树中某结点无左孩子, 则该结点一定是 (4)。

(4) : A. 度为 1 的结点 B. 度为 2 的结点 C. 分支结点 D. 叶子结点

5. 深度为  $k$  的完全二叉树至多有 (5) 个结点, 至少有 (6) 个结点。

(5)-(6) : A.  $2^{k-1}-1$  B.  $2^{k-1}$  C.  $2^k-1$  D.  $2^k$

6. 前序序列为 ABC 的不同二叉树有 (7) 种不同形态。

(7) : A. 3 B. 4 C. 5 D. 6

7. 若二叉树的前序序列为 DABCEFG 中序序列为 BACDFGE 则其后序序列为 (8), 层次序列为 (9)。

(8)-(9) : A. BCAGFED B. DAEBCFG C. ABCDEFG D. BCAEFGD

8. 在具有 200 个结点的完全二叉树中, 设根结点的层次编号为 1, 则层次编号为 60 的结点, 其左孩子结点的层次编号为 (10), 右孩子结点的层次编号为 (11), 双亲结点的层次编号为 (12)。

(10)-(12) : A. 30 B. 60 C. 120 D. 121

9. 遍历一棵具有  $n$  个结点的二叉树, 在前序序列、中序序列和后序序列中所有叶子结点的相对次序 (13)。

(13) : A. 都不相同 B. 完全相同 C. 前序和中序相同 D. 中序与后序相同

10. 在由 4 棵树组成的森林中, 第一、第二、第三和第四棵树组成的结点个数分别为 30, 10, 20, 5, 当把森林转换成二叉树后, 对应的二叉树中根结点的左子树中结点个数为 (14), 根结点的右子树中结点个数为 (15)。

(14) —(15) : A. 20 B. 29 C. 30 D. 35

11. 具有  $n$  个结点 ( $n>1$ ) 的二叉树的前序序列和后序序列正好相反, 则该二叉树中除叶子结点外每个结点 (16)。

(16) : A. 仅有左孩子 B. 仅有右孩子 C. 仅有一个孩子 D. 都有左、右孩子

12. 判断线索二叉树中  $p$  结点有右孩子的条件是 (17)。

(17) : A.  $p!=\text{NULL}$  B.  $p->\text{rchild}!=\text{NULL}$  C.  $p->\text{rtag}=0$  D.  $p->\text{rtag}=1$

13 . 将一棵树转换成二叉树，树的前根序列与其对应的二叉树的 (18) 相等。树的后根序列与其对应的二叉树的 (19) 相同。

(18) —(19) : A . 前序序列 B . 中序序列 C . 后序序列 D . 层次序列

14 . 设数据结构  $(D, R)$ ,  $D=\{d1, d2, d3, d4, d5, d6\}$ ,  $R=\{<d4, d2>, <d2, d1>, <d2, d3>, <d4, d6>, <d6, d5>\}$ , 这个结构的图形是 (20) ; 用 (21) 遍历方法可以得到序列  $\{d1, d2, d3, d4, d5, d6\}$ 。

(20) : A . 线性表 B . 二叉树 C . 队列 D . 栈

(21) : A . 前序 B . 中序 C . 后序 D . 层次

15 . 对于树中任一结点  $x$ , 在前根序列中序号为  $pre(x)$ , 在后根序列中序号为  $post(x)$ , 若树中结点  $x$  是结点  $y$  的祖先, 下列 (22) 条件是正确的。

(22) : A.  $pre(x)<pre(y)$  且  $post(x)<post(y)$

B .  $pre(x)<pre(y)$  且  $post(x)>post(y)$

C.  $pre(x)>pre(y)$  且  $post(x)<post(y)$

D .  $pre(x)>pre(y)$  且  $post(x)>post(y)$

16 . 每棵树都能惟一地转换成对应的二叉树, 由树转换的二叉树中, 一个结点  $N$  的左孩子是它在原树对应结点的 (23), 而结点  $N$  的右孩子是它在原树里对应结点的 (24)。

(23) —(24) : A . 最左孩子 B . 最右孩子 C . 右邻兄弟 D . 左邻兄弟

17 . 二叉树在线索化后, 仍不能有效求解的问题是 (25)。

(25) : A . 前序线索树中求前序直接后继结点

B . 中序线索树中求中序直接前驱结点

C . 中序线索树中求中序直接后继结点

D . 后序线索树中求后序直接后继结点

18 . 一棵具有 124 个叶子结点的完全二叉树, 最多有 (26) 个结点。

(26) : A . 247 B . 248 C . 249 D . 250

19 . 实现任意二叉树的后序遍历的非递归算法而不使用栈结构, 最有效的存储结构是采用 (27)。

(27) : A . 二叉链表 B . 孩子链表 C . 三叉链表 D . 顺序表

## 二、填空题

1. 树中任意结点允许有 (1) 孩子结点, 除根结点外, 其余结点 (2) 双亲结点。

2. 若一棵树的广义表表示为  $A(B(E, F), C(C(H, I, J, K), L), D(M(N)))$ 。则该树的度为 (3), 树的深度为 (4), 树中叶子结点个数为 (5)。

3 . 若树  $T$  中度为 1、2、3、4 的结点个数分别为 4、3、2、2, 则  $T$  中叶子结点的个数是 (6)。

4 . 一棵具有  $n$  个结点的二叉树, 若它有  $m$  个叶子结点, 则该二叉树中度为 1 的结点个数是 (7)。

5 . 深度为  $k(k>0)$  的二叉树至多有 (8) 个结点, 第  $i$  层上至多有 (9) 个结点。

6 . 已知二叉树有 52 个叶子结点, 度为 1 的结点个数为 30 则总结点个数为 (10)。

7 . 已知二叉树中有 30 个叶子结点, 则二叉树的总结点个数至少是 (11)。

8 . 高度为 6 的完全二叉树至少有 (12) 个结点。

9 . 一个含有 68 个结点的完全二叉树, 它的高度是 (13)。

10 . 已知一棵完全二叉树的第 6 层上有 6 个结点 (根结点的层数为 1), 则总的结点个数至少是 (14), 其中叶子结点个数是 (15)。

11 . 已知完全二叉树第 6 层上有 10 个叶子结点, 则这棵二叉树的结点总数最多是 (16)。

12 . 一棵树转换成二叉树后, 这棵二叉树的根结点一定没有 (17) 孩子, 若树中有  $m$  个分支结点, 则与其对应的二叉树中无右孩子的结点个数为 (18)。

13 . 若用二叉链表示具有  $n$  个结点的二叉树, 则有 (19) 个空链域。

14 . 具有  $m$  个叶子结点的哈夫曼树, 共有 (20) 个结点。

15 . 树的后根遍历序列与其对应的二叉树的 (21) 遍历序列相同。

16 . 线索二叉树的左线索指向其 (22), 右线索指向其 (23)。

## 三、应用题

1 . 具有  $n$  个结点的满二叉树的叶子结点个数是多少 ?

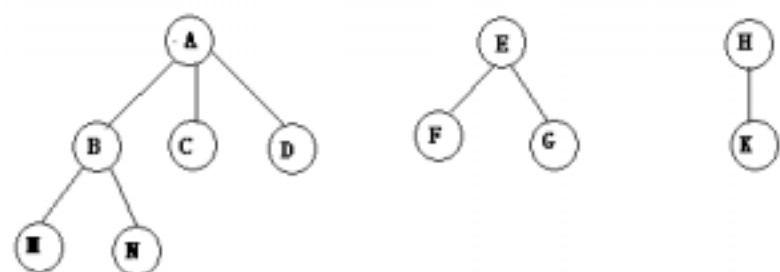
2 . 列出前序遍历序列是 ABC 的所有不同的二叉树。

3 . 已知二叉树的层次遍历序列为 ABCDEFGHIJK 中序序列为 DBGEHJACIKF 请构造一棵二叉树。

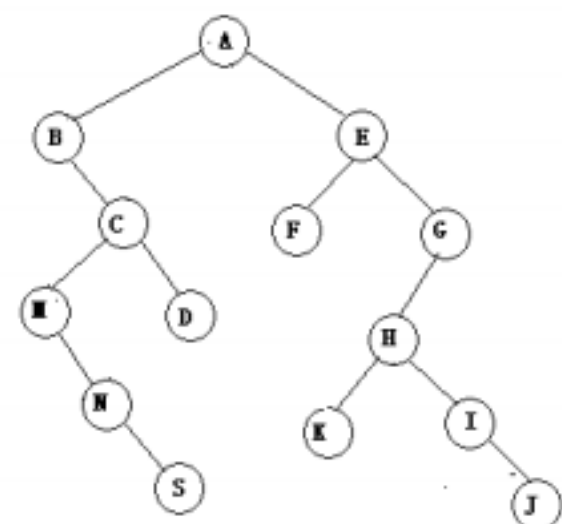
- 4 . 已知二叉树的中序遍历序列是 ACBDGHFE后序遍历序列是 ABDCFHEG请构造一棵二叉树。
- 5 . 已知二叉树的前序、中序和后序遍历序列如下，其中有一些看不清的字母用 \* 表示，请先填写 \* 处的字母，再构造一棵符合条件的二叉树，最后画出带头结点的中序线索链表。

- (1) 前序遍历序列是： \*BC\*\*\*G\*
- (2) 中序遍历序列是： CB\*EAGH\*
- (3) 后序遍历序列是： \*EDB\*\*FA

- 6 . 将下图所示的森林转换成一棵二叉树，并画出这棵二叉树的顺序存储结构。



- 7 . 将下图所示的二叉树还原成森林。



- 8 . 对于给定的一组权值 {3, 5, 6, 7, 9}，请构造相应的哈夫曼树，并计算其带权路径长度。

#### 四、算法设计题

- 1 . 请设计一个算法，求以孩子兄弟链表表示的树中叶子结点个数。
- 2 . 请编写一个算法，实现将以二叉链表存储的二叉树中所有结点的左、右孩子进行交换。
- 3 . 请编写一个算法，将以二叉链表存储的二叉树输出其广义表表示形式。
- 4 . 请编写一个算法，判断以二叉链表存储的二叉树是否为完全二叉树。
- 5 . 假设二叉树采用链接存储方式存储，编写一个二叉树先序遍历和后序遍历的非递归算法。
- 6 . 已知一棵二叉树用二叉链表存储， t 指向根结点， p 指向树中任一结点。请编写一个非递归算法，实现求从根结点 t 到结点 p 之间的路径。

## 第六章 图

### 一、单项选择题

- 1 . 下面关于图的存储结构的叙述中正确的是 (1)。
- (1) : A . 用邻接矩阵存储图占用空间大小只与图中顶点有关，与边数无关  
B . 用邻接矩阵存储图占用空间大小只与图中边数有关，而与顶点数无关  
C . 用邻接表存储图占用空间大小只与图中顶点数有关，而与边数无关  
D . 用邻接表存储图占用空大小只与图中边数有关，而与顶点数无关
- 2 . 下面关于对图的操作的说法不正确的是 (2)。
- (2) : A: 寻找关键路径是关于带权有向图的操作  
B . 寻找关键路径是关于带权无向图的操作  
C . 连通图的生成树不一定是惟一的  
D . 带权无向图的最小生成树不一定是惟一的
- 3 . 下面的各种图中，哪个图的邻接矩阵一定是对称的 (3)。
- (3) : A . AOE网 B . AOV网 C . 无向图 D . 有向图
- 4 . 在 AOE网中关于关键路径叙述正确的是 (4)。
- (4) : A . 从开始顶点到完成顶点的具有最大长度的路径，关键路径长度是完成整个工程所需的最短时间  
B . 从开始顶点到完成顶点的具有最小长度的路径，关键路径长度是完成整个工程所需的最短时间

C. 从开始顶点到完成顶点的具有最大长度的路径，关键路径长度是完成整个工程所需的最长时间

D. 从开始顶点到完成顶点的具有最小长度的路径，关键路径长度是完成整个工程所需的最短时间

5. 一个具有  $n$  个顶点  $e$  条边的图中，所有顶点的度数之和等于 (5)。

(5) : A.  $n$  B.  $2n$  C.  $e$  D.  $2e$

6. 具有 8 个顶点的无向图最多有 (6) 条边。

(6) : A. 8 B. 28 C. 56 D. 72

7. 具有 8 个顶点的有向图最多有 (7) 条边。

(7) : A. 8 B. 28 C. 56 D. 78

8. 深度优先遍历类似于二叉树的 (8)。

(8) : A. 前序遍历 B. 中序遍历 C. 后序遍历 D. 层次遍历

9. 广度优先遍历类似于二叉树的 (9)。

(9) : A. 前序遍历 B. 中序遍历 C. 后序遍历 D. 层次遍历

10. 任一个连通图的生成树 (10)。

(10) : A. 可能不存在 B. 只有一棵 C. 一棵或多棵 D. 一定有多棵

11. 下列关于连通图的 BFS 和 DFS 生成树高度论述正确的是 (11)。

- (11) : A. BFS 生成树高度 < DFS 生成树的高度  
B. BFS 生成树高度 = DFS 生成树的高度  
C. BFS 生成树高度 > DFS 生成树的高度  
D. BFS 生成树高度 ≤ DFS 生成树的高度

12.  $G$  是一个非连通无向图，共有 28 条边，则该图至少有解 (12) 个顶点。

(12) : A. 7 B. 8 C. 9 D. 10

## 二、判断题

1. 一个图的邻接矩阵表示是惟一的。 ( )
2. 一个图的邻接表表示是惟一的。 ( )
3. 无向图的邻接矩阵一定是对称矩阵。 ( )
4. 有向图的邻接矩阵一定是对称矩阵。 ( )
5. 有向图用邻接表表示，顶点  $v_i$  的出度是对应顶点  $v_j$  链表中结点个数。 ( )
6. 有向图用邻接表表示，顶点  $v_i$  的度是对应顶点  $v_j$  链表中结点个数。 ( )
7. 有向图用邻接矩阵表示，删除所有从顶点  $i$  出发的弧的方法是，将邻接矩阵的  $i$  行全部元素置为 0。 ( )
8. 若从无向图中任一顶点出发，进行一次深度优先搜索，就可以访问图中所有顶点，则该图一定是连通的。 ( )
9. 在非连通图的遍历过程中，调用深度优先搜索算法的次数等于图中连通分量的个数。 ( )
10. 具有  $n$  个顶点的有向强连通图的邻接矩阵中至少有  $n$  个小于 1 的非零元素。 ( )
11. 一个连通图的生成树是一个极小连通子图。 ( )
12. 在有数值相同的权值存在时，带权连通图的最小生成树可能不惟一。 ( )
13. 在 AOE 网中仅存在一条关键路径。 ( )
14. 若在有向图的邻接矩阵中，主对角线以下的元素均为 0，则该图一定存在拓扑有序序列。 ( )
15. 若图  $G$  的邻接表表示时，表中有奇数个边结点，则该图一定是有向图。 ( )
16. 在 AOE 网中，任何一个关键活动提前完成，整个工程都会提前完成。 ( )
17. 图的深度优先遍历序列一定是惟一的。 ( )
18. 有向无环图的拓扑有序序列一定是惟一的。 ( )
19. 拓扑排序输出的顶点个数小于图中的顶点个数，则该图一定存在环。 ( )

## 三、填空题

1. 在一个具有  $n$  个顶点的完全无向图和完全有向图中分别包含有 (1) 和 (2) 条边。
2. 具有  $n$  个顶点的连通图至少具有 (3) 条边。
3. 具有  $n$  个顶点  $e$  条边的有向图和无向图用邻接表表示，则邻接表的边结点个数分别为 (4) 和 (5) 条。
4. 在有向图的邻接表和逆邻接表中，每个顶点链表中链接着该顶点的所有 (6) 和 (7) 结

点。

5 . 若  $n$  个顶点的连通图是一个环，则它有 (8) 棵生成树。

6 . 图的逆邻接表存储结构只适用于 (9) 。

7 .  $n$  个顶点  $e$  条边的图采用邻接矩阵存储，深度优先遍历算法的时间复杂度是 (10) ，广度优先算法的时间复杂度是 (11) 。

8 .  $n$  个顶点  $e$  条边的图，采用邻接表存储，广度优先遍历算法的时间复杂度是 (12) ，深度优先遍历算法的时间复杂度是 (13) 。

9 . 若要求一个稀疏图的最小生成树，最好用 (14) 算法求解。

10 . 若要求一个稠密图的最小生成树，最好用 (15) 算法求解。

#### 四、应用题

1 . 已知图  $G=(V, E)$ ，其中  $V=\{a, b, c, d, e\}$ ， $E=\{(a, b), \{a, c\}, \{a, d\}, (b, c), (d, c), (b, e), (c, e), (d, e)\}$  要求：

- (1) 画出图  $G$ ；
- (2) 给出图  $G$  的邻接矩阵；
- (3) 给出图  $G$  的邻接表；
- (4) 给出图  $G$  的所有拓扑有序序列。

2 . 已知带权无向图的邻接表见下图，要求：

0	a	→	1   9	→	3   3	→	4   2	→	5   8	→	^
1	b	→	0   9	→	2   2	→	5   3	→	^		
2	c	→	1   2	→	3   7	→	6   8	→	^		
3	d	→	0   3	→	2   7	→	4   4	→	6   9	→	^
4	e	→	0   2	→	3   4	→	5   4	→	6   1	→	^
5	f	→	0   8	→	1   3	→	4   4	→	6   5	→	^
6	g	→	2   8	→	3   9	→	4   1	→	5   5	→	^

(1) 画出图  $G$ ；

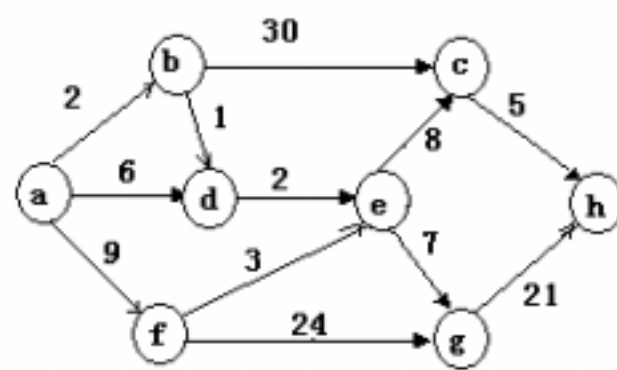
(2) 各画一棵从顶点  $a$  出发的深度优先生成树和广度优先生成树。

(3) 给出用 prim 算法从顶点  $a$  出发构造最少生成树的过程。

(4) 给出用 Kruscal 算法构造最小生成树的过程。

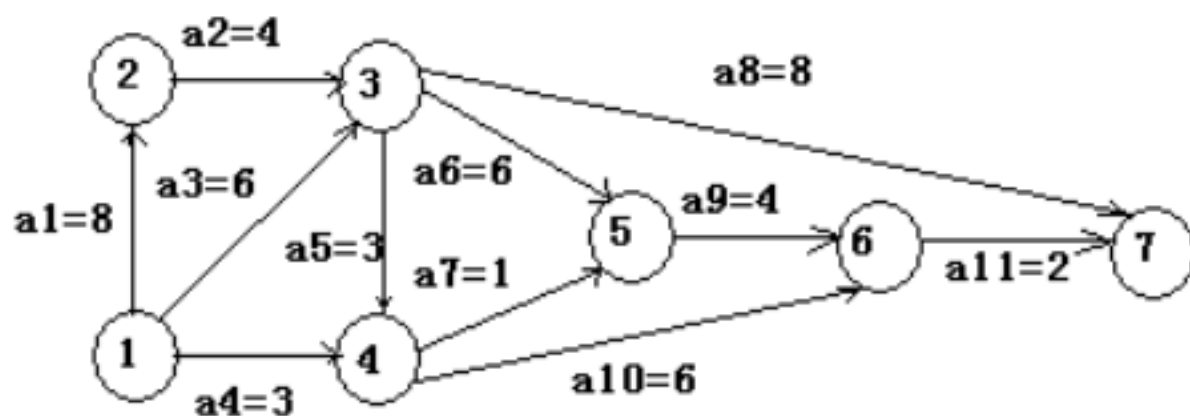
3 . 已知带权有向图如右图所示，要求：

- (1) 给出图  $G$  的邻接矩阵；
- (2) 给出图  $G$  的一个拓扑有序序列；
- (3) 求从顶点  $a$  出发到其余各顶点的最短路径。



4 . 已知带权有向图  $G$  见下图，图中边上的权活动  $a_i$  需要的天数，要求：

- (1) 求每项活动的最早和最晚开工时间；
- (2) 完成此项工程最少需要多少天；
- (3) 给出图  $G$  的关键路径。



#### 四、算法题

1. 编写根据无向图  $G$  的邻接表，判断图  $G$  是否连通的算法。

2. 编写根据有向图的邻接表，分别设计实现以下要求的算法：

- (1) 求出图  $G$  中每个顶点的出度；



- (2) 求出图 G 中出度最大的一个顶点，输出该顶点编号；  
 (3) 计算图 G 中出度为 0 的顶点数；  
 (4) 判断图 G 中是否存在边  $i, j$ 。

## 第七章 查找

### 一、单项选择题

1. 在长度为  $n$  的线性表中进行顺序查找，在等概率的情况下，查找成功的平均查找长度是 (1) 。
- (1) : A.  $n$  B.  $n(n+1)/2$  C.  $(n-1)/2$  D.  $(n+1)/2$
2. 在长度为  $n$  的顺序表中进行顺序查找，查找失败时需与键值比较次数是 (2) 。
- (2) : A.  $n$  B. 1 C.  $n-1$  D.  $n+1$
3. 对线性表进行顺序查找时，要求线性表的存储结构是 (3) 。
- (3) : A. 倒排表 B. 索引表 C. 顺序表或链表 D. 散列表
4. 对线性表用二分法查找时要求线性表必须是 (4) 。
- (4) : A. 顺序表 B. 单链表 C. 顺序存储的有序表 D. 散列表
5. 在有序表 A[80] 上进行二分法查找，查找失败时，需对键值进行最多比较次数是 (5) 。
- (5) : A. 20 B. 40 C. 10 D. 7
6. 在长度为  $n$  的有序顺序表中，采用二分法查找，在等概率的情况下，查找成功的平均查找长度是 (6) 。
- (6) : A.  $O(n^2)$  B.  $O(n \log_2 n)$  C.  $O(n)$  D.  $O(\log_2 n)$
7. 设顺序表为 {4, 6, 12, 32, 40, 42, 50, 60, 72, 78, 80, 90, 98}，用二分法查找 72，需要进行的比较次数是 (7) 。
- (7) : A. 2 B. 3 C. 4 D. 5
8. 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，则应采用的查找方法是 (8) 。
- (8) : A. 顺序查找 B. 二分法查找 C. 分块查找 D. 都不行
9. 在采用线性探查法处理冲突的散列表中进行查找，查找成功时所探测位置上的键值 (9) 。
- (9) : A. 一定都是同义词 B. 一定都不是同义词  
C. 不一定是同义词 D. 无任何关系
10. 在查找过程中，若同时还要做插入、删除操作，这种查找称为 (10) 。
- (10) : A. 静态查找 B. 动态查找 C. 内部查找 D. 外部查找

### 二、填空题

1. 在有序表 A[30] 上进行二分查找，则需要对键值进行 4 次、5 次比较查找成功的记录个数分别为 (1)、(2)，在等概率的情况下，查找成功的平均查找长度是 (3) 。
2. 散列法存储的基本思想是由 (4)，确定记录的存储地址。
3. 对一棵二叉排序树进行 (5) 遍历，可以得到一个键值从小到大次序排列的有序序列。
4. 对有序表 A[80] 进行二分查找，则对应的判定树高度为 (6)，判定树前 6 层的结点个数为 (7)，最高一层的结点个数是 (8)，查找给定值 K，最多与键值比较 (9) 次。
5. 对于表长为  $n$  的线性表要进行顺序查找，则平均时间复杂度为 (10)，若采用二分查找，则平均时间复杂度为 (11) 。
6. 在散列表中，装填因子。值越大，则插入记录时发生冲突的可能性 (12) 。
7. 在散列表的查找过程中与给定值 K 进行比较的次数取决于 (13)、(14) 和 (15)。
8. 在使用分块查找时，除表本身外，尚需建立一个 (16)，用来存放每一块中最高键值及 (17) 。
9. 若表中有 10000 个记录，采用分块查找时，用顺序查找确定记录所在的块，则分成 (18) 块最好，每块的最佳长度 (19)，在这种情况下，查找成功的平均检索长度为 (20) 。
10. 用  $n$  个键值构造一棵二叉排序树，最低高度是 (21) 。
11. 设有散列函数 H 和键值  $k_1, k_2$ ，若  $k_1 \neq k_2$ ，且  $H(k_1) = H(k_2)$ ，则称  $k_1, k_2$  是 (22) 。
12. 设有  $n$  个键值互为同义词，若用线性探查法处理冲突，把这  $n$  个键值存于表长为  $m(m > n)$  的散列表中，至少要进行 (23) 次探查。
13. 高度为 6 的平衡二叉排序树，其每个分支结点的平衡因子均为 0，则该二叉树共有 (24) 个结点。
14.  $m$  阶 B-树，除根结点外的分支结点最多有 (25) 棵子树，最少有 (26) 棵子树，最多有 (27) 个键值，最少有 (28) 个键值。



15 . m阶 B+树, 除根结点外的每个结点最多有 (29) 棵子树值, 最少有 (30) 棵子树。

### 三、应用题

1 . 画出对长度为 13 的有序表进行二分查找的一棵判定树, 并求其等概率时查找成功的平均查找长度。查找失败时需对键值的最多比较次数。

2 . 将整数序列 {8, 6, 3, 1, 2, 5, 9, 7, 4} 中的数依次插入到一棵空的二叉排序树中, 求在等概率情况下, 查找成功的平均查找长度, 查找失败时对键值的最多比较次数。再画出从二叉排序树中删除结点 6 和 8 的结果。

3 . 将整数序列 {8, 6, 3, 1, 2, 5, 9, 7, 41} 中的数依次插入到一棵空的平衡二叉排序树中, 求在等概率情况下, 查找成功的平均检索长度, 查找失败时对键值的最多比较次数。

4 . 按不同的输入顺序输入 4, 5, 6, 建立相应的不同形态的二叉排序树。

5 . 设有键值序列 {25, 40, 33, 47, 12, 66, 72, 87, 94, 22, 5, 58} 散列表长为 12, 散列函数为  $H(\text{key}) = \text{key} \% 11$ , 用拉链法处理冲突, 请画出散列表, 在等概率情况下, 求查找成功的平均查找长度和查找失败的平均查找长度。

6 . 已知一个散列函数为  $H(\text{key}) = \text{key} \% 13$ , 采用双散列法处理冲突,  $H1(\text{key}) = \text{key} \% 11 + 1$ , 探查序列为  $di = (H(\text{key}) + i * H1(\text{key})) \% m, i = 1, 2, \dots, m-1$ , 散列表见表 1, 要求回答下列问题:

(1) 对表中键值 21, 57, 45 和 50 进行查找时, 所需进行的比较次数各为多少?

(2) 在等概率情况下查找时, 查找成功的平均查找长度是多少?

0 1 2 3 4 5 6 7 8 9 10 11 12

						50	21		57		45	37	
--	--	--	--	--	--	----	----	--	----	--	----	----	--

表 1 散列表

### 四、算法设计题

1. 设二叉树用二叉链表表示, 且每个结点的键值互不相同, 请编写判别该二叉树是否为二叉排序树的非递归算法。

## 第八章 排序

### 一、单项选择题

1 . 对  $n$  个不同的记录按排序码值从小到大次序重新排列, 用冒泡 (起泡) 排序方法, 初始序列在 (1) 情况下, 与排序码值总比较次数最少, 在 (2) 情况下, 与排序码值总比较次数最多; 用直接插入排序方法, 初始序列在 (3) 情况下, 与排序码值总比较次数最少, 在 (4) 情况下, 与排序码值总比较次数最多; 用快速排序方法在 (5) 情况下, 与排序码值总比较次数最少, 在 (5) 情况下与排序码值总比较次数最多。

(1)-(6) :

A . 按排序码值从小到大排列 B . 按排序码值从大到小排列

C . 随机排列 (完全无序) D . 基本按排序码值升序排列

2 . 用冒泡排序方法对  $n$  个记录按排序码值从小到大排序时, 当初始序列是按排序码值从大到小排列时, 与码值总比较次数是 (7)。

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

3 . 下列排序方法中, 与排序码值总比较次数与待排序记录的初始序列排列状态无关的是 (8)。

(8) : A . 直接插入排序 B . 冒泡排序 C . 快速排序 D . 直接选择排序

4 . 将 6 个不同的整数进行排序, 至少需要比较 (9) 次, 至多需要比较 (10) 次。

(9)-(10) : A . 5 B . 6 C . 15 D . 21

5 . 若需要时间复杂度在  $O(n \log_2 n)$  内, 对整数数组进行排序, 且要求排序方法是稳定的, 则可选的排序方法是 (11)。

(11) : A . 快速排序 B . 归并排序 C . 堆排序 D . 直接插入排序

6 . 当待排序的整数是有序序列时, 采用 (12) 方法比较好, 其时间复杂度为  $O(n)$ , 而采用 (13) 方法却正好相反, 达到最坏情况下时间复杂度为  $O(n^2)$ ; 无论待排序序列排列是否有序, 采用 (14) 方法的时间复杂度都是  $O(n^2)$ 。

(12)-(14) : A . 快速排序 B . 冒泡排序 C . 归并排序 D . 直接选择排序

7 . 堆是一种 (15) 排序。

(15) : A . 插入 B . 选择 C . 交换 D . 归并

8 . 若一组记录的排序码值序列为 {40, 80, 50, 30, 60, 70}, 利用堆排序方法进行排序, 初建的大顶堆是 (16)。

- (16) : A. 80, 40, 50, 30, 60, 70 B. 80, 70, 60, 50, 40, 30  
C. 80, 70, 50, 40, 30, 60 D. 80, 60, 70, 30, 40, 50
9. 若一组记录的排序码值序列为 {50, 80, 30, 40, 70, 60} 利用快速排序方法, 以第一个记录为基准, 得到一趟快速排序的结果为 (17)。
- (17) : A. 30, 40, 50, 60, 70, 80 B. 40, 30, 50, 80, 70, 60  
C. 50, 30, 40, 70, 60, 80 D. 40, 50, 30, 70, 60, 80
10. 下列几种排序方法中要求辅助空间最大的是 (18)。
- (18) : A. 堆排序 B. 直接选择排序 C. 归并排序 D. 快速排序
11. 已知 A[m] 中每个数组元素距其最终位置不远, 采用下列 (19) 排序方法最节省时间。
- (19) : A. 直接插入 B. 堆 C. 快速 D. 直接选择
12. 设有 10000 个互不相等的无序整数, 若仅要求找出其中前 10 个最大整数, 最好采用 (20) 排序方法。
- (20) : A. 归并 B. 堆 C. 快速 D. 直接选择
13. 在下列排序方法中不需要对排序码值进行比较就能进行排序的是 (21)。
- (21) : A. 基数排序 B. 快速排序 C. 直接插入排序 D. 堆排序
14. 给定排序码值序列为 {F, B, J, C, E, A, I, D, C, H}, 对其按字母的字典序列的次序进行排列, 希尔 (Shell) 排序的第一趟 ( $d_1=5$ ) 结果应为 (22), 冒泡排序 (大数下沉) 的第一趟排序结果应为 (23), 快速排序的第一趟排序结果为 (24), 二路归并排序的第一趟排序结果是 (25)。
- (22)-(25) : A. {B, F, C, J, A, E, D, I, C, H}  
B. {C, B, D, A, E, F, I, C, J, H}  
C. {B, F, C, E, A, I, D, C, H, J}  
D. {A, B, D, C, E, F, I, J, C, H}

## 二、填空题

1. 内部排序方法按排序采用的策略可划分为五类: (1) (2) (3) (4) (5)。
2. 快速排序平均情况下的时间复杂度为 (6), 其最坏情况下的时间复杂度为 (7)。
3. 当待排序的记录个数  $n$  很大时, 应采用平均时间复杂度为 (8) 即 (9)、(10)、(11), 在这些方法中当排序码值是随机分布时, 采用 (12) 排序方法的平均时间复杂度最小。当希望排序方法是稳定时, 应采用 (13) 排序方法, 若只从节省空间考虑, 最节省空间的是 (14) 方法。
4. 对一组整数 {60, 40, 90, 20, 10, 70, 50, 80} 进行直接插入排序时, 当把第 7 个整数 50 插入到有序表中时, 为寻找插入位置需比较 (15) 次。
5. 从未排序序列中挑选最小 (最大) 元素, 并将其依次放到已排序序列的一端, 称为 (16) 排序。
6. 对  $n$  个记录进行归并排序, 所需要的辅助存储空间是 (17), 其平均时间复杂度是 (18), 最坏情况下的时间复杂度是 (19)。
7. 对  $n$  个记录进行冒泡排序, 最坏情况下的时间复杂度是 (20)。
8. 对 20 个记录进行归并排序时, 共需进行 (21) 趟归并, 在第三趟归并时是把最大长度为 (22) 的有序表两两归并为长度为 (23) 的有序表。

## 三、应用题

1. 举例说明本章介绍的排序方法中哪些是不稳定的 ?
2. 已知排序码值序列 {17, 18, 60, 40, 7, 32, 73, 65, 85}, 请写出冒泡排序每一趟的排序结果。
3. 对于排序码值序列 {10, 18, 14, 13, 16, 12, 11, 9, 15, 8}, 给出希尔排序 ( $d_1=5$ ,  $d_2=2$ ,  $d_3=1$ ) 的每一趟排序结果。
4. 判断下列序列是否为大顶堆? 若不是, 则把它们调整为大顶堆。  
(1) {90, 86, 48, 73, 35, 40, 42, 58, 66, 20}  
(2) {12, 70, 34, 66, 24, 56, 50, 90, 86, 36}

## 四、算法设计题

1. 在带表头结点的单链表上, 编写一个实现直接选择排序的算法。
2. 请编写一个快速排序的非递归算法。

## 附录：

大连理工大学 2002 年硕士入学试题

数据结构部分（共 50 分）

### 一、算法填空题（20 分）

1. 对以下函数填空，实现将头指针为  $h$  的单链表逆置，即原链表的第一个结点变成逆置后新链表的最后一个结点，原链表的第二个结点变成新链表的倒数第二个结点，如此等等，直到最后一个结点作为新链表的第一个结点，并返回指向该结点的指针。设单链表结点类型的定义为

```
typedef struct node
{int data ;
  struct node *next ;
}NODE ;
NODE *dlbzn(NODE*h)
{ NODE *p , *q ;
  q=NULL ;
  while(h)
  {
    p=h ;
    h=h->next ;
    _____ ;
    _____ ;
  }
  return q ;
}
```

2. 假设算术表达式由字符串  $b$  表示，其中可以包含三种括号：圆括号和方括号及花括号，其嵌套的顺序随意，如  $\{[]([])\}$ 。请对以下函数填空，实现判别给定表达式中所含括号是否正确配对出现的算法。

```
#define M 10
int khjc(char *b)
{ char sM} ;
  int i , j=0 , f=1 ;
  j=0 ;
  for(i=0 ; f&& b[i] !=? 0? ; i++)
  { switch(b[i])
    { case '?': _____ ; break ;
      case '[': _____ ; break ;
      case '{': _____ ; break ;
      case ')': _____ ;
      case ']': _____ ;
      case '?': if(j==0;||b[i]!= _____) f=0 ;
    }
  }
  return f&& _____ ;
}
```

3. 对以下函数填空，实现以带头结点的单链表  $h$  为存储结构的直接选择排序。设单链表结点类型的定义为

```
typedef struct node
{ int key ;
  struct node *next ;
}NODE ;
void pxx(NODE *h)
{ NODE *p , *q , *m ;
  int z ;
  p=h->next ;
  while(p!=NULL)
  { q=p->next ;
```

```

        m=p ;
        while(q!=NULL)
            { if(m->key>q->key)      _____ ;
              _____ ;
            }
        if(p!=m)
        { x=p->key ;
          p->key=m->key ;
          m->key=x ;
        }
        _____ ; }
    }

```

## 二、算法设计题 (30 分)

请用类 C 或类 PASCAL 语言设计实现下列功能的算法。

1. 设二叉排序树以二叉链表为存储结构，请编写一个非递归算法，从大到小输出二叉排序树中所有其值不小于 X 的键值。(10 分)
2. 设由 n 个整数组成一个大根堆 (即第一个数是堆中的最大值)，请编写一个时间复杂度为  $O(\log_2 n)$  的算法，实现将整数 X 插入到堆中，并保证插入后仍是大根堆。(10 分)
3. 请编写一个算法，判断含 n 个顶点和 e 条边的有向图中是否存在环。并分析算法的时间复杂度。(10 分)

大连理工大学 2003 年硕士入学试题  
数据结构部分 (共 75 分)

## 一、回答下列问题 (20 分)

1. 循环队列用数组  $A[0 \dots m-1]$  存放其数据元素。设 tail 指向其实际的队尾，front 指向其实际队首的前一个位置，则当前队列中的数据元素有多少个？如何进行队空和队满的判断？
2. 设散列表的地址空间为 0~10，散列函数为  $H(\text{key}) = \text{key} \% 11$  (% 为求余函数)，采用线性探查法解决冲突，并将键值序列 {15, 36, 50, 27, 19, 48} 依次存储到散列表中，请画出相应的散列表；当查找键值 48 时需要比较多少次？
3. 什么是 m 阶 B-树？在什么情况下向一棵 m 阶 B-树中插入一个关键字会产生结点分裂？在什么情况下从一棵 m 阶 B-树中删除一个关键字会产生结点合并？
4. 什么是线索二叉树？一棵二叉树的中序遍历序列为 djbaechif，前序遍历序列为 abdjcephi，请画出该二叉树的后序线索二叉树。

## 二、请用类 C 或类 PASCAL 语言进行算法设计，并回答相应问题。(45 分)

1. 设计一非递归算法采用深度优先搜索对无向图进行遍历，并对算法中的无向图的存储结构予以简单说明。
2. 用链式存储结构存放一元多项式  $P_n(x) = P_1x^{e_1} + P_2x^{e_2} + \dots + P_nx^{e_n}$ ，其中  $P_i$  是指数为  $e_i$  的项的非零系数，且满足  $0 \leq e_1 \leq e_2 \leq \dots \leq e_n = n$ 。请设计算法将多项式  $B = P_{n2}(x)$  加到多项式  $A = P_{n1}(x)$ ，同时对算法及链表的结点结构给出简单注释。要求利用  $P_{n1}(x)$  和  $P_{n2}(x)$  的结点产生最后求得的和多项式，不允许另建和多项式的结点空间。
3. (1) {R1, R2, ..., Rn} 为待排序的记录序列，请设计算法对 {R1, R2, ..., Rn} 按关键字的非递减次序进行快速排序。  
(2) 若待排序的记录的关键字集合是 {30, 4, 48, 25, 95, 13, 90, 27, 18}，请给出采用快速排序的第一趟、第二趟排序结果。  
(3) 若对 (2) 中给出的关键字集合采用堆排序，请问初建的小根堆是什么？  
(4) 当给定的待排序的记录的关键字基本有序时，应采用堆排序还是快速排序？为什么？

## 三、算法填空 (10 分)

1. 一棵树以孩子兄弟表示法存储，递归算法 numberofleaf 计算并返回根为 r 的树中叶子结点的个数 (NULL 代表空指针)。

```

typedef struct node { struct node *firstchild, *nextbrother; } TFNNode;
int numberofleaf(TFNNode *r)
{ int num;
  if(r==NULL) num=0;
  else
    if(r->firstchild==NULL)

```

```

        num=_____+numberofleaf ;
        (r->nextbrother) ;
    else _____ ;
    return(num) ;
}

```

2. 在根结点为  $r$  的二叉排序树中，插入数据域值为  $x$  的结点，要求插入新结点后的树仍是一棵二叉排序树 (NULL 代表空指针)。

二叉排序树的结点结构为

```

typedef struct node
{ int key ;
  struct node *lc,*rc ;
}BiNode ;
BiNode *insert(BiNode *r,int x)
{ BiNode *p,*q,*s;
  s=(BiNode*)malloc(sizeof(BiNode));
  s->key=x;
  s->lc=NULL;s->rc=NULL;
  q=NULL;
  if(r==NULL) {r=s;return(r);}
  p=r;
  while(_____)
  { q=p;
    if(_____) p=p->lc;
    else p=p->rc
  }
  if(x<q->key) _____;
  else _____;
  return;
}

```

#### 清华大学 2001 年数据结构与程序设计试题

试题内容：

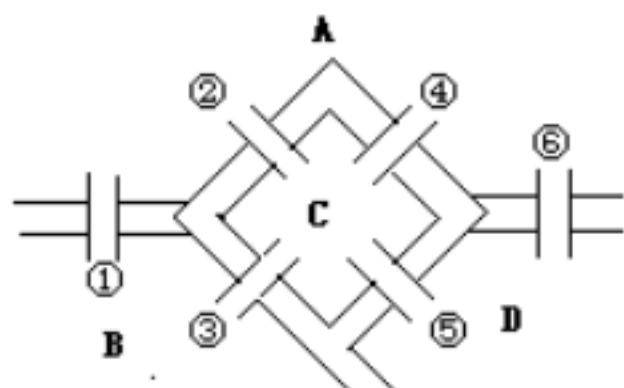
一、试给出下列有关并查集 (mfsets) 的操作序列的运算结果：

union(1, 2), union(3, 4), union(3, 5), union(1, 7), union(3, 6),  
union(8, 9), union(1, 8), union(3, 10), union(3, 11), union(3, 12),  
union(3, 13), union(14, 15), union(16, 0), union(14, 16), union(1, 3),  
union(1, 14)。(union 是合并运算，在以前的书中命名为 merge)

要求：

- (1) 对于 union( $i, j$ )，以  $i$  作为  $j$  的双亲； (5 分)
- (2) 按  $i$  和  $j$  为根的树的高度实现 union( $i, j$ )，高度大者为高度小者的双亲； (5 分)
- (3) 按  $i$  和  $j$  为根的树的结点个数实现 union( $i, j$ )，结点个数大者为结点个数小者的双亲。 (5 分)

二、设在 4 地(A, B, C, D) 之间架设有 6 座桥，如下图所示：



要求从某一地出发，经过每座桥恰巧一次，最后仍回到原地。

- (1) 试就以上图形说明：此问题有解的条件是什么？ (5 分)
- (2) 设图中的顶点数为  $n$ ，试用 C 或 Pascal 描述与求解此问题有关的数据结构并编写一个算法，找出满足要求的一条回路。 (10 分)

三、针对以下情况确定非递归的归并排序的运行时间 (数据比较次数与移动次数)：

- (1) 输入的  $n$  个数据全部有序； (5 分)

(2)输入的 n 个数据全部逆向有序； (5 分)

(3)随机地输入几个数据。 (5 分)

四、简单回答有关 AVL 树的问题。

(1)在有 N 个结点的 AVL 树中，为结点增加一个存放结点高度的数据成员，那么每一个结点需要增加多少个字位 (bit)?(5 分)

(2)若每一个结点中的高度计数器有 8bit，那么这样的 AVL 树可以有多少层？最少有多少个关键码?(5 分)

五、一个散列表包含 hashSize=13个表项，其下标从 0 到 12，采用线性探查法解决冲突。请按以下要求，将下列关键码散列到表中。

10 100 32 45 58 126 3 29 200 400 0

(1)散列函数采用除留余数法，用% hashSize(取余运算)将各关键码映像到表中。请指出每一个产生冲突的关键码可能产生多少次冲突。(7 分)

(2)散列函数采用先将关键码各位数字折叠相加，再用% hashSize 将相加的结果映像到表中的办法。请指出每一个产生冲突的关键码可能产生多少次冲突。(8 分)

六、设一棵二叉树的结点定义为

```
struct BinTreeNode{
    ElemType data ;
    BinTreeNode *leftChild,*rightChild ;
}
```

现采用输入广义表表示建立二叉树。具体规定如下：

(1)树的根结点作为由子树构成的表的表名放在表的最前面；

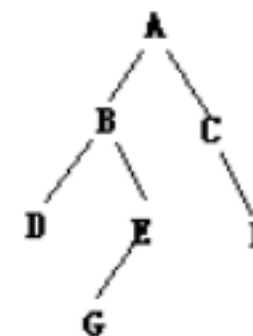
(2)每个结点的左子树和右子树用逗号隔开。若仅有右子树，没有左子树，逗号不能省略。

(3)在整个广义表表示输入的结尾加上一个特殊的符号 (例如“#”)表示输入结果。；

例如，对于如右图所示的二叉树，其广义表表示为 A(B(D，

E(G，))，C(，F))

此算法的基本思路是：依次从保存广义表的字符串 ls 中输入每个遇到的是字母（假定以字母作为结点的值），则表示是结点的值，应为它新的结点，并把该结点作为左子女当 (k=1) 或右子女 (当 k=2) 链接到其双上。若遇到的是左括号“(”，则表明子表的开始，将 A 置为 1；若遇到号”)”，则表明子表结束。若遇到的是逗号“，”，则表示以左子女为根理完毕，应接着处理以右子女为根的子树，将 A 置为 2。



字符。若建立一个亲结点的是右括的子树处

在算法中使用了一个栈 s，在进入子表之前，将根结点指针进栈，以便括号内的子女链接之用。在子表处理结束时退栈。相关的栈操作如下：

MakeEmpty(s) 置空栈

Push(s, p)元素 p 进栈

Pop(s)退栈

Top(s)存取栈顶元素的函数

下面给出了建立二叉树的算法，其中有 5 个语句缺失。请阅读此算法并把缺失的语句补上。(每空 3 分)

```
Void CreateBinTree(BinTreeNode *&BT,char ls)
{
    Stack<BinTreeNode*>s ; MakeEmpty(s) ;
    BT=NULL ; //置二叉树
    BinTreeNode *p ;
    int k ;
    istream ins(ls) ; //把串 ls 定义为输入字符串流对象 ins
    char ch ;
    ins>>ch ; //从 ins 顺序读入一个字符
    while(ch!= '#' )
    {
        //逐个字符处理，直到遇到 ?#?为止
        switch(ch)
        { case?(?: ____ (1)
            k=1 ;
            break ;
```

```

        case 3: pop(s);
            break;
        case 4: (2)
            break;
        default: p=new BinTreeNode;
            (3)
            p->leftChild=NULL;
            p->rightChild=NULL;
            if(BT==NULL)
                (4)
            else if(k==1) top(s)->leftChild=p;
            else top(s)->rightChild=p;
        }
        (5)
    }
}

```

七、下面是一个用 C 编写的快速排序算法。为了避免最坏情况，取基准记录 pivot 采用从 left, right 和 mid= (left+right)/2 中取中间值，并交换到 right 位置的办法。数组 a 存放待排序的一组记录，数据类型为 Type，left 和 right 是待排序子区间的最左端点和最右端点。

```

Void quicksort(Type a[], int left, int right)
{
    Type temp;
    if(left<right)
    {
        Type pivot=median3(a, left, right);
        int i=left, j=right-1;
        for(;;)
        {
            while(i<j&& a[i]<pivot) i++;
            while(i<j&& pivot<a[j]) j--;
            if(i<j)
            {
                temp=a[i]; a[j]=a[i]; a[i]=temp;
                i++; j--;
            }
            else break;
        }
        if(a[i]>pivot)
        {
            temp=a[i]; a[i]=a[right]; a[right]=temp;
        }
        quicksort(a, left, i-1); //递归排序左子区间
        quicksort(a, i+1, right); //递归排序右子区间
    }
}

```

(1)用 C 或 Pascal 实现三者取中子程序 median3(a, left, right); (5 分)

(2)改写 quicksort 算法，不用栈消去第二个递归调用 quicksort(a, j+1, right); (5 分)

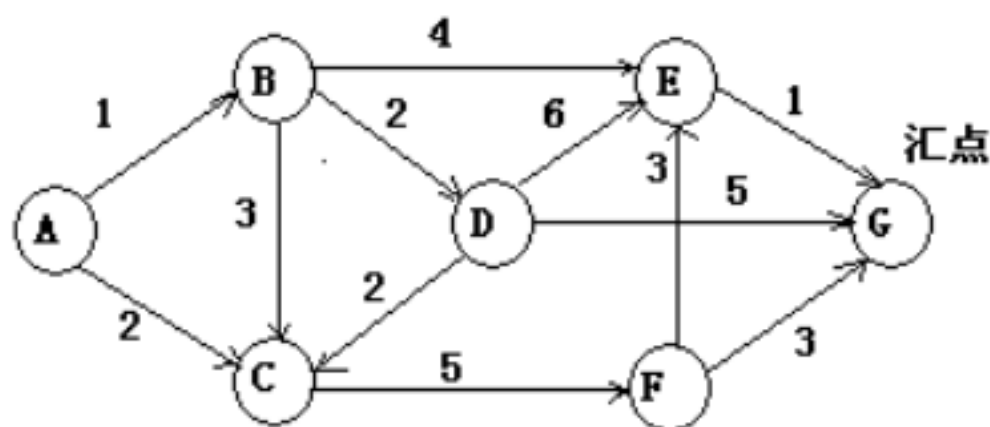
(3)继续改写 quicksort 算法，用栈消去剩下的递归调用。 (5 分)

上海交通大学 2001 年数据结构及程序设计试题

注意：程序设计请采用 C 语言，程序应有注解及说明。回答问题简洁、清晰全面。不得采用类 C 之类的语言写程序。

一、参见下图，该有向图是 AOE 网络。该图上已标出了源点及汇点，并给出了活动 (边) 的权值。





请求出该 AOE 网络的关键路径，以及事件（结点）的最早发生时间及最迟发生时间。（本题 8 分）

二、已知某二叉树的每个结点，要么其左、右子树皆为空，要么其左、右子树皆不空。又知该二叉树的前序序列为（即先根次序）：J、F、D、B、A、C、E、H、X、I、K；后序序列为（即后根次序）：A、C、B、E、D、X、I、H、F、K、J。请给出该二叉树的中序序列（即中根次序），并画出相应的二叉树树形。（本题 8 分）

三、回答下列问题：（本题 10 分）

- 1) 具有  $N$  个结点且互不相似的二叉树的总数是多少？
- 2) 具有  $N$  个结点且不同形态的树的总数是多少？
- 3) 对二叉树而言，如果它的叶子结点总数为  $N_0$ ，度为 2 的结点的总数为  $N_2$ ，则  $N_0$  和  $N_2$  之间的关系如何？

4) 二叉树是否是结点的度最多为 2 的树？请说明理由。

5) 具有  $n$  片叶子的哈夫曼树（即赫夫曼树）中，结点总数为多少？

四、在外部分类时，为了减少读、写的次数，可以采用  $k$  路平衡归并的最佳归并树模式。当初始归并段的总数不足时，可以增加长度为零的“虚段”。请问增加的“虚段”的数目为多少？请推导之。设初始归并段的总数为  $m$ 。（本题 8 分）

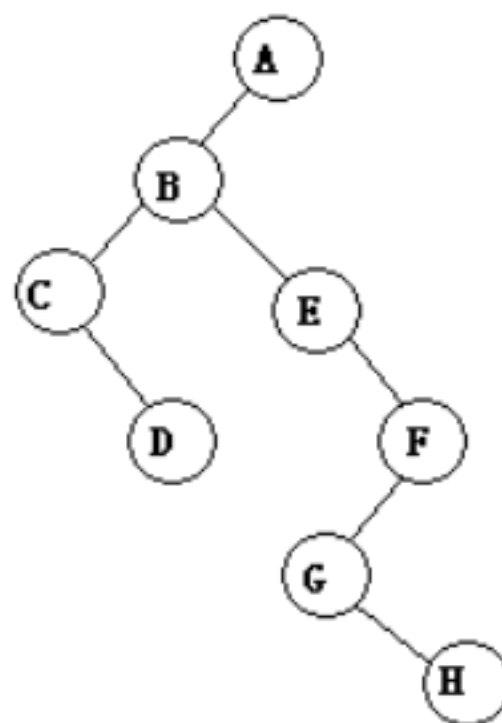
五、对平衡的排序二叉树进行删除结点的操作，必须保证删除之后平衡树中的每个结点的平衡因子是  $+1$ ， $-1$ ， $0$  三种情况之一，且该树仍然是排序二叉树。现假定删除操作是在  $p$  结点的左子树上进行的，且该左子树原高为  $h-1$ ，现变为  $h-2$ 。因此，必须从  $p$  的左子树沿着到根的方向回溯调整结点的平衡因子，并进行树形的调整。设  $p$  是调整时遇到的第一个平衡因子力图由  $-1$  变成  $-2$  的最年轻的“前辈”结点。我们知道，以  $p$  为根的子树经调整后，高度有可能减少 1。试用图形把调整前及调整后的相关结点的平衡因子、树形表示出来；仅仅针对调整后子树的高度减少 1 的情况即可。注意，罗列出所有可能的情况。上图可供参考。（本题 10 分）

六、某算法由下述方程表示。请求出该算法的时间复杂性的级别（以大  $O$  形式表示）。注意  $n > 7$  求解问题的规模，设  $n$  是 3 的正整数幂。（本题 8 分）

$$T_n = \begin{cases} 1 & \text{如果 } n=1 \\ 5T(n/3)+n & \text{如果 } n>1 \end{cases}$$

七、如右图所示，该二叉有序树变换成的相对应的二叉树。试给出原来的有序树形状。并回答以下问题：

- 1) 原有序树是度为多少的树？
- 2) 原有序树的叶子结点有哪几个？
- 3) 是否所有的二叉树都可以找到相对应的有序树？什么条件？（本题 5 分）



树是某棵树的形状

必须满足

的每个结点查找长度是解。（本题

八、在排序二叉树上进行查找操作时，设对树中点查找概率相同。设由  $n$  个结点构成的序列生成的排是“随机”的。试求出在成功查找的情况下，平均查找多少？为了简单起见，最后得到的递推式可不予求（8 分）

九、设从键盘每次输入两个字符。如 A、B，则着一条由数据场之值为字符 A 的结点到数据场之值为结点的有向边。依此输入这些

表示存在字符 B 的

有向边，直至出现字符 ! 为止。试设计一个程序，生成该有向图的邻接表及逆邻接表。必须交待所用的结构、变量、加以适当注解。（本题 20 分）

十、设二叉树中结点的数据场之值为一字符。采用二叉链表的方式存储该二叉树中的所有结点，设  $p$  为指向树根结点的指针。设计一个程序在该二叉树中寻找数据场之值为  $key$  ( $key$  为一变量，变量内容为一字符) 的那个结点的所有祖先。设二叉树中结点数据场之值互不重复。 (本题 15 分)

注意：有些书上将二叉树的二叉链表存储形式称之为标准存储形式。

#### 南开大学 2001 年数据结构试题

##### 一、选择题 (每小题 3 分，共 21 分)

在下列各题中，每题之后均有若干个备选答案，请选出所有正确的答案，填入“\_\_\_\_\_”处。答案请写在答题纸上。

- 任何一个无向连通图的最小生成树 \_\_\_\_\_。  
只有一棵          有一棵或多棵  
一定有多棵          可能不存在
- 已知一棵非空二叉树的 \_\_\_\_\_，则能够惟一确定这棵二叉树。  
先序遍历序列和后序遍历序列  
先序遍历序列和中序遍历序列  
先序遍历序列  
中序遍历序列
- 使用指针实现二叉树时，如果结点的个数为  $n$ ，则非空的指针域个数为 \_\_\_\_\_。  
 $n-1$            $2n-1$            $n+1$            $2n+1$
- 设队列存储于一个一维数组中，数组下标范围是  $1-n$ ，头尾指针分别为  $f$  和  $r$ ， $f$  指向第一个元素的前一个位置， $r$  指向队列中的最后一个元素，则队列中元素个数为 \_\_\_\_\_。  
 $r-f$            $r-f+1$            $(r-f+1) \bmod n$            $(r-f+n) \bmod n$
- 任意一个 AOE 网络的关键路径 \_\_\_\_\_。  
一定有多条          只有一条          可能不只一条          可能不存在
- 下列排序算法中，在每一趟都能选出一个元素放到其最终位置上的是 \_\_\_\_\_。  
插入排序          希尔排序          快速排序          堆排序
- 任意一个有向图的拓扑排序序列 \_\_\_\_\_。  
一定有多种          只有一种          可能不存在          以上都不对

二、(7 分)已知散列表地址空间为  $0-8$ ，哈希函数为  $H(k)=k \bmod 7$ ，采用线性探测再散列法解决冲突。将下面关键字数据依次填入该散列表中，同时将查找每个关键字所需的比较次数  $m$  填入下表中，并求等概率下的平均查找长度。

关键字值：100, 20, 21, 35, 3, 78, 99, 45

	0	1	2	3	4	5	6	7	8
A									
	100	20	21	35	3	78	99	45	
m									

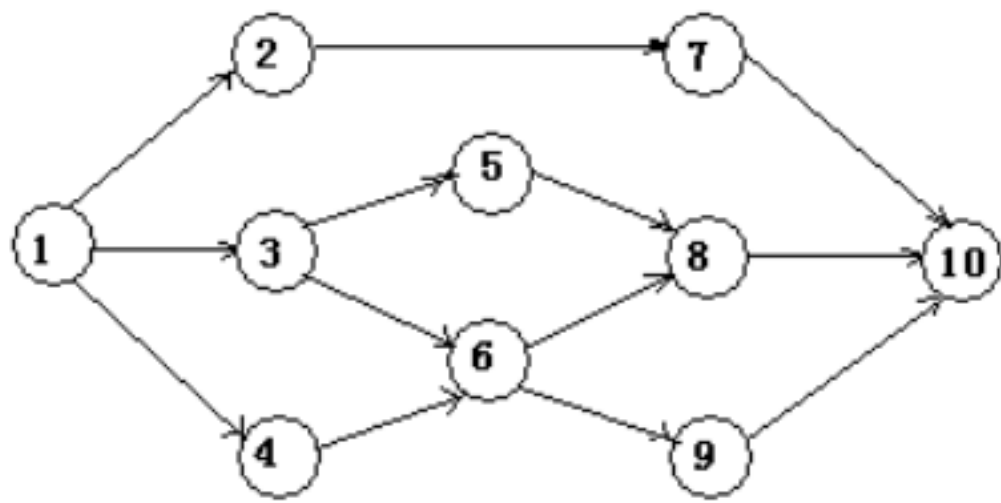
##### 三、(12 分)回答下列问题：

- (3 分)什么叫基数排序？
- (3 分)什么是 AVL 树中的平衡因子，它有什么特点？
- (6 分) $n$  个元素的序列  $\{k_1, k_2, \dots, k_n\}$  满足什么条件才能称之为堆？简述对它进行堆排序的过程。

四、(10 分)顺序给出以下关键字：63、23、31、26、7、91、53、15、72、52、49、68，构造 3 阶 B-树。要求从空树开始，每插入一个关键字，画出一个树形。

五、(6 分)设有向无环图  $G$  如下图所示：

试写出图  $G$  的六种不同的拓扑排序序列。



六、(10 分) 设二叉树以二叉链表表示，各结点的结构如下所示：

left	data	subsum	right
------	------	--------	-------

其中 left、right 分别为指向该结点左、右孩子的指针，data 为存储关键字值的整数域，subsum 中存储以该结点为根的子树中所有关键字值之和。试使用 C 或 C++ 语言设计算法，计算所给树 T 中所有结点的 subsum 值。

七、(12 分) 给出一个带头的单链表 L，L 的每个结点中存放一个整数。现给定一个阈值 K，将 L 分成两个子表 L1 和 L2，其中 L1 中存放 L 中所有关键字值大于等于 K 的结点，L2 中存放 L 中所有关键字值小于 K 的结点。试编程实现这个过程。要求，使用 C 或 C++ 语言实现算法，L1 和 L2 仍占用 L 的存储空间。

八、(10 分) 设有一维整数数组 A，试使用 C 或 C++ 语言设计算法，将 A 中所有的奇数排在所有偶数之前。要求，时间复杂度尽可能地少，结果仍放在 A 原来的存储空间。

九、(12 分) 简述哈夫曼编码的构造过程。

#### 华东理工大学 2001 年数据结构与程序设计试题

##### 一、单选题 (10 分)

1. 若某线性表中最常用的操作是在最后一个元素之后插入一个元素和删除第一个元素，则采用\_\_\_\_存储方式最节省运算时间。  
A. 单链表 B. 仅有头指针的单循环链表  
C. 双链表 D. 仅有尾指针的单循环链表
2. 若在线性表中采用折半查找法查找元素，该线性表应该\_\_\_\_。  
A. 元素按值有序 B. 采用顺序存储结构  
C. 元素按值有序，且采用顺序存储结构  
D. 元素按值有序，且采用链式存储结构
3. 对于给定的结点序列 abcdef，规定进栈只能从序列的左端开始。通过栈的操作，能得到的序列为\_\_\_\_。  
A. abcfed B. cabfed C. abcfde D. cbafde
4. 已知一算术表达式的中缀形式为  $A+B^*C-D/E$ ，后缀形式为  $ABC^*+DE/-$ ，其前缀形式为\_\_\_\_。  
A.  $-A+B^*C/DE$  B.  $-A+B^*CD/E$   
C.  $-+^*ABC/DE$  D.  $-+A^*BC/DE$
5. 如果  $n_1$  和  $n_2$  是二叉树 T 中两个不同结点， $n_2$  为  $n_1$  的后代，那么按遍历二叉树 T 时，结点  $n_2$  一定比结点  $n_1$  先被访问。  
A. 前序 B. 中序 C. 后序 D. 层次序
6. 具有 65 个结点的完全二叉树其深度为\_\_\_\_。(根的层次号为 1)  
A. 8 B. 7 C. 6 D. 5
7. 已知二叉树的前序遍历序列是 abdgcefh，中序遍历序列是 dgbaechf，它的后序遍历序列是\_\_\_\_。  
A. bdgcefha B. gdbecfha C. bdgechfa D. gdbehfca
8. 对于前序遍历和后序遍历结果相同的二叉树为\_\_\_\_。  
A. 一般二叉树  
B. 只有根结点的二叉树  
C. 根结点无左孩子的二叉树  
D. 根结点无右孩子的二叉树

9. 对于前序遍历与中序遍历结果相同的二叉树为 \_\_\_\_\_。

- 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

10. 在有  $n$  个叶子的哈夫曼树中，其结点总数为 \_\_\_\_\_。

- A. 不确定
- B.  $2n$
- C.  $2n+1$
- D.  $2n-1$

## 二、是非题 (10 分)

1. 顺序存储方式只能用于存储线性结构。
2. 消除递归不一定需要使用栈。
3. 将一棵树转换成相应的二叉树后，根结点没有右子树。
4. 完全二叉树可以采用顺序存储结构实现，存储非完全二叉树则不能。
5. 在前序遍历二叉树的序列中，任何结点的子树的所有结点都是直接跟在该结点之后。
6. 邻接表法只能用于有向图的存储，而邻接矩阵法对于有向图和无向图的存储都适用。
7. 在一个有向图的邻接表中，如果某个顶点的链表为空，则该顶点的度一定为零。
8. 二叉树为二叉排序树的充分必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。
9. 最佳二叉排序树的任何子树都是最佳二叉排序树。
10. 对两棵具有相同关键字集合的而形状不同的二叉排序树，按中序遍历它们得到的序列的顺序是一样的。

## 三、问答题

(应届生限做 2, 3, 4, 5 题；在职生任选做四题；共 40 分)

1. (10 分)什么是广义表？请简述广义表与线性表的主要区别。
2. (10 分)下图表示一个地区的通讯网，边表示城市间的通讯线路，边上的权表示架设线路花费的代价。

请分别给出该图的邻接表和邻接矩阵，要求每种存储结构能够表达出该图的全部信息，并分别对这二种形式中每个部分的含义（物理意义）予以简要说明。

若假设每个域（包括指针域）的长度为一个字节，请分别计算出这二种结构所占用的空间大小。

3. (10 分)已知如下所示长度为 12 的表：

(Jan, Feb, Mar, Apr, May, June, July, Aug, Sep, Oct, Nov, Dec)

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，请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。

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。

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. (10 分)在起泡排序过程中，有的关键字在某趟排序中可能朝着与最终排序相反的方向移动，试举例说明之。快速排序过程中有没有这种现象？

5. (10 分)调用下列函数  $f(n)$ ，回答下列问题：

试指出  $f(n)$  值的大小，并写出  $f(n)$  值的推导过程；

假定  $n=5$ ，试指出  $f(5)$  值的大小和执行  $f(5)$  的输出结果。

## C 函数

```
int f(int n)
{
    int i, j, k, sum=0;
    for(j=1; j<n+1; j++)
        {
            for(j=n; j>i-1; j--)
                for(k=1; k<j+1; k++)
                    sum++;
            printf("sum=%d\n", sum);
        }
    return(sum);
}
```

## Pascal 函数

```

FUNCTION f(n : integer) : integer ;
VAR i , j , k , sum , integer ;
BEGIN
    sum := 0 ;
    FOR i := 1 TO n DO
        BEGIN
            FOR j := n DOWN TO i DO
                FOR k := 1 TO j DO
                    sum := sum + 1
                writeln( 'sum=', sum)
            END ;
        f := sum
    END ;

```

#### 四、编写算法

(应届生限做 1、2、3、4 题，在职生任选四题，每题 10 分，共 40 分)

1. (10 分)利用两个栈 S1 和 S2 模拟一个队列，写出入队和出队的算法 (可用栈的基本操作)。

栈的操作有：

makeEmpty(s : stack) ;	置空栈
push(s : stack ; value : datatype) ;	新元素 value 进栈
pop(s : stack) : datatype ;	出栈，返回栈顶值
isEmpty(s : stack) : boolean ;	判栈空否

队列的操作有

enQueue(s1 : stack ; s2 : stack ; value : datatype) ;	元素 value 进队
deQueue(s1 : stack ; s2 : stack) : datatype ;	出队列，返回队头值

2. (10 分)试写出给定的二叉树进行层次遍历的算法，在遍历时使用一个链接队列。

3. (10 分)试写一个算法，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 ( $i < j$ )。假设分别基于下述策略：

- 1) 图的深度优先搜索；
- 2) 图的广度优先搜索；

4. (10 分)设二叉排序树中关键字由 1 至 1000 的整数构成，现要检索关键字为 531 的结点，下述关键字序列中哪些可能是二叉排序树上搜索到的序列，哪些不可能是二叉排序树上搜索到的序列，哪些不可能是二叉排序树上搜索到的序列？

- a) 800, 399, 588, 570, 500, 520, 566, 531
- b) 730, 355, 780, 390, 701, 401, 530, 531
- c) 732, 321, 712, 385, 713, 392, 531
- d) 8, 578, 555, 340, 433, 551, 550, 450, 531

通过对上述序列的分析，试写一个算法判定给定的关键字序列 (假定关键字互不相同) 是否可能是二叉排序树的搜索序列。若可能则返回真，否则返回假。可假定被判定的序列已存入数组中。

5. (10 分)编写一个在非空的带表头结点的单链表上实现就地排序的程序。(不额外申请新的链表空间)

#### 华中理工大学 2001 年数据结构试题

说明：在有数据类型定义和算法设计的各题中，任取 C 语言、PASCAL 语言之一作答，但不许使用所谓“类 C”或“类 PASCAL”。

一、选择题 (从下列各题四个备选答案中选出一至四个正确答案，将其代号 (A, B, C, D) 写在题干前面的括号内。答案选错或未选全者，该题不得分。每小题 1 分，共计 20 分)

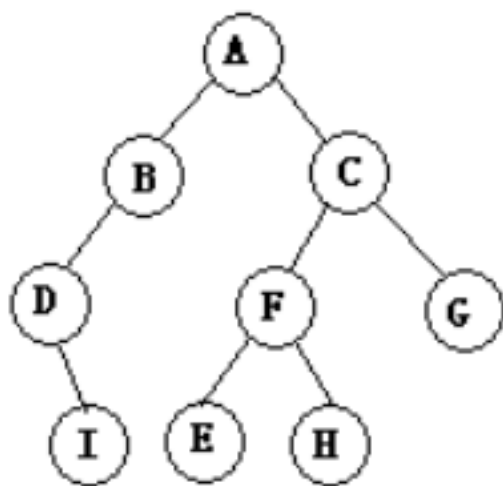
- ( ) 1. 一个算法具有 \_\_\_\_ 等特点。
 

A. 可行性	B. 至少有一个输入量
C. 确定性	D. 健壮性
- ( ) 2. \_\_\_\_ 是一个线性表。
 

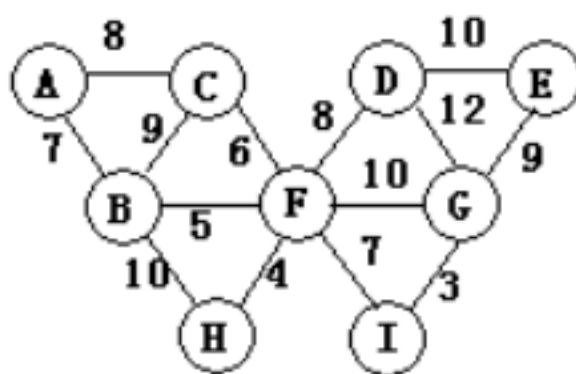
A. (A, B, C, D)	B. { ?A?, ?B?, ?C?, ?D? }
C. (1, 2, 3, ...)	D. (40, -22, 88)
- ( ) 3. 可使用 \_\_\_\_ 作压缩稀疏矩阵的存储结构。

- A . 邻接矩阵      B . 三元组表  
C . 邻接表      D . 十字链表
- ( )4 . 采用一维数组表示顺序存储结构时 , 可用它来表示 \_\_\_\_。
- A . 字符串      B . 2 度树      C . 二叉树      D . 无向图
- ( )5 . 假设进入栈的元素序列为 d , c , a , b , e , 那么可得到出栈的元素序列 \_\_\_\_。
- A . a , b , c , d , e      B . a , b , e , d , c  
C . d , b , a , e , c      D . d , b , a , c , e
- ( )6 . 对队列的操作有 \_\_\_\_。
- A . 在队首插入元素      B . 删除值最小的元素  
C . 按元素的大小排序      D . 判断是否还有元素
- ( )7 . 假设有 60 行 70 列的二维数组 a[1 . . 60 , 1 . . 70] , 以列序为主序顺序存储 , 其基地址为 10000 , 每个元素占 2 个存储单元 , 那么第 32 行第 58 列的元素 a[32 , 58] 的存储地址为 \_\_\_\_。(无第 0 行第 0 列元素)
- A . 16902      B . 16904  
C . 14454      D . 答案 A , B , C 均不对
- ( )8 . \_\_\_\_ 是 C 语言中 "abcd321ABCD" 的子串。
- A . abed      B . 32IAB      C . "abeABC "      D . '2IAB "
- ( )9 . 在下列各广义表中 , 长度为 3 的广义表有 \_\_\_\_。
- A . (a , b , c , ())      B . ((g) , (a , b , c , d , f) , ())  
C . (a , (b , (c)))      D . ((()))
- ( )10 . 一个堆 (heap) 又是一棵 \_\_\_\_。
- A . 二叉排序树      B . 完全二叉树  
C . 平衡二叉树      D . 哈夫曼 (Huffman) 树
- ( )11 . 折半查找有序表 (4 , 6 , 10 , 20 , 30 , 50 , 70 , 88 , 100) , 若查找元素 58 , 则它将依次与表中元素 \_\_\_\_ 比较大小 , 查找结果是失败。
- A . 20 , 70 , 30 , 50      B . 30 , 88 , 70  
C . 20 , 50      D . 30 , 88 , 50 , 70
- ( )12 . 对 27 个记录的有序表作折半查找 , 当查找失败时 , 至少需要比较 \_\_\_\_ 次关键字。
- A . 3      B . 4      C . 5      D . 6
- ( )13 . 具有 12 个结点的完全二叉树有 \_\_\_\_。
- A . 5 个叶子      B . 5 个度为 2 的结点  
C . 7 个分支结点      D . 2 个度为 1 的结点
- ( )14 . 具有  $n(n>0)$  个结点的完全二叉树的深度为 \_\_\_\_。
- A .  $\lceil \log_2(n) \rceil$       B .  $\lceil \log_2(n+1) \rceil$   
C .  $\lceil \log_2(n) \rceil + 1$       D .  $\lceil \log_e(n) \rceil + 1$
- ( )15 . 用 5 个权值 {3 , 2 , 4 , 5 , 1} 构造的哈夫曼 (Huffman) 树的带权路径长度是 \_\_\_\_。
- A . 32      B . 33      C . 34      D . 15
- ( )16 . 对 14 个记录的表进行 2 路归并排序 , 共需移动 \_\_\_\_ 次记录。
- A . 42      B . 56      C . 91      D . 84
- ( )17 . 对有 n 个记录的表作快速排序 , 在最坏情况下 , 算法的时间复杂度是 \_\_\_\_。
- A .  $O(n)$       B .  $O(n^2)$       C .  $O(n \log_2 n)$       D .  $O(2^n)$
- ( )18 . 在最好情况下 , 下列算法中 \_\_\_\_ 排序算法所需比较关键字次数最少。
- A . 冒泡      B . 归并      C . 快速      D . 直接插入
- ( )19 . 置换选择排序的功能是 \_\_\_\_。
- A . 选出最大的元素      B . 产生初始归并段  
C . 产生有序文件      D . 置换某个记录
- ( )20 . 可在 \_\_\_\_ 构造一个散列文件。
- A . 内存      B . 软盘      C . 磁带      D . 硬盘

二、试用 2 种不同表示法画出下列二叉树 B1 的存储结构 , 并评述这 2 种表示法的优、缺点。(共 10 分)



二题图 B1



三题图 G

三、对图 G：

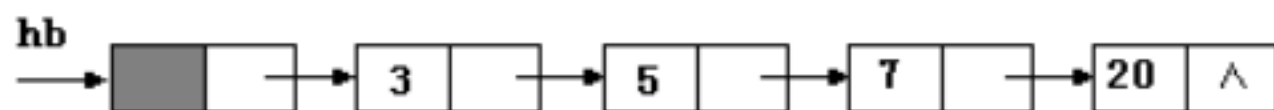
1. 从顶点 A 出发，求图 G 的一棵深度优先生成树；
2. 从顶点 B 出发，求图 G 的一棵广度优先生成树；
3. 求图 G 的一棵最小生成树。（共 12 分）

四、设哈希 (Hash) 函数为： $H(k)=k \text{MOD} 14$ ，其中  $k$  为关键字 (整数)，MOD 为取模运算，用线性探测再散列法处理冲突，在地址范围为 0~14 散列区中，试用关键字序列 (19, 27, 26, 28, 29, 40, 64, 21, 15, 12) 造一个哈希表，回答下列问题：

1. 画出该哈希表的存储结构图；
2. 若查找关键字 40，必须依次与表中哪些关键字比较大小？
3. 假定每个关键字的查找概率相同，试求查找成功时的平均查找长度。（共 13 分）

五、给定头指针为  $ha$  的单链表 A，和头指针为  $hb$  的递增有序单链表 B。试利用表 A 和表 B 的结点，将表 A 和表 B 归并为递增有序表 C，其头指针为  $hc$ ，允许有相同 data 值 (数据元素) 的结点存在，表 A，B，C 均带表头结点。

例如，给定：



将它们归并为：



1. 在下列 C 算法 merge 中有下划线的位置填空，使之成为完整的算法，要求在填空后加注释，以提高算法的可读性。

2. 假定单链表 A 和 B 的长度分别为  $m$  和  $n(m>0, n>0)$ ，试分别就最好情况、最坏情况和平均性能，分析算法 merge 的时间复杂度。（共 15 分）

结点类型定义如下：

```
struct node
{
    int data ;
    struct node *next ;
};
```

算法如下：

```
Void merge(struct node*ha , struct node *hb , struct node*hc)
{
    struct node *pc , *qc , *pa , *fa ;
    int search ;
    hc=hb ; hb=NULL ;
    pa=ha->next ; free(ha) ;
    while(pa)
    {
        pc=hc ; qc=hc->next ;
        search=_____ ;
        while(qc&&search)
        {
            if(pa->data<=qc->data)
```



```

        _____ ;
    else
        { pc=qc ;
          qc=_____ ;
        }
    }
    fa=_____ ;
    pa=_____ ;
    fa->next=_____ ;
    _____ ;
}
}

```

六、设下图二叉树 B2 的存储结构为二叉链表，root 为根指针，结点结构为：(lchild, data, rchild) 其中：lchild, rchild 分别为指向左右孩子的指针，data 为字符型。(共 10 分)

1. 对下列二叉树 B2，执行下列算法 traversal(root)，试指出其输出结果；
2. 假定二叉树 B2 共有 n 个结点，试分析算法 traversal 的时间复杂度。结点类型定义如下：

```

struct node
{ char data ;
  struct node *lchild , *rchild ;
};

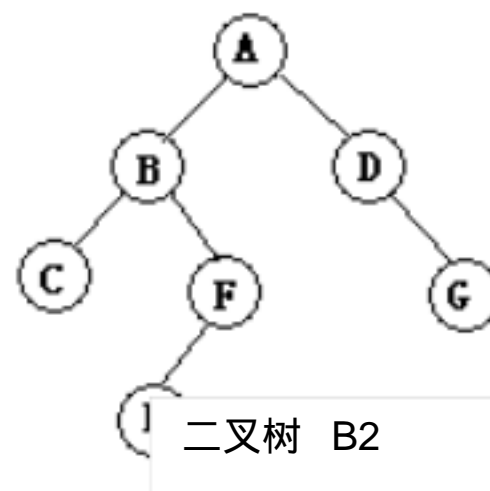
```

算法如下：

```

void traversal(struct node *root)
{ if(root)
  { printf( "%c ", root->data) ;
    traversal(root->lchild) ;
    printf( "%c ", root->data) ;
    traversal(root->rchild) ;
  }
}

```



七、算法设计题 (要求：(1)写出所用数据结构的类型定义和变量说明；(2)写出算法，并在相关位置加注释，以提高算法的可读性。)

1. 试设计一算法：输入一个有 m 行 n 列的整数矩阵，然后将每一行的元素按非减次序输出。例如，若输入：

```

4 , 3 , 5 , 6 , 2
9 , 8 , 1 , 2 , 8
7 , 1 , 2 , 3 , 8

```

则输出如下结果：

```

2 , 3 , 4 , 5 , 6
1 , 2 , 8 , 8 , 9
1 , 2 , 3 , 7 , 8

```

2. 如果字符串的一个子串 (其长度大于 1) 的各个字符均相同，则称之为等值子串。试设计一算法：输入字符串 S，以 '!' 为结束标志，如果串 S 中不存在等值子串，则输出信息：“无等值子串”，否则求出 (输出) 一个长度最大的等值子串。

例如，若 S = "abcl23abcl23!"，则输出：“无等值子串”；

又如，若 S = "abcaabccccdddddadaadd!"，则输出等值子串：“ddddd”。(共 20 分)

北京理工大学 2001 年程序设计 (含数据结构) 试题

第二部分 数据结构 (共 50 分)

一、选择题 (12 分，每题 2 分)

1. 下列数据中哪些是非线性数据结构 \_\_\_\_\_。  
A. 栈 B. 队列 C. 完全二叉树 D. 堆
2. 静态链表中指针表示的是 \_\_\_\_\_。  
A. 内存地址 B. 数组下标  
C. 下一元素地址 D. 左、右孩子地址

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

- A. 仅修改队头指针
- B. 仅修改队尾指针
- C. 队头, 队尾指针都要修改
- D. 队头, 队尾指针都可能要修改

4. 在下列排序算法中, 哪一个算法的时间复杂度与初始排列无关 \_\_\_\_\_。

- A. 直接插入排序
- B. 起泡排序
- C. 快速排序
- D. 直接选择排序

5. 二叉树第  $i$  层结点的结点个数最多是 (设根的层数为 1) \_\_\_\_\_。

- A.  $2^{i-1}$
- B.  $2^i - 1$
- C.  $2i$
- D.  $2i - 1$

6. 树的后根遍历序列等同于该树对应的二叉树的 \_\_\_\_\_。

- A. 先序序列
- B. 中序序列
- C. 后序序列

二、填空 (8 分, 每空 1 分)

1. 数据结构中评价算法的两个重要指标是\_\_\_\_\_。

2. 顺序存储结构是通过 \_\_\_\_\_ 表示元素之间的关系的。链式存储结构是通过 \_\_\_\_\_ 表示元素之间的关系的。

3. AOV 网中, 结点表示 \_\_\_\_\_, 边表示 \_\_\_\_\_。AOE 网中, 结点表示 \_\_\_\_\_, 边表示 \_\_\_\_\_。

4. 哈夫曼树是 \_\_\_\_\_。

三、求解下列问题 (25 分, 每题 5 分)

1. 请用 C 语言给出线性链表的类型定义。

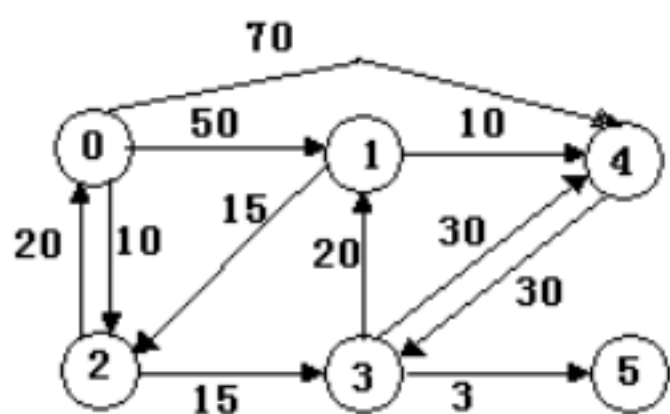
2. 已知二叉树的先序序列: CBHEGAF, 中序序列: HBGEACF, 试构造该二叉树。

3. 设散列表的地址空间为 0 到 12 的存储单元, 散列函数为:  $h(key) = key \bmod 13$ , 用链地址法解决冲突, 初始时散列表为空。现依次将关键字 25, 33, 48, 25, 43, 38, 39 插入散列表, 试画出完成上述所有插入操作后散列表的状态, 并计算在等概率的情况下, 在该表中查找成功的平均查找长度。

4. 给出如下图所示的图形。

1) 试写出该图的邻接表;

2) 试写出该图从结点 0 出发的深度优先遍历序列和广度优先遍历序列, 并画出遍历过程中所经的路径。



5. 全国统考答题

对上图, 按迪杰斯特拉算法求出从结点 0 到其余各点的最短路径, 并给出在求解过程中数组 distance 的变化状态。

6. 单独考试答题 (可在 5 或 6 中任选一题做)

给出关键字序列 27, 18, 21, 77, 26, 45, 66, 34 试写出快速排序的过程。

四、算法题 (12 分)(请在算法的主要步骤上加注释)

1. 下面是用 C 语言编写的对不带头结点的单链表进行就地置逆的算法, 该算法用 L 返回置逆后链表的头指针。试在空缺处填入适当的语句。

```
void reverse(linklist&L)
{
    p=NULL ; q=L ;
    while(q!=NULL)
    {
        _____q->next=p ;
        p=q ;
    }
}
```

}

## 2. 全国统考答题

设二叉树用二叉链表存储，试编写按层输出二叉树结点的算法。

## 3. 单独考试答题 (可在 2 或 3 中任选一题做)

试编写在带头结点的单链表中删除 (一个)最小值结点的 (高效)算法。

void delete(Linklist &L)

## 北京邮电大学 2001 年数据结构试题

### 一、选择题 (10 分，每题 2 分)

1. B<sup>+</sup>树是应用在 ( ) 文件系统中。

ISAM

VSAM

2. 将一棵树 t 转换为孩子 - 兄弟链表表示的二叉树 h，则 t 的后根遍历是 h 的 ( )。

前序遍历

中序遍历

后序遍历

3. 一个有向无环图的拓扑排序序列 ( ) 是惟一的。

一定

不一定

4. 将 10 个元素散列到 100000 个单元的哈希表中，则 ( ) 产生冲突。

一定会

一定不会

仍可能会

5. 若要求尽可能快地对序列进行稳定的排序，则应选 ( )。

快速排序

归并排序

冒泡排序

### 二、填空题 (20 分，每空 2 分)

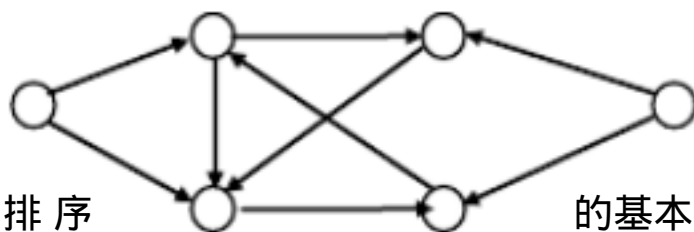
1. 数据的逻辑结构是指 \_\_\_\_\_。

2. 区分循环队列的满与空，只有两类办法，它们是 \_\_\_\_\_ 和 \_\_\_\_\_。

3. 用一维数组 B 以列优先存放带状矩阵 A 中的非零元素  $A[i, j](1 \leq i \leq n, i-2 \leq j \leq i+2)$ ，B 中的第 8 个元素是 A 中第 \_\_\_\_\_ 行、第 \_\_\_\_\_ 列的元素。

4. 字符串 'ababaaab' 的 nextval 函数值为\_\_\_\_\_。

5. 下图中的强连通分量的个数为 \_\_\_\_\_ 个。



6. 外部排序的基本方法是归并排序，但在之前必须先生成 \_\_\_\_\_。

7. 若不考虑基数排序，则在排序过程中，主要进行的两种基本操作是关键字的 \_\_\_\_\_ 和记录的 \_\_\_\_\_。

### 三、简答题 (15 分，每题 5 分)

1. 特殊矩阵和稀疏矩阵哪种压缩存储后会失去随机存取的功能 ? 为什么 ?

2. 试问线索二叉树的目的何在 ?

3. 在多关键字排序时，LSD 和 MSD 两种方法的特点是什么 ?

### 四、应用题 (25 分，每题 5 分)

1. 画出按下表中元素的顺序构造的平衡二叉排序树，并求其在等概率的情况下查找成功的平均查找长度。

{15, 12, 24, 3, 27, 21, 18, 6, 36, 33, 30, 26, 3}

2. 假设用于通信的电文由字符集 {a, b, c, d, e, f, g} 中的字母构成，它们在电文中出现的频率分别为 {0.31, 0.16, 0.10, 0.08, 0.11, 0.20, 0.04}。

(1) 为这 7 个字母设计哈夫曼编码；

(2) 对这 7 个字母进行等长编码，至少需要几位二进制数 ? 哈夫曼编码比等长编码使电文总长压缩多少 ?

3. 试推导当总盘数为 n 时的 Hanoi 塔的移动次数。

4. 一棵满 k 叉树，按层次遍历存储在一维数组中，试计算结点下标为 u 的结点的第 i 个孩子的下标以及结点下标为 v 的结点的父母结点的下标。

5. 有一图的邻接矩阵如下，试给出用弗洛伊德算法求各点间最短距离的矩阵序列  $A^{(1)}$ 、 $A^{(2)}$ 、 $A^{(3)}$  和  $A^{(4)}$ 。

$$A = \begin{bmatrix} 0 & 2 & & & \\ & 0 & 1 & 6 & \\ 5 & & 0 & 4 & \\ 3 & & & 0 & \end{bmatrix}$$

五、算法 (30 分, 每题 10 分)

1. 在单链表中, 每个结点含有 5 个正整型的数据元素 (若最后一个结点的数据元素不满 5 个, 以值 0 填充), 试编写一算法查找值为  $n(n>0)$  的数据元素所在的结点指针以及在该结点中的序号, 若链表中不存在该数据元素则返回空指针。

2. 将一组数据元素按哈希函数  $H(\text{key})$  散列到哈希表  $m(0 \dots m)$  中, 用线性探测法处理冲突 ( $H(\text{key})+1$ 、 $H(\text{key})+2$ 、 $\dots$ 、 $H(\text{key})-1$ ), 假设空单元用 EMPTY 表示, 删除操作是将哈希表中结点标志位从 INUSE 标记为 DELETED, 试写出该散列表的查找、插入和删除三个基本操作算法。

3. 给出算法将二叉链表表示的表达式二叉树按中缀表达式输出, 并加上相应的括号。

北京航空航天大学 2001 年 数据结构与程序设计试题

一、问答题 (本题 10 分)

一般情况下, 线性表可以采用哪几种存储结构? 请分别叙述每一种存储结构的构造原理与特点。

二、(本题 10 分)

已知 AOE 网为  $G=(V, E)$ ,  $V=\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$ ,  $E=\{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}\}$ , 其中:

$a_1: (v_1, v_2)_5$      $a_2: (v_1, v_3)_6$      $a_3: (v_2, v_5)_3$      $a_4: (v_3, v_4)_3$   
 $a_5: (v_3, v_5)_6$      $a_6: (v_4, v_5)_3$      $a_7: (v_4, v_7)_1$      $a_8: (v_4, v_8)_4$   
 $a_9: (v_5, v_6)_4$      $a_{10}: (v_5, v_7)_1$      $a_{11}: (v_6, v_{10})_4$      $a_{12}: (v_7, v_9)_5$   
 $a_{13}: (v_8, v_9)_2$      $a_{14}: (v_9, v_{10})_2$

注: 顶点偶对右下角的数字表示边上的权值。

请按下述过程指出所有关键路径:

ee[1:10]:

le[1:10]:

e[1:14]:

l[1:14]:

其中, ee[i] 与 le[i] 分别表示事件  $v_i$  的最早发生时间与最晚发生时间; e[i] 与 l[i] 分别表示活动  $a_i$  的最早开始时间与最晚开始时间。

三、(本题共 30 分, 每小题 10 分)

欲建立一文献库, 其正文 (文献本身) 存放在一个双向循环链表的各个链结点中。

1. 为便于链结点的插入、删除操作, 以及按题目、发表日期、发表者名称、主题词 (假设每文最多给出三个主题词) 进行检索, 请设计该链表的链结点结构 (给出链结点每个域的名称, 并说明该域内存放什么信息。注: 以下各小题设计链结点结构也这样要求)。画出整个链表结构的示意图。

2. 设计一个三级索引结构, 其中第三级索引称为题目索引, 是按文献题目构造的稠密索引, 通过该级索引并根据给定题目可得到每个文献的存放地址; 该级索引按文献学科类分类存放。第二级索引称为中类索引, 是题目索引的索引, 指出同一中类的文献题目索引的存放位置 (例如农林类、气象类,, , 古代史类、近代史类,, )。第一级索引称为大类索引, 指出同一大类 (如: 自然科学类、历史类,, ) 的文献的中类索引的存放位置。请设计每一级索引的结点结构, 并画出该索引的整体示意图。

3. 再设计一种三级索引结构, 其中第三级索引仍是题目索引 (与 2 题所述相同), 第二级索引把具有相同主题词的文献题目索引地址组织在一个单链表中。第一级索引称为主题词索引, 用文献给出的主题词作关键字组成杂凑表, 即该级索引为一个杂凑表, 能够指出具有同一主题词的文献题目索引的索引链表的第一个链结点的存储位置。该杂凑表采用链地址法处理冲突。请设计每一级索引的结点结构, 并画出该索引的整体示意图。

四、(本题 10 分)

已知非空线性链表由 list 指出, 链结点的构造为



请写一算法, 将链表中数据域值最小的那个链结点移至链表的最前面。要求: 不得额外申请新的链结点。

五、(本题 15 分, 第 1 小题 5 分, 第 2 小题 10 分)

已知求两个正整数  $m$  与  $n$  的最大公因子的过程用自然语言可以表述为反复执行如下动作:

第一步: 若  $n$  等于零, 则返回  $m$ ;

第二步: 若  $m$  小于  $n$ , 则  $m$  与  $n$  相互交换; 否则, 保存  $m$ , 然后将  $n$  送  $m$ , 将保存的  $m$  除以  $n$

的余数送  $n$ 。

1. 将上述过程用递归函数表达出来 (设求  $x$  除以  $y$  的余数可以用  $x \text{MOD} y$  形式表示)。
2. 写出求解该递归函数的非递归算法。

六、(本题 10 分)

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  $pos$ 。请用 C 语言实现该函数。假设分配给字符串  $s$  的空间足够容纳字符串  $t$  插入。(说明: 不得使用任何库函数。)

七、(本题 15 分)

命令 `sgrep` 用来在文件中查找给定字符串, 并输出串所在行及行号。

命令格式为: `sgrep[-i]filename searchstring`

其中: `-i`: 表示查找的大小写无关, 省略时表示大小写相关。

`filename`: 给定文件名。

`searchstring`: 所要查找的串。

用 C 语言实现该程序, 该程序应具有一定的错误处理能力。(提示: 使用命令行参数)

注意: 除文件及输入/出操作可使用库函数外, 其他不允许使用库函数。

### 西安交通大学 2001 年数据结构试题

一、判断下列叙述是否正确, 正确的填  $\square$ , 不正确的填  $\times$  (10 分)

1. 数据对象就是一组数据元素的集合。 ( )
2. 任何一棵前序线索二叉树, 都可以不用栈实现前序遍历。 ( )
3. 就平均查找长度而言, 分块查找最小, 折半查找次之, 顺序查找最大。 ( )
4. 用 shell 方法排序时, 若关键字的排列杂乱无序, 则效率最高。 ( )
5. 在  $m$  阶 B-树中, 所有非终端结点至少包含  $\lceil m/2 \rceil$ 。 ( )

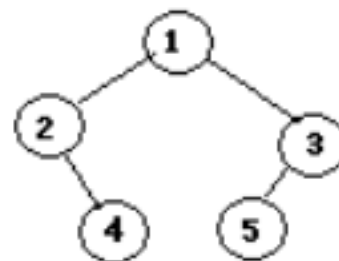
二、选择填空题 (15 分)

1. 假设以数组  $A[m \dots n]$  存放循环队列的元素, 其头指针是  $front$ , 当前队列有  $k$  个元素, 则队列的尾指针为 ( )。

- A.  $(front+k) \bmod (n-m+1)$
- B.  $(m+k) \bmod n + front$
- C.  $(front-m+k) \bmod (n-m+1) + m$
- D.  $(front-m+k) \bmod (n-m+1)$

2. 已知二叉树如右图所示, 此二叉树的顺序存储的结点序列是 ( )。

- A. 123 45
- B. 12345
- C. 12 435
- D. 24153



3. 若用冒泡排序对关键字序列 {20, 17, 11, 8, 6, 2} 从小到大进行排序, 则需要交换的总次数为 ( )。

- A. 3
- B. 6
- C. 12
- D. 15

4. 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.  $head = \text{NULL}$
- B.  $head \rightarrow \text{next} = \text{NULL}$
- C.  $head \rightarrow \text{next} == head$
- D.  $head \neq \text{NULL}$

5. 已知串  $s = \text{'ABCDEFGH'}$ , 则  $s$  的所有不同子串的个数为 ( )。

- A. 8
- B. 9
- C. 36
- D. 37

三、填空题 (15 分)

1. 假设一个 12 阶的上三角矩阵  $A$  按行优先顺序压缩存储在一维数组  $B$  中, 则非零元素  $a_{8,9}$  在  $B$  中的存储位置  $k = \underline{\hspace{2cm}}$ 。

2. 在拓扑排序中, 拓扑序列的第一个顶点必定是 入度为 0 的顶点。

3. 由六个分别带权值为 5, 12, 9, 30, 7, 16 的叶子结点构造一棵哈夫曼 (Huffman) 树, 该树的结点个数为 15, 树的带权路径长度为 259。

4. 在 链队列 的情况下, 链队列的出队操作需要修改尾指针。

5. 对表长为  $n$  的顺序表进行分块查找, 若以顺序查找确定块, 且每块长度为  $s$ , 则在等概率查找的情况下, 查找成功时的平均查找长度为  $\frac{n}{s} + 1$ 。

四、简答题 (50 分)

1. (6 分) 已知广义表  $L : ((a, b), (c, (d, (e)))) , f)$ 。

(1) 试利用广义表取表头  $head(L)$  和取表尾  $tail(L)$  的基本运算, 将原子  $c$  从下列广义表中分解出

来，请写出运算表达式。

(2)请给出下列表达式的运算结果：

hem(tail(head(tail(L))))          tail(tail(head(L)))

2 . (10 分)已知一个图  $G=(V, E)$ ，其中：

$V=\{a, b, c, d, e, f\}$ ；

$E=\{<a, b>, <a, d>, <a, e>, <d, e>, <e, b>, <c, b>, <c, e>, <c, f>, <f, e>\}$

(1)请画出该图，并写出邻接矩阵。

(2)根据邻接矩阵，分别同从顶点  $a$  出发的深度优先和广度优先遍历序列。

(3)画出由此得到的深度优先和广度优先生成树 (或森林)。

3 . (6 分)已知一个栈  $s$  的输入序列为  $abcd$ ，下面两个序列能否通过栈的  $PUSH$  和  $POP$  操作输出；如果能，请写出操作序列；如果不能，请说明原因。

dbca          cbda

4 . (5 分)设模式串  $t="ababacababa"$ ，试求出  $t$  的  $next$  和  $nextval$  函数值。

5 . (7 分)已知一组关键字  $K=\{20, 15, 4, 18, 9, 6, 25, 12, 3, 22\}$ ，请判断此序列是否为堆？如果不是，请调整为堆。

6 . (8 分)对于表达式  $(a-b+c)*d/(e+f)$

(1)画出它的中序二叉树，并标出该二叉树的前序线索；

(2)给出它的前缀表达式和后缀表达式。

7 . (8 分)设散列长度为 9，散列函数为  $H(k)=k \bmod 9$ ，给出关键字序列：23, 45, 14, 17, 9, 29, 37, 18, 25, 41, 33，采用链地址法解决冲突。

(1)请画出散列表；

(2)求出查找各关键字的比较次数；

(3)计算在等概率情况下，查找成功的平均查找长度。

五、算法填空 (10 分)

已知图的邻接表存储结构描述如下：

CONST N= 图的顶点数：

TYPE arcptr=    arcnode；

    arcnode=RECORD

        adjvex : integer；

        nextarc : arcptr

    END；

    vexnode=RECORD

        vexdata : char；

        firstarc : arcptr

    END；

Graph=ARRAY[1 .. N]of vexnode；

下面是一个图的深度优先遍历的非递归算法，请在 \_\_\_\_\_ 处填上适当内容，使其成为一个完整算法。

PROCEDURE traver\_dfs(g : graph)；

VAR visited : ARRAY[1 .. N] of BOOLEAN；

s : STACK；{s 为一个栈}

p : arcptr；

BEGIN

    FOR i : =1 TO N DO visited[i] : = \_\_\_\_\_

    INISTACK(S)；

    FOR i=1 TO N DO

        BEGIN

            BEGIN

                visited[i] : =true；

                visit(i)；     //访问第 i 个顶点

                IF g[i].firstarc   NIL THEN

                    PUSH(d, g[i].firstarc)

            END



```

        WHILE _____ DO
            BEGIN
                p : POP(s) ;
                IF p .nextarc = NIL THEN
                    PUSH(s , p .nextarc) ;
                    j : =p .adjvex ;
                    IF NOT visited[j] THEN
                        BEGIN
                            visited[j] : =true ;
                            visit(j) ;
                            IF e[j].firstarc = NIL THEN
                                _____
                            END
                        END
                    END
                END
            END
        END
    
```

中国科学院软件研究所 2001 年数据结构试题

一、单项选择题 (每空 2 分, 共 20 分)

- 下列函数中渐近时间复杂度最小的是 ( )。
 

A .  $T_1(n)=n\log_2 n+5000n$       B .  $T_2(n)=n^2-8000n$   
 C .  $T_3(n)=n^{\log_{22} 1}-6000n$       D .  $T_4(n)=2n\log_2 n-7000n$
- 线性表的静态链表存储结构与顺序存储结构相比优点是 ( )。
 

A . 所有的操作算法实现简单  
 B . 便于随机存取  
 C . 便于插入和删除  
 D . 便于利用零散的存储器空间
- 设栈的输入序列为  $1, 2, \dots, n$ , 输出序列为  $a_1, a_2, \dots, a_n$ , 若存在  $1 \leq k \leq n$  使得  $a_k=n$ , 则当  $k \leq i \leq n$  时,  $a_i$  为 ( )。
 

A .  $n-i+1$       B .  $n-(i-k)$       C . 不确定
- 设高度为  $h$  的二叉树 (根的层数为 1) 上只有度为 0 和度为 2 的节点, 则此类二叉树中所包含的节点数至少为 ( )。
 

A .  $2h$       B .  $2h-1$       C .  $2h+1$       D .  $h+1$
- 设指针  $p$  指向线索树中的某个结点, 则查找  $*p$  在某种次序下的前驱或后继不能获得加速的是 ( )。
 

A . 前序线索树中查找  $*p$  的前序后继  
 B . 中序线索树中查找  $*p$  的中序后继  
 C . 中序线索树中查找  $*p$  的中序前继  
 D . 后序线索树中查找  $*p$  的后序前继
- 假定有  $k$  个关键字互为同义词, 若用线性探测法将这  $k$  个关键字存入散列表中, 至少需要进行 ( ) 次探测。
 

A .  $k-1$       B .  $k$       C .  $k+1$       D .  $k(k+1)/2$
- 若待排序元素基本有序, 则下列排序中平均速度最快的排序是 ( ) ; 若要求辅助空间为  $O(1)$ , 则平均速度最快的排序是 ( ) ; 若要求排序是稳定的, 且关键字为实数, 则平均速度最快的排序是 ( )。
 

A . 直接插入排序      B . 直接选择排序      C . Shell 排序  
 D . 冒泡排序      E . 快速排序      F . 堆排序  
 G . 归并排序      H . 基数排序
- 对于多关键字而言, ( ) 是一种方便而又高效的文件组织文式。
 

A . 顺序文件      B . 索引文件      C . 散列文件      D . 倒排文件

二、问答题 (共 25 分)

- 设  $A[-2 : 6; -3 : 6]$  是一个用行主序存储的二维数组, 已知  $A[-2, -3]$  的起始存储位置为  $\text{loc}(-2, -3)=1000$ , 每个数组元素占用 4 个存储单元, 求: (6 分)
  - $A[4, 5]$  的起始存储位置  $\text{loc}(4, 5)$  ;

- 2) 起始存储位置为 1184 的数组元素的下标。
2. 给定二叉树的先序和后序遍历序列, 能否重构出该二叉树? 若能, 试证明之, 否则给出反例。(6 分)
3. 在含有  $n$  个关键字的  $m$  阶 B-树中进行查找, 至多读盘多少次? 完全平衡的二叉排序树的读盘次数大约比它大多少倍? (设两种树中的每个节点均是一个存储块)。(8 分)
4. 用向量表示的循环队列的队首和队尾位置分别为 1 和  $\text{max\_size}$ , 试给出判断队列为空和为满的边界条件。(5 分)

### 三、阅读程序题 (共 10 分)

1. 设  $G$  是一个有向无环图, 试指出下述算法的功能, 输出的序列是  $G$  的什么序列? (10 分)

```
void Demo (ALGraph G)
{
    //G 是图的逆邻接表, 向量 outdegree 的各分量初值为 0。
    for(i=0 ; i<G.NodeNum ; i++)
        for(p=G.adjlist[i].firstedge ; p ; p=p->next)
            //扫描 i 的入边表
            outdegree[p->adjvex]++ ; //设 p->adjvex=j, 则将 <i, j>的起点
                                    j 的出度加 1
    initStack(&s) ; //设置空栈 s
    for(i=0 ; i<G.NodeNum ; i++)
        if(outdegree[i]==0)
            Push(&s, i) ; //出度为 0 的顶点 i 入栈
    while(!StackEmpty(s)) //栈 s 非空
    {
        i=Pop(&Q) ; //出栈, 相当于删去顶点 i
        printf( "%c ", G.adjlist[i].vertex) ; //输出顶点 i
        for(p=G.adjlist[i].firstedge ; p ; p=p->next)
        {
            //扫描 i 的入边表
            j=p->adjvex ; //j 是 i 的入边 <j, i>的起点
            outdegree[j]-- ; //j 的出度减 1, 即删去 i 的入边 <j, i>
            if(!outdegree[j])
                Push(&s, j) ; //若 j 的出度为 0, 则令其入栈
        }
    }
    //endfor
    //endwhile
}
//endDemo
```

### 四、算法题 (每题 15 分, 共 45 分)

1. 试设计算法在  $O(n)$  时间内将数组  $A[1 \dots n]$  划分为左右两个部分, 使得左边的所有元素值均为奇数, 右边的所有元素值均为偶数, 要求所使用的辅助存储空间大小为  $O(1)$ 。
2. 试写一递归算法, 从大到小输出二叉排序中所有的值不小于  $x$  的关键字, 要求算法的时间为  $O(h+m)$ , 其中  $h$  为树的高度,  $m$  为输出的关键字个数。
3. 设  $G$  是以邻接表表示的无向图,  $v_0$  是  $G$  中的一个顶点,  $k$  是一个正的常数。要求写一算法打印出图中所有与  $v_0$  有简单路径相通, 且路径长度小于等于  $k$  的所有顶点 (不含  $v_0$ ), 路径长度由路径上的边数来定义。

## 参考答案

### 第一章

1. (a) 是线性结构, 对应的数据逻辑结构图见图 1-2。  
 (b) 是树形结构, 对应的数据逻辑结构图见图 1-3。  
 (c) 是有向图, 其数据逻辑结构图见图 1-4。

2. (a)  $O(m \times n)$

(b)  $s+=b[i][j]$  的重复执行次数是  $n(n+1)/2$ , 时间复杂度是  $O(n^2)$

(c)  $O(\log_2 n)$

3. D 4 . D 5 . D

### 第二章

#### 一、单项选择题

- (1) ~ (5) DACAA (6) ~ (10) BAAAD

#### 二、填空题

(1) 链域 (2) 插入 (3) 删除 (4)  $n-i+1$  (5)  $n-1$  (6)  $n$  / 2 (7)  $(n-1)$  / 2  
 (8)  $h=NULL$  (9)  $h->next=NULL$  (10)  $p->next=h$  (11)  $p->next->prior=s$   
 (12)  $p->next=s$  (13)  $p->prior->next$  (14)  $p->prior=s$  (15)  $p->next->prior=p$   
 (16)  $p->prior->next=p$  (17)  $p->next->prior=p->prior$  (18)  $r->next=p$   
 (19)  $r=p$  (20)  $r->next=q->next$

### 三、应用题

1. 顺序表：逻辑上相邻的数据元素其物理存储位置必须紧邻。其优点是：节省存储，可以随机存取。

链表：逻辑上相邻的数据元素其物理存储位置不要求紧邻，用指针来描述结点间的逻辑关系，故插入和删除运算比较方便。

2. 在带表头结点的单链表中，头指针指向表头结点，不带表头结点的单链表中，头指针指向表中第一个结点（又称首元），当进行查找运算时，必须从头指针开始去顺次访问表中每一个元素。

头结点是另设一个结点，其指针域指向存放表中第一个元素的结点，设置头结点的好处是：无需对在第一个结点前插入一个结点或删除一个结点时进行特殊处理，也可以对空表和非空表的运算进行统一处理。

### 四、算法设计题

1. 先判断表满否，若表未滿，则从最后一个元素  $a_n$  开始，逐个与  $x$  进行比较，若  $a_i > x$  ( $1 \leq i \leq n$ )，则将  $a_i$  后移一个位置，直到  $a_i \leq x$ ，最后把  $x$  插入到这个位置，表长  $+1$ 。算法描述如下：

```
#define M 100
int insort(Slist *L, int z)
{int i;
  if(L->n>=M)
  {printf("overflow");
   return 0; }
  else
  for(i=L->n;i>=0;i--)
  if(L->data[i]>x
    L->data[i+1]=L->data[i];
  else break;
  L->data[i+1]=x;
  L->n++;
  return 1;
}
```

2. 逐个查找单链表中的结点  $x$ ，并计数。

```
int number(lnode *h,int x)
{ int n=0;
  while(h)
  {if(h->data==x)
    n++;
    h=h->next;
  }
  return n;
}
```

3. 前插法建立带表头结点的单链表算法中的  $tag$  为输入数据结束标志。

```
Lnode *createhh(int tag)
{ int x;
  Lnode *p, *h=(Lnode *)malloc(sizeof(Lnode));
  h->next=NULL;
  printf("input x: ");
  scanf("%d",&x);
  while(x!=tag)
  {p=(Lnode *)malloc(sizeof(Lnode));
```

```

p->data=x;
p->next=h->next;
h->next=p;
scanf(    "%d",&x);
}
return h;
}

```

4. 先建立一个表头结点，用尾插法建立该单链表。然后将尾结点的指针域值置为表中第一个结点的首地址，最后释放表头结点。算法描述如下：

```

Lnode *createht(int tag)
{ int x    ;
  Lnode * p , *r,* h=(Lnode *)malloc(sizeof(Lnode))    ;
  r=h;
  printf(    "input x:    ") ;
  scanf(    "%d",&x);
  while(x!=tag)
  {p=(Lnode*)malloc(sizeof(Lnode));
   p->data=x;
   r->next=p;
   r=p;
   scanf(    "%d",&x);
  }
  r->next=h->next;
  free(h);
  return r;
}

```

5. 设 p 指向待逆置链表中的第一个结点，先将表头结点的链域置空。顺次取出待逆置链表中的第一个结点，用前插法插入到带表头结点的单链表中。

```

Void reverseh(Lnode *h)
{ Lnode *s,*p=h->next;
  h->next=NULL;
  while(p)
  {s=p;p=p->next;
   s->next=h->next;
   h->next=s;
  }
}

```

6. 逐个检测顺序表中其值在 x 和 y 之间的元素，并计数 k, 再将其值大于 y 的元素向前移动 k 个元素。算法描述如下：

```

void deletexy(Slist *a,int x,int y)
{ int i,k=0;
  for(i=0;i<a->n;i++)
  if(a->data[i]>=x&&a->data[i]<=y)
    k++;
  else
    a->data[i-k]=a->data[i];
  a->n=a->n-k;}

```

### 第三章

#### 一、单项选择题

(1)-(5) CACCB (6)-(7)BC

#### 二、填空题

(1) 栈顶 (2) 栈底 (3) 队尾 (4) 队首 (5)O(n) (6)O(1) (7)O(1)

(8)O(1) (9)front=rear (10)(rear+1) % m=front (11)(rear-front+m) % m

(12) 向是的两端

### 三、应用题

1 . 栈和队列是操作位置受限的线性表，即对插入和删除运算的位置加以限制。栈是仅允许在表的一端进行插入和删除运算的线性表，因而是后进先出表。而队列是只允许在表的一端插入，另一端进行删除运算的线性表，因而是先进先出表。

2 . 循环队列的优点是解决了 “假上溢” 问题。

设用数组  $A[m]$  来存放循环队列，设  $front$  指向实际队首， $rear$  指向新元素应插入的位置时， $front=rear$  即可能是队空，又可能是队满，为了区分队空和队满可采用下列三种方案：

(1) 用 “少用一个元素空间” 的方案，初态  $front=rear=0$ ，判断队空的条件是  $front=rear$ ，判断队满的条件是  $(rear+1) \% m=front$ 。

(2) 用 “设队空标志” 的方法，如设  $front=-1$  时为队空标志，初态是  $front=-1$ ， $rear=0$ 。当  $front=rear$  时则队满。

(3) 用  $count$  存放表中元素个数，当  $count=0$  时为队空，当  $count=m$  时为队满。

3 . 若进栈序列为  $a, b, c, d$ ，全部可能的出栈序列是：

$abcd$ ， $abdc$ ， $acbd$ ， $acdb$ ， $adcb$ ， $bacd$ ， $badc$ ， $bcad$ ， $bcda$ ， $bdca$ ， $cbad$ ， $cbda$ ， $cdba$ ， $dcba$ ，

不可能的出栈序列为： $adbc$ ， $bdac$ ， $cdab$ ， $cadb$ ， $dabc$ ， $dacb$ ， $dbac$ ， $dbca$ ， $dcab$ 。

4 . 由于队列中操作遵循的原则是先进先出，出队元素的顺序是  $bdcfea$ ，故元素进队的顺序也是  $bdcfea$ ，元素进队的顺序与元素出栈的顺序相同，在栈中存取数据遵循的原则是后进先出，要得到出栈序列  $bdcfea$ ，栈的容量至少是应该存放 3 个元素的空间。

5. 3 4 25 6 15+/-8\*+

### 四、算法设计题

1 . 解本题的基本思想是：先将顺序队列  $q$  中所有元素出队，并依次进入顺序栈  $s$  中，然后出栈，再依次入队。设队列中的元素为  $a_1, a_2, \dots, a_n$ ，出队并进栈的序列为  $a_1, a_2, \dots, a_n$ ，出栈并入队的序列为  $a_n, a_{n-1}, \dots, a_1$ ，可见顺序队列  $q$  中所有元素已逆置了。顺序队列的类型定义为：

```
#define MAX 100
typedef struct
{int data[MAX];
  int front, rear;
}Squeue;
```

算法描述如下：

```
void invert(Squeue *q)
{int s[MAX], top=0;
 while(q->front<q->rear)
  s[top++]=q->data[++q->front];
 q->front=-1; q->rear=0;
 while(top>0)
  q->data[q->rear++]=s[--top];
}
```

2.

(1) 递归算法：

```
int f1(int n)
{ int f;
  if(n==0) return (1);
  else
    return(n*f1(n/2));
}
```

(2) 将上述递归函数转换成非递归函数时，使用了一个栈，用于存储调用参数和函数值，这里采用一维数组  $s[n]$  作为栈，用一个栈顶指针  $top$ 。算法如下：

```
#define maxlen 200;
int f2(int n)
{ int s[maxlen];
```

```

int top=0,fval=1;
while(n!=0)
{
    s[top]=n;
    n=n/2;
    top++;
}
while(top>=0)
{
    fval=fval*s[top];
    top--;
}
return fval;
}

```

3. 仅设队尾指针的带表头结点的循环链表，找队首结点为 rear->next->next。

(1) 初始化

```

Lnode *int queue()
{
    Lnode *rear;
    rear=(Lnode*)malloc(sizeof(Lnode));
    rear->next=rear;
    return rear;
}

```

(2) 入队算法

```

Lnode *enqueue(Lnode *rear,int x)
{
    Lnode *p=(Lnode*)malloc(sizeof(Lnode));
    p->data=x;
    p->next=rear->next;
    rear->next=p;
    return p;
}

```

(3) 出队算法

```

int outqueue(Lnode *rear,int *x)
{
    Lnode *p;
    if(rear->next==rear)
    {
        printf("queue is empty ");
        return 0;
    }
    p=rear->next->next;
    *x=p->data;
    rear->next->next=p->next;
    free(p);
    return 1;
}

```

#### 第四章

##### 一、单项选择题

(1) —(5)DBDBD (6)-(10)BDBBA (11)-(13)CAA

##### 二、填空题

(1) 两个串的长度相等且对应位置上的字符相同。

(2) 是由零个字符组成的串 (3)0

(4) "goodmorning " (5) "nin " (6)4 (7) "goto " (8)m

(9)m(n-m+1) (10)Loc(a[0][0])=1000-88=912

(11)n(n+1)/2+1 (12) n(n+1)/2 (13)i(i-1)/2+j-1

(14)(2n-j+2)(j-1)/2+i-j (15)3 (16)4 (17)a(18)((b),((c,(d))))

(19) 三元组表 (20) 十字链表 (21) 子表 (22) 子表



### 三、应用题

1. 用联接、取子串、置换运算将串  $S = "(xyz)+^*$  转化为  $T = "(x+z)^*y"$ 。

$C1 = \text{substr}(S, 3, 1) = "y"$

$C2 = \text{substr}(S, 6, 1) = "+"$

$C3 = \text{substr}(S, 7, 1) = "*"$

$C4 = \text{concat}(C3, C1) = "*y"$

$T = \text{replace}(S, 3, 1, C2) = \text{replace}(S, 3, 1, \text{substr}(S, 6, 1)) = "(x+z)+x"$

$T = \text{replace}(T, 6, 2, \text{concat}(C3, C1)) = \text{replace}(T, 6, 2, \text{concat}(\text{substr}(S, 7, 1), \text{substr}(S, 3, 1))) =$

$"(x+z)^*y"$ 。

若只用取子串和联接操作进行的转换过程是：

$C1 = \text{concat}(\text{substr}(S, 1, 2), \text{substr}(S, 6, 1)) = "(x+"$

$C2 = \text{concat}(\text{substr}(S, 7, 1), \text{substr}(S, 3, 1)) = "*y"$

$T = \text{concat}(\text{concat}(\text{concat}(\text{substr}(S, 1, 2), \text{substr}(S, 6, 1)), \text{substr}(S, 4, 2)), \text{concat}(\text{substr}(S, 7, 1), \text{substr}(S, 3, 1)))$

2. 串  $S$  的长度为  $n$ ，其中的字符各不相同，所以  $S = "(x+z)^*y"$  中含 1 个字符的子串有  $n$  个，2 个字符的子串有  $n-1$  个……，含  $n-2$  个字符的子串有 3 个，含  $n-1$  个字符的子串有 2 个，共有非平凡子串的个数是  $n(n+1)/2-1$ 。

3. 串  $T$  的 next 和 nextval 函数值见下表：

下标 j	1	2	3	4	5	6	7	8	9	10	11
串 T	a	b	c	a	A	c	c	B	a	c	a
next[j]	0	1	1	1	2	2	1	1	1	2	1
nextval[j]	0	1	1	0	2	2	1	1	0	2	0

4. 特殊矩阵是指具有相同值的矩阵元素或零元素的分布具有一定规律，可以将其压缩存储在一维数组中，矩阵元素  $a_{ij}$  的下标  $i$  和  $j$  与其在一维数中存放的下标  $k$  之间存在一一对应关系，故不会失去随机存取功能。

稀疏矩阵中零元素的分布没有一定规律，可以将非零元素存于三元组表中，非零元素  $a_{ij}$  在三元组中的存放位置与  $i$ 、 $j$  没有对应关系，故失去随机存取功能。

5.  $N$  阶对称矩阵  $A$  中， $a_{ij} = a_{ji}$ ，可以仅存放下三角元素（或上三角元素）。设  $r = \max(i, j)$ ， $c = \min(i, j)$ ，

$$k = r(r-1)/2 + c - 1;$$

例如，4 阶对称矩阵  $A$ ，按行优先顺序存于一维数组  $F[10]$  中，

0	1	2	3	4	5	6	7	8	9
a11	a21	a22	a31	a32	a33	a41	a42	a43	a44

当  $i=3$ ， $j=2$  时， $k=3*2/2+2-1=4$ ;

当  $i=2$ ， $j=3$  时， $k=4$

(2) 当对称矩阵  $A$  按列优先顺序压缩存放，若仅存放上三角元素，则有：

$$k = r(r-1)/2 + c - 1$$

例如，4 阶对称矩阵  $A$ ，按列优先顺序存于一维数组  $F[10]$  中，

0	1	2	3	4	5	6	7	8	9
a11	a12	a22	a13	a23	a33	a14	a24	a34	a44

当  $i=1$ ， $j=3$  时， $k=3*2/2+1-1=3$ ；当  $i=3$ ， $j=1$  时， $k=3$

## 第五章

### 一、单项选择题

(1)-(5)BBCDC (6)-(10)BCABC (11) —(15)DABBD (16)-(19)CCABB  
(20)-(24)BBBAC (25)-(27)DBC

### 二、填空题

(1) 有零个或多个 (2) 有且仅有一个

(3) 根据树的广义表表示，可以画出这棵树，该树的度为 4。

(4) 树的深度为 4

(5) 树中叶子结点个数为 8

(6)  $n_0 = 14$  (7)  $n - 2m + 1$  (8)  $2^{k-1}$  (9)  $2^{i-1}$  (10) 133 (11) 59

(12)  $2^5 = 32$  (13)  $\lceil \log_2(n+1) \rceil = \lceil \log_2 69 \rceil = 7$  (14)  $2^5 - 1 + 6 = 37$  (15) 19

(16)  $2^7 - 1 - 20 = 107$  (17) 右 (18)  $m+1$  (19)  $n+1$  (20)  $2m-1$

(21) 中序 (22) 直接前驱结点 (23) 直接后继结点

### 三、应用题

1. 具有  $n$  个结点的满二叉树中只有度为 2 和度为 0 的结点, 故  $n = 2n_0 - 1, n_0 = (n + 1) / 2$ 。

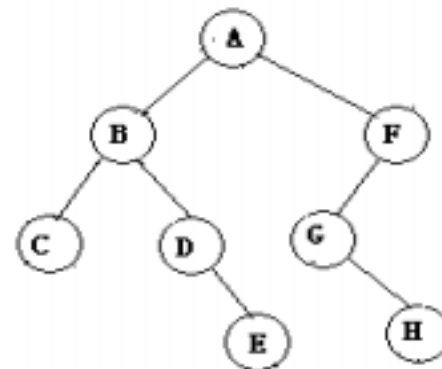
5. 构造一棵二叉树应该先确定根结点, 由后序序列最后一个字母 A 确定根结点, 可将前序序列的第一个字母 \* 改为 A。在中序遍历序列中 A 的左子树中含 4 个字母, 右子树中含 3 个字母, 在后序遍历序列中左子树的后序遍历子序列为 \*EDB, 可将左子树的后序遍历序列和中序遍历序列互相补足, 分别是 CEDB 和 CBDE。由这两个子序列可知左子树的根结点为 B, D 又为以 B 为根结点的右子树的根结点, E 为 D 的右孩子, 所以在前序序列中 A 的左子树的前序序列为: BCDE。在根结点 A 的右子树中, 由后序遍历序列可确定其根结点为 F, 再确定其中序序列为 GHF, 进一步确定其前序序列为 FGH, 补足 \* 处后, 三个遍历序列为:

(1) 前序遍历序列是: ABCDEFGH

(2) 中序遍历序列是: CBDEAGHF

(3) 后序遍历序列是: CEDBHGFA

由前序遍历序列和中序遍历序列可构造一



棵二叉树如右图。

### 四、算法设计题

1. 求叶子结点个数。

算法描述如下:

```
int leafnum(Csnode *t, int n)
{
    Gsnode *p;
    if(t == NULL) return 0;
    p = t->fc;
    if(p == NULL)
        n++;
    while(p)
    {
        n = leaf(p, n);
        p = p->ns;
    }
    return n;
}
```

2. 交换左右孩子。算法如下:

```
void exchange(Btnode *t)
{
    Btnode *p;
    if(t)
    {
        exchange(t->lchild);
        exchange(t->rchild);
        if(t->lchild || t->rchild)
        {
            p = t->lchild;
            t->lchild = t->rchild;
            t->rchild = p;
        }
    }
}
```

3. void printbtree(Btnode \*r)

```
{
    if(r)
    {
        printf(" %c", r->data);
        if(r->lchild || r->rchild)
            printf(" ( ");
        printbtree(r->lchild);
        if(r->rchild)
            printf(" , ");
        printbtree(r->rchild);
        printf(" ) ");
    }
}
```

```

4. #define MAX 100
int checkt(Btnode *t)
{
    Btnode *s[MAX];
    int i,n=0;
    for(i=0;i<MAX;i++)
        s[i]=NULL;
    if(t==NULL)
        return 0;
    i=0;
    S[0]=t;
    While(i<=n)
    {
        if(!s[i])
            return 0;
        if(s[i]->lchild)
        {
            n=2*i+1;
            s[n]=s[i]->lchild;
        }
        if(s[i]->rchild)
        {
            n=2*i+2;
            s[n]=s[i]->rchild;
        }
        i++;
    }
    return 1;
}

```

5. (1) 前序遍历二叉树的非递归算法的基本思想是：从二叉树的根结点开始，沿左支一直走到没有左孩子的结点为止，在走的过程中访问所遇结点，并把非空右孩子进栈。当找到没有左孩子的结点时，从栈顶退出某结点的右孩子，此时该结点的左子树已遍历结束，然后按上述过程遍历该结点的右子树，如此重复，直到二叉树中所有结点都访问完毕为上。算法如下：

```

void preorder(Btnode *t)
{
    int l=0;
    Btnode *p,*s[M];
    P=t;
    Do
    {
        while(p)
        {
            printf("    %c\t",p->data);
            if(p->rchild!=NULL)
                s[i++]=p->rchild;
            p=p->lchild;
        }
        if(i>0)
            p=s[--i];
    }while(i>0||p!=NULL);
}

```

(2) 后序遍历二叉树的非递归算法的基本思想：采用一个栈保存返回的结点，先遍历根节点的所有结节点并入栈，出栈一个结点，然后遍历该结点的右结点并入栈，再遍历该右结点的所有左结点并入栈，当一个结点的左右子树均访问后再访问该结点，如此这样，直到栈为空为止。其中的难点是如何判断一个结点  $t$  的右结点已访问过，为此用  $p$  保存已访问过的结点（初值为  $NULL$ ），若  $t \rightarrow \text{right} = p$  成立（在后序遍历中， $t$  的右节点一定是在  $t$  之前访问），说明  $t$  的左右子树均已访问，现在应访问  $t$ 。算法如下：

```

void postorder(Btree *t)
{
    Btree *s[maxsize];
    Btree *p;
}

```

```

int flag,top=-1;
Do
{ while(t)
  { top++;
    s[top]=t;
    t=t->left;
  }
  p=NULL;
  flag=1;
  while(top!=-1&&flag)
  { t=s[top];
    if(t->right==p)
    { printf(      "%d" ,t->data);
      top--;
      p=t;
    }
    else
    { t=t->right;
      flag=1;
    }
  }
}while(top!=-1)
}

```

6. 求二叉树中从根结点到 p 结点之间路径长度的算法描述如下：

```

#define MAX 100
void path(Btnode *t,Btnode *p)
{ Btnode *s[MAX],*q=t;
  int b[MAX],top=-1;
  do {
    while(q)
    { s[++top]=q;
      b[top]=0;
      q=q->lchild;
    }
    if(top>=0&&b[top]==1)
    { q=s[top];
      if(q==p)
      { printf(      "根结点到 q 结点的路径是：  ") ;
        for(l=0;l<=top;l++)
          printf(      "%c",s[l]->data);
        return;
      }
      else top--;
    }
    if(top>=0)
    { p=s[top]->rchild;
      b[top]=1;
    }
  }while(top>0);
}

```

7. 判断以二叉链表存储的二叉树是否为完全二叉树的算法描述如下：

```

#define MAX 100
void checkt(BTnode *t)

```

```

{ Btnode *s[MAX];
  int i,n=0;
  for(i=0;i<MAX;i++)
    s[i]=MAX;
  if(t==NULL)
    return 0;
  s[0]=t;
  while(i<=n)
  { if(!s[i])
    return 0;
    if(s[i]->lchild)
    { n=2*i+1;
      s[n]=s[i]->lchild;
    }
    if(s[i]->rchild)
    { n=2*i+2;
      s[n]=s[i]->rchild;
    }
    i++;
  }
  return 1;
}

```

## 第六章

### 一、单项选择题

(1)-(4)AACA (5)-(9)DBCAD (10)-(11)CB

(12) 图 G 是一个非连通分量，至少有两个连通分量。含 8 个顶点的完全无向图共需 28 条边，另外 9 个顶点构成一个连通分量，所以至少含 9 个顶点。

### 二、判断题

- 1 . 正确 2 . 错误 3 . 正确 4 . 错误 5 . 正确 6 . 错误 7 . 正确 8 . 正确 9 . 正确  
 10 . 正确 ( 若 n 个顶点依次首尾相接构成一个环的有向强连通图，至少含 n 条边，即邻接矩阵中至少有 n 个小于 u 的非零元素 )。  
 11 . 错误 12 . 正确 13 . 错误 14 . 正确 15 . 正确  
 16 . 错误 17 . 错误 18 . 错误 19 . 正确

### 三、填空题

(1) $n(n-1)/2$  (2) $n(n-1)$  (3) $n-1$  (4)e (5)2e (6)出边 (7)入边 (8)n (9)有向图  
 (10) $O(n^2)$  (11) $O(n^2)$  (12) $O(n+e)$  (13) $O(n+e)$  (14)Kruscal (15)prim

### 四、算法题

1. 将图的深度优先遍历或广度优先遍历算法稍加改造，另设 m 保存访问的顶点个数，若已访问顶点个数 m 小于图中顶点个数 n，则图 G 不连通，否则为连通。现将深度优先遍历的非递归算法改造如下：

采用邻接表表示，类型定义如下：

```

#define MaxV 20
typedef struct node
{int adjvex;
  struct node *next;
}Enode;
typedef struct
{int data;
  Enode *firste;
}Vnode;
typedef struct
{Vnode adjlist[MaxV];

```

```

    int n;
    int e;
}ALGraph;
int connect(ALGraph *G)
{int i,j, Visited[MaxV],m=0,top=0;
  Enode * p,* S[MaxV];
  For(i=0;i<n;i++)
    Visited[i]=0;
  Visited[0]=0;
  Printf(    "%c",G->adjlist[0].data);
  m++;
  S[top++]=p->next;
  j=p->adjvex;
  if(visited[j]==0)
  {visited[j]=1;
   printf(    "%c",G->adjlist[j].data);
   m++;
   s[top++]=G->adjlist[j]->firste;
  }
}
}
if(m<G->a)
  return(0);
else
  return(1);}

```

- 2 . (1) 解：顺序计算每个顶点链表的表中结点个数，就是该顶点的出度  
算法如下：

```

void outdegree(graph g,int v)
{arcnode *p;
  int n=0;
  p=g.adjlist[v].firstarc;
  while(p)
  { n++;
   p=p->nextarc;}
  return n;
}
void outds(graph g)
{ int i;
  printf(    "各顶点出度： 、 \n ") ;
  for(i=0;i<g.vexnum;i++)
    printf(    "顶点 %d: %d\n",i,outdegree(g,i));
}

```

```

(2)void maxoutds(graph g)
{ int maxv=0,maxds=0i,x;
  for(i=0;i<g.vexnum;i++)
  { x=outdegredd(g,i);
   if(x>maxds)
    {maxds=x;maxv=1;}
  }
  printf(    "最大出度：顶点  %d的出度  %d",maxv,maxds);
}

```

```

(3)void zerods(graph g)

```



```

{ int i,x;
  printf(      “出度为 0 的顶点： \n ”);
  for(i=0;i<g.vexnum;i++)
  { x=outdegree(g,i);
    if(x==0)
      printf(      “%d”,i);
    }
  }
}

(4)void arc(graph g)
{ int i,j;
  arcnode *p;
  printf(      “输入边： \n ”);
  scanf(      “%d,%d”,i,j);
  p=g.adjlist[i].firstarc;
  while(p!=NULL&& p->adjvex!=j)
    p=p->nextarc;
  if(p==NULL)
    printf(      “不存在！ ”) ;
  else
    printf(      “存在 <%d,%d>边： \n ” , i,j)  ;
}

```

## 第七章

### 一、单项选择题

(1) — (5) DDCCD (6) — (9) DCCC

### 二、填空题

(1) 8 (2) 15 (3) 4.3 (4) 中序 (5) 记录的键值 (6) 7  
 (7) 63 (8) 17 (9) 17 (10)  $O(n)$  (11)  $O(\log_2 n)$  (12) 越大  
 (13) 散列函数 (14) 处理冲突方法 (15) 装填因子 (16) 索引表  
 (17) 该块的起始地址 (18) 100 (19) 100 (20) 101 (21)  $\lceil \log_2(n+1) \rceil$   
 (22) 同义词 (23)  $n(n+1)/2$  (24) 63 (25)  $m$  (26)  $\lceil m/2 \rceil$  (27)  $m-1$   
 (28)  $\lceil m/2 \rceil - 1$  (29)  $m$  (30)  $\lceil m/2 \rceil$

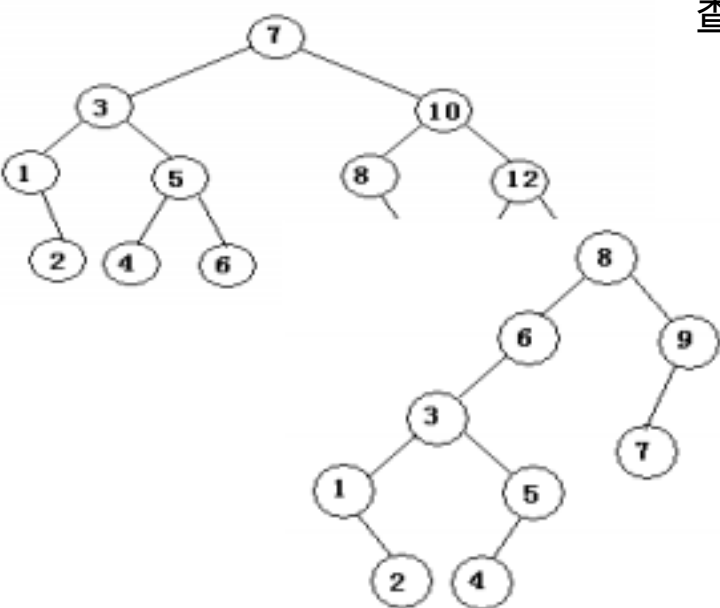
### 三、应用题

1. 判定树见下图。

查找成功的平均查找长度

$$ASL = (1 + 2 \times 2 + 3 \times 4 + 4 \times 6) / 13 = 41/13$$

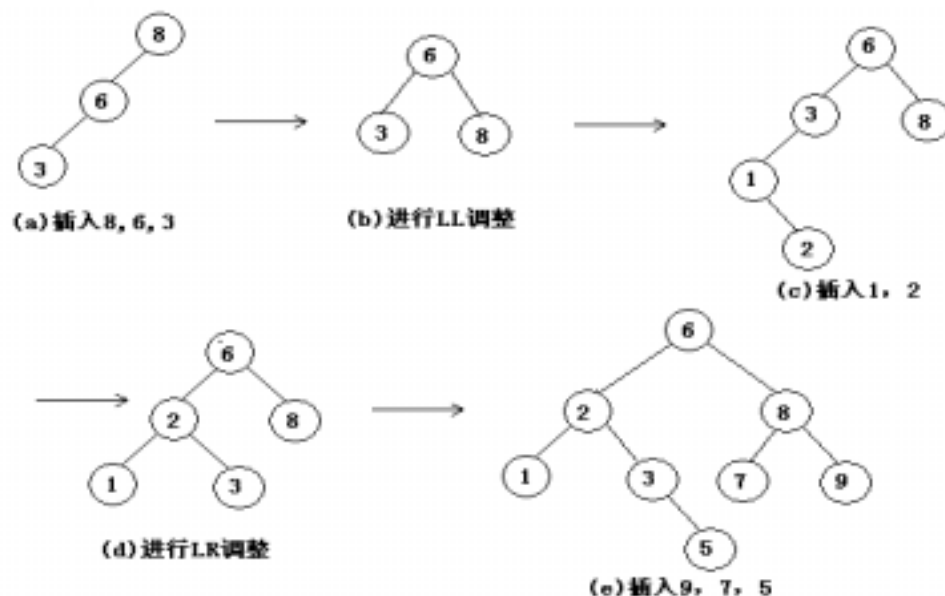
查找失败时与键值最多比较次数为 4。



2. 二叉排序树见右图。在等概率情况下，  
 查找成功的平均查找长度  $ASL = 29/9 = 3.2$   
 查找失败时，与键值最多比较次数是 5。

等概率情况下，查  
 $ASL =$   
 $/9 = 2.8$   
 查找失败时，与键

### 3. 构造一棵平衡二叉排序树

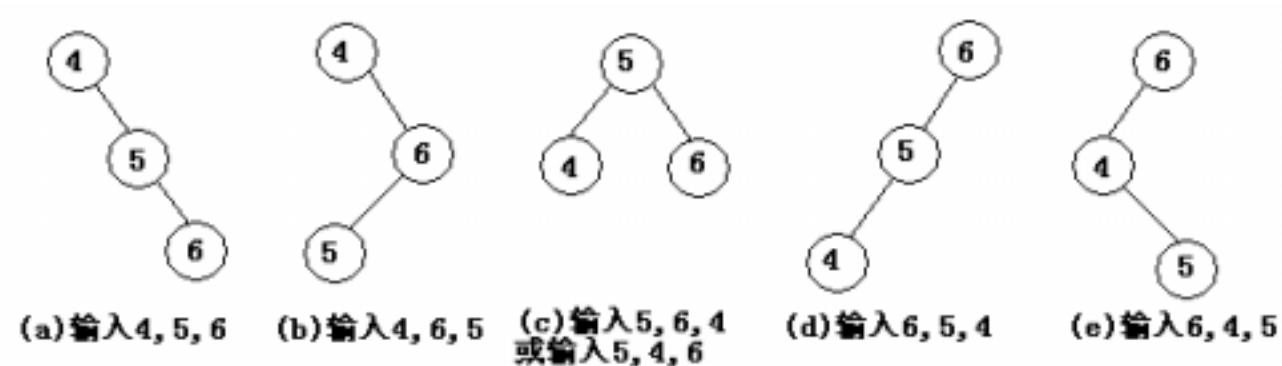


的过程见右图。  
 找成功的平均查找长度  
 $(1 + 2 \times 2 + 3 \times 4 + 4 \times 2)$   
 值的最多比较次数是 4。

### 4. 按不同的输

入顺序输入 4、5、6，建

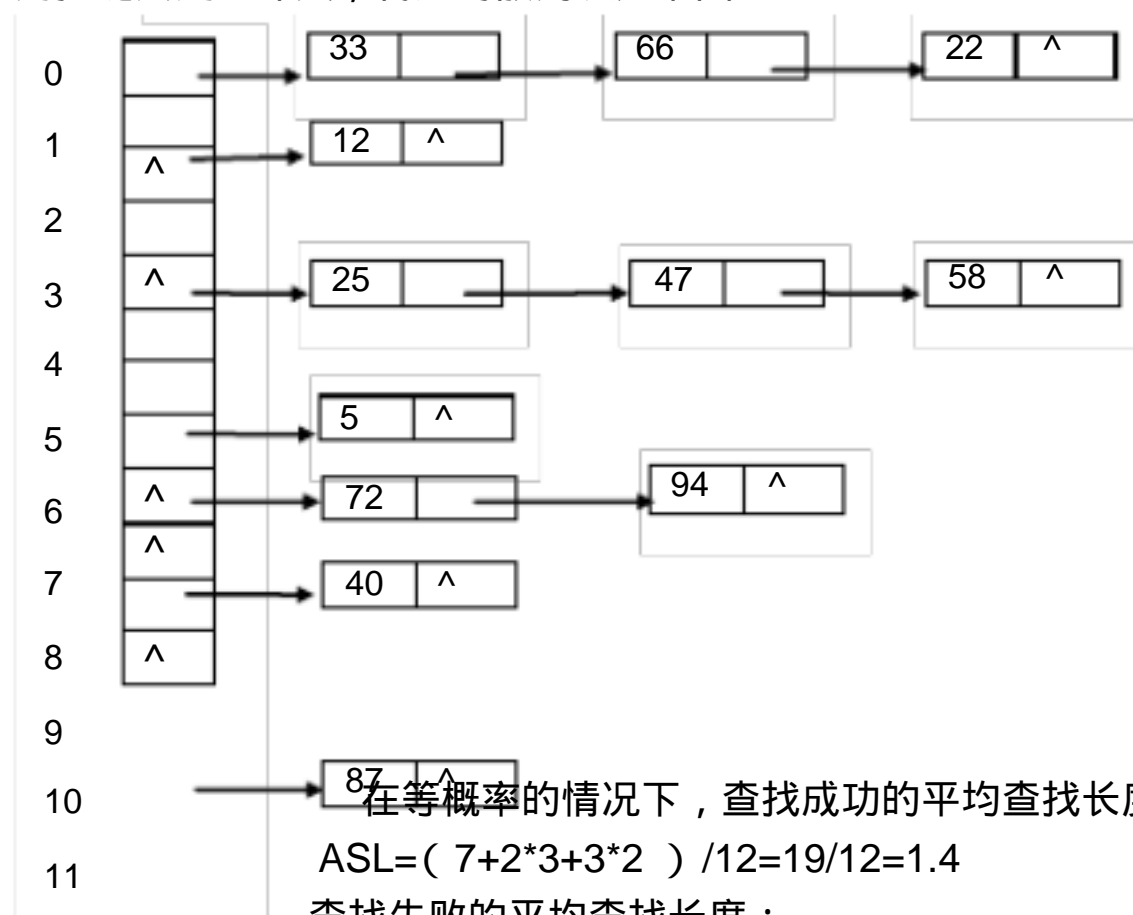
立相应的不同形态的二叉排序树见下图。



5. 键值序列：{25, 40, 33, 47, 12, 66, 72, 87, 94, 22, 5, 58}

散列地址：3 7 0 3 1 0 6 10 6 0 5 3

用拉链法处理冲突，构造的散列表如下图：



6. 已知散列表是：

0	1	2	3	4	5	6	7	8	9	10	11	12
						50	21		57		45	37

查找 key = 21 57 45 50

$h_0 = key \% 13 = 8 \quad 5 \quad 6 \quad 11$

$d = key \% 11 + 1 = 11 \quad 3 \quad 2 \quad 7$

$h_1 = (h_0 + d) \% 13 = 6 \quad 8 \quad 8 \quad 5$

$h_2 = (h_1 + d) \% 13 = \quad \quad 10$

查找成功比较次数 2 2 3 2

查找成功的平均查找长度  $ASL = (1 + 2 + 2 + 2 + 3) / 5 = 2$

#### 四、算法设计

1. 按中序遍历二叉排序树可得到一个按键值从小到大排列的有序表，利用这个特点来判别二叉树是否为二叉排序树，算法如下：

```
#define max 99
#define min 0
int judge(BTnode *bt)
{ BTnode *s[max], *p=bt;
  int top=0, preal=min;
  do {
    while(p)
      {s[top++] = p;
       p = p->lchild;
      }
  }
```

```

if(top>0)
{p=s[--top];
if(p->data<preval)
return 0;
preval=p->data;
p=p->rchild;
}
}while(p||top>0)
return 1;
}

```

## 第八章

### 一、单项选择题

(2) — (5) DDCCD (6) — (9) DCCC

### 二、填空题

(1) 8 (2) 15 (3) 4.3 (4) 中序 (5) 记录的键值 (6) 7  
 (7) 63 (8) 17 (9) 17 (10)  $O(n)$  (11)  $O(\log_2 n)$  (12) 越大  
 (13) 散列函数 (14) 处理冲突方法 (15) 装填因子 (16) 索引表  
 (17) 该块的起始地址 (18) 100 (19) 100 (20) 101 (21)  $\lceil \log_2(n+1) \rceil$   
 (22) 同义词 (23)  $n(n+1)/2$  (24) 63 (25)  $m$  (26)  $\lceil m/2 \rceil$  (27)  $m-1$   
 (28)  $\lceil m/2 \rceil + 1$  (29)  $m$  (30)  $\lceil m/2 \rceil$

### 三、应用题

1. 判定树见下图。

查找成功的平均查找长度

$$ASL = (1 + 2 \times 2 + 3 \times 4 + 4 \times 6) / 13 = 41/13$$

查找失败时与键值最多比较次数为 4。

2. 二叉排序树见右图。在等概率情况下，

查找成功的平均查找长度  $ASL = 29/9 = 3.2$

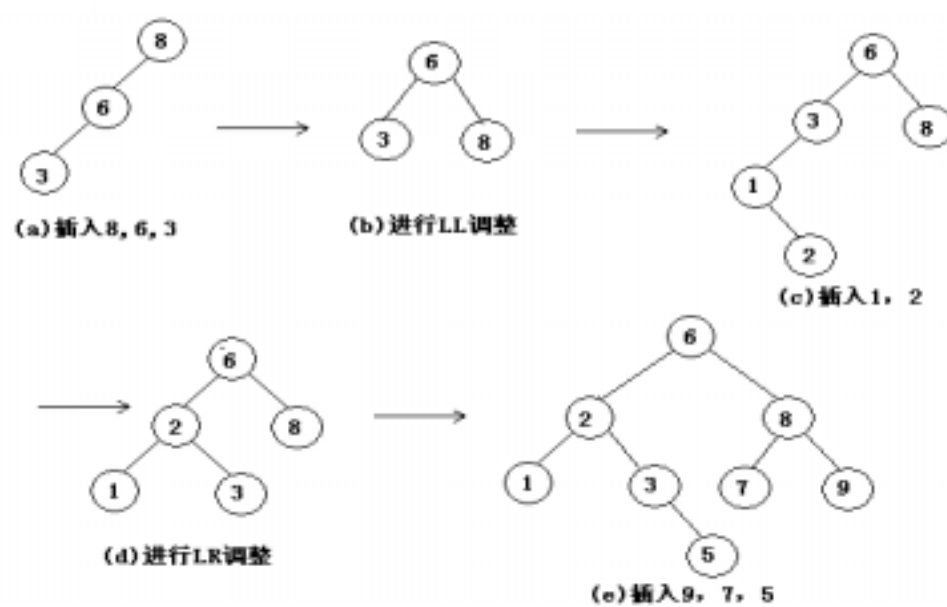
查找失败时，与键值最多比较次数是 5。

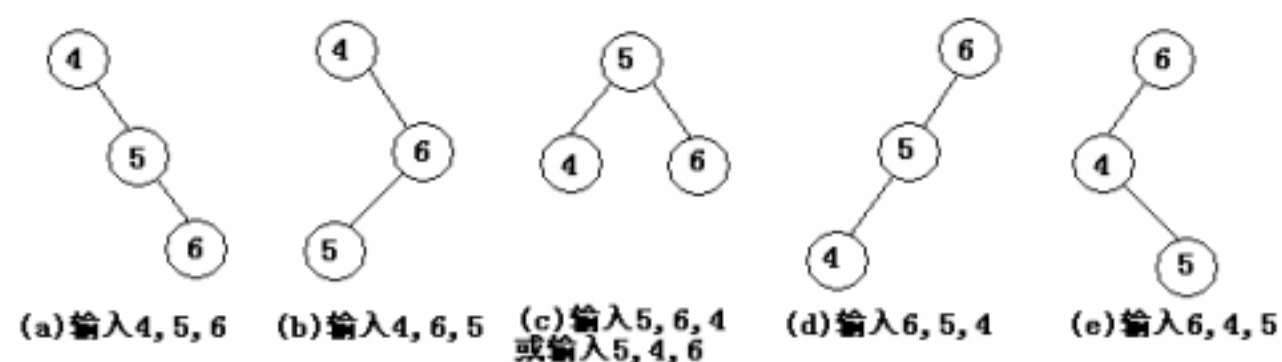
3. 构造一棵平衡二叉排序树

查找成功的平均查找长度  
 (  $1 + 2 \times 2 + 3 \times 4 + 4 \times 2$  )

值的最多比较次数是 4。  
 入顺序输入 4、5、6，建的二叉排序树见下图。

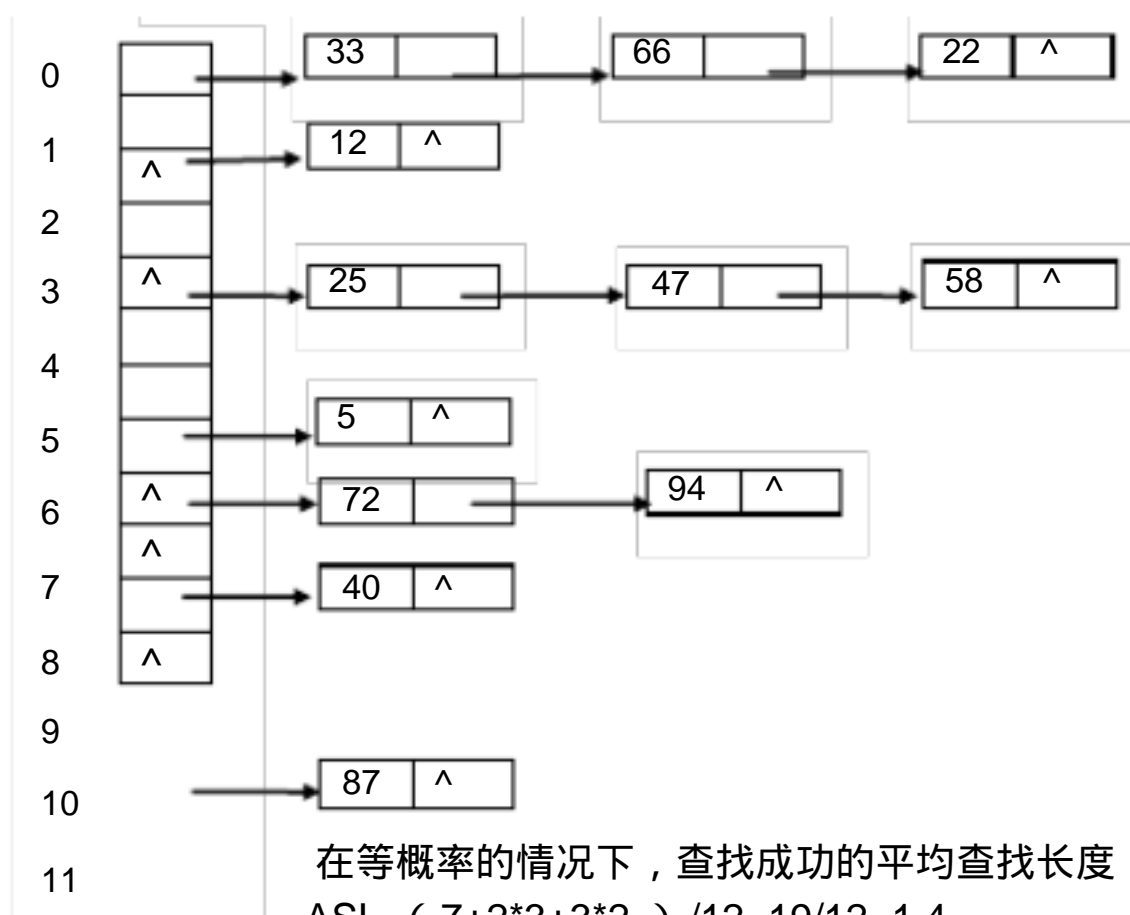
的过程见右图。  
 等概率情况下，查  
 $ASL =$   
 /9 2.8  
 查找失败时，与键  
 4. 按不同的输  
 立相应的不同形态





5. 键值序列：{25, 40, 33, 47, 12, 66, 72, 87, 94, 22, 5, 58}  
 散列地址：3 7 0 3 1 0 6 10 6 0 5 3

用拉链法处理冲突，构造的散列表如下图：



在等概率的情况下，查找成功的平均查找长度：  
 $ASL = (7 + 2 \times 3 + 3 \times 2) / 12 = 19 / 12 = 1.4$

查找失败的平均查找长度：

$$ASL = (4 + 2 \times 1 + 3 \times 2) / 12 = 1$$

6. 已知散列表是：

1	1	2	3	4	5	6	7	8	9	10	11	12
						50	21		57		45	37

查找 key = 21 57 45 50

$$h_0 = \text{key} \% 13 = 8 \quad 5 \quad 6 \quad 11$$

$$d = \text{key} \% 11 + 1 = 11 \quad 3 \quad 2 \quad 7$$

$$h_1 = (h_0 + d) \% 13 = 6 \quad 8 \quad 8 \quad 5$$

$$h_2 = (h_1 + d) \% 13 = \quad \quad 10$$

查找成功比较次数 2 2 3 2

$$\text{查找成功的平均查找长度 } ASL = (1 + 2 + 2 + 2 + 3) / 5 = 2$$

#### 四、算法设计

1. 按中序遍历二叉排序树可得到一个按键值从小到大排列的有序表，利用这个特点来判别二叉树是否为二叉排序树，算法如下：

```
#define max 99
#define min 0
int judge(BTnode *bt)
{ BTnode *s[max], *p=bt;
  int top=0, preVal=min;
  do {
    while(p)
    { s[top++] = p;
      p = p->lchild;
    }
    if(top>0)
```

```

    {p=s[--top];
    if(p->data<preval)
        return 0;
    preval=p->data;
    p=p->rchild;
    }
    }while(p||top>0)
    return 1;
}

```

## 第九章

### 一、单项选择题

(1)-(5) ABABC (6)-(10) ADDAC (11)-(15) BBADB

(16)-(20) DBCAB (21)-(25) ADCBA

### 二、填空题

(1) 插入排序 (2) 交换排序 (3) 选择排序 (4) 归并排序 (5) 基数排序

(6)  $O(n \log_2 n)$  (7)  $O(n^2)$  (8)  $O(n \log_2 n)$  (9) 快速 (10) 堆 (11) 归并

(12) 快速 (13) 归并 (14) 堆 (15) 3 (16) 选择 (17)  $O(n)$  (18)  $O(n \log_2 n)$  (19)  $O(n \log_2 n)$

(20)  $n^2$  (22) 5 (23) 4 (24) 8

### 三、应用题

1. 解 不稳定的排序方法有快速排序、直接选择排序、堆排序，分别举例说明如下：

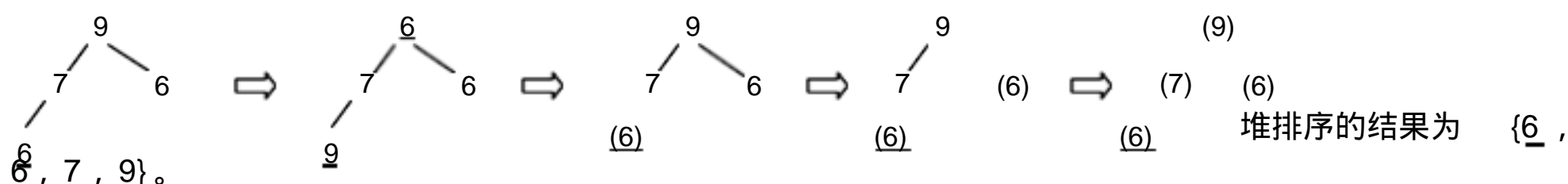
(1) 快速排序

给定键值序列 {9, 6, 6} 进行快速排序。

排序结果为 {6, 6, 9}。

(2) 堆排序

给定键值序列为 {9, 7, 6, 6} 进行堆排序（小顶堆）的过程见下图。



(3) 直接选择排序，例：给定排序码值序列为 {6, 6, 4}，直接选择排序后结果为 {4, 6, 6}。

(4) 希尔排序，例给定排序码值序列为 {4, 2, 2, 5}，设  $d_1=2, d_2=1$  时，排序结果为 {2, 2, 4, 5}。

2. 解：冒泡排序：

第一趟：17, 18, 40, 7, 32, 60, 65, 73, 85

第二趟：17, 18, 7, 32, 40, 60, 65, 73

第三趟：17, 7, 18, 32, 40, 60, 65

第四趟：7, 17, 18, 32, 40, 60

第五趟：7, 17, 18, 32, 40（没有发生交换，结束排序）

3. 解：希尔排序：

原序列：10 18 14 13 16 12 11 9 15 8  $d_1=5$

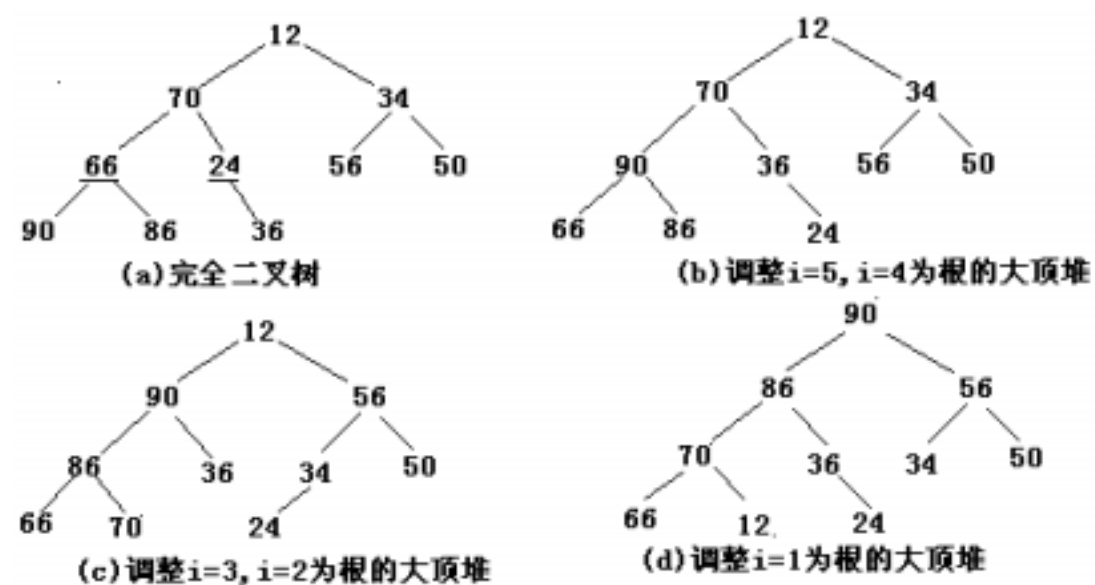
一趟排序结果：10 11 9 13 8 12 18 14 15 16  $d_2=2$

二趟排序结果：{8 11 9 12 10 13 15 14 18 16}  $d_3=1$

结果

三趟排序结果 {8, 9, 10, 11, 12, 13, 14, 15, 16, 18}

4. 初始序列 (1) {90, 86, 48, 73, 35, 40, 42, 58, 66, 20} 对应的完全二叉树见下图：



它是一个大顶堆。

初始序列 ( 2 ) { 12 , 70 , 34 , 66 , 24 , 56 , 90 , 86 , 36 } 对应的完全二叉树不是堆 , 将其调整成堆的过程见左图 :

#### 四、算法设计题

1 . 单链表结点类型定义为 :

```
typedef struct node
{int key;
 struct node *next;
}Lnode;
```

直接选择排序是每次从  $n-i+1$  个结点中选择码值最小者 , 与第  $i$  个结点的码值进行交换 , 设  $p$  指向第  $i$  个结点 ,  $min$  指向无序表中码值最小的结点 , 算法描述如下 :

```
Void selesort1(Lnode *h)
{Lnode *p,*q,*min;
 int x;
 p=h->next;
 while(p)
{min=p;
 q=p->next;
 while(q)
 {if(q->key<min->key)
 m=q;
 q=q->next;
 }
 if(min!=p)
 {x=p->key;
 p->key=min->key;
 min->key=x;
 }
 p=p->next;
 }
```



```

        p=p->next;
    }
}

2. 快速排序非递归算法描述如下（其中 s 为顺序栈）：
void Quicksort1(RecType t[],int n)
{int top,low=0,high=n,i,j,*s;
  RecType x;
  S=(int *)malloc(sizeof(int)*2*n);
  Top=0;
  Do{
while(low<high)
{i=low;j=high;x=r[i];
  do{
    while((r[i].key>=x.key)&&(i<j))
      j--;
    if(i<j){
      r[j]=r[i];j--;}
    }while(i<j);
    r[i]=x;
    if((i+1)<high)
      {s[++top]=i+1;
       s[++top]=high;}
    high=i-1;
  if(top>0)
    {low=s[top--];
     high=s[top--];}
  }while(top>0||low<high);
}
```

目录

目录 .....	1
说明 : .....	1
第一章 绪论 .....	2
第二章 线性表 .....	5
第三章 栈与队列 .....	19
第四章 串 .....	30
第五章 数组和广义表 .....	42
第六章 树和二叉树 .....	56
第七章 图 .....	75
第八章 动态存储管理 .....	96
第九章 查找 .....	100
第十章 内部排序 .....	111

说明：

1. 本文是对严蔚敏《数据结构 (c 语言版 )习题集》一书中所有算法设计题目的解决方案 ,主要作者为一

具.以

下网友 :biwier,szm99,siice, 龙抬头 ,iamkent,zames,birdthinking,lovebuaa 等为答案的修订和完善工作提出了宝

贵意见 ,在此表示感谢 ;

2. 本解答中的所有算法均采用类 c 语言描述 ,设计原则为面向交流、面向阅读 ,作者不保证程序能够上机正

常运行 (这种保证实际上也没有任何意义 );

3. 本解答原则上只给出源代码以及必要的注释 ,对于一些难度较高或思路特殊的题目将给出简要的分析说

明,对于作者无法解决的题目将给出必要的讨论 .目前尚未解决的题目有 : 5.20, 10.40;

4. 请读者在自己已经解决了某个题目或进行了充分的思考之后 ,再参考本解答 ,以保证复习效果 ;

5. 由于作者水平所限 ,本解答中一定存在不少这样或者那样的错误和不足 ,希望读者们在阅读中多动脑、勤

思考,争取发现和纠正这些错误 ,写出更好的算法来 .请将你发现的错误或其它值得改进之处向作者报告 :

## 第一章 绪论

1.16

void print\_descending(int x,int y,int z)// 按从大到小顺序输出三个数

```
{
    scanf("%d,%d,%d",&x,&y,&z);
    if(x<y) x<->y; //<-> 为表示交换的双目运算符 ,以下同
    if(y<z) y<->z;
    if(x<y) x<->y; // 冒泡排序
    printf("%d %d %d",x,y,z);
} //print_descending
```

1.17

Status fib(int k,int m,int &f)// 求 k 阶斐波那契序列的第 m 项的值 f

```
{
    int tempd;
    if(k<2||m<0) return ERROR;
    if(m<k-1) f=0;
    else if (m==k-1 || m==k) f=1;
    else
    {
        for(i=0;i<=k-2;i++) temp[i]=0;
        temp[k-1]=1;temp[k]=1; // 初始化
        sum=1;
        j=0;
        for(i=k+1;i<=m;i++,j++) // 求出序列第 k 至第 m 个元素的值
            temp[i]=2*sum-temp[j];
        f=temp[m];
    }
}
```

```

    }
    return OK;
} //fib
分析：k 阶斐波那契序列的第 m 项的值  $f[m]=f[m-1]+f[m-2]+\dots+f[m-k]$ 
 $=f[m-1]+f[m-2]+\dots+f[m-k]+f[m-k-1]-f[m-k-1]$ 
 $=2*f[m-1]-f[m-k-1]$ 

```

所以上述算法的时间复杂度仅为  $O(m)$ 。如果采用递归设计，将达到  $O(k^m)$ 。即使采用暂存中间结果的方法，也将达到  $O(m^2)$ 。

1.18

严蔚敏《数据结构习题集》解答

```

typedef struct{
    char *sport;
    enum{male,female} gender;
    char schoolname; //校名为 'A','B','C','D' 或 'E'
    char *result;
    int score;
} resulttype;

```

```

typedef struct{
    int malescore;
    int femalescore;
    int totalscore;
} scoretype;

```

void summary(resulttype result[ ])// 求各校的男女总分和团体总分，假设结果已经储存在 result[ ] 数组中

```

{
    scoretype score[MAXSIZE];
    i=0;
    while(result[i].sport!=NULL)
    {
        switch(result[i].schoolname)
        {
            case 'A':
                score[ 0 ].totalscore+=result[i].score;
                if(result[i].gender==0) score[ 0 ].malescore+=result[i].score;
                else score[ 0 ].femalescore+=result[i].score;
                break;
            case 'B':
                score[ 0 ].totalscore+=result[i].score;
                if(result[i].gender==0) score[ 0 ].malescore+=result[i].score;
                else score[ 0 ].femalescore+=result[i].score;

```

```

        break;
    }
    i++;
}
for(i=0;i<5;i++)
{
    printf("School %d:\n",i);
    printf("Total score of male:%d\n",score[i].malescore);
    printf("Total score of female:%d\n",score[i].femalescore);
    printf("Total score of all:%d\n\n",score[i].totalscore);
}
} //summary

```

### 1.19

```

Status algo119(int a[ARRSIZE])// 求  $i! \cdot 2^i$  序列的值且不超过 maxint
{
    last=1;
    for(i=1;i<=ARRSIZE;i++)
    {
        a[i-1]=last*2*i;
        if((a[i-1]/last)!=(2*i)) return OVERFLOW;
        last=a[i-1];
        return OK;
    }
} //algo119

```

分析:当某一项的结果超过了 maxint 时,它除以前面一项的商会发生异常。

### 1.20

```

void polyvalue()
{
    float temp;
    float *p=a;
    printf("Input number of terms:");
    scanf("%d",&n);
    printf("Input value of x:");
    scanf("%f",&x);
    printf("Input the %d coefficients from a0 to a%d:\n",n+1,n);
    p=a;xp=1;sum=0; //xp 用于存放 x 的 i 次方
    for(i=0;i<=n;i++)
    {
        scanf("%f",&temp);
        sum+=xp*(temp);
        xp*=x;
    }
}

```

```

    }
    printf("Value is:%f",sum);
} //polyvalue

```

## 第二章 线性表

### 2.10

```

Status DeleteK(SqList &a,int i,int k) // 删除线性表 a 中第 i 个元素起的 k 个元素
{
    if(i<1||k<0||i+k-1>a.length) return INFEASIBLE;
    for(count=1;i+count-1<=a.length-k;count++) // 注意循环结束的条件
        a.elem[i+count-1]=a.elem[i+count+k-1];
    a.length-=k;
    return OK;
} //DeleteK

```

### 2.11

```

Status Insert_SqList(SqList &va,int x) // 把 x 插入递增有序表 va 中
{
    if(va.length+1>va.listsize) return ERROR;
    va.length++;
    for(i=va.length-1;va.elem[i]>x&& i>=0;i--)
        va.elem[i+1]=va.elem[i];
    va.elem[i+1]=x;
    return OK;
} //Insert_SqList

```

### 2.12

```

int ListComp(SqList A,SqList B) // 比较字符表 A 和 B,并用返回值表示结果 ,值为 1,表示 A>B;
// 值为 -1,表示 A<B; 值为 0,表示 A=B
{
    for(i=1;i<=A.length&&i<=B.length;i++)
        if(A.elem[i]!=B.elem[i])
            return A.elem[i]>B.elem[i]?1:-1;
    if(A.length==B.length) return 0;
    return A.length>B.length?1:-1; //当两个字符表可以互相比较的部分完全相同时 ,哪个较长 ,
// 哪个就较大
} //ListComp

```

### 2.13

```

LNode* Locate(LinkList L,int x) // 链表上的元素查找 ,返回指针

```

```

{
    for(p=l->next;p&& p->data!=x;p=p->next);

    return p;
} //Locate

```

2.14

```

int Length(LinkList L) // 求链表的长度
{
    for(k=0,p=L;p->next;p=p->next,k++);
    return k;
} //Length

```

2.15

```

void ListConcat(LinkList ha,LinkList hb,LinkList &hc) // 把链表 hb 接在 ha 后面形成链表 hc
{
    hc=ha;p=ha;
    while(p->next) p=p->next;
    p->next=hb;
} //ListConcat

```

2.16

见书后答案 .

2.17

```

Status Insert(LinkList &L,int i,int b) // 在无头结点链表 L 的第 i 个元素之前插入元素 b
{
    p=L;q=(LinkList*)malloc(sizeof(LNode));
    q->data=b;
    if(i==1)
    {
        q->next=p;L=q; // 插入在链表头部
    }
    else
    {
        while(--i>1) p=p->next;
        q->next=p->next;p->next=q; // 插入在第 i 个元素的位置
    }
} //Insert

```



2.18

Status Delete(LinkList &L,int i)// 在无头结点链表 L 中删除第 i 个元素

```
{
    严蔚敏《数据结构习题集》解答

    if(i==1) L=L->next; // 删除第一个元素
    else
    {
        p=L;
        while(--i>1) p=p->next;
        p->next=p->next->next; // 删除第 i 个元素
    }
} //Delete
```

2.19

第 7 页 共 124 页

Status Delete\_Between(Linklist &L,int mink,int maxk)// 删除元素递增排列的链表 L 中  
值大于

mink 且小于 maxk 的所有元素

```
{
    p=L;
    while(p->next->data<=mink) p=p->next; //p 是最后一个不大于 mink 的元素
    if(p->next) //如果还有比 mink 更大的元素
    {
        q=p->next;
        while(q->data<maxk) q=q->next; //q 是第一个不小于 maxk 的元素
        p->next=q;
    }
} //Delete_Between
```

2.20

Status Delete\_Equal(Linklist &L)// 删除元素递增排列的链表 L 中所有值相同的元素

```
{
    p=L->next;q=p->next; //p,q 指向相邻两元素
    while(p->next)
    {
        if(p->data!=q->data)
        {
            p=p->next;q=p->next; // 当相邻两元素不相等时 ,p,q 都向后推一步
        }
        else
        {
            while(q->data==p->data)
```

```

        {
            free(q);
            q=q->next;
        }
        p->next=q;p=q;q=p->next; // 当相邻元素相等时删除多余元素
    }//else

    }//while
} //Delete_Equal

```

## 2.21

```

void reverse(SqList &A)// 顺序表的就地逆置
{
    for(i=1,j=A.length;i<j;i++,j--)
        A.elem[i]<->A.elem[j];
} //reverse

```

## 2.22

第 8 页 共 124 页

```

void LinkList_reverse(Linklist &L)// 链表的就地逆置 ;为简化算法 ,假设表长大于 2
{
    p=L->next;q=p->next;s=q->next;p->next=NULL;
    while(s->next)
    {
        q->next=p;p=q;
        q=s;s=s->next; //把 L 的元素逐个插入新表表头
    }
    q->next=p;s->next=q;L->next=s;
} //LinkList_reverse

```

分析 :本算法的思想是 ,逐个地把 L 的当前元素 q 插入新的链表头部 ,p 为新表表头 .

## 2.23

```

void merge1(LinkList &A,LinkList &B,LinkList &C)// 把链表 A 和 B 合并为 C,A 和
B 的元素间
隔排列 ,且使用原存储空间
{
    p=A->next;q=B->next;C=A;
    while(p&&q)
    {
        s=p->next;p->next=q; // 将 B 的元素插入
        if(s)
        {
            t=q->next;q->next=s; // 如 A 非空 ,将 A 的元素插入

```

```

    }
    p=s;q=t;
  }//while
} //merge1

```

## 2.24

第 9 页 共 124 页

```

void reverse_merge(LinkList &A,LinkList &B,LinkList &C)// 把元素递增排列的链表 A
和 B 合
并为 C,且 C 中元素递减排列 ,使用原空间
{
    pa=A->next;pb=B->next;pre=NULL; //pa 和 pb 分别指向 A,B 的当前元素
    while(pa||pb)
    {
        if(pa->data<pb->data||!pb)
        {
            pc=pa;q=pa->next;pa->next=pre;pa=q; //将 A 的元素插入新表
        }
        else
        {
            pc=pb;q=pb->next;pb->next=pre;pb=q; // 将 B 的元素插入新表
        }
        pre=pc;
    }
    C=A;A->next=pc; // 构造新表头
} //reverse_merge

```

分析 :本算法的思想是 ,按从小到大的顺序依次把 A 和 B 的元素插入新表的头部 pc 处,最后处理 A 或 B 的剩余元素 .

## 2.25

```

void SqList_Intersect(SqList A,SqList B,SqList &C)// 求元素递增排列的线性表 A 和 B
的元素
的交集并存入 C 中
{
    i=1;j=1;k=0;
    while(A.elem[i]&&B.elem[j])
    {
        if(A.elem[i]<B.elem[j]) i++;
        if(A.elem[i]>B.elem[j]) j++;
        if(A.elem[i]==B.elem[j])
        {
            C.elem[++k]=A.elem[i]; // 当发现了一个在 A,B 中都存在的元素 ,

```

```

        i++;j++; // 就添加到 C 中
    }
} //while
} //SqList_Intersect

```

2.26

```

void LinkList_Intersect(LinkList A,LinkList B,LinkList &C)// 在链表结构上重做上题
{
    p=A->next;q=B->next;
    pc=(LNode*)malloc(sizeof(LNode));

    C=pc;
    while(p&&q)
    {
        if(p->data<q->data) p=p->next;
        else if(p->data>q->data) q=q->next;
        else
        {
            s=(LNode*)malloc(sizeof(LNode));
            s->data=p->data;
            pc->next=s;pc=s;
            p=p->next;q=q->next;
        }
    }
} //while
} //LinkList_Intersect

```

2.27

第 10 页 共 124 页

```

void SqList_Intersect_True(SqList &A,SqList B)// 求元素递增排列的线性表 A 和 B 的
元素的交
集并存回 A 中
{
    i=1;j=1;k=0;
    while(A.elem[i]&&B.elem[j])
    {
        if(A.elem[i]<B.elem[j]) i++;
        else if(A.elem[i]>B.elem[j]) j++;
        else if(A.elem[i]!=A.elem[k])
        {
            A.elem[++k]=A.elem[i]; // 当发现了一个在 A,B 中都存在的元素
            i++;j++; // 且 C 中没有 ,就添加到 C 中
        }
        else {i++;j++;}
    }
} //while

```

```

    while(A.elem[k]) A.elem[k++]=0;
} //SqList_Intersect_True

```

2.28

void LinkList\_Intersect\_True(LinkList &A, LinkList B) // 在链表结构上重做上题

```

{
    p=A->next; q=B->next; pc=A;
    while(p && q)
    {
        if(p->data < q->data) p=p->next;
        else if(p->data > q->data) q=q->next;
        else if(p->data != pc->data)

```

严蔚敏《数据结构习题集》解答

```

        {
            pc=pc->next;
            pc->data=p->data;
            p=p->next; q=q->next;
        }
    } //while
} //LinkList_Intersect_True

```

2.29

void SqList\_Intersect\_Delete(SqList &A, SqList B, SqList C)

```

{
    i=0; j=0; k=0; m=0; //i 指示 A 中元素原来的位置, m 为移动后的位置
    while(i < A.length && j < B.length && k < C.length)
    {
        if(B.elem[j] < C.elem[k]) j++;
        else if(B.elem[j] > C.elem[k]) k++;
        else
        {
            same=B.elem[j]; //找到了相同元素 same
            while(B.elem[j]==same) j++;
            while(C.elem[k]==same) k++; //j, k 后移到新的元素
            while(i < A.length && A.elem[i] < same)
                A.elem[m++]=A.elem[i++]; //需保留的元素移动到新位置
            while(i < A.length && A.elem[i]==same) i++; //跳过相同的元素
        }
    } //while
    while(i < A.length)
        A.elem[m++]=A.elem[i++]; //A 的剩余元素重新存储。
    A.length=m;
} // SqList_Intersect_Delete

```

分析:先从 B 和 C 中找出共有元素,记为 same,再在 A 中从当前位置开始,凡小于 same 的元素均保留(存到新的位置),等于 same 的就跳过,到大于 same 时就再找下一个 same.

2.30

void LinkList\_Intersect\_Delete(LinkList &A,LinkList B,LinkList C)// 在链表结构上重做上题

```
{
    p=B->next;q=C->next;r=A->next;
    while(p&&q&&r)
    {
        if(p->data<q->data) p=p->next;
        else if(p->data>q->data) q=q->next;
        else
        {
```

严蔚敏《数据结构习题集》解答

```
            u=p->data; //确定待删除元素 u
            while(r->next->data<u) r=r->next; // 确定最后一个小于 u 的元素指针 r
            if(r->next->data==u)
            {
                s=r->next;
                while(s->data==u)
                {
                    t=s;s=s->next;free(t); // 确定第一个大于 u 的元素指针 s
                }
                r->next=s; //删除 r 和 s 之间的元素
            }
            while(p->data==u) p=p->next;
            while(q->data==u) q=q->next;
        }
    }
} //LinkList_Intersect_Delete
```

2.31

Status Delete\_Pre(CiLNode \*s)// 删除单循环链表中结点 s 的直接前驱

```
{
    p=s;
    while(p->next->next!=s) p=p->next; // 找到 s 的前驱的前驱 p
    p->next=s;
    return OK;
} //Delete_Pre
```

2.32

```

Status DuLNode_Pre(DuLinkList &L)// 完成双向循环链表结点的 pre 域
{
    for(p=L;!p->next->pre;p=p->next) p->next->pre=p;
    return OK;
} //DuLNode_Pre

```

### 2.33

Status LinkList\_Divide(LinkList &L,CiList &A,CiList &B,CiList &C)// 把单链表 L 的元素按类

型分为三个循环链表 .CiList 为带头结点的单循环链表类型 .

```

{
    s=L->next;
    A=(CiList*)malloc(sizeof(CiLNode));p=A;
    B=(CiList*)malloc(sizeof(CiLNode));q=B;
    C=(CiList*)malloc(sizeof(CiLNode));r=C; // 建立头结点
    while(s)

```

严蔚敏《数据结构习题集》解答

```

{
    if(isalphabet(s->data))
    {
        p->next=s;p=s;
    }
    else if(isdigit(s->data))
    {
        q->next=s;q=s;
    }
    else
    {
        r->next=s;r=s;
    }
} //while
p->next=A;q->next=B;r->next=C; // 完成循环链表
} //LinkList_Divide

```

### 2.34

void Print\_XorLinkedList(XorLinkedList L)// 从左向右输出异或链表的元素值

```

{
    p=L.left;pre=NULL;
    while(p)
    {
        printf("%d",p->data);
        q=XorP(p->LRPtr,pre);
        pre=p;p=q; // 任何一个结点的 LRPtr 域值与其左结点指针进行异或运算即得到其

```



右结点指  
针

```
}  
} //Print_XorLinkedList
```

2.35

Status Insert\_XorLinkedList(XorLinkedList &L,int x,int i)// 在异或链表 L的第 i 个元素  
前插入元  
素 x

```
{  
    p=L.left;pre=NULL;  
    r=(XorNode*)malloc(sizeof(XorNode));  
    r->data=x;  
    if(i==1) // 当插入点在最左边的情况  
    {  
        p->LRPtr=XorP(p.LRPtr,r);  
        r->LRPtr=p;  
        L.left=r;  
        return OK;  
    }  
    j=1;q=p->LRPtr; // 当插入点在中间的情况  
    while(++j<i&&q)  
    {  
        q=XorP(p->LRPtr,pre);  
        pre=p;p=q;  
    } //while // 在 p,q 两结点之间插入  
    if(!q) return INFEASIBLE; //i 不可以超过表长  
    p->LRPtr=XorP(XorP(p->LRPtr,q),r);  
    q->LRPtr=XorP(XorP(q->LRPtr,p),r);  
    r->LRPtr=XorP(p,q); // 修改指针  
    return OK;  
} //Insert_XorLinkedList
```

2.36

第 14 页 共 124 页

Status Delete\_XorLinkedList(XorLinkedList &L,int i)// 删除异或链表 L 的第 i 个元素

```
{  
    p=L.left;pre=NULL;  
    if(i==1) // 删除最左结点的情况  
    {  
        q=p->LRPtr;  
        q->LRPtr=XorP(q->LRPtr,p);
```

```

        L.left=q;free(p);
        return OK;
    }
    j=1;q=p->LRPtr;
    while(++j<i&&q)
    {
        q=XorP(p->LRPtr,pre);
        pre=p;p=q;
    }//while // 找到待删结点 q
    if(!q) return INFEASIBLE; //i 不可以超过表长
    if(L.right==q) //q 为最右结点的情况
    {
        p->LRPtr=XorP(p->LRPtr,q);
        L.right=p;free(q);
        return OK;
    }
    r=XorP(q->LRPtr,p); //q 为中间结点的情况 ,此时 p,r 分别为其左右结点
    p->LRPtr=XorP(XorP(p->LRPtr,q),r);
    r->LRPtr=XorP(XorP(r->LRPtr,q),p); // 修改指针
    free(q);
    return OK;
} //Delete_XorLinkedList

```

## 2.37

第 15 页 共 124 页

```

void OEReform(DuLinkedList &L)// 按 1,3,5,...4,2 的顺序重排双向循环链表 L 中的所有结点
{
    p=L.next;
    while(p->next!=L&&p->next->next!=L)
    {
        p->next=p->next->next;
        p=p->next;
    } // 此时 p 指向最后一个奇数结点
    if(p->next==L) p->next=L->pre->pre;
    else p->next=L->pre;
    p=p->next; // 此时 p 指向最后一个偶数结点
    while(p->pre->pre!=L)
    {
        p->next=p->pre->pre;
        p=p->next;
    }
    p->next=L; // 按题目要求调整了 next 链的结构 ,此时 pre 链仍为原状
    for(p=L;p->next!=L;p=p->next) p->next->pre=p;
}

```

```
L->pre=p; // 调整 pre 链的结构 ,同 2.32 方法
} //OEReform
分析 :next 链和 pre 链的调整只能分开进行 .如同时进行调整的话 ,必须使用堆栈保存
偶数结点
的指针 ,否则将会破坏链表结构 ,造成结点丢失 .
```

## 2.38

```
DuLNode * Locate_DuList(DuLinkedList &L,int x)// 带 freq 域的双向循环链表上的查
找
{
    p=L.next;
    while(p.data!=x&&p!=L) p=p->next;
    if(p==L) return NULL; // 没找到
    p->freq++;q=p->pre;
    while(q->freq<=p->freq&&p!=L) q=q->pre; // 查找插入位置
    if(q!=p->pre)
    {
        p->pre->next=p->next;p->next->pre=p->pre;
        q->next->pre=p;p->next=q->next;
        q->next=p;p->pre=q; // 调整位置
    }
    return p;
} //Locate_DuList
```

## 2.39

第 16 页 共 124 页

```
float GetValue_SqPoly(SqPoly P,int x0)// 求升幂顺序存储的稀疏多项式的值
{
    PolyTerm *q;
    xp=1;q=P.data;
    sum=0;ex=0;
    while(q->coef)
    {
        while(ex<q->exp) xp*=x0;
        sum+=q->coef*xp;
        q++;
    }
    return sum;
} //GetValue_SqPoly
```

## 2.40

```
void Subtract_SqPoly(SqPoly P1,SqPoly P2,SqPoly &P3)// 求稀疏多项式 P1 减 P2 的
差式 P3
```

```

{
    PolyTerm *p,*q,*r;
    Create_SqPoly(P3); //建立空多项式 P3
    p=P1.data;q=P2.data;r=P3.data;
    while(p->coef&&q->coef)
    {
        if(p->exp<q->exp)
        {
            r->coef=p->coef;
            r->exp=p->exp;
            p++;r++;
        }
        else if(p->exp>q->exp)
        {
            r->coef=-q->coef;
            r->exp=q->exp;
            q++;r++;
        }
        else
        {
            if((p->coef-q->coef)!=0) // 只有同次项相减不为零时才需要存入 P3 中
            {
                r->coef=p->coef-q->coef;
                r->exp=p->exp;r++;
            }//if
            p++;q++;
        }//else
    }//while

    while(p->coef) // 处理 P1 或 P2 的剩余项
    {
        r->coef=p->coef;
        r->exp=p->exp;
        p++;r++;
    }
    while(q->coef)
    {
        r->coef=-q->coef;
        r->exp=q->exp;
        q++;r++;
    }
} //Subtract_SqPoly

```

2.41

void QiuDao\_LinkedPoly(LinkPoly &L)// 对有头结点循环链表结构存储的稀疏多项

式 L 求

导

```
{
    p=L->next;
    if(!p->data.exp)
    {
        L->next=p->next;p=p->next; // 跳过常数项
    }
    while(p!=L)
    {
        p->data.coef*=p->data.exp--;// 对每一项求导
        p=p->next;
    }
} //QiuDao_LinkedPoly
```

2.42

void Divide\_LinkedPoly(LinkedList &L,&A,&B)// 把循环链表存储的稀疏多项式 L 拆成只含奇次项的 A 和只含偶次项的 B

```
{
    p=L->next;
    A=(PolyNode*)malloc(sizeof(PolyNode));
    B=(PolyNode*)malloc(sizeof(PolyNode));
    pa=A;pb=B;
    while(p!=L)
    {
        if(p->data.exp!=2*(p->data.exp/2))
        {
            pa->next=p;pa=p;
        }
        else
        {
            pb->next=p;pb=p;
        }
        p=p->next;
    } //while
    pa->next=A;pb->next=B;
} //Divide_LinkedPoly
```

### 第三章 栈与队列

3.15

```

typedef struct{
    Elemtype *base[2];
    Elemtype *top[2];
}BDStacktype; // 双向栈类型

Status Init_Stack(BDStacktype &twS,int m)// 初始化一个大小为 m 的双向栈 twS
{
    twS.base[0]=(Elemtype*)malloc(sizeof(Elemtype));
    twS.base[1]=twS.base[0]+m;
    twS.top[0]=twS.base[0];
    twS.top[1]=twS.base[1];
    return OK;
}

Status push(BDStacktype &twS,int i,Elemtype x)//x 入栈 ,i=0 表示低端栈 ,i=1 表示高端栈
{
    if(twS.top[0]>twS.top[1]) return OVERFLOW; // 注意此时的栈满条件
    if(i==0) *twS.top[0]++=x;
    else if(i==1) *twS.top[1]--=x;
    else return ERROR;
    return OK;
}

Status pop(BDStacktype &twS,int i,Elemtype &x)//x 出栈 ,i=0 表示低端栈 ,i=1 表示高端栈
{
    if(i==0)
    {
        if(twS.top[0]==twS.base[0]) return OVERFLOW;
        x=*--twS.top[0];
    }
    else if(i==1)
    {
        if(twS.top[1]==twS.base[1]) return OVERFLOW;
        x=*++twS.top[1];
    }
    else return ERROR;
    return OK;
}

```

### 3.16

```

void Train_arrange(char *train)// 这里用字符串 train 表示火车 , 'H'表示硬席 , 'S'表示软席
{

```

```

p=train;q=train;
InitStack(s);
while(*p)
{
    if(*p=='H') push(s,*p); // 把'H'存入栈中
    else *(q++)=*p; // 把'S'调到前部
    p++;
}
while(!StackEmpty(s))
{
    pop(s,c);
    *(q++)=c; // 把'H'接在后部
}
} //Train_arrange

```

### 3.17

int IsReverse()//判断输入的字符串中 '&'前和 '&'后部分是否为逆串 ,是则返回 1,否则返回 0

```

{
    InitStack(s);
    while((e=getchar())!='&')
    {
        if(e=='@') return 0; //不允许在 ?&?之前出现 ?@?
        push(s,e);
    }
    while( (e=getchar())!='@')
    {
        if(StackEmpty(s)) return 0;
        pop(s,c);
        if(e!=c) return 0;
    }
    if(!StackEmpty(s)) return 0;
    return 1;
} //IsReverse

```

### 3.18

Status Bracket\_Test(char \*str)//判别表达式中小括号是否匹配

```

{
    count=0;
    for(p=str;*p;p++)
    {
        if(*p=='(') count++;

        else if(*p==')') count--;
    }
}

```



```

        if (count<0) return ERROR;
    }
    if(count) return ERROR; // 注意括号不匹配的两种情况
    return OK;
} //Bracket_Test

```

3.19

```

Status AllBrackets_Test(char *str) // 判别表达式中三种括号是否匹配
{
    InitStack(s);
    for(p=str;*p;p++)
    {
        if(*p=='('||*p=='['||*p=='{') push(s,*p);
        else if(*p==')'||*p==']'||*p=='}')
        {
            if(StackEmpty(s)) return ERROR;
            pop(s,c);
            if(*p==')' && c!='(') return ERROR;
            if(*p==']' && c!='[') return ERROR;
            if(*p=='}' && c!='{') return ERROR; // 必须与当前栈顶括号匹配
        }
    }
} //for
if(!StackEmpty(s)) return ERROR;
return OK;
} //AllBrackets_Test

```

3.20

```

typedef struct {
    .
    int x;
    int y;
} coordinate;

```

```

void Repaint_Color(int g[m][n],int i,int j,int color) // 把点 (i,j) 相邻区域的颜色置换为 color
{
    old=g[i][j];
    InitQueue(Q);
    EnQueue(Q,{i,j});
    while(!QueueEmpty(Q))
    {
        DeQueue(Q,a);
        x=a.x;y=a.y;

```

严蔚敏《数据结构习题集》解答

```

    if(x>1)
        if(g[x-1][y]==old)
        {
            g[x-1][y]=color;
            EnQueue(Q,{x-1,y}); // 修改左邻点的颜色
        }
    if(y>1)
        if(g[x][y-1]==old)
        {
            g[x][y-1]=color;
            EnQueue(Q,{x,y-1}); // 修改上邻点的颜色
        }
    if(x<m)
        if(g[x+1][y]==old)
        {
            g[x+1][y]=color;
            EnQueue(Q,{x+1,y}); // 修改右邻点的颜色
        }
    if(y<n)
        if(g[x][y+1]==old)
        {
            g[x][y+1]=color;
            EnQueue(Q,{x,y+1}); // 修改下邻点的颜色
        }
    }//while
} //Repaint_Color

```

第 22 页 共 124 页

分析 :本算法采用了类似于图的广度优先遍历的思想 ,用两个队列保存相邻同色点的横坐标和纵坐标 .递归形式的算法该怎么写呢 ?

### 3.21

```

void NiBoLan(char *str,char *new)// 把中缀表达式 str 转换成逆波兰式 new
{
    p=str;q=new; // 为方便起见 ,设 str 的两端都加上了优先级最低的特殊符号
    InitStack(s); //s 为运算符栈
    while(*p)
    {
        if(*p 是字母 )) *q++=*p; // 直接输出
        else
        {
            c=gettop(s);
            if(*p 优先级比 c 高) push(s,*p);
            else

```

```

{
    while(gettop(s) 优先级不比 *p 低)

```

严蔚敏《数据结构习题集》解答

```

        {
            pop(s,c);*(q++)=c;
        }//while
        push(s,*p); //运算符在栈内遵循越往栈顶优先级越高的原则
    }//else
}//else
p++;
}//while
}//NiBoLan // 参见编译原理教材

```

### 3.22

int GetValue\_NiBoLan(char \*str)// 对逆波兰式求值

```

{
    p=str;InitStack(s); //s 为操作数栈
    while(*p)
    {
        if(*p 是数 ) push(s,*p);
        else
        {
            pop(s,a);pop(s,b);
            r=compute(b,*p,a); //假设 compute 为执行双目运算的过程
            push(s,r);
        }//else
        p++;
    }//while
    pop(s,r);return r;
}//GetValue_NiBoLan

```

### 3.23

第 23 页 共 124 页

Status NiBoLan\_to\_BoLan(char \*str,stringtype &new)// 把逆波兰表达式 str 转换为波兰式 new

```

{
    p=str;Initstack(s); //s 的元素为 stringtype 类型
    while(*p)
    {
        if(*p 为字母 ) push(s,*p);
        else
        {
            if(StackEmpty(s)) return ERROR;

```

```

    pop(s,a);
    if(StackEmpty(s)) return ERROR;
    pop(s,b);
    c=link(link(*p,b),a);
    push(s,c);

```

严蔚敏《数据结构习题集》解答

```

    }//else
    p++;
} //while
pop(s,new);
if(!StackEmpty(s)) return ERROR;
return OK;
} //NiBoLan_to_BoLan

```

第 24 页 共 124 页

分析 :基本思想见书后注释 . 本题中暂不考虑串的具体操作的实现 ,而将其看作一种抽象数据

类型 stringtype,对其可以进行连接操作 :c=link(a,b).

3.24

```

Status g(int m,int n,int &s)// 求递归函数 g 的值 s
{
    if(m==0&&n>=0) s=0;
    else if(m>0&&n>=0) s=n+g(m-1,2*n);
    else return ERROR;
    return OK;
} //g

```

3.25

```

Status F_recursive(int n,int &s)// 递归算法
{
    if(n<0) return ERROR;
    if(n==0) s=n+1;
    else
    {
        F_recurve(n/2,r);
        s=n*r;
    }
    return OK;
} //F_recursive

```

```

Status F_nonrecursive(int n,int s)//非递归算法
{
    if(n<0) return ERROR;

```

```

if(n==0) s=n+1;
else
{
    InitStack(s); //s 的元素类型为 struct {int a;int b;}
    while(n!=0)
    {
        a=n;b=n/2;
        push(s,{a,b});

```

严蔚敏《数据结构习题集》解答

```

        n=b;
    }//while
    s=1;
    while(!StackEmpty(s))
    {
        pop(s,t);
        s*=t.a;
    }//while
}
return OK;
} //F_nonrecursive

```

3.26

```

float Sqrt_recursive(float A,float p,float e)// 求平方根的递归算法
{
    if(abs(p^2-A)<=e) return p;
    else return sqrt_recurve(A,(p+A/p)/2,e);
} //Sqrt_recurve

```

```

float Sqrt_nonrecursive(float A,float p,float e)// 求平方根的非递归算法
{
    while(abs(p^2-A)>=e)
        p=(p+A/p)/2;
    return p;
} //Sqrt_nonrecursive

```

3.27

第 25 页 共 124 页

这一题的所有算法以及栈的变化过程请参见《数据结构 (pascal 版)》,作者不再详细写出.

3.28

```

void InitCiQueue(CiQueue &Q)// 初始化循环链表表示的队列 Q

```

```

{
    Q=(CiLNode*)malloc(sizeof(CiLNode));
    Q->next=Q;
} //InitCiQueue

void EnCiQueue(CiQueue &Q,int x)// 把元素 x 插入循环链表表示的队列 Q,Q 指向队
尾元
素,Q->next 指向头结点 ,Q->next->next 指向队头元素
{
    p=(CiLNode*)malloc(sizeof(CiLNode));
    p->data=x;
    p->next=Q->next; // 直接把 p 加在 Q 的后面
    Q->next=p;
    Q=p; //修改尾指针
}

```

第 26 页 共 124 页

```

Status DeCiQueue(CiQueue &Q,int x)// 从循环链表表示的队列 Q 头部删除元素 x
{
    if(Q==Q->next) return INFEASIBLE; // 队列已空
    p=Q->next->next;
    x=p->data;
    Q->next->next=p->next;
    free(p);
    return OK;
} //DeCiQueue

```

3.29

```

Status EnCyQueue(CyQueue &Q,int x)// 带 tag 域的循环队列入队算法
{
    if(Q.front==Q.rear&&Q.tag==1) //tag 域的值为 0 表示 "空",1 表示 "满"
        return OVERFLOW;
    Q.base[Q.rear]=x;
    Q.rear=(Q.rear+1)%MAXSIZE;
    if(Q.front==Q.rear) Q.tag=1; // 队列满
} //EnCyQueue

```

```

Status DeCyQueue(CyQueue &Q,int &x)// 带 tag 域的循环队列出队算法
{
    if(Q.front==Q.rear&&Q.tag==0) return INFEASIBLE;
    Q.front=(Q.front+1)%MAXSIZE;
    x=Q.base[Q.front];
    if(Q.front==Q.rear) Q.tag=1; // 队列空
}

```

```

    return OK;
} // DeCyQueue

```

分析 : 当循环队列容量较小而队列中每个元素占的空间较多时 , 此种表示方法可以节约较多的存储空间 , 较有价值 .

3.30

Status EnCyQueue(CyQueue &Q, int x) // 带 length 域的循环队列入队算法

```

{
    if(Q.length == MAXSIZE) return OVERFLOW;
    Q.rear = (Q.rear + 1) % MAXSIZE;
    Q.base[Q.rear] = x;
    Q.length++;
}

```

严蔚敏《数据结构习题集》解答

```

    return OK;
} // EnCyQueue

```

Status DeCyQueue(CyQueue &Q, int &x) // 带 length 域的循环队列出队算法

```

{
    if(Q.length == 0) return INFEASIBLE;
    head = (Q.rear - Q.length + 1) % MAXSIZE; // 详见书后注释
    x = Q.base[head];
    Q.length--;
} // DeCyQueue

```

3.31

第 27 页 共 124 页

int Palindrome\_Test() // 判别输入的字符串是否回文序列 , 是则返回 1, 否则返回 0

```

{
    InitStack(S); InitQueue(Q);
    while((c = getchar()) != '@')
    {
        Push(S, c); EnQueue(Q, c); // 同时使用栈和队列两种结构
    }
    while(!StackEmpty(S))
    {
        Pop(S, a); DeQueue(Q, b);
        if(a != b) return ERROR;
    }
    return OK;
} // Palindrome_Test

```

3.32



```

void GetFib_CyQueue(int k,int n)// 求 k 阶斐波那契序列的前 n+1 项
{
    InitCyQueue(Q); // 其 MAXSIZE 设置为 k
    for(i=0;i<k-1;i++) Q.base[i]=0;
    Q.base[k-1]=1; // 给前 k 项赋初值
    for(i=0;i<k;i++) printf("%d",Q.base[i]);
    for(i=k;i<=n;i++)
    {
        m=i%k;sum=0;
        for(j=0;j<k;j++) sum+=Q.base[(m+j)%k];
        Q.base[m]=sum; // 求第 i 项的值存入队列中并取代已无用的第一项
        printf("%d",sum);
    }
}
//GetFib_CyQueue
严蔚敏《数据结构习题集》解答

```

### 3.33

```

Status EnDQueue(DQueue &Q,int x)// 输出受限的双端队列的入队操作
{
    if((Q.rear+1)%MAXSIZE==Q.front) return OVERFLOW; // 队列满
    avr=(Q.base[Q.rear-1]+Q.base[Q.front])/2;
    if(x>=avr) // 根据 x 的值决定插入在队头还是队尾
    {
        Q.base[Q.rear]=x;
        Q.rear=(Q.rear+1)%MAXSIZE;
    } // 插入在队尾
    else
    {
        Q.front=(Q.front-1)%MAXSIZE;
        Q.base[Q.front]=x;
    } // 插入在队头
    return OK;
}
//EnDQueue

```

```

Status DeDQueue(DQueue &Q,int &x)// 输出受限的双端队列的出队操作
{
    if(Q.front==Q.rear) return INFEASIBLE; // 队列空
    x=Q.base[Q.front];
    Q.front=(Q.front+1)%MAXSIZE;
    return OK;
}
//DeDQueue

```

### 3.34

void Train\_Rearrange(char \*train)// 这里用字符串 train 表示火车 , 'P'表示硬座 , 'H'表示硬卧 , 'S'表示软卧 , 最终按 PSH 的顺序排列

```
{
    r=train;
    InitDQueue(Q);
    while(*r)
    {
        if(*r=='P')
        {
            printf("E");
            printf("D"); // 实际上等于不入队列 ,直接输出 P 车厢
        }
        else if(*r=='S')
        {
            printf("E");
            EnDQueue(Q,*r,0); //0 表示把 S 车厢从头端入队列
```

严蔚敏《数据结构习题集》解答

```
    }
    else
    {
        printf("A");
        EnDQueue(Q,*r,1); //1 表示把 H 车厢从尾端入队列
    }
} //while
while(!DQueueEmpty(Q))
{
    printf("D");
    DeDQueue(Q);
} //while // 从头端出队列的车厢必然是先 S 后 H 的顺序
} //Train_Rearrange
```

第 29 页 共 124 页

严蔚敏《数据结构习题集》解答

## 第四章 串

### 4.10

```
void String_Reverse(Stringtype s,Stringtype &r)// 求 s 的逆串 r
{
    StrAssign(r,' '); // 初始化 r 为空串
    for(i=Strlen(s);i-->0)
    {
```

```

        StrAssign(c,SubString(s,i,1));
        StrAssign(r,Concat(r,c)); // 把 s 的字符从后往前添加到 r 中
    }
} //String_Reverse

```

4.11

第 30 页 共 124 页

void String\_Subtract(Stringtype s,Stringtype t,Stringtype &r) // 求所有包含在串 s 中而 t 中没有的字符构成的新串 r

```

{
    StrAssign(r,"");
    for(i=1;i<=Strlen(s);i++)
    {
        StrAssign(c,SubString(s,i,1));
        for(j=1;j<i&&StrCompare(c,SubString(s,j,1));j++); // 判断 s 的当前字符 c 是否第一次出现
        if(i==j)
        {
            for(k=1;k<=Strlen(t)&&StrCompare(c,SubString(t,k,1));k++); // 判断当前字符是否包含在 t 中
            if(k>Strlen(t)) StrAssign(r,Concat(r,c));
        }
    }
} //for
} //String_Subtract

```

4.12

int Replace(Stringtype &S,Stringtype T,Stringtype V) // 将串 S 中所有子串 T 替换为 V,并返回置换次数

```

{
    for(n=0,i=1;i<=Strlen(S)-Strlen(T)+1;i++) // 注意 i 的取值范围
        if(!StrCompare(SubString(S,i,Strlen(T)),T)) // 找到了与 T 匹配的子串
        { // 分别把 T 的前面和后面部分保存为 head 和 tail
            StrAssign(head,SubString(S,1,i-1));
            StrAssign(tail,SubString(S,i+Strlen(T),Strlen(S)-i-Strlen(T)+1));
            StrAssign(S,Concat(head,V));
            StrAssign(S,Concat(S,tail)); // 把 head,V,tail 连接为新串
            i+=Strlen(V); // 当前指针跳到插入串以后
            n++;
        } //if
    return n;
}

```

严蔚敏《数据结构习题集》解答

} //Replace

第 31 页 共 124 页

分析 :i+=Strlen(V); 这一句是必需的 ,也是容易忽略的 .如省掉这一句 ,则在某些情况下 ,会

引起

不希望的后果，虽然在大多数情况下没有影响。请思考：设  $S='place'$ ,  $T='ace'$ ,  $V='face'$ ，则省掉  $i+=\text{Strlen}(V)$ ；运行时会出现什么结果？

4.13

$\text{int Delete\_SubString}(\text{Stringtype } \&s, \text{Stringtype } t) //$  从串  $s$  中删除所有与  $t$  相同的子串，并返回删除

次数

```
{
    for(n=0,i=1;i<=Strlen(s)-Strlen(t)+1;i++)
        if(!StrCompare(SubString(s,i,Strlen(t)),t))
        {
            StrAssign(head,SubString(S,1,i-1));
            StrAssign(tail,SubString(S,i+Strlen(t),Strlen(s)-i-Strlen(t)+1));
            StrAssign(S,Concat(head,tail)); //把 head,tail 连接为新串
            n++;
        }//if
    return n,
} //Delete_SubString
```

4.14

$\text{Status NiBoLan\_to\_BoLan}(\text{Stringtype } str, \text{Stringtype } \&\text{new}) //$  把前缀表达式  $str$  转换为后缀式

$\text{new}$

```
{
    Initstack(s); //s 的元素为 Stringtype 类型
    for(i=1;i<=Strlen(str);i++)
    {
        r=SubString(str,i,1);
        if(r 为字母 ) push(s,r);
        else
        {
            if(StackEmpty(s)) return ERROR;
            pop(s,a);
            if(StackEmpty(s)) return ERROR;
            pop(s,b);
            StrAssign(t,Concat(r,b));
            StrAssign(c,Concat(t,a)); //把运算符 r,子前缀表达式 a,b 连接为新子前缀表达式
            c
            push(s,c);
        }
    } //for
    pop(s,new);
    if(!StackEmpty(s)) return ERROR;
    return OK;
} //NiBoLan_to_BoLan
```

分析：基本思想见书后注释 3.23。请读者用此程序取代作者早些时候对 3.23 题给出的程序。

## 4.15

严蔚敏《数据结构习题集》解答

第 32 页 共 124 页

void StrAssign(Stringtype &T,char chars&#x2013;)// 用字符数组 chars 给串 T 赋值,Stringtype 的定义  
见课本

```
{
    for(i=0,T[0]=0;chars[i];T[0]++,i++) T[i+1]=chars[i];
} //StrAssign
```

## 4.16

char StrCompare(Stringtype s,Stringtype t)// 串的比较 ,s>t 时返回正数 ,s=t 时返回 0,s<t 时返回负数

```
{
    for(i=1;i<=s[0]&& i<=t[0]&& s[i]==t[i];i++);
    if(i>s[0]&& i>t[0]) return 0;
    else if(i>s[0]) return -t[i];
    else if(i>t[0]) return s[i];
    else return s[i]-t[i];
} //StrCompare
```

## 4.17

int String\_Replace(Stringtype &S,Stringtype T,Stringtype V);// 将串 S 中所有子串 T 替换为 V,并  
返回替换次数

```
{
    for(n=0,i=1;i<=S[0]-T[0]+1;i++)
    {
        for(j=i,k=1;T[k]&&S[j]==T[k];j++,k++);
        if(k>T[0]) // 找到了与 T 匹配的子串 :分三种情况处理
        {
            if(T[0]==V[0])
                for(l=1;l<=T[0];l++) // 新子串长度与原子串相同时 :直接替换
                    S[i+l-1]=V[l];
            else if(T[0]<V[0]) // 新子串长度大于原子串时 :先将后部右移
            {
                for(l=S[0];l>=i+T[0];l--)
                    S[l+V[0]-T[0]]=S[l];
                for(l=1;l<=V[0];l++)
                    S[i+l-1]=V[l];
            }
            else //新子串长度小于原子串时 :先将后部左移
            {
                for(l=i+V[0];l<=S[0]+V[0]-T[0];l++)
                    S[l]=S[l-V[0]+T[0]];
            }
        }
        n++;
    }
    return n;
}
```

```

        for(l=1;l<=V[0];l++)
            S[i+l-1]=V[l];
    }
    S[0]=S[0]-T[0]+V[0];
    i+=V[0];n++;
} //if

```

严蔚敏《数据结构习题集》解答

```

    } //for
    return n;
} //String_Replace

```

4.18

```

typedef struct {
    char ch;
    int num;
} mytype;

void StrAnalyze(Stringtype S) // 统计串 S 中字符的种类和个数
{
    mytype T[MAXSIZE]; // 用结构数组 T 存储统计结果
    for(i=1;i<=S[0];i++)
    {
        c=S[i];j=0;
        while(T[j].ch&&T[j].ch!=c) j++; // 查找当前字符 c 是否已记录过
        if(T[j].ch) T[j].num++;
        else T[j]={c,1};
    } //for
    for(j=0;T[j].ch;j++)
        printf("%c: %d\n",T[j].ch,T[j].num);
} //StrAnalyze

```

4.19

第 33 页 共 124 页

```

void Subtract_String(Stringtype s,Stringtype t,Stringtype &r) // 求所有包含在串 s 中而 t
中没有的
字符构成的新串 r
{
    r[0]=0;
    for(i=1;i<=s[0];i++)
    {
        c=s[i];
        for(j=1;j<i&& s[j]!=c;j++); // 判断 s 的当前字符 c 是否第一次出现
        if(i==j)
        {
            for(k=1;k<=t[0]&&t[k]!=c;k++); // 判断当前字符是否包含在 t 中
            if(k>t[0]) r[++r[0]]=c;
        }
    }
}

```

```

    }//for
} //Subtract_String
4.20
int SubString_Delete(Stringtype &s,Stringtype t)// 从串 s 中删除所有与 t 相同的子串 ,并返回删除
次数
{
    for(n=0,i=1;i<=s[0]-t[0]+1;i++)
    {
        for(j=1;j<=t[0]&& s[i+j-1]==t[j];j++);

```

严蔚敏《数据结构习题集》解答

```

        if(j>m) // 找到了与 t 匹配的子串
        {
            for(k=i;k<=s[0]-t[0];k++) s[k]=s[k+t[0]]; // 左移删除
            s[0]-=t[0];n++;
        }
    } //for
    return n;
} //Delete_SubString
4.21
typedef struct{

```

```

    char ch;
    LStrNode *next;
} LStrNode,*LString; // 链串结构
void StringAssign(LString &s,LString t)// 把串 t 赋值给串 s
{
    s=malloc(sizeof(LStrNode));
    for(q=s,p=t->next;p;p=p->next)
    {
        r=(LStrNode*)malloc(sizeof(LStrNode));
        r->ch=p->ch;
        q->next=r;q=r;
    }
    q->next=NULL;
} //StringAssign

```

第 34 页 共 124 页

```

void StringCopy(LString &s,LString t)// 把串 t 复制为串 s.与前一个程序的区别在于 ,串
s 业已存
在.
{
    for(p=s->next,q=t->next;p&& q;p=p->next,q=q->next)
    {
        p->ch=q->ch;pre=p;
    }
}

```



```

while(q)
{
    p=(LStrNode*)malloc(sizeof(LStrNode));
    p->ch=q->ch;
    pre->next=p;pre=p;
}
p->next=NULL;
} //StringCopy
char StringCompare(LString s,LString t)// 串的比较 ,s>t 时返回正数 ,s=t 时返回 0,s<t 时返回负数
{
    for(p=s->next,q=t->next;p&&q&&p->ch==q->ch;p=p->next,q=q->next);
    if(!p&&!q) return 0;
    else if(!p) return -(q->ch);

```

严蔚敏《数据结构习题集》解答

```

    else if(!q) return p->ch;
    else return p->ch-q->ch;
} //StringCompare
int StringLen(LString s)// 求串 s 的长度 (元素个数 )
{
    for(i=0,p=s->next;p;p=p->next,i++);
    return i;
} //StringLen
LString * Concat(LString s,LString t)// 连接串 s 和串 t 形成新串 ,并返回指针
{
    p=malloc(sizeof(LStrNode));
    for(q=p,r=s->next;r;r=r->next)
    {
        q->next=(LStrNode*)malloc(sizeof(LStrNode));
        q=q->next;
        q->ch=r->ch;
    } //for // 复制串 s
    for(r=t->next;r;r=r->next)
    {
        q->next=(LStrNode*)malloc(sizeof(LStrNode));
        q=q->next;
        q->ch=r->ch;
    } //for // 复制串 t
    q->next=NULL;
    return p;
} //Concat

```

```

LString * Sub_String(LString s,int start,int len)// 返回一个串 ,其值等于串 s 从 start 位置起长为

```

len 的子串

```
{
    p=malloc(sizeof(LStrNode));q=p;
    for(r=s;start;start--,r=r->next); // 找到 start 所对应的结点指针 r
    for(i=1;i<=len;i++,r=r->next)
    {
        q->next=(LStrNode*)malloc(sizeof(LStrNode));
        q=q->next;
        q->ch=r->ch;
    } // 复制串 t
    q->next=NULL;
    return p;
} // Sub_String
```

4.22

void LString\_Concat(LString &t,LString &s,char c) // 用块链存储结构,把串 s 插入到串 t 的字符

c

之后

```
{
```

严蔚敏《数据结构习题集》解答

```
    p=t.head;
    while(p&&!(i=Find_Char(p,c))) p=p->next; // 查找字符 c
    if(!p) // 没找到
    {
        t.tail->next=s.head;
        t.tail=s.tail; // 把 s 连接在 t 的后面
    }
    else
    {
        q=p->next;
        r=(Chunk*)malloc(sizeof(Chunk)); // 将包含字符 c 的节点 p 分裂为两个
        for(j=0;j<i;j++) r->ch[j]='#'; // 原节点 p 包含 c 及其以前的部分
        for(j=i;j<CHUNKSIZE;j++) // 新节点 r 包含 c 以后的部分
        {
            r->ch[j]=p->ch[j];
            p->ch[j]='#'; // p 的后半部分和 r 的前半部分的字符改为无效字符 '#'
        }
        p->next=s.head;
        s.tail->next=r;
        r->next=q; // 把串 s 插入到结点 p 和 r 之间
    } // else
    t	curlen+=s	curlen; // 修改串长
    s	curlen=0;
} // LString_Concat
```

```
int Find_Char(Chunk *p,char c)// 在某个块中查找字符 c,如找到则返回位置是第几个字符,如没找到则返回 0
```

```
{
    for(i=0;i<CHUNKSIZE&&p->ch[i]!=c;i++);
    if(i==CHUNKSIZE) return 0;
    else return i+1;
} //Find_Char
```

#### 4.23

```
int LString_Palindrome(LString L)// 判断以块链结构存储的串 L 是否为回文序列 ,是则返回 1, 否则返回 0
```

```
{
    InitStack(S);
    p=S.head;i=0;k=1; //i 指示元素在块中的下标 ,k 指示元素在整个序列中的序号 (从 1 开始)
    for(k=1;k<=S.curlen;k++)
    {
        if(k<=S.curlen/2) Push(S,p->ch[i]); // 将前半段的字符入串
        else if(k>(S.curlen+1)/2)
        {
            Pop(S,c); //将后半段的字符与栈中的元素相匹配
```

严蔚敏《数据结构习题集》解答

```
        if(p->ch[i]!=c) return 0; // 失配
    }
}
```

第 37 页 共 124 页

```
    if(++i==CHUNKSIZE) // 转到下一个元素 ,当为块中最后一个元素时 ,转到下一块
    {
        p=p->next;
        i=0;
    }
} //for
return 1; //成功匹配
} //LString_Palindrome
```

#### 4.24

```
void HString_Concat(HString s1,HString s2,HString &t)// 将堆结构表示的串 s1 和 s2 连接为新串
```

```
t
{
    if(t.ch) free(t.ch);
    t.ch=malloc((s1.length+s2.length)*sizeof(char));
    for(i=1;i<=s1.length;i++) t.ch[i-1]=s1.ch[i-1];
    for(j=1;j<=s2.length;j++,i++) t.ch[i-1]=s2.ch[j-1];
    t.length=s1.length+s2.length;
```

```
}//HString_Concat
```

4.25

int HString\_Replace(HString &S,HString T,HString V)// 堆结构串上的置换操作 ,返回置换次数

```
{
    for(n=0,i=0;i<=S.length-T.length;i++)
    {
        for(j=i,k=0;k<T.length&&S.ch[j]==T.ch[k];j++,k++);
        if(k==T.length) // 找到了与 T 匹配的子串 :分三种情况处理
        {
            if(T.length==V.length)
                for(l=1;l<=T.length;l++) // 新子串长度与原子串相同时 :直接替换
                    S.ch[i+l-1]=V.ch[l-1];
            else if(T.length<V.length) // 新子串长度大于原子串时 :先将后部右移
            {
                for(l=S.length-1;l>=i+T.length;l--)
                    S.ch[l+V.length-T.length]=S.ch[l];
                for(l=0;l<V.length;l++)
                    S[i+l]=V[l];
            }
            else //新子串长度小于原子串时 :先将后部左移
            {
                for(l=i+V.length;l<S.length+V.length-T.length;l++)
                    S.ch[l]=S.ch[l-V.length+T.length];
                for(l=0;l<V.length;l++)
                    S[i+l]=V[l];
            }
            S.length+=V.length-T.length;
            i+=V.length;n++;
        }
    }
    return n;
}
//HString_Replace
```

4.26

第 38 页 共 124 页

Status HString\_Insert(HString &S,int pos,HString T)// 把 T 插入堆结构表示的串 S 的第 pos 个字符之前

```
{
    if(pos<1) return ERROR;
    if(pos>S.length) pos=S.length+1;// 当插入位置大于串长时 ,看作添加在串尾
    S.ch=realloc(S.ch,(S.length+T.length)*sizeof(char));
    for(i=S.length-1;i>=pos-1;i--)
```

```

        S.ch[i+T.length]=S.ch[i]; // 后移为插入字符串让出位置
    for(i=0;i<T.length;i++)
        S.ch[pos+i-1]=T.ch[pos]; // 插入串 T
    S.length+=T.length;
    return OK;
} // HString_Insert
4.27
int Index_New(Stringtype s,Stringtype t) // 改进的定位算法
{
    i=1;j=1;
    while(i<=s[0]&& j<=t[0])
    {
        if((j!=1&&s[i]==t[j])||(j==1&&s[i]==t[j]&&s[i+t[0]-1]==t[t[0]]))
        { // 当 j==1 即匹配模式串的第一个字符时 ,需同时匹配其最后一个
            i=i+j-2;
            j=1;
        }
        else
        {
            i++;j++;
        }
    } // while
    if(j>t[0]) return i-t[0];
} // Index_New

```

```

4.28
void LGet_next(LString &T) // 链串上的 get_next 算法
{
    p=T->succ;p->next=T;q=T;
    while(p->succ)

```

严蔚敏《数据结构习题集》解答

```

    {
        if(q==T||p->data==q->data)
        {
            p=p->succ;q=q->succ;
            p->next=q;
        }
        else q=q->next;
    } // while
} // LGet_next
4.29

```

LStrNode \* LIndex\_KMP(LString S,LString T,LStrNode \*pos) // 链串上的 KMP 匹配算法,返回  
值为匹配的子串首指针

```

{
    p=pos;q=T->succ;
    while(p&&q)
    {
        if(q==T||p->chdata==q->chdata)
        {
            p=p->succ;
            q=q->succ;
        }
        else q=q->next;
    }//while
    if(!q)
    {
        for(i=1;i<=Strlen(T);i++)
            p=p->next;
        return p;
    } // 发现匹配后 ,要往回找子串的头
    return NULL;
} //LIndex_KMP

```

4.30

void Get\_LRepSub(Stringtype S)// 求 S 的最长重复子串的位置和长度

```

{
    for(maxlen=0,i=1;i<S[0];i++)// 串 S2 向右移 i 格
    {
        for(k=0,j=1;j<=S[0]-i;j++)//j 为串 S2 的当前指针 ,此时串 S1 的当前指针为 i+j,两指针同步移动
        {
            if(S[j]==S[j+i]) k++; // 用 k 记录连续相同的字符数
            else k=0; //失配时 k 归零
            if(k>maxlen) // 发现了比以前发现的更长的重复子串
            {
                lrs1=j-k+1;lrs2=mrs1+i;maxlen=k; // 作记录
            }
        }
    }//for
} //for
if(maxlen)
{
    printf("Longest Repeating Substring length:%d\n",maxlen);
    printf("Position1:%d Position 2:%d\n",lrs1,lrs2);
}
else printf("No Repeating Substring found!\n");
} //Get_LRepSub

```

严蔚敏《数据结构习题集》解答

分析:i 代表 "错位值".本算法的思想是,依次把串 S 的一个副本 S2 向右错位平移 1 格,2 格,3 格,...

与自身 S1 相匹配,如果存在最长重复子串,则必然能在此过程中被发现.用变量

lrs1,lrs2,maxlen 来记录已发现的最长重复子串第一次出现位置,第二次出现位置和长度.题目

中未说明 "重复子串"是否允许有重叠部分,本算法假定允许.如不允许,只需在第二个 for 语句

的循环条件中加上  $k \leq i$  即可.本算法时间复杂度为  $O(\text{Strlen}(S)^2)$ .

4.31

void Get\_LPubSub(Stringtype S,Stringtype T)// 求串 S 和串 T 的最长公共子串位置和长度

```
{
    if(S[0]>=T[0])
    {
        StrAssign(A,S);StrAssign(B,T);
    }
    else
    {
        StrAssign(A,T);StrAssign(B,S);
    } // 为简化设计,令 S 和 T 中较长的那个为 A,较短的那个为 B
    for(maxlen=0,i=1-B[0];i<A[0];i++)
    {
        if(i<0) //i 为 B 相对于 A 的错位值,向左为负,左端对齐为 0,向右为正
        {
            jmin=1;jmax=i+B[0];
        } //B 有一部分在 A 左端的左边
        else if(i>A[0]-B[0])
        {
            jmin=i;jmax=A[0];
        } //B 有一部分在 A 右端的右边
        else
        {
            jmin=i;jmax=i+B[0];
        } //B 在 A 左右两端之间.

        //以上是根据 A 和 B 不同的相对位置确定 A 上需要匹配的区间 (与 B 重合的区间)的端
        点:jmin,jmax.
        for(k=0,j=jmin;j<=jmax;j++)
        {
            if(A[j]==B[j-i]) k++;
            else k=0;
```

严蔚敏《数据结构习题集》解答



```

        if(k>maxlen)
        {
            lps1=j-k+1;lps2=j-i-k+1;maxlen=k;
        }
    }//for
} //for
if(maxlen)
{
    if(S[0]>=T[0])
    {
        lpsS=lps1;lpsT=lps2;
    }
    else
    {
        lpsS=lps2;lpsT=lps1;
    } // 将 A,B 上的位置映射回 S,T 上的位置
    printf("Longest Public Substring length:%d\n",maxlen);
    printf("Position in S:%d    Position in T:%d\n",lpsS,lpsT);
} //if
else printf("No Repeating Substring found!\n");
} //Get_LPubSub

```

第 41 页 共 124 页

分析 :本题基本思路与上题同 .唯一的区别是 ,由于 A,B 互不相同 ,因此 B 不仅要向右错位 ,而且

还要向左错位 ,以保证不漏掉一些情况 .当 B 相对于 A 的位置不同时 ,需要匹配的区间的计算公式也各不相同 ,请读者自己画图以帮助理解 .本算法的时间复杂度是  $O(\text{strlen}(s) * \text{strlen}(t))$ 。

严蔚敏《数据结构习题集》解答

## 第五章 数组和广义表

5.18

第 42 页 共 124 页

```

void RSh(int A[n],int k)// 把数组 A 的元素循环右移 k 位 ,只用一个辅助存储空间
{
    for(i=1;i<=k;i++)
        if(n%i==0&& k%i==0) p=i;// 求 n 和 k 的最大公约数 p
    for(i=0;i<p;i++)
    {
        j=i;l=(i+n-k)%n;temp=A[j];
        while(l!=i)
        {
            A[j]=A[l];
            j=l;l=(j+n-k)%n;
        }
    }
}

```

```

    }// 循环右移一步
    A[j]=temp;
  }//for
} //RSh

```

分析:要把 A 的元素循环右移 k 位,则 A[0] 移至 A[k],A[k] 移至 A[2k]..... 直到最终回到 A[0].

然而这并没有全部解决问题,因为有可能有的元素在此过程中始终没有被访问过,而是被跳了过去.分析可知,当 n 和 k 的最大公约数为 p 时,只要分别以 A[0],A[1],...A[p-1] 为起点执行上述

算法,就可以保证每一个元素都被且仅被右移一次,从而满足题目要求.也就是说,A 的所

有元

素分别处在 p 个"循环链"上面.举例如下:

n=15,k=6,则 p=3.

第一条链:A[0]->A[6],A[6]->A[12],A[12]->A[3],A[3]->A[9],A[9]->A[0].

第二条链:A[1]->A[7],A[7]->A[13],A[13]->A[4],A[4]->A[10],A[10]->A[1].

第三条链:A[2]->A[8],A[8]->A[14],A[14]->A[5],A[5]->A[11],A[11]->A[2].

恰好使所有元素都右移一次.

虽然未经数学证明,但作者相信上述规律应该是正确的.

5.19

void Get\_Saddle(int A[m][n])// 求矩阵 A 中的马鞍点

```

{
    for(i=0;i<m;i++)
    {
        for(min=A[i][0],j=0;j<n;j++)
            if(A[i][j]<min) min=A[i][j]; // 求一行中的最小值
        for(j=0;j<n;j++)
            if(A[i][j]==min) // 判断这个(些)最小值是否鞍点
            {
                for(flag=1,k=0;k<m;k++)
                    if(min<A[k][j]) flag=0;
                if(flag)
                    printf("Found a saddle element!\nA[%d][%d]=%d",i,j,A[i][j]);
            }
    }
} //for
} //Get_Saddle

```

严蔚敏《数据结构习题集》解答

```

    }
} //for
} //Get_Saddle

```

5.20

int exps[MAXSIZE]; //exps 数组用于存储某一项的各变元的指数

int maxm,n; //maxm 指示变元总数,n 指示一个变元的最高指数

第 43 页 共 124 页

void Print\_Poly\_Descend(int \*a,int m)// 按降幂顺序输出 m 元多项式的项,各项的系数已经按

照题目要求存储于 m 维数组中,数组的头指针为 a

```

{
    maxm=m;
    for(i=m*n;i>=0;i--) // 按降幂次序 ,可能出现的最高项次数为 mn
        Get_All(a,m,i,0); // 确定并输出所有次数为 i 的项
} //Print_Poly_Descend
void Get_All(int *a,int m,int i,int seq) // 递归求出所有和为 i 的 m 个自然数
{
    if(seq==maxm) Print_Nomial(a,exps); // 已经求完时 ,输出该项
    else
    {
        min=i-(m-1)*n; // 当前数不能小于 min
        if(min<0) min=0;
        max=n<i?n:i; // 当前数不能大于 max
        for(j=min;j<=max;j++)
        {
            exps[seq]=j; //依次取符合条件的数
            Get_All(a,m-1,i-j,seq+1); // 取下一个数
        }
    }
} //else
exps[seq]=0; //返回
} //Get_All
void Print_Nomial(int *a,int exps[ ]) // 输出一个项 ,项的各变元的指数已经存储在数组 exps 中
{
    pos=0;
    for(i=0;i<maxm;i++) // 求出该项的系数在 m 维数组 a 中低下标优先的存储位置 pos
    {
        pos*=n;
        pos+=exps[i];
    }
    coef=(a+pos); // 取得该系数 coef
    if(!coef) return; // 该项为 0 时无需输出
    else if(coef>0) printf("+"); // 系数为正时打印加号
    else if(coef<0) printf("-"); // 系数为负时打印减号
    if(abs(coef)!=1) printf("%d",abs(coef)); // 当系数的绝对值不为 1 时打印系数
    for(i=0;i<maxm;i++)
        if(exps[i]) // 打印各变元及其系数

```

严蔚敏《数据结构习题集》解答

```

{
    printf("x");
    printf("%d",i);
    printf("E");
    if(exps[i]>1) printf("%d",exp[i]); // 系数为 1 时无需打印
}

```

```
}//Print_Nomial
```

第 44 页 共 124 页

分析:本算法的关键在于如何按照降幂顺序输出各项。这里采用了一个递归函数来找到所有满

足和为  $i$  的  $m$  个自然数作为各变元的指数。只要先取第一个数为  $j$ , 然后再找到所有满足和为  $i-j$  的  $m-1$  个自然数就行了。要注意  $j$  的取值范围必须使剩余  $m-1$  个自然数能够找到, 所以不能

小于  $i-(m-1)*\max n$ , 也不能大于  $i$ 。只要找到了一组符合条件的数, 就可以在存储多项式系数的数组中确定对应的项的系数的位置, 并且在系数不为 0 时输出对应的项。

## 5.21

void TSMatrix\_Add(TSMatrix A,TSMatrix B,TSMatrix &C)// 三元组表示的稀疏矩阵加法

```
{
    C.mu=A.mu;C.nu=A.nu;C.tu=0;
    pa=1;pb=1;pc=1;
    for(x=1;x<=A.mu;x++) // 对矩阵的每一行进行加法
    {
        while(A.data[pa].i<x) pa++;
        while(B.data[pb].i<x) pb++;
        while(A.data[pa].i==x&&B.data[pb].i==x) // 行列值都相等的元素
        {
            if(A.data[pa].j==B.data[pb].j)
            {
                ce=A.data[pa].e+B.data[pb].e;
                if(ce) // 和不为 0
                {
                    C.data[pc].i=x;
                    C.data[pc].j=A.data[pa].j;
                    C.data[pc].e=ce;
                    pa++;pb++;pc++;
                }
            }
            else if(A.data[pa].j>B.data[pb].j)
            {
                C.data[pc].i=x;
                C.data[pc].j=B.data[pb].j;
                C.data[pc].e=B.data[pb].e;
                pb++;pc++;
            }
            else
            {

```

严蔚敏《数据结构习题集》解答

```
C.data[pc].i=x;
```

```

        C.data[pc].j=A.data[pa].j;
        C.data[pc].e=A.data[pa].e
        pa++;pc++;
    }
} //while
while(A.data[pa]==x) // 插入 A 中剩余的元素 (第 x 行)
{
    C.data[pc].i=x;
    C.data[pc].j=A.data[pa].j;
    C.data[pc].e=A.data[pa].e
    pa++;pc++;
}
while(B.data[pb]==x) // 插入 B 中剩余的元素 (第 x 行)
{
    C.data[pc].i=x;
    C.data[pc].j=B.data[pb].j;
    C.data[pc].e=B.data[pb].e;
    pb++;pc++;
}
} //for
C.tu=pc;
} //TSMatrix_Add
5.22

```

第 45 页 共 124 页

```

void TSMatrix_Addto(TSMatrix &A,TSMatrix B)// 将三元组矩阵 B 加到 A 上
{
    for(i=1;i<=A.tu;i++)
        A.data[MAXSIZE-A.tu+i]=A.data[i];/ 把 A 的所有元素都移到尾部以腾出位置
    pa=MAXSIZE-A.tu+1;pb=1;pc=1;
    for(x=1;x<=A.mu;x++) // 对矩阵的每一行进行加法
    {
        while(A.data[pa].i<x) pa++;
        while(B.data[pb].i<x) pb++;
        while(A.data[pa].i==x&&B.data[pb].i==x)// 行列值都相等的元素
        {
            if(A.data[pa].j==B.data[pb].j)
            {
                ne=A.data[pa].e+B.data[pb].e;
                if(ne) // 和不为 0
                {
                    A.data[pc].i=x;
                    A.data[pc].j=A.data[pa].j;
                    A.data[pc].e=ne;
                    pa++;pb++;pc++;
                }
            }
        }
    }
}

```

严蔚敏《数据结构习题集》解答

```

    }
} //if
else if(A.data[pa].j>B.data[pb].j)
{
    A.data[pc].i=x;
    A.data[pc].j=B.data[pb].j;
    A.data[pc].e=B.data[pb].e;
    pb++;pc++;
}
else
{
    A.data[pc].i=x;
    A.data[pc].j=A.data[pa].j;
    A.data[pc].e=A.data[pa].e;
    pa++;pc++;
}
} //while
while(A.data[pa]==x) // 插入 A 中剩余的元素 (第 x 行)
{
    A.data[pc].i=x;
    A.data[pc].j=A.data[pa].j;
    A.data[pc].e=A.data[pa].e;
    pa++;pc++;
}
while(B.data[pb]==x) // 插入 B 中剩余的元素 (第 x 行)
{
    A.data[pc].i=x;
    A.data[pc].j=B.data[pb].j;
    A.data[pc].e=B.data[pb].e;
    pb++;pc++;
}
} //for
A.tu=pc;
for(i=A.tu;i<MAXSIZE;i++) A.data[i]={0,0,0}; // 清除原来的 A 中记录
} //TSMatrix_Addto

```

5.23

```

typedef struct{
    int j;
    int e;
} DSElem;

typedef struct{
    DSElem data[MAXSIZE];

```

```

    int cpot[MAXROW]; // 这个向量存储每一行在二元组中的起始位置
    int mu,nu,tu;

```

```
    } DMatrix; // 二元组矩阵类型
```

第 47 页 共 124 页

```
Status DMatrix_Locate(DMatrix A,int i,int j,int &e)// 求二元组矩阵的元素 A[i][j] 的值
e
{
    for(s=A.cpot[i];s<A.cpot[i+1]&&A.data[s].j!=j;s++);// 注意查找范围
    if(s<A.cpot[i+1]&&A.data[s].j==j) // 找到了元素 A[i][j]
    {
        e=A.data[s];
        return OK;
    }
    return ERROR;
} //DMatrix_Locate
```

5.24

```
typedef struct{
    int seq; //该元素在以行为主序排列时的序号
    int e;
} SElem;

typedef struct{
    SElem data[MAXSIZE];
    int mu,nu,tu;
} SMatrix; // 单下标二元组矩阵类型

Status SMatrix_Locate(SMatrix A,int i,int j,int &e)// 求单下标二元组矩阵的元素 A[i][j]
的值 e
{
    s=i*A.nu+j+1;p=1;
    while(A.data[p].seq<s) p++; // 利用各元素 seq 值逐渐递增的特点
    if(A.data[p].seq==s) // 找到了元素 A[i][j]
    {
        e=A.data[p].e;
        return OK;
    }
    return ERROR;
} //SMatrix_Locate
```

5.25

```
typedef enum{0,1} bool;
typedef struct{
    int mu,nu;
    int elem[MAXSIZE];
    bool map[mu][nu];
} BMatrix; // 用位图表示的矩阵类型

void BMatrix_Add(BMatrix A,BMatrix B,BMatrix &C)// 位图矩阵的加法
{
```



```

C.mu=A.mu;C.nu=A.nu;
pa=1;pb=1;pc=1;
for(i=0;i<A.mu;i++) // 每一行的相加
    for(j=0;j<A.nu;j++) // 每一个元素的相加

```

严蔚敏《数据结构习题集》解答

```

{
    if(A.map[i][j]&&B.map[i][j]&&(A.elem[pa]+B.elem[pb]))//      结果不为 0
    {
        C.elem[pc]=A.elem[pa]+B.elem[pb];
        C.map[i][j]=1;
        pa++;pb++;pc++;
    }
    else if(A.map[i][j]&&!B.map[i][j])
    {
        C.elem[pc]=A.elem[pa];
        C.map[i][j]=1;
        pa++;pc++;
    }
    else if(!A.map[i][j]&&B.map[i][j])
    {
        C.elem[pc]=B.elem[pb];
        C.map[i][j]=1;
        pb++;pc++;
    }
}
}
} //BMatrix_Add

```

5.26

第 48 页 共 124 页

```

void Print_OLMatrix(OLMatrix A)// 以三元组格式输出十字链表表示的矩阵
{
    for(i=0;i<A.mu;i++)
    {
        if(A.rhead[i])
            for(p=A.rhead[i];p;p=p->right) // 逐次遍历每一个行链表
                printf("%d %d %d\n",i,p->j,p->e;
    }
}
} //Print_OLMatrix

```

5.27

```

void OLMatrix_Add(OLMatrix &A,OLMatrix B)// 把十字链表表示的矩阵 B 加到 A
上
{
    for(j=1;j<=A.nu;j++) cp[j]=A.thead[j]; // 向量 cp 存储每一列当前最后一个元素的指
针
    for(i=1;i<=A.mu;i++)

```

```

{
    pa=A.rhead[i];pb=B.rhead[i];pre=NULL;
    while(pb)
    {
        if(pa==NULL||pa->j>pb->j) // 新插入一个结点
        {
            p=(OLNode*)malloc(sizeof(OLNode));
            if(!pre) A.rhead[i]=p;

```

严蔚敏《数据结构习题集》解答

```

        else pre->right=p;
        p->right=pa;pre=p;
        p->i=i;p->j=pb->j;p->e=pb->e; // 插入行链表中
        if(!A.chead[p->j])
        {
            A.chead[p->j]=p;
            p->down=NULL;
        }
        else
        {
            while(cp[p->j]->down) cp[p->j]=cp[p->j]->down;
            p->down=cp[p->j]->down;
            cp[p->j]->down=p;
        }
        cp[p->j]=p; // 插入列链表中
    }//if
    else if(pa->j<pb->j)
    {
        pre=pa;
        pa=pa->right;
    } //pa 右移一步
    else if(pa->e+pb->e)
    {
        pa->e+=pb->e;
        pre=pa;pa=pa->right;
        pb=pb->right;
    } // 直接相加
    else
    {
        if(!pre) A.rhead[i]=pa->right;
        else pre->right=pa->right;
        p=pa;pa=pa->right; //从行链表中删除
        if(A.chead[p->j]==p)
            A.chead[p->j]=cp[p->j]=p->down;
        else cp[p->j]->down=p->down; // 从列链表中删除
        free (p);

```

```

        }//else
    }//while
}//for

```

```

}//OLMatrix_Add

```

分析 :本题的具体思想在课本中有详细的解释说明 .

5.28

第 49 页 共 124 页

```

void MPList_PianDao(MPList &L)// 对广义表存储结构的多元多项式求第一变元的偏导
{
    严蔚敏《数据结构习题集》解答

```

```

    for(p=L->hp->tp;p&& p->exp;pre=p,p=p->tp)
    {
        if(p->tag) Mul(p->hp,p->exp);
        else p->coef*=p->exp; // 把指数乘在本结点或其下属结点上
        p->exp--;
    }
    pre->tp=NULL;
    if(p) free (p); // 删除可能存在的常数项

```

```

}//MPList_PianDao

```

```

void Mul(MPList &L,int x)// 递归算法 ,对多元多项式 L 乘以 x
{
    for(p=L;p;p=p->tp)
    {
        if(!p->tag) p->coef*=x;
        else Mul(p->hp,x);
    }
}
}//Mul

```

5.29

第 50 页 共 124 页

```

void MPList_Add(MPList A,MPList B,MPList &C)// 广义表存储结构的多元多项式相加的递归算法
{
    C=(MPLNode*)malloc(sizeof(MPLNode));
    if(!A->tag&&!B->tag) // 原子项 ,可直接相加
    {
        C->coef=A->coef+B->coef;
        if(!C->coef)
        {
            free(C);
            C=NULL;
        }
    }
    }//if
    else if(A->tag&&B->tag) // 两个多项式相加

```

```

{
    p=A;q=B;pre=NULL;
    while(p&&q)
    {
        if(p->exp==q->exp)
        {
            C=(MPLNode*)malloc(sizeof(MPLNode));
            C->exp=p->exp;
            MPLList_Add(A->hp,B->hp,C->hp);
            pre->tp=C;pre=C;
            p=p->tp;q=q->tp;
        }
    }
}

```

严蔚敏《数据结构习题集》解答

```

        else if(p->exp>q->exp)
        {
            C=(MPLNode*)malloc(sizeof(MPLNode));
            C->exp=p->exp;
            C->hp=A->hp;
            pre->tp=C;pre=C;
            p=p->tp;
        }
        else
        {
            C=(MPLNode*)malloc(sizeof(MPLNode));
            C->exp=q->exp;
            C->hp=B->hp;
            pre->tp=C;pre=C;
            q=q->tp;
        }
    }//while
    while(p)
    {
        C=(MPLNode*)malloc(sizeof(MPLNode));
        C->exp=p->exp;
        C->hp=p->hp;
        pre->tp=C;pre=C;
        p=p->tp;
    }
    while(q)
    {
        C=(MPLNode*)malloc(sizeof(MPLNode));
        C->exp=q->exp;
        C->hp=q->hp;
        pre->tp=C;pre=C;
        q=q->tp;
    }
}

```

```

    } // 将其同次项分别相加得到新的多项式    ,原理见第二章多项式相加一题
} //else if
else if(A->tag&&!B->tag) // 多项式和常数项相加
{
    x=B->coef;
    for(p=A;p->tp->tp;p=p->tp);

```

第 51 页 共 124 页

```

    if(p->tp->exp==0) p->tp->coef+=x; // 当多项式中含有常数项时    ,加上常数项
    if(!p->tp->coef)
    {
        free(p->tp);
        p->tp=NULL;
    }

```

严蔚敏《数据结构习题集》解答

```

    else
    {
        q=(MPLNode*)malloc(sizeof(MPLNode));
        q->coef=x;q->exp=0;
        q->tag=0;q->tp=NULL;
        p->tp=q;
    } // 否则新建常数项    ,下同
} //else if
else
{
    x=A->coef;
    for(p=B;p->tp->tp;p=p->tp);
    if(p->tp->exp==0) p->tp->coef+=x;
    if(!p->tp->coef)
    {
        free(p->tp);
        p->tp=NULL;
    }
    else
    {
        q=(MPLNode*)malloc(sizeof(MPLNode));
        q->coef=x;q->exp=0;
        q->tag=0;q->tp=NULL;
        p->tp=q;
    }
} //else

```

} //MPList\_Add

5.30

int GList\_Getdeph(GList L) // 求广义表深度的递归算法

```

{

```

```

    if(!L->tag) return 0; // 原子深度为 0
    else if(!L) return 1; // 空表深度为 1
    m=GList_Getdeph(L->ptr.hp)+1;
    n=GList_Getdeph(L->ptr.tp);
    return m>n?m:n;
} //GList_Getdeph

```

5.31

```

void GList_Copy(GList A,GList &B) // 复制广义表的递归算法
{
    if(!A->tag) // 当结点为原子时 ,直接复制
    {
        B->tag=0;
        B->atom=A->atom;
    }

```

第 52 页 共 124 页

严蔚敏《数据结构习题集》解答

```

    else //当结点为子表时
    {
        B->tag=1;
        if(A->ptr.hp)
        {
            B->ptr.hp=malloc(sizeof(GLNode));
            GList_Copy(A->ptr.hp,B->ptr.hp);
        } // 复制表头
        if(A->ptr.tp)
        {
            B->ptr.tp=malloc(sizeof(GLNode));
            GList_Copy(A->ptr.tp,B->ptr.tp);
        } // 复制表尾
    } //else
} //GList_Copy

```

5.32

第 53 页 共 124 页

```

int GList_Equal(GList A,GList B) // 判断广义表 A 和 B 是否相等 ,是则返回 1,否则返回 0
{ // 广义表相等可分三种情况 :
    if(!A&&!B) return 1; // 空表是相等的
    if(!A->tag&&!B->tag&&A->atom==B->atom) return 1; // 原子的值相等
    if(A->tag&&B->tag)
        if(GList_Equal(A->ptr.hp,B->ptr.hp)&&GList_Equal(A->ptr.tp,B->ptr.tp))
            return 1; //表头表尾都相等
    return 0;
} //GList_Equal

```

5.33

void GList\_PrintElem(GList A,int layer)// 递归输出广义表的原子及其所在层次 ,layer 表示当前层次

```
{
    if(!A) return;
    if(!A->tag) printf("%d %d\n",A->atom,layer);
    else
    {
        GList_PrintElem(A->ptr.hp,layer+1);
        GList_PrintElem(A->ptr.tp,layer); // 注意尾表与原表是同一层次
    }
}
//GList_PrintElem
```

5.34

void GList\_Reverse(GList A)// 递归逆转广义表 A

```
{
    GLNode *ptr[MAX_SIZE];
    if(A->tag&&A->ptr.tp) // 当 A 不为原子且表尾非空时才需逆转
    {
        for(i=0,p=A;p;p=p->ptr.tp,i++)
```

严蔚敏《数据结构习题集》解答

```
    {
        if(p->ptr.hp) GList_Reverse(p->ptr.hp); //逆转各子表
        ptr[i]=p->ptr.hp;
    }
    for(p=A;p;p=p->ptr.tp) // 重新按逆序排列各子表的顺序
        p->ptr.hp=ptr[--i];
}
//GList_Reverse
```

5.35

Status Create\_GList(GList &L)// 根据输入创建广义表 L,并返回指针

```
{
    scanf("%c",&ch);
    if(ch==' ')
    {
        L=NULL;
        scanf("%c",&ch);
        if(ch!='') return ERROR;
        return OK;
    }
    L=(GList)malloc(sizeof(GLNode));
    L->tag=1;
    if(isalphabet(ch)) // 输入是字母
    {
        p=(GList)malloc(sizeof(GLNode)); // 建原子型表头
        p->tag=0;p->atom=ch;
```

```

        L->ptr.hp=p;
    }
    else if(ch=='(') Create_GList(L->ptr.hp); // 建子表型表头
    else return ERROR;
    scanf ("%c",&ch);
    if(ch==')') L->ptr.tp=NULL;
    else if(ch==',') Create_GList(L->ptr.tp); // 建表尾
    else return ERROR;
    return OK;
} //Create_GList

```

分析 :本题思路见书后解答 .

5.36

void GList\_PrintList(GList A) // 按标准形式输出广义表

```

{
    if(!A) printf("()"); // 空表
    else if(!A->tag) printf("%d",A->atom); // 原子
    else
    {
        printf("(");

```

第 54 页 共 124 页

严蔚敏《数据结构习题集》解答

```

        for(p=A;p;p=p->ptr.tp)
        {
            GList_PrintList(p->ptr.hp);
            if(p->ptr.tp) printf(","); // 只有当表尾非空时才需要打印逗号
        }
        printf(")");
    } //else
} //GList_PrintList

```

5.37

第 55 页 共 124 页

void GList\_DelElem(GList &A,int x) // 从广义表 A 中删除所有值为 x 的原子

```

{
    if(A&&A->ptr.hp)
    {
        if(A->ptr.hp->tag) GList_DelElem(A->ptr.hp,x);
        else if(!A->ptr.hp->tag&&A->ptr.hp->atom==x)
        {
            q=A;
            A=A->ptr.tp; //删去元素值为 x 的表头
            free(q);
            GList_DelElem(A,x);
        }
    }
}

```



```

    if(A&&A->ptr.tp) GList_DelElem(A->ptr.tp,x);
} //GList_DelElem

```

5.39

```

void GList_PrintElem_LOrder(GList A) // 按层序输出广义表 A 中的所有元素
{
    InitQueue(Q);
    for(p=L;p;p=p->ptr.tp) EnQueue(Q,p);
    while(!QueueEmpty(Q))
    {
        DeQueue(Q,r);
        if(!r->tag) printf("%d",r->atom);
        else
            for(r=r->ptr.hp;r;r=r->ptr.tp) EnQueue(Q,r);
    } //while
} //GList_PrintElem_LOrder

```

分析 :层序遍历的问题 ,一般都是借助队列来完成的 ,每次从队头取出一个元素的同时把它下

一层的孩子插入队尾 .这是层序遍历的基本思想 .

严蔚敏《数据结构习题集》解答

## 第六章 树和二叉树

6.33

第 56 页 共 124 页

```

int Is_Descendant_C(int u,int v) // 在孩子存储结构上判断 u 是否 v 的子孙 ,是则返回 1,
否则返回
0
{
    if(u==v) return 1;
    else
    {
        if(L[v])
            if (Is_Descendant(u,L[v])) return 1;
        if(R[v])
            if (Is_Descendant(u,R[v])) return 1; // 这是个递归算法
    }
    return 0;
} //Is_Descendant_C

```

6.34

```

int Is_Descendant_P(int u,int v) // 在双亲存储结构上判断 u 是否 v 的子孙 ,是则返回 1,否则返回
回
0
{
    for(p=u;p!=v&& p=T[p]);
}

```

```

        if(p==v) return 1;
        else return 0;
    }//Is_Descendant_P

```

6.35

这一题根本不需要写什么算法，见书后注释：两个整数的值是相等的。

6.36

```

int Bitree_Sim(Bitree B1, Bitree B2) // 判断两棵树是否相似的递归算法
{
    if(!B1 && !B2) return 1;
    else if(B1 && B2 && Bitree_Sim(B1->lchild, B2->lchild) && Bitree_Sim(B1->rchild, B2->rchild))
        return 1;
    else return 0;
} // Bitree_Sim

```

6.37

```

void PreOrder_Nonrecursive(Bitree T) // 先序遍历二叉树的非递归算法
{
    InitStack(S);
    Push(S, T); // 根指针进栈
    while(!StackEmpty(S))
    {
        while(Gettop(S, p) && p)
        {

```

严蔚敏《数据结构习题集》解答

```

            visit(p->data);
            push(S, p->lchild);
        } // 向左走到尽头
        pop(S, p);
        if(!StackEmpty(S))
        {
            pop(S, p);
            push(S, p->rchild); // 向右一步
        }
    } // while
} // PreOrder_Nonrecursive

```

6.38

```

typedef struct {
    BTNode* ptr;
    enum {0, 1, 2} mark;
} PMType; // 有 mark 域的结点指针类型

void PostOrder_Stack(BiTree T) // 后续遍历二叉树的非递归算法，用栈
{
    PMType a;
    InitStack(S); // S 的元素为 PMType 类型
    Push(S, {T, 0}); // 根结点入栈

```

```

while(!StackEmpty(S))
{
    Pop(S,a);
    switch(a.mark)
    {
        case 0:
            Push(S,{a.ptr,1}); // 修改 mark 域
            if(a.ptr->lchild) Push(S,{a.ptr->lchild,0}); // 访问左子树
            break;
        case 1:
            Push(S,{a.ptr,2}); // 修改 mark 域
            if(a.ptr->rchild) Push(S,{a.ptr->rchild,0}); // 访问右子树
            break;
        case 2:
            visit(a.ptr); // 访问结点 ,返回
    }
}
} //while
} //PostOrder_Stack

```

第 57 页 共 124 页

分析 :为了区分两次过栈的不同处理方式 ,在堆栈中增加一个 mark 域 ,mark=0 表示刚刚访问此

结点 ,mark=1 表示左子树处理结束返回 ,mark=2 表示右子树处理结束返回 .每次根据栈顶元素的 mark 域值决定做何种动作 .

6.39

typedef struct {

严蔚敏《数据结构习题集》解答

```

        int data;
        EBTNode *lchild;
        EBTNode *rchild;
        EBTNode *parent;
        enum {0,1,2} mark;

```

第 58 页 共 124 页

} EBTNode,EBitree; // 有 mark 域和双亲指针域的二叉树结

点类型

void PostOrder\_Nonrecursive(EBitree T) // 后序遍历二叉树的非递归算法 ,不用栈

```

{
    p=T;
    while(p)
        switch(p->mark)
        {
            case 0:
                p->mark=1;
                if(p->lchild) p=p->lchild; // 访问左子树
                break;

```

```

        case 1:
            p->mark=2;
            if(p->rchild) p=p->rchild; // 访问右子树
            break;
        case 2:
            visit(p);
            p->mark=0; //恢复 mark 值
            p=p->parent; //返回双亲结点
    }
} //PostOrder_Nonrecursive
分析:本题思路与上一题完全相同,只不过结点的 mark 值是储存在结点中的,而不是暂
存在堆
栈中,所以访问完毕后要将 mark 域恢复为 0,以备下一次遍历。

```

6.40

```

typedef struct {
    int data;
    PBTNode *lchild;
    PBTNode *rchild;
    PBTNode *parent;
} PBTNode, PBitree; // 有双亲指针域的二叉树结点类型

void Inorder_Nonrecursive(PBitree T) // 不设栈非递归遍历有双亲指针的二叉树
{
    p=T;
    while(p->lchild) p=p->lchild; // 向左走到尽头
    while(p)
    {
        visit(p);
        if(p->rchild) // 寻找中序后继:当有右子树时
        {

```

严蔚敏《数据结构习题集》解答

```

            p=p->rchild;
            while(p->lchild) p=p->lchild; // 后继就是在右子树中向左走到尽头
        }

```

第 59 页 共 124 页

```

        else if(p->parent->lchild==p) p=p->parent; // 当自己是双亲的左孩子时后继就是双
        亲
        else
        {
            p=p->parent;
            while(p->parent&& p->parent->rchild==p) p=p->parent;
            p=p->parent;
        } // 当自己是双亲的右孩子时后继就是向上返回直到遇到自己是在其左子树中的
        祖先
    } //while
}

```

```

} // Inorder_Nonrecursive
6.41
int c, k; // 这里把 k 和计数器 c 作为全局变量处理
void Get_PreSeq(Bitree T) // 求先序序列为 k 的结点的值
{
    if(T)
    {
        c++; // 每访问一个子树的根都会使前序序号计数器加 1
        if(c == k)
        {
            printf("Value is %d\n", T->data);
            exit(1);
        }
        else
        {
            Get_PreSeq(T->lchild); // 在左子树中查找
            Get_PreSeq(T->rchild); // 在右子树中查找
        }
    }
} // if
} // Get_PreSeq
main()
{
    ...
    scanf("%d", &k);
    c = 0; // 在主函数中调用前, 要给计数器赋初值 0
    Get_PreSeq(T, k);
    ...
} // main
6.42
int LeafCount_BiTree(Bitree T) // 求二叉树中叶子结点的数目
{
    if(!T) return 0; // 空树没有叶子
    else if(!T->lchild && !T->rchild) return 1; // 叶子结点

```

严蔚敏《数据结构习题集》解答

第 60 页 共 124 页

```

        else return Leaf_Count(T->lchild) + Leaf_Count(T->rchild); // 左子树的叶子数加上右子
树的叶
子数
} // LeafCount_BiTree
6.43
void Bitree_Revolute(Bitree T) // 交换所有结点的左右子树
{
    T->lchild <-> T->rchild; // 交换左右子树
    if(T->lchild) Bitree_Revolute(T->lchild);

```

```

        if(T->rchild) Bitree_Revolute(T->rchild); // 左右子树再分别交换各自的左右子树
    } // Bitree_Revolute

```

6.44

```

int Get_Sub_Depth(Bitree T, int x) // 求二叉树中以值为 x 的结点为根的子树深度
{
    if(T->data == x)
    {
        printf("%d\n", Get_Depth(T)); // 找到了值为 x 的结点, 求其深度
        exit 1;
    }
    else
    {
        if(T->lchild) Get_Sub_Depth(T->lchild, x);
        if(T->rchild) Get_Sub_Depth(T->rchild, x); // 在左右子树中继续寻找
    }
} // Get_Sub_Depth

```

```

int Get_Depth(Bitree T) // 求子树深度的递归算法
{
    if(!T) return 0;
    else
    {
        m = Get_Depth(T->lchild);
        n = Get_Depth(T->rchild);
        return (m > n ? m : n) + 1;
    }
} // Get_Depth

```

6.45

```

void Del_Sub_x(Bitree T, int x) // 删除所有以元素 x 为根的子树
{
    if(T->data == x) Del_Sub(T); // 删除该子树
    else
    {
        if(T->lchild) Del_Sub_x(T->lchild, x);
        if(T->rchild) Del_Sub_x(T->rchild, x); // 在左右子树中继续查找
    } // else
} // Del_Sub_x

```

严蔚敏《数据结构习题集》解答

```

void Del_Sub(Bitree T) // 删除子树 T
{
    if(T->lchild) Del_Sub(T->lchild);
    if(T->rchild) Del_Sub(T->rchild);
    free(T);
} // Del_Sub

```

6.46

```

void Bitree_Copy_Nonrecursive(Bitree T, Bitree &U) // 非递归复制二叉树

```

```

{
    InitStack(S1);InitStack(S2);
    push(S1,T); //根指针进栈
    U=(BTNode*)malloc(sizeof(BTNode));
    U->data=T->data;
    q=U;push(S2,U);
    while(!StackEmpty(S))
    {
        while(Gettop(S1,p)&&p)
        {
            q->lchild=(BTNode*)malloc(sizeof(BTNode));
            q=q->lchild;q->data=p->data;
            push(S1,p->lchild);
            push(S2,q);
        } // 向左走到尽头
        pop(S1,p);
        pop(S2,q);
        if(!StackEmpty(S1))
        {
            pop(S1,p);pop(S2,q);
            q->rchild=(BTNode*)malloc(sizeof(BTNode));
            q=q->rchild;q->data=p->data;
            push(S1,p->rchild); // 向右一步
            push(S2,q);
        }
    } //while
} //BiTree_Copy_Nonrecursive

```

分析 :本题的算法系从 6.37 改写而来 .

6.47

void LayerOrder(Bitree T) // 层序遍历二叉树

```

{
    InitQueue(Q); // 建立工作队列
    EnQueue(Q,T);
    while(!QueueEmpty(Q))
    {
        DeQueue(Q,p);

```

第 61 页 共 124 页

严蔚敏《数据结构习题集》解答

```

        visit(p);
        if(p->lchild) EnQueue(Q,p->lchild);
        if(p->rchild) EnQueue(Q,p->rchild);
    }
} //LayerOrder

```

6.48

```
int found=FALSE;
```

```
Bitree* Find_Near_Ancient(Bitree T, Bitree p, Bitree q) // 求二叉树 T 中结点 p 和 q 的最近共同祖
```

```
先
```

```
{
    Bitree pathp[ 100 ], pathq[ 100 ] // 设立两个辅助数组暂存从根到 p, q 的路径
    Findpath(T, p, pathp, 0);
    found=FALSE;
    Findpath(T, q, pathq, 0); // 求从根到 p, q 的路径放在 pathp 和 pathq 中
    for(i=0; pathp[i]==pathq[i] && pathp[i]; i++); // 查找两条路径上最后一个相同结点
    return pathp[--i];
}
```

```
//Find_Near_Ancient
```

```
void Findpath(Bitree T, Bitree p, Bitree path[ ], int i) // 求从 T 到 p 路径的递归算法
```

```
{
    if(T==p)
    {
        found=TRUE;
        return; // 找到
    }
    path[i]=T; // 当前结点存入路径
    if(T->lchild) Findpath(T->lchild, p, path, i+1); // 在左子树中继续寻找
    if(T->rchild && !found) Findpath(T->rchild, p, path, i+1); // 在右子树中继续寻找
    if(!found) path[i]=NULL; // 回溯
}
```

```
//Findpath
```

6.49

```
int IsFull_Bitree(Bitree T) // 判断二叉树是否完全二叉树 , 是则返回 1, 否则返回 0
```

```
{
    InitQueue(Q);
    flag=0;
    EnQueue(Q, T); // 建立工作队列
    while(!QueueEmpty(Q))
    {
        DeQueue(Q, p);
        if(!p) flag=1;
        else if(flag) return 0;
        else
        {
            EnQueue(Q, p->lchild);
            EnQueue(Q, p->rchild); // 不管孩子是否为空 , 都入队列
        }
    }
}
```

严蔚敏《数据结构习题集》解答

```
}
} // while
return 1;
```



```
}//IsFull_Bitree
```

第 63 页 共 124 页

分析 :该问题可以通过层序遍历的方法来解决 .与 6.47 相比 ,作了一个修改 ,不管当前结点是否

有左右孩子 ,都入队列 .这样当树为完全二叉树时 ,遍历时得到是一个连续的不包含空指针的序列 .反之 ,则序列中会含有空指针 .

6.50

Status CreateBitree\_Triplet(Bitree &T)// 输入三元组建立二叉树

```
{
    if(getchar()!='^') return ERROR;
    T=(BTNode*)malloc(sizeof(BTNode));
    p=T;p->data=getchar();
    getchar(); //滤去多余字符
    InitQueue(Q);
    EnQueue(Q,T);
    while((parent=getchar())!='^'&&(child=getchar())&&(side=getchar()))
    {
        while(QueueHead(Q)!=parent&&!QueueEmpty(Q)) DeQueue(Q,e);
        if(QueueEmpty(Q)) return ERROR; // 未按层序输入
        p=QueueHead(Q);
        q=(BTNode*)malloc(sizeof(BTNode));
        if(side=='L') p->lchild=q;
        else if(side=='R') p->rchild=q;
        else return ERROR; //格式不正确
        q->data=child;
        EnQueue(Q,q);
    }
    return OK;
}
} //CreateBitree_Triplet
```

6.51

Status Print\_Expression(Bitree T)// 按标准形式输出以二叉树存储的表达式

```
{
    if(T->data 是字母 ) printf("%c",T->data);
    else if(T->data 是操作符 )
    {
        if(!T->lchild||!T->rchild) return ERROR; // 格式错误
        if(T->lchild->data 是操作符 &&T->lchild->data 优先级低于 T->data)
        {
            printf("(");
            if(!Print_Expression(T->lchild)) return ERROR;
            printf(")");
        } // 注意在什么情况下要加括号
        else if(!Print_Expression(T->lchild)) return ERROR;
```

严蔚敏《数据结构习题集》解答

```

        if(T->rchild->data 是操作符 &&T->rchild->data 优先级低于 T->data)
        {
            printf("(");
            if(!Print_Expression(T->rchild)) return ERROR;
            printf(")");
        }
        else if(!Print_Expression(T->rchild)) return ERROR;
    }
    else return ERROR; //非法字符
    return OK;
} //Print_Expression

```

6.52

```

typedef struct{
    BTNode node;
    int layer;
} BTNRecord; // 包含结点所在层次的记录类型

int FanMao(Bitree T) // 求一棵二叉树的 "繁茂度"
{
    int countd; //count 数组存放每一层的结点数
    InitQueue(Q); //Q 的元素为 BTNRecord 类型
    EnQueue(Q,{T,0});
    while(!QueueEmpty(Q))
    {
        DeQueue(Q,r);
        count[r.layer]++;
        if(r.node->lchild) EnQueue(Q,{r.node->lchild,r.layer+1});
        if(r.node->rchild) EnQueue(Q,{r.node->rchild,r.layer+1});
    } // 利用层序遍历来统计各层的结点数
    h=r.layer; // 最后一个队列元素所在层就是树的高度
    for(maxn=count[0],i=1;count[i];i++)
        if(count[i]>maxn) maxn=count[i]; // 求层最大结点数
    return h*maxn;
} //FanMao

```

第 64 页 共 124 页

分析:如果不允许使用辅助数组,就必须在遍历的同时求出层最大结点数,形式上会复杂一些,你能写出来吗?

6.53

```

int maxh;
Status Printpath_MaxdepthS1(Bitree T) // 求深度等于树高度减一的最靠左的结点
{
    Bitree pathd;
    maxh=Get_Depth(T); //Get_Depth 函数见 6.44
    if(maxh<2) return ERROR; // 无符合条件结点
    Find_h(T,1);
    return OK;
}

```

```
//Printpath_MaxdepthS1
void Find_h(Bitree T,int h)// 寻找深度为 maxh-1 的结点
{
    path[h]=T;
    if(h==maxh-1)
    {
        for(i=1;path[i];i++) printf("%c",path[i]->data);
        exit; //打印输出路径
    }
    else
    {
        if(T->lchild) Find_h(T->lchild,h+1);
        if(T->rchild) Find_h(T->rchild,h+1);
    }
    path[h]=NULL; // 回溯
}
//Find_h
```

6.54

第 65 页 共 124 页

```
Status CreateBitree_SqList(Bitree &T,SqList sa)// 根据顺序存储结构建立二叉链表
{
    Bitree ptr[sa.last+1]; // 该数组储存与 sa 中各结点对应的树指针
    if(!sa.last)
    {
        T=NULL; // 空树
        return;
    }
    ptr[1]=(BTNode*)malloc(sizeof(BTNode));
    ptr[1]->data=sa.elem[1]; // 建立树根
    T=ptr[1];
    for(i=2;i<=sa.last;i++)
    {
        if(!sa.elem[i]) return ERROR; // 顺序错误
        ptr[i]=(BTNode*)malloc(sizeof(BTNode));
        ptr[i]->data=sa.elem[i];
        j=i/2; // 找到结点 i 的双亲 j
        if(i-j*2) ptr[j]->rchild=ptr[i]; //i 是 j 的右孩子
        else ptr[j]->lchild=ptr[i]; //i 是 j 的左孩子
    }
    return OK;
}
//CreateBitree_SqList
```

6.55

```
int DescNum(Bitree T)// 求树结点 T 的子孙总数填入 DescNum 域中,并返回该数
{
```

```

    if(!T) return -1;
    else d=(DescNum(T->lchild)+DescNum(T->rchild)+2); // 计算公式

```

严蔚敏《数据结构习题集》解答

```

    T->DescNum=d;
    return d;
} // DescNum

```

第 66 页 共 124 页

分析 :该算法时间复杂度为  $O(n)$ ,  $n$  为树结点总数 .注意 :为了能用一个统一的公式计算子孙数

目,所以当  $T$  为空指针时 ,要返回 -1 而不是 0.

6.56

BTNode \*PreOrder\_Next(BTNode \*p)// 在先序后继线索二叉树中查找结点  $p$  的先序后继,并返回指针

```

{
    if(p->lchild) return p->lchild;
    else return p->rchild;
} // PreOrder_Next

```

分析 :总觉得不会这么简单 .是不是哪儿理解错了 ?

6.57

Bitree PostOrder\_Next(Bitree p)// 在后序后继线索二叉树中查找结点  $p$  的后序后继 ,并返回指针

```

{
    if(p->rtag) return p->rchild; //p 有后继线索
    else if(!p->parent) return NULL; //p 是根结点
    else if(p==p->parent->rchild) return p->parent; //p 是右孩子
    else if(p==p->parent->lchild&& p->parent->rtag)
        return p->parent; //p 是左孩子且双亲没有右孩子
    else //p 是左孩子且双亲有右孩子
    {
        q=p->parent->rchild;
        while(q->lchild||!q->rtag)
        {
            if(q->lchild) q=q->lchild;
            else q=q->rchild;
        } // 从 p 的双亲的右孩子向下走到底
        return q;
    } //else
} // PostOrder_Next

```

6.58

Status Insert\_BiThrTree(BiThrTree &T,BiThrTree &p,BiThrTree &x)// 在中序线索二叉树  $T$  的结点  $p$  下插入子树  $x$

```

{
    if(p->ltag) //x 作为 p 的左子树
    {
        s=p->lchild; //s 为 p 的前驱
        p->ltag=Link;
        p->lchild=x;
        q=x;
        while(q->lchild&&!q->ltag) q=q->lchild;

        q->lchild=s; // 找到子树中的最左结点 ,并修改其前驱指向 s
        x->rtag=Thread;
        x->rchild=p; //x 的后继指向 p
    }
    else if(p->rtag) //x 作为 p 的右子树
    {
        s=p->rchild; //s 为 p 的后继
        p->rtag=Link;
        p->rchild=x;
        q=x;
        while(q->lchild&&!q->ltag) q=q->lchild;
        q->lchild=p; // 找到子树中的最左结点 ,并修改其前驱指向 p
        x->rtag=Thread;
        x->rchild=s; //x 的后继指向 p 的后继
    }
    else//x 作为 p 的左子树 ,p 的左子树作为 x 的右子树
    {
        s=p->lchild;t=s;
        while(t->lchild&&!t->ltag) t=t->lchild;
        u=t->lchild; // 找到 p 的左子树的最左结点 t 和前驱 u
        p->lchild=x;
        x->rtag=Link;
        x->rchild=s; //x 作为 p 的左子树 ,p 的左子树 s 作为 x 的右子树
        t->lchild=x;
        q=x;
        while(q->lchild&&!q->ltag) q=q->lchild;
        q->lchild=u; // 找到子树中的最左结点 ,并修改其前驱指向 u
    }
    return OK;
}
} //Insert_BiThrTree

```

6.59

```

void Print_CSTree(CSTree T)// 输出孩子兄弟链表表示的树 T 的各边
{
    for(child=T->firstchild;child;child=child->nextsib)
    {
        printf("(%c,%c)",T->data,child->data);
    }
}

```

```

        Print_CSTree(child);
    }
} //Print_CSTree

```

6.60

```

int LeafCount_CSTree(CSTree T) // 求孩子兄弟链表表示的树 T 的叶子数目
{
    if(!T->firstchild) return 1; // 叶子结点
    else

```

第 67 页 共 124 页

严蔚敏《数据结构习题集》解答

```

    {
        count=0;
        for(child=T->firstchild;child;child=child->nextsib)
            count+=LeafCount_CSTree(child);
        return count; // 各子树的叶子数之和
    }
} //LeafCount_CSTree

```

6.61

```

int GetDegree_CSTree(CSTree T) // 求孩子兄弟链表表示的树 T 的度
{
    if(!T->firstchild) return 0; // 空树
    else
    {
        degree=0;
        for(p=T->firstchild;p;p=p->nextsib) degree++; // 本结点的度
        for(p=T->firstchild;p;p=p->nextsib)
        {
            d=GetDegree_CSTree(p);
            if(d>degree) degree=d; // 孩子结点的度的最大值
        }
        return degree;
    } //else
} //GetDegree_CSTree

```

6.62

```

int GetDepth_CSTree(CSTree T) // 求孩子兄弟链表表示的树 T 的深度
{
    if(!T) return 0; // 空树
    else
    {
        for(maxd=0,p=T->firstchild;p;p=p->nextsib)
            if((d=GetDepth_CSTree(p))>maxd) maxd=d; // 子树的最大深度
        return maxd+1;
    }
} //GetDepth_CSTree

```

6.63

int GetDepth\_CTree(CTree A)// 求孩子链表表示的树 A 的深度

```
{
    return SubDepth(A.r);
} //GetDepth_CTree
int SubDepth(int T)// 求子树 T 的深度
{
    if(!A.nodes[T].firstchild) return 1;
    for(sd=1,p=A.nodes[T].firstchild;p;p=p->next)
        if((d=SubDepth(p->child))>sd) sd=d;
```

第 68 页 共 124 页

严蔚敏《数据结构习题集》解答

```
    return sd+1;
} //SubDepth
```

6.64

int GetDepth\_PTree(PTree T)// 求双亲表表示的树 T 的深度

```
{
    maxdep=0;
    for(i=0;i<T.n;i++)
    {
        dep=0;
        for(j=i;j>=0;j=T.nodes[j].parent) dep++; // 求每一个结点的深度
        if(dep>maxdep) maxdep=dep;
    }
    return maxdep;
} //GetDepth_PTree
```

6.65

char Pred,Ind; // 假设前序序列和中序序列已经分别储存在数组 Pre 和 In 中

第 69 页 共 124 页

Bitree Build\_Sub(int Pre\_Start,int Pre\_End,int In\_Start,int In\_End)// 由子树的前序和中序序列建立

立其二叉链表

```
{
    sroot=(BTNode*)malloc(sizeof(BTNode)); // 建根
    sroot->data=Pre[Pre_Start];
    for(i=In_Start;In[i]!=sroot->data;i++); // 在中序序列中查找子树根
    leftlen=i-In_Start;
    rightlen=In_End-i; // 计算左右子树的大小
    if(leftlen)
    {
        lroot=Build_Sub(Pre_Start+1,Pre_Start+leftlen,In_Start,In_Start+leftlen-1);
        sroot->lchild=lroot;
    } // 建左子树 ,注意参数表的计算
    if(rightlen)
```

```

{
    rroot=Build_Sub(Pre_End-rightlen+1,Pre_End,In_End-rightlen+1,In_End);
    sroot->rchild=rroot;
} // 建右子树 ,注意参数表的计算
return sroot; //返回子树根
} //Build_Sub
main()
{
    ...
    Build_Sub(1,n,1,n); // 初始调用参数 ,n 为树结点总数
    ...
}

```

分析:本算法利用了这样一个性质 ,即一棵子树在前序和中序序列中所占的位置总是连续的 .  
 因此 ,就可以用起始下标和终止下标来确定一棵子树 .Pre\_Start,Pre\_End,In\_Start 和 In\_End 分

严蔚敏《数据结构习题集》解答

第 70 页 共 124 页

别指示子树在前序子序列里的起始下标 ,终止下标 ,和在中序子序列里的起始和终止下标 .

6.66

```

typedef struct{
    CSNode *ptr;
    CSNode *lastchild;
} NodeMsg; // 结点的指针和其最后一个孩子的指针
Status Build_CSTree_PTree(PTree T)// 由树 T 的双亲表构造其孩子兄弟链表
{
    NodeMsg Tree[MAXSIZE];
    for(i=0;i<T.n;i++)
    {
        Tree[i].ptr=(CSNode*)malloc(sizeof(CSNode));
        Tree[i].ptr->data=T.node[i].data; // 建结点
        if(T.nodes[i].parent>=0) // 不是树根
        {
            j=T.nodes[i].parent; // 本算法要求双亲表必须是按层序存储
            if(!(Tree[j].lastchild)) // 双亲当前还没有孩子
                Tree[j].ptr->firstchild=Tree[i].ptr; // 成为双亲的第一个孩子
            else //双亲已经有了孩子
                Tree[j].lastchild->nextsib=Tree[i].ptr; // 成为双亲最后一个孩子的下一个兄弟
            Tree[j].lastchild=Tree[i].ptr; // 成为双亲的最后一个孩子
        } //if
    } //for
} //Build_CSTree_PTree

```

6.67

```

typedef struct{

```



```

        char data;
        CSNode *ptr;
        CSNode *lastchild;
    } NodeInfo; // 结点数据 ,结点指针和最后一个孩子的指针
Status CreateCSTree_Duplet(CSTree &T)// 输入二元组建立树的孩子兄弟链表
{
    NodeInfo Treed;
    n=1;k=0;
    if(getchar()!='^') return ERROR; // 未按格式输入
    if((c=getchar())=='^') T=NULL; // 空树
    Tree[0].ptr=(CSNode*)malloc(sizeof(CSNode));
    Tree[0].data=c;
    Tree[0].ptr->data=c;
    while((p=getchar())!='^'&&(c=getchar())!='^')
    {
        Tree[n].ptr=(CSNode*)malloc(sizeof(CSNode));
        Tree[n].data=c;
        Tree[n].ptr->data=c;
        严蔚敏《数据结构习题集》解答
    }

```

```

        for(k=0;Tree[k].data!=p;k++); // 查找当前边的双亲结点
        if(Tree[k].data!=p) return ERROR; // 未找到 :未按层序输入
        r=Tree[k].ptr;
        if(!r->firstchild)
            r->firstchild=Tree[n].ptr;
        else Tree[k].lastchild->nextsib=Tree[n].ptr;
        Tree[k].lastchild=Tree[n].ptr; // 这一段含义同上一题
        n++;
    }//while
    return OK;
} //CreateCSTree_Duplet
6.68

```

第 71 页 共 124 页

```

Status CreateCSTree_Degree(char node[ ],int degree[ ])//由结点的层序序列和各结点的
度构造
树的孩子兄弟链表
{
    CSNode * ptr[MAXSIZE]; // 树结点指针的辅助存储
    ptr[0]=(CSNode*)malloc(sizeof(CSNode));
    i=0;k=1; //i 为当前结点序号 ,k 为当前孩子的序号
    while(node[i])
    {
        ptr[i]->data=node[i];
        d=degree[i];
        if(d)
    }

```

```

{
    ptr[k]=(CSNode*)malloc(sizeof(CSNode)); //k    为当前孩子的序号
    ptr[i]->firstchild=ptr[k]; //    建立    i    与第一个孩子    k    之间的联系
    for(j=2;j<=d;j++)
    {
        ptr[k]=(CSNode*)malloc(sizeof(CSNode));
        ptr[k-1]->nextsib=ptr[k]; //    当结点的度大于    1    时,为其孩子建立兄弟链表
        k++;
    }//for
    ptr[k-1]->nextsib=NULL;
} //if
i++;
} //while
} //CreateCSTree_Degree

```

```
void Print_BiTree(BiTree T,int i)//    按树状打印输出二叉树的元素    ,i 表示结点所在层次    ,初次调用时 i=0
```

```
if(T->rchild) Print_BiTree(T->rchild,i+1);
for(j=1;j<=i;j++) printf(" "); // 打印 i 个空格以表示出层次
printf("%c\n",T->data); // 打印 T 元素,换行
```

```
    if(T->lchild) Print_BiTree(T->rchild,i+1);
} //Print_BiTree
```

分析 :该递归算法实际上是带层次信息的中序遍历 ,只不过按照题目要求 ,顺序为先右后左.

```
Status CreateBiTree_GLlist(BiTree &T)// 由广义表形式的输入建立二叉链表
```

```
c=getchar();
if(c=='#') T=NULL; // 空子树
else
{
    T=(CSNode*)malloc(sizeof(CSNode));
    T->data=c;
    if(getchar()!='(') return ERROR;
    if(!CreateBiTree_GList(pl)) return ERROR;
    T->lchild=pl;
    if(getchar()!='(',')') return ERROR;
    if(!CreateBiTree_GList(pr)) return ERROR;
    T->rchild=pr;
    if(getchar()!=')') return ERROR; // 这些语句是为了保证输入符合    A(B,C) 的格式
}
```

```

        return OK;
    } //CreateBiTree_GList
6.71
void Print_CSTree(CSTree T,int i) // 按凹入表形式打印输出树的元素 ,i 表示结点所在层次,初次
调用时 i=0
{
    for(j=1;j<=i;j++) printf(" "); // 留出 i 个空格以表现出层次
    printf("%c\n",T->data); // 打印元素 ,换行
    for(p=T->firstchild;p;p=p->nextsib)
        Print_CSTree(p,i+1); // 打印子树
} //Print_CSTree
6.72
void Print_CTree(int e,int i) // 按凹入表形式打印输出树的元素 ,i 表示结点所在层次
{
    for(j=1;j<=i;j++) printf(" "); // 留出 i 个空格以表现出层次
    printf("%c\n",T.nodes[e].data); // 打印元素 ,换行
    for(p=T.nodes[e].firstchild;p;p=p->next)
        Print_CSTree(p->child,i+1); // 打印子树
} //Print_CSTree
main()
{
    ...
    Print_CTree(T.r,0); // 初次调用时 i=0
    ...
}
严蔚敏《数据结构习题集》解答

} //main
6.73
char c; //全局变量 ,指示当前字符

```

第 73 页 共 124 页

```

Status CreateCSTree_GList(CSTree &T) // 由广义表形式的输入建立孩子兄弟链表
{
    c=getchar();
    T=(CSNode*)malloc(sizeof(CSNode));
    T->data=c;
    if((c=getchar())=='(') // 非叶结点
    {
        if(!CreateCSTree_GList(fc)) return ERROR; // 建第一个孩子
        T->firstchild=fc;
        for(p=fc;c==',';p->nextsib=nc,p=nc) // 建兄弟链
            if(!CreateCSTree_GList(nc)) return ERROR;
        p->nextsib=NULL;
        if((c=getchar())!=')') return ERROR; // 括号不配对
    }
}

```

```

else T->firstchild=NULL; // 叶子结点
return OK;

```

```

} // CreateBiTree_GList

```

分析: 书后给出了两个间接递归的算法, 事实上合成一个算法在形式上可能更好一些. 本算法

另一个改进之处在于加入了广义表格式是否合法的判断.

6.74

```

void PrintGlist_CSTree(CSTree T) // 按广义表形式输出孩子兄弟链表表示的树

```

```

{
    printf("%c", T->data);
    if(T->firstchild) // 非叶结点
    {
        printf("(");
        for(p=T->firstchild; p; p=p->nextsib)
        {
            PrintGlist_CSTree(p);
            if(p->nextsib) printf(","); // 最后一个孩子后面不需要加逗号
        }
        printf(")");
    } // if
} // PrintGlist_CSTree

```

6.75

```

char c;

```

```

int pos=0; // pos 是全局变量, 指示已经分配到了哪个结点

```

```

Status CreateCTree_GList(CTree &T, int &i) // 由广义表形式的输入建立孩子链表

```

```

{
    c=getchar();
    T.nodes[pos].data=c;

```

严蔚敏《数据结构习题集》解答

```

i=pos++; // i 是局部变量, 指示当前正在处理的子树根

```

```

if((c=getchar())=='(') // 非叶结点

```

```

{
    CreateCTree_GList();
    p=(CTBox*)malloc(sizeof(CTBox));
    T.nodes[i].firstchild=p;
    p->child=pos; // 建立孩子链的头
    for(; c==','; p=p->next) // 建立孩子链
    {
        CreateCTree_GList(T, j); // 用 j 返回分配得到的子树根位置
        p->child=j;
        p->next=(CTBox*)malloc(sizeof(CTBox));
    }
    p->next=NULL;
    if((c=getchar())!=')') return ERROR; // 括号不配对
} // if

```

```

        else T.nodes[i].firstchild=NULL; // 叶子结点
    return OK;
} // CreateBiTree_GList

```

第 74 页 共 124 页

分析 :该算法中 ,pos 变量起着 "分配 "结点在表中的位置的作用 ,是按先序序列从上向下分配 ,因此树根 T.r 一定等于 0,而最终的 pos 值就是结点数 T.n.

6.76

```

void PrintGList_CTree(CTree T,int i) // 按广义表形式输出孩子链表表示的树
{
    printf("%c",T.nodes[i].data);
    if(T.nodes[i].firstchild) // 非叶结点
    {
        printf("(");
        for(p=T->firstchild;p;p=p->nextsib)
        {
            PrintGlist_CSTree(T,p->child);
            if(p->nextsib) printf(","); // 最后一个孩子后面不需要加逗号
        }
        printf(")");
    } // if
} // PrintGlist_CTree

```

严蔚敏《数据结构习题集》解答

## 第七章 图

7.14

第 75 页 共 124 页

```

Status Build_AdjList(ALGraph &G) // 输入有向图的顶点数 ,边数 ,顶点信息和边的信息建立邻接表
{
    InitALGraph(G);
    scanf("%d",&v);
    if(v<0) return ERROR; // 顶点数不能为负
    G.vexnum=v;
    scanf("%d",&a);
    if(a<0) return ERROR; // 边数不能为负
    G.arcnum=a;
    for(m=0;m<v;m++)
        G.vertices[m].data=getchar(); // 输入各顶点的符号
    for(m=1;m<=a;m++)
    {

```

```

t=getchar();h=getchar(); //t 为弧尾 ,h 为弧头
if((i=LocateVex(G,t))<0) return ERROR;
if((j=LocateVex(G,h))<0) return ERROR; // 顶点未找到
p=(ArcNode*)malloc(sizeof(ArcNode));
if(!G.vertices[i].firstarc) G.vertices[i].firstarc=p;
else
{
    for(q=G.vertices[i].firstarc;q->nextarc;q=q->nextarc);
    q->nextarc=p;
}
p->adjvex=j;p->nextarc=NULL;
} //while
return OK;
} //Build_AdjList

```

7.15

//本题中的图 G 均为有向无权图 ,其余情况容易由此写出

Status Insert\_Vex(MGraph &G, char v) // 在邻接矩阵表示的图 G 上插入顶点 v

```

{
    if(G.vexnum+1)>MAX_VERTEX_NUM return INFEASIBLE;
    G.vexs[++G.vexnum]=v;
    return OK;
} //Insert_Vex

```

Status Insert\_Arc(MGraph &G,char v,char w) // 在邻接矩阵表示的图 G 上插入边 (v,w)

```

{
    if((i=LocateVex(G,v))<0) return ERROR;
    if((j=LocateVex(G,w))<0) return ERROR;

```

严蔚敏《数据结构习题集》解答

```

    if(i==j) return ERROR;
    if(!G.arcs[i][j].adj)
    {
        G.arcs[i][j].adj=1;
        G.arcnum++;
    }
    return OK;
} //Insert_Arc

```

第 76 页 共 124 页

Status Delete\_Vex(MGraph &G,char v) // 在邻接矩阵表示的图 G 上删除顶点 v

```

{
    n=G.vexnum;
    if((m=LocateVex(G,v))<0) return ERROR;
    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++)
    {
        G.arcs[i][m]=G.arcs[i][n];

```

```

        G.arcs[m][i]=G.arcs[n][i]; // 将边的关系随之交换
    }
    G.arcs[m][m].adj=0;
    G.vexnum--;
    return OK;
} //Delete_Vex
分析 :如果不把待删除顶点交换到最后一个顶点的话 ,算法将会比较复杂 ,而伴随着大量
元素
的移动 ,时间复杂度也会大大增加 .
Status Delete_Arc(MGraph &G,char v,char w)// 在邻接矩阵表示的图 G 上删除边 (v,w)
{
    if((i=LocateVex(G,v))<0) return ERROR;
    if((j=LocateVex(G,w))<0) return ERROR;
    if(G.arcs[i][j].adj)
    {
        G.arcs[i][j].adj=0;
        G.arcnum--;
    }
    return OK;
} //Delete_Arc

```

7.16

//为节省篇幅 ,本题只给出 Insert\_Arc 算法 .其余算法请自行写出 .

```

Status Insert_Arc(ALGraph &G,char v,char w)// 在邻接表表示的图 G 上插入边 (v,w)
{
    if((i=LocateVex(G,v))<0) return ERROR;
    if((j=LocateVex(G,w))<0) return ERROR;
    p=(ArcNode*)malloc(sizeof(ArcNode));
    p->adjvex=j;p->nextarc=NULL;
    if(!G.vertices[i].firstarc) G.vertices[i].firstarc=p;

```

严蔚敏《数据结构习题集》解答

```

    else
    {
        for(q=G.vertices[i].firstarc;q->q->nextarc;q=q->nextarc)
            if(q->adjvex==j) return ERROR; // 边已经存在
        q->nextarc=p;
    }
    G.arcnum++;
    return OK;
} //Insert_Arc

```

7.17

第 77 页 共 124 页

//为节省篇幅 ,本题只给出较为复杂的 Delete\_Vex 算法 .其余算法请自行写出 .

```

Status Delete_Vex(OLGraph &G,char v)// 在十字链表表示的图 G 上删除顶点 v
{

```

```

if((m=LocateVex(G,v))<0) return ERROR;
n=G.vexnum;
for(i=0;i<n;i++) // 删除所有以 v 为头的边
{
    if(G.xlist[i].firstin->tailvex==m) // 如果待删除的边是头链上的第一个结点
    {
        q=G.xlist[i].firstin;
        G.xlist[i].firstin=q->hlink;
        free(q);G.arcnum--;
    }
    else //否则
    {
        for(p=G.xlist[i].firstin;p&& p->hlink->tailvex!=m;p=p->hlink);
        if(p)
        {
            q=p->hlink;
            p->hlink=q->hlink;
            free(q);G.arcnum--;
        }
    }
}
}
//for
for(i=0;i<n;i++) // 删除所有以 v 为尾的边
{
    if(G.xlist[i].firstout->headvex==m) // 如果待删除的边是尾链上的第一个结点
    {
        q=G.xlist[i].firstout;
        G.xlist[i].firstout=q->tlink;
        free(q);G.arcnum--;
    }
    else //否则
    {

```

严蔚敏《数据结构习题集》解答

```

        for(p=G.xlist[i].firstout;p&& p->tlink->headvex!=m;p=p->tlink);
        if(p)
        {
            q=p->tlink;
            p->tlink=q->tlink;
            free(q);G.arcnum--;
        }
    }
}
}
//for
for(i=m;i<n;i++) // 顺次用结点 m 之后的顶点取代前一个顶点
{
    G.xlist[i]=G.xlist[i+1]; // 修改表头向量
    for(p=G.xlist[i].firstin;p;p=p->hlink)

```



```

        p->headvex--;
    for(p=G.xlist[i].firstout;p;p=p->tlink)
        p->tailvex--; // 修改各链中的顶点序号
    }
    G.vexnum--;
    return OK;
} //Delete_Vex

```

7.18

//为节省篇幅 ,本题只给出 Delete\_Arc 算法 .其余算法请自行写出 .

第 78 页 共 124 页

Status Delete\_Arc(AMLGraph &G,char v,char w)//// 在邻接多重表表示的图 G 上删除边 (v,w)

```

{
    if((i=LocateVex(G,v))<0) return ERROR;
    if((j=LocateVex(G,w))<0) return ERROR;
    if(G.adjmulist[i].firstedge->jvex==j)
        G.adjmulist[i].firstedge=G.adjmulist[i].firstedge->ilink;
    else
    {
        for(p=G.adjmulist[i].firstedge;p&& p->ilink->jvex!=j;p=p->ilink);
        if (!p) return ERROR; // 未找到
        p->ilink=p->ilink->ilink;
    } // 在 i 链表中删除该边
    if(G.adjmulist[j].firstedge->ivex==i)
        G.adjmulist[j].firstedge=G.adjmulist[j].firstedge->jlink;
    else
    {
        for(p=G.adjmulist[j].firstedge;p&& p->jlink->ivex!=i;p=p->jlink);
        if (!p) return ERROR; // 未找到
        q=p->jlink;
        p->jlink=q->jlink;
        free(q);
    } // 在 j 链表中删除该边
}

```

严蔚敏《数据结构习题集》解答

```

    G.arcnum--;
    return OK;
} //Delete_Arc

```

7.19

第 79 页 共 124 页

Status Build\_AdjMulist(AMLGraph &G)// 输入有向图的顶点数 ,边数 ,顶点信息和边的信息建

立邻接多重表

```

{
    InitAMLGraph(G);
}

```

```

scanf("%d",&v);
if(v<0) return ERROR; // 顶点数不能为负
G.vexnum=v;
scanf("%d",&a);
if(a<0) return ERROR; // 边数不能为负
G.arcnum=a;
for(m=0;m<v;m++)
    G.adjmulist[m].data=getchar(); // 输入各顶点的符号
for(m=1;m<=a;m++)
{
    t=getchar();h=getchar(); //t 为弧尾 ,h 为弧头
    if((i=LocateVex(G,t))<0) return ERROR;
    if((j=LocateVex(G,h))<0) return ERROR; // 顶点未找到
    p=(EBox*)malloc(sizeof(EBox));
    p->ivex=i;p->jvex=j;
    p->ilink=NULL;p->jlink=NULL; // 边结点赋初值
    if(!G.adjmulist[i].firstedge) G.adjmulist[i].firstedge=p;
    else
    {
        q=G.adjmulist[i].firstedge;
        while(q)
        {
            r=q;
            if(q->ivex==i) q=q->ilink;
            else q=q->jlink;
        }
        if(r->ivex==i) r->ilink=p; // 注意 i 值既可能出现在边结点的 ivex 域中 ,
        else r->jlink=p; // 又可能出现在边结点的 jvex 域中
    } //else // 插入 i 链表尾部
    if(!G.adjmulist[j].firstedge) G.adjmulist[j].firstedge=p;
    else
    {
        q=G.adjmulist[j].firstedge;
        while(q)
        {
            r=q;
            if(q->jvex==j) q=q->jlink;
            else q=q->ilink;
        }
        if(r->jvex==j) r->jlink=p;
        else r->ilink=p;
    } //else // 插入 j 链表尾部
} //for
return OK;

```

严蔚敏《数据结构习题集》解答

```
}//Build_AdjList
```

7.20

第 80 页 共 124 页

```
int Pass_MGraph(MGraph G)// 判断一个邻接矩阵存储的有向图是不是可传递的,是则返回 1,
回
否则返回 0
{
    for(x=0;x<G.vexnum;x++)
        for(y=0;y<G.vexnum;y++)
            if(G.arcs[x][y])
            {
                for(z=0;z<G.vexnum;z++)
                    if(z!=x&&G.arcs[y][z]&&!G.arcs[x][z]) return 0;// 图不可传递的条件
            }//if
    return 1;
}
}Pass_MGraph
```

分析:本算法的时间复杂度大概是  $O(n^2*d)$ .

7.21

```
int Pass_ALGraph(ALGraph G)// 判断一个邻接表存储的有向图是不是可传递的,是则返回 1,
否则返回 0
{
    for(x=0;x<G.vexnum;x++)
        for(p=G.vertices[x].firstarc;p;p=p->nextarc)
        {
            y=p->adjvex;
            for(q=G.vertices[y].firstarc;q;q=q->nextarc)
            {
                z=q->adjvex;
                if(z!=x&&!is_adj(G,x,z)) return 0;
            }//for
        }//for
}
}Pass_ALGraph
```

```
int is_adj(ALGraph G,int m,int n)// 判断有向图 G 中是否存在边 (m,n),是则返回 1,否则
返回 0
```

```
{
    for(p=G.vertices[m].firstarc;p;p=p->nextarc)
        if(p->adjvex==n) return 1;
    return 0;
}
}is_adj
```

严蔚敏《数据结构习题集》解答

7.22

```
int visited[MAXSIZE]; // 指示顶点是否在当前路径上
```

第 81 页 共 124 页

```
int exist_path_DFS(ALGraph G,int i,int j)// 深度优先判断有向图 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; //i 就是 j
    else
    {
        visited[i]=1;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            k=p->adjvex;
            if(!visited[k]&&exist_path(k,j)) return 1; //i 下游的顶点到 j 有路径
        }
    }
} //exist_path_DFS
```

### 7.23

int exist\_path\_BFS(ALGraph G,int i,int j)// 广度优先判断有向图 G 中顶点 i 到顶点 j 是否有  
路径,  
是则返回 1,否则返回 0

```
{
    int visited[MAXSIZE];
    InitQueue(Q);
    EnQueue(Q,i);
    while(!QueueEmpty(Q))
    {
        DeQueue(Q,u);
        visited[u]=1;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            k=p->adjvex;
            if(k==j) return 1;
            if(!visited[k]) EnQueue(Q,k);
        }
    }
    return 0;
} //exist_path_BFS
```

### 7.24

void STTraverse\_Nonrecursive(Graph G)// 非递归遍历强连通图 G

```
{
    int visited[MAXSIZE];
    InitStack(S);
    Push(S,GetVex(S,1)); //将第一个顶点入栈
    visit(1);
    visited =1;
```

严蔚敏《数据结构习题集》解答

```
while(!StackEmpty(S))
```

```

{
    while(Gettop(S,i)&& i)
    {
        j=FirstAdjVex(G,i);
        if(j&&!visited[j])
        {
            visit(j);
            visited[j]=1;
            Push(S,j); //向左走到尽头
        }
    } //while
    if(!StackEmpty(S))
    {
        Pop(S,j);
        Gettop(S,i);
        k=NextAdjVex(G,i,j); // 向右走一步
        if(k&&!visited[k])
        {
            visit(k);
            visited[k]=1;
            Push(S,k);
        }
    } //if
} //while
} //Straverse_Nonrecursive

```

第 82 页 共 124 页

分析 :本算法的基本思想与二叉树的先序遍历非递归算法相同 ,请参考 6.37.由于是强连通图 ,  
 所以从第一个结点出发一定能够访问到所有结点 .

7.25

见书后解答 .

7.26

Status TopoNo(ALGraph G)// 按照题目要求顺序重排有向图中的顶点

```

{
    int new[MAXSIZE],indegree[MAXSIZE]; // 储存结点的新序号
    n=G.vexnum;
    FindInDegree(G,indegree);
    InitStack(S);
    for(i=1;i<G.vexnum;i++)
        if(!indegree[i]) Push(S,i); // 零入度结点入栈
    count=0;
    while(!StackEmpty(S))
    {
        Pop(S,i);
        new[i]=n--; // 记录结点的拓扑逆序序号
    }
}

```

```

        count++;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            k=p->adjvex;
            if(!(--indegree[k])) Push(S,k);
        }//for
    }//while
    if(count<G.vexnum) return ERROR; // 图中存在环
    for(i=1;i<=n;i++) printf("Old No:%d New No:%d\n",i,new[i])
    return OK;
} //TopoNo

```

分析:只要按拓扑逆序对顶点编号,就可以使邻接矩阵成为下三角矩阵.

7.27

```
int visited[MAXSIZE];
```

第 83 页 共 124 页

int exist\_path\_len(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]=1;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            l=p->adjvex;
            if(!visited[l])
                if(exist_path_len(G,l,j,k-1)) return 1; // 剩余路径长度减一
        }//for
        visited[i]=0; // 本题允许曾经被访问过的结点出现在另一条路径中
    }//else
    return 0; //没找到
} //exist_path_len

```

7.28

```

int path[MAXSIZE],visited[MAXSIZE]; // 暂存遍历过程中的路径
int Find_All_Path(ALGraph G,int u,int v,int k)// 求有向图 G 中顶点 u 到 v 之间的所有简单路径,k
表示当前路径长度
{
    path[k]=u; // 加入当前路径中
    visited[u]=1;
    if(u==v) // 找到了一条简单路径
    {

```

```

        printf("Found one path!\n");
        for(i=0;path[i];i++) printf("%d",path[i]); // 打印输出
    }
    else

```

严蔚敏《数据结构习题集》解答

```

        for(p=G.vertices[u].firstarc;p;p=p->nextarc)
        {
            l=p->adjvex;
            if(!visited[l]) Find_All_Path(G,l,v,k+1); // 继续寻找
        }
        visited[u]=0;
        path[k]=0; // 回溯
    }//Find_All_Path
main()
{
    ...
    Find_All_Path(G,u,v,0); // 在主函数中初次调用 ,k 值应为 0
    ...
} //main

```

7.29

第 84 页 共 124 页

int GetPathNum\_Len(ALGraph G,int i,int j,int len)// 求邻接表方式存储的有向图 G 的顶点 i 到 j 之间长度为 len 的简单路径条数

```

{
    if(i==j&&len==0) return 1; // 找到了一条路径 ,且长度符合要求
    else if(len>0)
    {
        sum=0; //sum 表示通过本结点的路径数
        visited[i]=1;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            l=p->adjvex;
            if(!visited[l])
                sum+=GetPathNum_Len(G,l,j,len-1)// 剩余路径长度减一
        }//for
        visited[i]=0; // 本题允许曾经被访问过的结点出现在另一条路径中
    }//else
    return sum;
} //GetPathNum_Len

```

7.30

```

int visited[MAXSIZE];
int path[MAXSIZE]; // 暂存当前路径
int cycles[MAXSIZE][MAXSIZE]; // 储存发现的回路所包含的结点

```

```

int thiscycle[MAXSIZE]; // 储存当前发现的一个回路
int cycount=0; // 已发现的回路个数
void GetAllCycle(ALGraph G)// 求有向图中所有的简单回路
{
    for(v=0;v<G.vexnum;v++) visited[v]=0;
    for(v=0;v<G.vexnum;v++)
        if(!visited[v]) DFS(G,v,0); // 深度优先遍历

```

严蔚敏《数据结构习题集》解答

```

} //DFSTraverse
void DFS(ALGraph G,int v,int k)//k 表示当前结点在路径上的序号
{
    visited[v]=1;
    path[k]=v; // 记录当前路径
    for(p=G.vertices[v].firstarc;p;p=p->nextarc)
    {
        w=p->adjvex;
        if(!visited[w]) DFS(G,w,k+1);
        else //发现了一条回路
        {
            for(i=0;path[i]!=w;i++); // 找到回路的起点
            for(j=0;path[i+j];i++) thiscycle[j]=path[i+j]; // 把回路复制下来

```

第 85 页 共 124 页

```

            if(!exist_cycle()) cycles[cycount++]=thiscycle; // 如果该回路尚未被记录过 ,就添加到记录中

```

```

            for(i=0;i<G.vexnum;i++) thiscycle[i]=0; // 清空目前回路数组
        } //else
    } //for

```

```

    path[k]=0;
    visited[k]=0; // 注意只有当前路径上的结点 visited 为真 .因此一旦遍历中发现当前结点 visited 为真 ,即表示发现了一条回路
} //DFS

```

```

int exist_cycle()// 判断 thiscycle 数组中记录的回路在 cycles 的记录中是否已经存在
{

```

```

    int temp[MAXSIZE];
    for(i=0;i<cycount;i++) // 判断已有的回路与 thiscycle 是否相同
    { // 也就是 ,所有结点和它们的顺序都相同
        j=0;c=thiscycle; // 例如 ,142857 和 857142 是相同的回路
        for(k=0;cycles[i][k]!=c&&cycles[i][k]!=0;k++); // 在 cycles 的一个行向量中寻找等于 thiscycle

```

```

        第一个结点的元素
        if(cycles[i][k]) // 有与之相同的一个元素
        {
            for(m=0;cycles[i][k+m];m++)

```



```

        temp[m]=cycles[i][k+m];
    for(n=0;n<k;n++,m++)
        temp[m]=cycles[i][n]; // 调整 cycles 中的当前记录的循环相位并放入 temp
数组中
    if(!StrCompare(temp,thiscycle)) // 与 thiscycle 比较
        return 1; //完全相等
    for(m=0;m<G.vexnum;m++) temp[m]=0; // 清空这个数组
}
} //for
return 0; //所有现存回路都不与 thiscycle 完全相等
} //exist_cycle

```

分析:这个算法的思想是 ,在遍历中暂存当前路径 ,当遇到一个结点已经在路径之中时就表明存在一条回路 ;扫描路径向量 path 可以获得这条回路上的所有结点 .把结点序列 (例如,142857)

严蔚敏《数据结构习题集》解答

第 86 页 共 124 页

存入 thiscycle 中 ;由于这种算法中 ,一条回路会被发现好几次 ,所以必须先判断该回路是否已

经在 cycles 中被记录过 ,如果没有才能存入 cycles 的一个行向量中 .把 cycles 的每一个行向量

取出来与之比较 .由于一条回路可能有多种存储顺序 ,比如 142857 等同于 285714 和 571428,

所以还要调整行向量的次序 ,并存入 temp 数组 ,例如 ,thiscycle 为 142857 第一个结点为 1,cycles

的当前向量为 857142,则找到后者中的 1,把 1 后部分提到 1 前部分前面 ,最终在 temp 中得到

142857,与 thiscycle 比较 ,发现相同 ,因此 142857 和 857142 是同一条回路 ,不予存储 .这个算法

太复杂 ,很难保证细节的准确性 ,大家理解思路便可 .希望有人给出更加简捷的算法 .

7.31

```

int visited[MAXSIZE];
int finished[MAXSIZE];
int count; //count 在第一次深度优先遍历中用于指示 finished 数组的填充位置
void Get_SGraph(OLGraph G) // 求十字链表结构储存的有向图 G 的强连通分量
{
    count=0;
    for(v=0;v<G.vexnum;v++) visited[v]=0;
    for(v=0;v<G.vexnum;v++) // 第一次深度优先遍历建立 finished 数组
        if(!visited[v]) DFS1(G,v);
    for(v=0;v<G.vexnum;v++) visited[v]=0; // 清空 visited 数组
    for(i=G.vexnum-1;i>=0;i--) // 第二次逆向的深度优先遍历
    {
        v=finished(i);
        if(!visited[v])

```

```

        {
            printf("\n"); // 不同的强连通分量在不同的行输出
            DFS2(G,v);
        }
    }//for
} //Get_SGraph
void DFS1(OLGraph G,int v)// 第一次深度优先遍历的算法
{
    visited[v]=1;
    for(p=G.xlist[v].firstout;p;p=p->tlink)
    {
        w=p->headvex;
        if(!visited[w]) DFS1(G,w);
    }//for
    finished[++count]=v; // 在第一次遍历中建立 finished 数组
} //DFS1
void DFS2(OLGraph G,int v)// 第二次逆向的深度优先遍历的算法
{
    visited[v]=1;
    printf("%d",v); // 在第二次遍历中输出结点序号
    for(p=G.xlist[v].firstin;p;p=p->hlink)
    {
        w=p->tailvex;
        if(!visited[w]) DFS2(G,w);
    }//for
} //DFS2

```

严蔚敏《数据结构习题集》解答

第 87 页 共 124 页

分析 :求有向图的强连通分量的算法的时间复杂度和深度优先遍历相同 ,也为  $O(n+e)$ .

7.32

void Forest\_Prim(ALGraph G,int k,CSTree &T)// 从顶点 k 出发 ,构造邻接表结构的有向图 G 的最小生成森林 T,用孩子兄弟链表存储

```

{
    for(j=0;j<G.vexnum;j++) // 以下在 Prim 算法基础上稍作改动
        if(j!=k)
        {
            closedge[j]={k,Max_int};
            for(p=G.vertices[j].firstarc;p;p=p->nextarc)
                if(p->adjvex==k) closedge[j].lowcost=p->cost;
        }//if
    closedge[k].lowcost=0;
    for(i=1;i<G.vexnum;i++)
    {

```

```

        k=minimum(closedge);
        if(closedge[k].lowcost<Max_int)
        {
            Addto_Forest(T,closedge[k].adjvex,k); // 把这条边加入生成森林中
            closedge[k].lowcost=0;
            for(p=G.vertices[k].firstarc;p;p=p->nextarc)
                if(p->cost<closedge[p->adjvex].lowcost)
                    closedge[p->adjvex]={k,p->cost};
        }//if
        else Forest_Prim(G,k); //对另外一个连通分量执行算法
    }//for
} //Forest_Prim

void Addto_Forest(CSTree &T,int i,int j)// 把边 (i,j) 添加到孩子兄弟链表表示的树 T 中
{
    p=Locate(T,i); // 找到结点 i 对应的指针 p,过程略
    q=(CSTNode*)malloc(sizeof(CSTNode));
    q->data=j;
    if(!p) // 起始顶点不属于森林中已有的任何一棵树
    {
        p=(CSTNode*)malloc(sizeof(CSTNode));
        p->data=i;
        for(r=T;r->nextsib;r=r->nextsib);
        r->nextsib=p;
        p->firstchild=q;
    } // 作为新树插入到最右侧
}

```

严蔚敏《数据结构习题集》解答

```

        else if(!p->firstchild) // 双亲还没有孩子
            p->firstchild=q; // 作为双亲的第一个孩子
        else //双亲已经有了孩子
        {
            for(r=p->firstchild;r->nextsib;r=r->nextsib);
            r->nextsib=q; // 作为双亲最后一个孩子的兄弟
        }
    } //Addto_Forest

main()
{
    ...
    T=(CSTNode*)malloc(sizeof(CSTNode)); // 建立树根
    T->data=1;
    Forest_Prim(G,1,T);
    ...
} //main

```

块而得到

的,其时间复杂度为  $O(n^2)$ .

7.33

```
typedef struct {
    int vex; // 结点序号
    int ecno; // 结点所属的连通分量号
} VexInfo;
VexInfo vexs[MAXSIZE]; // 记录结点所属连通分量号的数组
void Init_VexInfo(VexInfo &vexs[], int vexnum) // 初始化
{
    for(i=0; i<vexnum; i++)
        vexs[i] = {i, i}; // 初始状态 : 每一个结点都属于不同的连通分量
} // Init_VexInfo
int is_ec(VexInfo vexs[], int i, int j) // 判断顶点 i 和顶点 j 是否属于同一个连通分量
{
    if(vexs[i].ecno == vexs[j].ecno) return 1;
    else return 0;
} // is_ec
void merge_ec(VexInfo &vexs[], int ec1, int ec2) // 合并连通分量 ec1 和 ec2
{
    for(i=0; i<vexs[i].vex; i++)
        if(vexs[i].ecno == ec2) vexs[i].ecno == ec1;
} // merge_ec
void MinSpanTree_Kruscal(Graph G, EdgeSetType &EdgeSet, CSTree &T) // 求图的最小生成树的
// 克鲁斯卡尔算法
{
    Init_VexInfo(vexs, G.vexnum);
    ecnum = G.vexnum; // 连通分量个数
    while(ecnum > 1)
    {
        GetMinEdge(EdgeSet, u, v); // 选出最短边
        if(!is_ec(vexs, u, v)) // u 和 v 属于不同连通分量
        {
            Addto_CSTree(T, u, v); // 加入到生成树中
            merge_ec(vexs, vexs[u].ecno, vexs[v].ecno); // 合并连通分量
            ecnum--;
        }
        DelMinEdge(EdgeSet, u, v); // 从边集中删除
    } // while
} // MinSpanTree_Kruscal
```

严蔚敏《数据结构习题集》解答

```
void Addto_CSTree(CSTree &T, int i, int j) // 把边 (i,j) 添加到孩子兄弟链表表示的树 T 中
{
    if(!is_ec(vexs, i, j)) // i 和 j 属于不同连通分量
    {
        Addto_CSTree(T, i, j); // 加入到生成树中
        merge_ec(vexs, vexs[i].ecno, vexs[j].ecno); // 合并连通分量
        ecnum--;
    }
}
```



```

DFS(G,v); // 从顶点 v 出发进行深度优先遍历
for(flag=1,w=0;w<G.vexnum;w++)

```

第 90 页 共 124 页

```

    if(!visited[w]) flag=0; // 如果 v 是根,则深度优先遍历可以访问到所有结点
    if(flag) printf("Found a root vertex:%d\n",v);
} //for
} //Get_Root, 这个算法要求图中不能有环,否则会发生误判

```

```

void DFS(ALGraph G,int v)
{
    visited[v]=1;
    for(p=G.vertices[v].firstarc;p;p=p->nextarc)
    {
        w=p->adjvex;
        if(!visited[w]) DFS(G,w);
    }
} //DFS

```

7.36

```

void Fill_MPL(ALGraph &G) // 为有向无环图 G 添加 MPL 域
{

```

```

    FindIndegree(G,indegree);
    for(i=0;i<G.vexnum;i++)
        if(!indegree[i]) Get_MPL(G,i); // 从每一个零入度顶点出发构建 MPL 域
} //Fill_MPL

```

```

int Get_MPL(ALGraph &G,int i) // 从一个顶点出发构建 MPL 域并返回其 MPL 值
{
    if(!G.vertices[i].firstarc)
    {
        G.vertices[i].MPL=0;
        return 0; // 零出度顶点
    }
    else
    {

```

严蔚敏《数据结构习题集》解答

```

        max=0;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            j=p->adjvex;
            if(G.vertices[j].MPL==0) k=Get_MPL(G,j);
            if(k>max) max=k; // 求其直接后继顶点 MPL 的最大者
        }
        G.vertices[i].MPL=max+1; // 再加一,就是当前顶点的 MPL
        return max+1;
    } //else
} //Get_MPL

```

7.37

```
int maxlen,path[MAXSIZE]; // 数组 path 用于存储当前路径
int mlp[MAXSIZE]; // 数组 mlp 用于存储已发现的最长路径
void Get_Longest_Path(ALGraph G)// 求一个有向无环图中最长的路径
{
    maxlen=0;
    FindIndegree(G,indegree);
    for(i=0;i<G.vexnum;i++)
    {
        for(j=0;j<G.vexnum;j++) visited[j]=0;
        if(!indegree[i]) DFS(G,i,0);// 从每一个零入度结点开始深度优先遍历
    }
    printf("Longest Path:");
    for(i=0;mlp[i];i++) printf("%d",mlp[i]); // 输出最长路径
}
//Get_Longest_Path
void DFS(ALGraph G,int i,int len)
{
    visited[i]=1;
    path[len]=i;
    if(len>maxlen&&!G.vertices[i].firstarc) // 新的最长路径
    {
        for(j=0;j<=len;j++) mlp[j]=path[j]; // 保存下来
        maxlen=len;
    }
    else
    {
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            j=p->adjvex;
            if(!visited[j]) DFS(G,j,len+1);
        }
    }
    //else
    path[i]=0;
}
```

第 91 页 共 124 页

严蔚敏《数据结构习题集》解答

```
visited[i]=0;
}
//DFS
```

7.38

第 92 页 共 124 页

```
void NiBoLan_DAG(ALGraph G)// 输出有向无环图形式表示的表达式的逆波兰式
{
    FindIndegree(G,indegree);
    for(i=0;i<G.vexnum;i++)
        if(!indegree[i]) r=i; // 找到有向无环图的根
}
```

```

    PrintNiBoLan_DAG(G,i);
} //NiBoLan_DAG
void PrintNiBoLan_DAG(ALGraph G,int i) // 打印输出以顶点 i 为根的表达式逆波兰式
{
    c=G.vertices[i].data;
    if(!G.vertices[i].firstarc) //c 是原子
        printf("%c",c);
    else //子表达式
    {
        p=G.vertices[i].firstarc;
        PrintNiBoLan_DAG(G,p->adjvex);
        PrintNiBoLan_DAG(G,p->nexarc->adjvex);
        printf("%c",c);
    }
} //PrintNiBoLan_DAG

```

7.39

```

void PrintNiBoLan_Bitree(Bitree T) // 在二叉链表存储结构上重做上一题
{
    if(T->lchild) PrintNiBoLan_Bitree(T->lchild);
    if(T->rchild) PrintNiBoLan_Bitree(T->rchild);
    printf("%c",T->data);
} //PrintNiBoLan_Bitree

```

7.40

```

int Evaluate_DAG(ALGraph G) // 给有向无环图表示的表达式求值
{
    FindIndegree(G, indegree);
    for(i=0; i<G.vexnum; i++)
        if(!indegree[i]) r=i; // 找到有向无环图的根
    return Evaluate_imp(G,i);
} //NiBoLan_DAG
int Evaluate_imp(ALGraph G,int i) // 求子表达式的值
{
    if(G.vertices[i].tag==NUM) return G.vertices[i].value;
    else
    {
        p=G.vertices[i].firstarc;

```

严蔚敏《数据结构习题集》解答

```

        v1=Evaluate_imp(G,p->adjvex);
        v2=Evaluate_imp(G,p->nextarc->adjvex);
        return calculate(v1,G.vertices[i].optr,v2);
    }
} //Evaluate_imp

```

分析:本题中,邻接表的 vertices 向量的元素类型修改如下:

```

struct {

```



```

enum tag{NUM,OPTR};
union {
    int value;
    char optr;
};
ArcNode * firstarc;
} Elemtype;

```

7.41

void Critical\_Path(ALGraph G)// 利用深度优先遍历求网的关键路径

```

{
    FindIndegree(G, indegree);
    for(i=0; i<G.vexnum; i++)
        if(!indegree[i]) DFS1(G, i); // 第一次深度优先遍历 :建立 ve
    for(i=0; i<G.vexnum; i++)
        if(!indegree[i]) DFS2(G, i); // 第二次深度优先遍历 :建立 vl
    for(i=0; i<=G.vexnum; i++)
        if(vl[i]==ve[i]) printf("%d", i); // 打印输出关键路径
}

```

//Critical\_Path

void DFS1(ALGraph G, int i)

```

{
    if(!indegree[i]) ve[i]=0;
    for(p=G.vertices[i].firstarc; p; p=p->nextarc)
    {
        dut=*p->info;
        if(ve[i]+dut>ve[p->adjvex])
            ve[p->adjvex]=ve[i]+dut;
        DFS1(G, p->adjvex);
    }
}

```

//DFS1

void DFS2(ALGraph G, int i)

```

{
    if(!G.vertices[i].firstarc) vl[i]=ve[i];
    else
    {
        for(p=G.vertices[i].firstarc; p; p=p->nextarc)
        {
            DFS2(G, p->adjvex);

```

第 93 页 共 124 页

严蔚敏《数据结构习题集》解答

```

        dut=*p->info;
        if(vl[p->adjvex]-dut<vl[i])
            vl[i]=vl[p->adjvex]-dut;
    }
}
//else

```

```
}//DFS2
```

7.42

第 94 页 共 124 页

```
void ALGraph_DIJ(ALGraph G,int v0,Pathmatrix &P,ShortestPathTable &D)// 在邻接表
存储结
```

## 构造上实现迪杰斯特拉算法

```

{
    for(v=0;v<G.vexnum;v++)
        D[v]=INFINITY;
    for(p=G.vertices[v0].firstarc;p;p=p->nextarc)
        D[p->adjvex]=*p->info; // 给 D 数组赋初值
    for(v=0;v<G.vexnum;v++)
    {
        final[v]=0;
        for(w=0;w<G.vexnum;w++) P[v][w]=0; // 设空路径
        if(D[v]<INFINITY)
        {
            P[v][v0]=1;
            P[v][v]=1;
        }
    }
}

D[v0]=0;final[v0]=1; // 初始化
for(i=1;i<G.vexnum;i++)
{
    min=INFINITY;
    for(w=0;w<G.vexnum;w++)
        if(!final[w])
            if(D[w]<min) // 尚未求出到该顶点的最短路径
            {
                v=w;
                min=D[w];
            }
    final[v]=1;
    for(p=G.vertices[v].firstarc;p;p=p->nextarc)
    {
        w=p->adjvex;
        if(!final[w]&&(min+(*p->info)<D[w])) // 符合迪杰斯特拉条件
        {
            D[w]=min+edgelen(G,v,w);
            P[w]=P[v];
            P[w][w]=1; // 构造最短路径
        }
    }
}
}

```

```

    }//for
} //ALGraph_DIJ

```

第 95 页 共 124 页

分析:本算法对迪杰斯特拉算法中直接取任意边长度的语句作了修改 .由于在原算法中 , 每次

循环都是对尾相同的边进行处理 ,所以可以用遍历邻接表中的一条链来代替 .

严蔚敏《数据结构习题集》解答

## 第八章 动态存储管理

8.11

```

typedef struct {
    char *start;
    int size;
} fmblock; // 空闲块类型

char *Malloc_Fdlf(int n)// 遵循最后分配者最先释放规则的内存分配算法
{
    while(Gettop(S,b)&& b.size<n)
    {
        Pop(S,b);
        Push(T,b); //从栈顶逐个取出空闲块进行比较
    }
    if(StackEmpty(S)) return NULL; // 没有大小足够的空闲块
    Pop(S,b);
    b.size-=n;
    if(b.size) Push(S,{b.start+n,b.size}); // 分割空闲块
    while(!StackEmpty(T))
    {
        Pop(T,a);
        Push(S,a);
    } // 恢复原来次序
    return b.start;
} //Malloc_Fdlf

mem_init()// 初始化过程
{
    ...
    InitStack(S);InitStack(T); //S 和 T 的元素都是 fmblock 类型
    Push(S,{MemStart,MemLen}); // 一开始 ,栈中只有一个内存整块
    ...
} //main

```

8.12

```

void Free_Fdlf(char *addr,int n)// 与上一题对应的释放算法
{
    while(Gettop(S,b)&& b.start<addr)

```

```

{
    Pop(S,b);
    Push(T,b);
} // 在按地址排序的栈中找到合适的插入位置
if(Gettop(T,b)&&(b.start+b.size==addr)) // 可以与上邻块合并
{
    Pop(T,b);

```

第 96 页 共 124 页

严蔚敏《数据结构习题集》解答

```

    addr=b.start;n+=b.size;
}
if(Gettop(S,b)&&(addr+n==b.start)) // 可以与下邻块合并
{
    Pop(S,b);
    n+=b.size;
}
Push(S,{addr,n}); // 插入到空闲块栈中
while(!StackEmpty(T))
{
    Pop(T,b);
    Push(S,b);
} // 恢复原来次序
} // Free_Fdlf

```

8.13

第 97 页 共 124 页

```

void Free_BT(Space &pav,Space p) // 在边界标识法的动态存储管理系统中回收空闲块
p
{
    n=p->size;
    f=p+n-1; //f 指向空闲块底部
    if((p-1)->tag&&(f+1)->tag) // 回收块上下邻块均为占用块
    {
        p->tag=0;f->tag=0;
        f->uplink=p;
        if(!pav)
        {
            p->llink=p;
            p->rlink=p;
        }
        else
        {
            q=pav->llink;
            p->llink=q;p->rlink=pav;
            q->rlink=p;pav->llink=p;

```

```

    }
    pav=p;
} //if
else if(!(p-1)->tag&&(f+1)->tag) // 上邻块为空闲块
{
    q=(p-1)->uplink;
    q->size+=n;
    f->uplink=q;
    f->tag=0;
}
else if((p-1)->tag&&!(f+1)->tag) // 下邻块为空闲块

```

严蔚敏《数据结构习题集》解答

```

{
    q=f+1;
    s=q->llink;t=q->rlink;
    p->llink=s;p->rlink=t;
    s->rlink=p;t->llink=p;
    p->size+=q->size;
    (q+q->size-1)->uplink=p;
    p->tag=0;
}
else //上下邻块均为空闲块
{
    s=(p-1)->uplink;
    t=f+1;
    s->size+=n+t->size;
    t->llink->rlink=t->rlink;
    t->rlink->llink=t->llink;
    (t+t->size-1)->uplink=s;
}
} //Free_BT, 该算法在课本里有详细的描述

```

8.14

void Free\_BS(freelist &avail,char \*addr,int n)// 伙伴系统的空闲块回收算法

```

{
    buddy=addr%(2*n)?(addr-n):(addr+n); // 求回收块的伙伴地址
    addr->tag=0;
    addr->kval=n;
    for(i=0;avail[i].nodesize<n;i++); // 找到这一大小的空闲块链
    if(!avail[i].first) // 尚没有该大小的空闲块
    {
        addr->llink=addr;
        addr->rlink=addr;
        avail[i].first=addr; // 作为唯一一个该大小的空闲块
    }
    else

```

```

{
    for(p=avail[i].first;p!=buddy&&p!=avail[i].first;p=p->rlink);//    寻找伙伴
    if(p==buddy) // 伙伴为空闲块 ,此时进行合并
    {
        if(p->rlink==p) avail[i].first=NULL;//    伙伴是此大小的唯一空闲块
        else
        {
            p->llink->rlink=p->rlink;
            p->rlink->llink=p->llink;
        } // 从空闲块链中删去伙伴
        new=addr>p?p:addr; //合并后的新块首址

```

第 98 页 共 124 页

严蔚敏《数据结构习题集》解答

```

        Free_BS(avail,new,2*n); // 递归地回收新块
    }//if
    else //伙伴为占用块 ,此时插入空闲块链头部
    {
        q=p->rlink;
        p->rlink=addr;addr->llink=p;
        q->llink=addr;addr->rlink=q;
    }
} //else
} //Free_BS
8.15

```

第 99 页 共 124 页

```

FBList *MakeList(char *highbound,char *lowbound)//    把堆结构存储的所有空闲块链
接成可
利用空间表 ,并返回表头指针
{
    p=lowbound;
    while(p->tag&&p<highbound) p++; //    查找第一个空闲块
    if(p>=highbound) return NULL; //    没有空闲块
    head=p;
    for(q=p;p<highbound;p+=cellsize) //    建立链表
        if(!p->tag)
        {
            q->next=p;
            q=p;
        } //if
    p->next=NULL;
    return head; //返回头指针
} //MakeList
8.16
void Mem_Contract(Heap &H)//    对堆 H 执行存储紧缩

```

```

{
    q=MemStart;j=0;
    for(i=0;i<Max_ListLen;i++)
        if(H.list[i].stadr->tag)
        {
            s=H.list[i].length;
            p=H.list[i].stadr;
            for(k=0;k<s;k++) *(q++)=*(p++); // 紧缩内存空间
            H.list[j].stadr=q;
            H.list[j].length=s; // 紧缩占用空间表
            j++;
        }
}
} //Mem_Contract
严蔚敏《数据结构习题集》解答

```

## 第九章 查找

9.25

第 100 页 共 124 页

```

int Search_Sq(SSTable ST,int key)//在有序表上顺序查找的算法 ,监视哨设在高下标端
{
    ST.elem[ST.length+1].key=key;
    for(i=1;ST.elem[i].key>key;i++);
    if(i>ST.length||ST.elem[i].key<key) return ERROR;
    return i;
} //Search_Sq

```

分析 :本算法查找成功情况下的平均查找长度为  $ST.length/2$ ,不成功情况下为  $ST.length$ .

9.26

```

int Search_Bin_Recursive(SSTable ST,int key,int low,int high)// 折半查找的递归算法
{
    if(low>high) return 0; // 查找不到时返回 0
    mid=(low+high)/2;
    if(ST.elem[mid].key==key) return mid;
    else if(ST.elem[mid].key>key)
        return Search_Bin_Recursive(ST,key,low,mid-1);
    else return Search_Bin_Recursive(ST,key,mid+1,high);
}
} //Search_Bin_Recursive

```

9.27

```

int Locate_Bin(SSTable ST,int key)// 折半查找 ,返回小于或等于待查元素的最后一个结
点号
{
    int *r;

```

```

    r=ST.elem;
if(key<r .key) return 0;
else if(key>=r[ST.length].key) return ST.length;
low=1;high=ST.length;
while(low<=high)
{
    mid=(low+high)/2;
    if(key>=r[mid].key&&key<r[mid+1].key) // 查找结束的条件
        return mid;
    else if(key<r[mid].key) high=mid;
    else low=mid;
} // 本算法不存在查找失败的情况 ,不需要 return 0;
} //Locate_Bin

```

9.28

```

typedef struct {
    int maxkey;
    int firstloc;
} Index;

typedef struct {
    int *elem;
    int length;
}

```

严蔚敏《数据结构习题集》解答

第 101 页 共 124 页

```

Index idx[MAXBLOCK]; // 每块起始位置和最大元素 ,其
中 idx[0]不利用 ,其内
容初始化为 {0,0} 以利于折半查找
int blknum; // 块的数目
} IdxSeqList; // 索引顺序表类型
int Search_IdxSeq(IdxSeqList L,int key)// 分块查找 ,用折半查找法确定记录所在块 ,块内采用顺
序查找法
{
    if(key>L.idx[L.blknum].maxkey) return ERROR; // 超过最大元素
    low=1;high=L.blknum;
    found=0;
    while(low<=high&&!found) // 折半查找记录所在块号 mid
    {
        mid=(low+high)/2;
        if(key<=L.idx[mid].maxkey&&key>L.idx[mid-1].maxkey)
            found=1;
        else if(key>L.idx[mid].maxkey)
            low=mid+1;
        else high=mid-1;
    }
    i=L.idx[mid].firstloc; // 块的下界
}

```



```

j=i+blksize-1; // 块的上界
temp=L.elem[i-1]; // 保存相邻元素
L.elem[i-1]=key; // 设置监视哨
for(k=j;L.elem[k]!=key;k--); // 顺序查找
L.elem[i-1]=temp; // 恢复元素
if(k<i) return ERROR; // 未找到
return k;

```

```

} // Search_IdxSeq

```

分析:在块内进行顺序查找时,如果需要设置监视哨,则必须先保存相邻块的相邻元素,以免数据丢失。

9.29

```

typedef struct {

```

```

    LNode *h; //h 指向最小元素
    LNode *t; //t 指向上次查找的结点
} CSList;

```

```

LNode *Search_CSList(CSList &L,int key) // 在有序单循环链表存储结构上的查找算法,假定每次查找都成功

```

```

{
    if(L.t->data==key) return L.t;
    else if(L.t->data>key)

```

严蔚敏《数据结构习题集》解答

```

        for(p=L.h,i=1;p->data!=key;p=p->next,i++);
    else
        for(p=L.t,i=L.tpos;p->data!=key;p=p->next,i++);
    L.t=p; // 更新 t 指针
    return p;
} // Search_CSList

```

第 102 页 共 124 页

分析:由于题目中假定每次查找都是成功的,所以本算法中没有关于查找失败的处理。由微积分可得,在等概率情况下,平均查找长度约为  $n/3$ 。

9.30

```

typedef struct {

```

```

    DLNode *pre;
    int data;
    DLNode *next;
} DLNode;

```

```

typedef struct {

```

```

    DLNode *sp;
    int length;
} DSList; // 供查找的双向循环链表类型

```

```

DLNode *Search_DSList(DSList &L,int key) // 在有序双向循环链表存储结构上的查找算法,假定每次查找都成功

```

```

{

```

```

p=L.sp;
if(p->data>key)
{
    while(p->data>key) p=p->pre;
    L.sp=p;
}
else if(p->data<key)
{
    while(p->data<key) p=p->next;
    L.sp=p;
}
return p;
} //Search_DSList

```

分析:本题的平均查找长度与上一题相同,也是  $n/3$ .

9.31

```

int last=0,flag=1;
int Is_BSTree(Bitree T) // 判断二叉树 T 是否二叉排序树,是则返回 1,否则返回 0
{
    if(T->lchild&&flag) Is_BSTree(T->lchild);
    if(T->data<last) flag=0; // 与其中序前驱相比较
    last=T->data;
    if(T->rchild&&flag) Is_BSTree(T->rchild);
    return flag;
}

```

严蔚敏《数据结构习题集》解答

```

} //Is_BSTree

```

9.32

```

int last=0;

```

第 103 页 共 124 页

```

void MaxLT_MinGT(BiTree T,int x) // 找到二叉排序树 T 中小于 x 的最大元素和大于
x 的最小
元素

```

```

{
    if(T->lchild) MaxLT_MinGT(T->lchild,x); // 本算法仍是借助中序遍历来实现
    if(last<x&&T->data==x) // 找到了小于 x 的最大元素
        printf("a=%d\n",last);
    if(last<=x&&T->data>x) // 找到了大于 x 的最小元素
        printf("b=%d\n",T->data);
    last=T->data;
    if(T->rchild) MaxLT_MinGT(T->rchild,x);
} //MaxLT_MinGT

```

9.33

```

void Print_NLT(BiTree T,int x) // 从大到小输出二叉排序树 T 中所有不小于 x 的元素
{
    if(T->rchild) Print_NLT(T->rchild,x);
}

```

```

    if(T->data<x) exit(); // 当遇到小于 x 的元素时立即结束运行
    printf("%d\n",T->data);
    if(T->lchild) Print_NLT(T->lchild,x); // 先右后左的中序遍历
} // Print_NLT

```

9.34

void Delete\_NLT(BiTree &T,int x) // 删除二叉排序树 T 中所有不小于 x 元素结点,并释放空间

```

{
    if(T->rchild) Delete_NLT(T->rchild,x);
    if(T->data<x) exit(); // 当遇到小于 x 的元素时立即结束运行
    q=T;
    T=T->lchild;
    free(q); // 如果树根不小于 x,则删除树根,并以左子树的根作为新的树根
    if(T) Delete_NLT(T,x); // 继续在左子树中执行算法
} // Delete_NLT

```

9.35

void Print\_Between(BiThrTree T,int a,int b) // 打印输出后继线索二叉排序树 T 中所有大于 a 且小于 b 的元素

```

{
    p=T;
    while(!p->ltag) p=p->lchild; // 找到最小元素
    while(p&& p->data<b)
    {
        if(p->data>a) printf("%d\n",p->data); // 输出符合条件的元素
        if(p->rtag) p=p->rtag;
        else
        {

```

严蔚敏《数据结构习题集》解答

```

            p=p->rchild;
            while(!p->ltag) p=p->lchild;
        } // 转到中序后继
    } // while
} // Print_Between

```

9.36

第 104 页 共 124 页

void BSTree\_Insert\_Key(BiThrTree &T,int x) // 在后继线索二叉排序树 T 中插入元素 x

```

{
    if(T->data<x) // 插入到右侧
    {
        if(T->rtag) // T 没有右子树时,作为右孩子插入
        {
            p=T->rchild;
            q=(BiThrNode*)malloc(sizeof(BiThrNode));
            q->data=x;

```

```

        T->rchild=q;T->rtag=0;
        q->rtag=1;q->rchild=p; // 修改原线索
    }
    else BSTree_Insert_Key(T->rchild,x);//T 有右子树时 ,插入右子树中
} //if
else if(T->data>x) // 插入到左子树中
{
    if(!T->lchild) //T 没有左子树时 ,作为左孩子插入
    {
        q=(BiThrNode*)malloc(sizeof(BiThrNode));
        q->data=x;
        T->lchild=q;
        q->rtag=1;q->rchild=T; // 修改自身的线索
    }
    else BSTree_Insert_Key(T->lchild,x);//T 有左子树时 ,插入左子树中
} //if
} //BSTree_Insert_Key

```

9.37

Status BSTree\_Delete\_key(BiThrTree &T,int x)// 在后继线索二叉排序树 T 中删除元素 x

```

{
    BTreeNode *pre,*ptr,*suc;//ptr 为 x 所在结点 ,pre 和 suc 分别指向 ptr 的前驱和后继
    p=T;last=NULL; //last 始终指向当前结点 p 的前一个 (前驱)
    while(!p->ltag) p=p->lchild; // 找到中序起始元素
    while(p)
    {
        if(p->data==x) // 找到了元素 x 结点
        {
            pre=last;
            ptr=p;

```

严蔚敏《数据结构习题集》解答

```

        }
        else if(last&&last->data==x) suc=p; // 找到了 x 的后继
        if(p->rtag) p=p->rtag;
        else
        {
            p=p->rchild;
            while(!p->ltag) p=p->lchild;
        } // 转到中序后继
        last=p;
    } //while // 借助中序遍历找到元素 x 及其前驱和后继结点
    if(!ptr) return ERROR; // 未找到待删结点
    Delete_BSTree(ptr); // 删除 x 结点
    if(pre&&pre->rtag)
        pre->rchild=suc; // 修改线索

```

```

        return OK;
    }//BSTree_Delete_key

```

第 105 页 共 124 页

void Delete\_BSTree(BiThrTree &T)// 课本上给出的删除二叉排序树的子树 T 的算法，按照线索

二叉树的结构作了一些改动

```

{
    q=T;
    if(!T->ltag&&T->rtag) // 结点无右子树 ,此时只需重接其左子树
        T=T->lchild;
    else if(T->ltag&&!T->rtag) // 结点无左子树 ,此时只需重接其右子树
        T=T->rchild;
    else if(!T->ltag&&!T->rtag) // 结点既有左子树又有右子树
    {
        p=T;r=T->lchild;
        while(!r->rtag)
        {
            s=r;
            r=r->rchild; // 找到结点的前驱 r 和 r 的双亲 s
        }
        T->data=r->data; // 用 r 代替 T 结点
        if(s!=T)
            s->rchild=r->lchild;
        else s->lchild=r->lchild; // 重接 r 的左子树到其双亲结点上
        q=r;
    }//else
    free(q); // 删除结点
} //Delete_BSTree

```

分析:本算法采用了先求出 x 结点的前驱和后继 ,再删除 x 结点的办法 ,这样修改线索时会比较

简单,直接让前驱的线索指向后继就行了 .如果试图在删除 x 结点的同时修改线索 ,则问题反而复杂化了 .

9.38

严蔚敏《数据结构习题集》解答

```

void BSTree_Merge(BiTree &T,BiTree &S)// 把二叉排序树 S 合并到 T 中
{
    if(S->lchild) BSTree_Merge(T,S->lchild);
    if(S->rchild) BSTree_Merge(T,S->rchild); // 合并子树
    Insert_Key(T,S); // 插入元素
} //BSTree_Merge

```

第 106 页 共 124 页

void Insert\_Node(Bitree &T,BTNode \*S)// 把树结点 S 插入到 T 的合适位置上

```

{
    if(S->data>T->data)

```

```

{
    if(!T->rchild) T->rchild=S;
    else Insert_Node(T->rchild,S);
}
else if(S->data<T->data)
{
    if(!T->lchild) T->lchild=S;
    else Insert_Node(T->lchild,S);
}
S->lchild=NULL; // 插入的新结点必须和原来的左右子树断绝关系
S->rchild=NULL; // 否则会导致树结构的混乱

```

}//Insert\_Node

分析 :这是一个与课本上不同的插入算法 .在合并过程中 ,并不释放或新建任何结点 ,而是采取

修改指针的方式来完成合并 .这样 ,就必须按照后序序列把一棵树中的元素逐个连接到另一棵树上 ,否则将会导致树的结构的混乱 .

9.39

void BSTree\_Split(BiTree &T,BiTree &A,BiTree & B,int x)// 把二叉排序树 T 分裂为两棵二叉排序树 A 和 B,其中 A 的元素全部小于等于 x,B 的元素全部大于 x

```

{
    if(T->lchild) BSTree_Split(T->lchild,A,B,x);
    if(T->rchild) BSTree_Split(T->rchild,A,B,x); // 分裂左右子树
    if(T->data<=x) Insert_Node(A,T);
    else Insert_Node(B,T); // 将元素结点插入合适的树中

```

}//BSTree\_Split

void Insert\_Node(Bitree &T,BTNode \*S)// 把树结点 S 插入到 T 的合适位置上

```

{
    if(!T) T=S; // 考虑到刚开始分裂时树 A 和树 B 为空的情况
    else if(S->data>T->data) // 其余部分与上一题同
    {
        if(!T->rchild) T->rchild=S;
        else Insert_Node(T->rchild,S);
    }
    else if(S->data<T->data)
    {
        if(!T->lchild) T->lchild=S;

```

严蔚敏《数据结构习题集》解答

```

        else Insert_Node(T->lchild,S);
    }
    S->lchild=NULL;
    S->rchild=NULL;

```

}//Insert\_Key

9.40

```

typedef struct {
    int data;

```

```

int bf;
int lsize; //lsize 域表示该结点的左子树的结点总数加 1
BlcNode *lchild,*rchild;
} BlcNode,*BlcTree; // 含 lsize 域的平衡二叉排序树类型

```

第 107 页 共 124 页

```

BTNode *Locate_BlclTree(BlclTree T,int k)// 在含 lsize 域的平衡二叉排序树 T 中确定
第 k 小的
结点指针
{
    if(!T) return NULL; //k 小于 1 或大于树结点总数
    if(T->lsize==k) return T; // 就是这个结点
    else if(T->lsize>k)
        return Locate_BlclTree(T->lchild,k); // 在左子树中寻找
    else return Locate_BlclTree(T->rchild,k-T->lsize); // 在右子树中寻找 ,注意要修改 k 的
值
} //Locate_BlclTree

```

9.41

```

typedef struct {
    enum {LEAF,BRANCH} tag; // 结点类型标识
    int keynum;
    BPLink parent; // 双亲指针
    int key[MAXCHILD]; // 关键字
    union {
        BPLink child[MAXCHILD]; // 非叶结点的孩子指针
        struct {
            rectype *info[MAXCHILD]; // 叶子结点的信息
            BPNODE *next; // 指向下一个叶子结点的链接
        } leaf;
    }
} BPNODE,*BPLink,*BPTree; //B+ 树及其结点类型

```

Status BPTree\_Search(BPTree T,int key,BPNODE \*ptr,int pos)//B+ 树中按关键字随机查找的算法 ,

返回包含关键字的叶子结点的指针 ptr 以及关键字在叶子结点中的位置 pos

```

{
    p=T;
    while(p.tag==BRANCH) // 沿分支向下查找
    {
        for(i=0;i<p->keynum&&key>p->key[i];i++); // 确定关键字所在子树
        if(i==p->keynum) return ERROR; // 关键字太大
        p=p->child[i];
    }
}

```

严蔚敏《数据结构习题集》解答

```

}
for(i=0;i<p->keynum&&key!=p->key[i];i++); // 在叶子结点中查找

```

```

    if(i==p->keynum) return ERROR; // 找不到关键字
    ptr=p;pos=i;
    return OK;
} //BPTree_Search

```

9.42

第 108 页 共 124 页

void TrieTree\_Insert\_Key(TrieTree &T,StringType key)// 在 Trie 树 T 中插入字符串

key,StringType 的结构见第四章

```

{
    q=(TrieNode*)malloc(sizeof(TrieNode));
    q->kind=LEAF;
    q->lf.k=key; // 建叶子结点
    klen=key[0];
    p=T;i=1;
    while(p&& i<=klen&&p->bh.ptr[ord(key[i])])
    {
        last=p;
        p=p->bh.ptr[ord(key[i])];
        i++;
    } // 自上而下查找
    if(p->kind==BRANCH) // 如果最后落到分支结点 (无同义词):
    {
        p->bh.ptr[ord(key[i])]=q; // 直接连上叶子
        p->bh.num++;
    }
    else //如果最后落到叶子结点 (有同义词):
    {
        r=(TrieNode*)malloc(sizeof(TrieNode)); // 建立新的分支结点
        last->bh.ptr[ord(key[i-1])]=r; // 用新分支结点取代老叶子结点和上一层的联系
        r->kind=BRANCH;r->bh.num=2;
        r->bh.ptr[ord(key[i])]=q;
        r->bh.ptr[ord(p->lf.k[i])]=p; // 新分支结点与新老两个叶子结点相连
    }
} //TrieTree_Insert_Key

```

分析:当自上而下的查找结束时,存在两种情况.一种情况,树中没有待插入关键字的同义词,此时只要新建一个叶子结点并连到分支结点上即可.另一种情况,有同义词,此时要把同义词的叶子结点与树断开,在断开的部位新建一个下一层的分支结点,再把同义词和新关键字的叶子结点连到新分支结点的下一层.

9.43

Status TrieTree\_Delete\_Key(TrieTree &T,StringType key)// 在 Trie 树 T 中删除字符串 key

```

{
    p=T;i=1;
    while(p&&p->kind==BRANCH&&i<=key[0]) // 查找待删除元素

```

严蔚敏《数据结构习题集》解答



```

{
    last=p;
    p=p->bh.ptr[ord(key[i])];
    i++;
}
if(p&& p->kind==LEAF&&p->lf.k=key) // 找到了待删除元素
{
    last->bh.ptr[ord(key[i-1])]=NULL;
    free(p);
    return OK;
}
else return ERROR; //没找到待删除元素
} //TrieTree_Delete_Key
9.44

```

第 109 页 共 124 页

```

void Print_Hash(HashTable H) // 按第一个字母顺序输出 Hash 表中的所有关键字 ,其中
处理冲突
采用线性探测开放定址法
{
    for(i=1;i<=26;i++)
        for(j=i;H.elem[j].key;j=(j+1)%hashsize[sizeindex]) // 线性探测
            if(H(H.elem[j].key)==i) printf("%s\n",H.elem[j]);
} //Print_Hash
int H(char *s) // 求 Hash 函数
{
    if(s) return s[0]-96; // 求关键字第一个字母的字母序号 (小写)
    else return 0;
} //H
9.45

```

```

typedef *LNode[MAXSIZE] CHashTable; // 链地址 Hash 表类型
Status Build_Hash(CHashTable &T,int m) // 输入一组关键字 ,建立 Hash 表,表长为 m,用链地址
法
处理冲突 .
{
    if(m<1) return ERROR;
    T=malloc(m*sizeof(WORD)); // 建立表头指针向量
    for(i=0;i<m;i++) T[i]=NULL;
    while((key=Inputkey())!=NULL) // 假定 Inputkey 函数用于从键盘输入关键字
    {
        q=(LNode*)malloc(sizeof(LNode));
        q->data=key;q->next=NULL;
        n=H(key);
        if(!T[n]) T[n]=q; // 作为链表的第一个结点
        else
        {

```

```

        for(p=T[n];p->next;p=p->next);
        p->next=q; //插入链表尾部 .本算法不考虑排序问题 .

```

严蔚敏《数据结构习题集》解答

```

    }
} //while
return OK;
} //Build_Hash
9.46

```

第 110 页 共 124 页

Status Locate\_Hash(HashTable H,int row,int col,KeyType key,int &k)// 根据行列值在 Hash 表表  
示的稀疏矩阵中确定元素 key 的位置 k

```

{
    h=2*(100*(row/10)+col/10); // 作者设计的 Hash 函数
    while(H.elem[h].key&&!EQ(H.elem[h].key,key))
        h=(h+1)%20000;
    if(EQ(H.elem[h].key,key)) k=h;
    else k=NULL;
} //Locate_Hash

```

分析 :本算法所使用的 Hash 表长 20000,装填因子为 50%,Hash 函数为行数前两位和列数前两位所组成的四位数再乘以二 ,用线性探测法处理冲突 .当矩阵的元素是随机分布时 ,查找的时间复杂度为  $O(1)$ .

严蔚敏《数据结构习题集》解答

## 第十章 内部排序

10.23

void Insert\_Sort1(SqList &L)// 监视哨设在高下标端的插入排序算法

```

{
    k=L.length;
    for(i=k-1;i--i) // 从后向前逐个插入排序
        if(L.r[i].key>L.r[i+1].key)
        {
            L.r[k+1].key=L.r[i].key; // 监视哨
            for(j=i+1;L.r[j].key>L.r[i].key;++j)
                L.r[j-1].key=L.r[j].key; // 前移
            L.r[j-1].key=L.r[k+1].key; // 插入
        }
} //Insert_Sort1

```

10.24

void BilInsert\_Sort(SqList &L)// 二路插入排序的算法

```

{
    int d[MAXSIZE]; // 辅助存储
    x=L.r.key;d =x;
    first=1;final=1;
    for(i=2;i<=L.length;i++)
    {
        if(L.r[i].key>=x) // 插入前部
        {
            for(j=final;d[j]>L.r[i].key;j--)
                d[j+1]=d[j];
            d[j+1]=L.r[i].key;
            final++;
        }
        else //插入后部
        {
            for(j=first;d[j]<L.r[i].key;j++)
                d[j-1]=d[j];
            d[(j-2)%MAXSIZE+1]=L.r[i].key;

```

第 111 页 共 124 页

first=(first-2)%MAXSIZE+1; // 这种形式的表达式是为了兼顾 first=1 的情况

```

    }
} //for
for(i=first,j=1;d[i];i=i%MAXSIZE+1,j++)// 将序列复制回去
    L.r[j].key=d[i];
} //BiInsert_Sort

```

10.25

void SLInsert\_Sort(SLList &L)// 静态链表的插入排序算法

严蔚敏《数据结构习题集》解答

```

{
    L.r[0].key=0;L.r[0].next=1;
    L.r[1].next=0; // 建初始循环链表
    for(i=2;i<=L.length;i++) // 逐个插入
    {
        p=0;x=L.r[i].key;
        while(L.r[L.r[p].next].key<x&&L.r[p].next)
            p=L.r[p].next;
        q=L.r[p].next;
        L.r[p].next=i;
        L.r[i].next=q;
    } //for
    p=L.r[0].next;
    for(i=1;i<L.length;i++) // 重排记录的位置
    {
        while(p<i) p=L.r[p].next;

```

```

        q=L.r[p].next;
        if(p!=i)
        {
            L.r[p]<->L.r[i];
            L.r[i].next=p;
        }
        p=q;
    }//for
} //SLInsert_Sort
10.26

```

第 112 页 共 124 页

```

void Bubble_Sort1(int a[ ],int n)// 对包含 n 个元素的数组 a 进行改进的冒泡排序
{
    change=n-1; //change 指示上一趟冒泡中最后发生交换的元素
    while(change)
    {
        for(c=0,i=0;i<change;i++)
            if(a[i]>a[i+1])
            {
                a[i]<->a[i+1];
                c=i+1; //c 指示这一趟冒泡中发生交换的元素
            }
        change=c;
    }//while
} //Bubble_Sort1
10.27

```

```

void Bubble_Sort2(int a[ ],int n)// 相邻两趟是反方向起泡的冒泡排序算法
{

```

```

    low=0;high=n-1; // 冒泡的上下界

```

严蔚敏《数据结构习题集》解答

```

    change=1;
    while(low<high&&change)
    {
        change=0;
        for(i=low;i<high;i++) // 从上向下起泡
            if(a[i]>a[i+1])
            {
                a[i]<->a[i+1];
                change=1;
            }
        high--; //修改上界
        for(i=high;i>low;i--) // 从下向上起泡
            if(a[i]<a[i-1])
            {

```

```

        a[i]<->a[i-1];
        change=1;
    }
    low++; // 修改下界
} //while
} //Bubble_Sort2

```

10.28

第 113 页 共 124 页

```

void Bubble_Sort3(int a[],int n)// 对上一题的算法进行化简 ,循环体中只包含一次冒泡
{
    int b[3]; //b[0] 为冒泡的下界 ,b[2]为上界 ,b[1]无用
    d=1;b[0]=0;b[2]=n-1; //d 为冒泡方向的标识 ,1 为向上 ,-1 为向下
    change=1;
    while(b[0]<b[2]&&change)
    {
        change=0;
        for(i=b[1-d];i!=b[1+d];i+=d) // 统一的冒泡算法
            if((a[i]-a[i+d])*d>0) // 注意这个交换条件
            {
                a[i]<->a[i+d];
                change=1;
            }
        b[1+d]-=d; // 修改边界
        d*=-1; // 换个方向
    } //while
} //Bubble_Sort3

```

10.29

```

void OE_Sort(int a[],int n)// 奇偶交换排序的算法
{
    change=1;
    while(change)

```

严蔚敏《数据结构习题集》解答

```

{
    change=0;
    for(i=1;i<n-1;i+=2) // 对所有奇数进行一趟比较
        if(a[i]>a[i+1])
        {
            a[i]<->a[i+1];
            change=1;
        }
    for(i=0;i<n-1;i+=2) // 对所有偶数进行一趟比较
        if(a[i]>a[i+1])
        {
            a[i]<->a[i+1];

```

```

        change=1;
    }
} //while
} //OE_Sort
分析 :本算法的结束条件是连续两趟比较无交换发生
10.30
typedef struct {
    int low;
    int high;
} boundary; //子序列的上下界类型
void QSort_NotRecurve(int SQList &L) // 快速排序的非递归算法
{
    low=1;high=L.length;
    InitStack(S); //S 的元素为 boundary 类型
    while(low<high&&!StackEmpty(S)) // 注意排序结束的条件
    {
        if(high-low>2) // 如果当前子序列长度大于 3 且尚未排好序
        {
            pivot=Partition(L,low,high); // 进行一趟划分
            if(high-pivot>pivot-low)
            {
                Push(S,{pivot+1,high}); // 把长的子序列边界入栈
                high=pivot-1; // 短的子序列留待下次排序
            }
        }
        else
        {
            Push(S,{low,pivot-1});
            low=pivot+1;
        }
    }
} //if

```

第 114 页 共 124 页

```

        else if(low<high&&high-low<3) // 如果当前子序列长度小于 3 且尚未排好序
        {

```

严蔚敏《数据结构习题集》解答

```

            Easy_Sort(L,low,high); // 直接进行比较排序
            low=high; // 当前子序列标志为已排好序
        }
        else //如果当前子序列已排好序但栈中还有未排序的子序列
        {
            Pop(S,a); //从栈中取出一个子序列
            low=a.low;
            high=a.high;
        }
    } //while

```

```

} // QSort_NotRecurve
int Partition(SQList &L, int low, int high) // 一趟划分的算法 , 与书上相同
{
    L.r[0] = L.r[low];
    pivotkey = L.r[low].key;
    while (low < high)
    {
        while (low < high && L.r[high].key >= pivotkey)
            high--;
        L.r[low] = L.r[high];
        while (low < high && L.r[low].key <= pivotkey)
            low++;
        L.r[high] = L.r[low];
    } // while
    L.r[low] = L.r[0];
    return low;
} // Partition

```

第 115 页 共 124 页

```

void Easy_Sort(SQList &L, int low, int high) // 对长度小于 3 的子序列进行比较排序
{
    if (high - low == 1) // 子序列只含两个元素
        if (L.r[low].key > L.r[high].key) L.r[low] <-> L.r[high];
    else // 子序列含有三个元素
    {
        if (L.r[low].key > L.r[low+1].key) L.r[low] <-> L.r[low+1];
        if (L.r[low+1].key > L.r[high].key) L.r[low+1] <-> L.r[high];
        if (L.r[low].key > L.r[low+1].key) L.r[low] <-> L.r[low+1];
    }
} // Easy_Sort

```

10.31

```

void Divide(int a[], int n) // 把数组 a 中所有值为负的记录调到非负的记录之前
{
    low = 0; high = n - 1;
    while (low < high)
    {

```

严蔚敏《数据结构习题集》解答

```

        while (low < high && a[high] >= 0) high--; // 以 0 作为虚拟的枢轴记录
        a[low] <-> a[high];
        while (low < high && a[low] < 0) low++;
        a[low] <-> a[high];
    }
} // Divide

```

10.32

```

typedef enum {RED, WHITE, BLUE} color; // 三种颜色

```

void Flag\_Arrange(color a[ ],int n)// 把由三种颜色组成的序列重排为按照红,白,蓝的顺序排列

```
{
    i=0;j=0;k=n-1;
    while(j<=k)
        switch(a[j])
        {
            case RED:
                a[i]<->a[j];
                i++;
                j++;
                break;
            case WHITE:
                j++;
                break;
            case BLUE:
                a[j]<->a[k];
                k--; //这里没有 j++;语句是为了防止交换后 a[j] 仍为蓝色的情况
        }
}
```

}//Flag\_Arrange

分析 :这个算法中设立了三个指针 .其中 ,j 表示当前元素 ;i 以前的元素全部为红色 ;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者后部 .

10.33

void LinkedList\_Select\_Sort(LinkedList &L)// 单链表上的简单选择排序算法

```
{
    for(p=L;p->next->next;p=p->next)
    {
        q=p->next;x=q->data;
        for(r=q,s=q;r->next;r=r->next) // 在 q 后面寻找元素值最小的结点
            if(r->next->data<x)
            {
                x=r->next->data;
                s=r;
            }
        if(s!=q) // 找到了值比 q->data 更小的最小结点 s->next
        {
            p->next=s->next;s->next=q;

            t=q->next;q->next=p->next->next;
            p->next->next=t;
        } // 交换 q 和 s->next 两个结点
    } //for
}
```

严蔚敏《数据结构习题集》解答



```

} //LinkedList_Select_Sort
10.34
void Build_Heap(Heap &H,int n)// 从低下标到高下标逐个插入建堆的算法
{
    for(i=2;i<n;i++)
    { // 此时从 H.r[1] 到 H.r[i-1] 已经是大顶堆
        j=i;
        while(j!=1) // 把 H.r[i] 插入
        {
            k=j/2;
            if(H.r[j].key>H.r[k].key)
                H.r[j]<->H.r[k];
            j=k;
        }
    } //for
} //Build_Heap
10.35
void TriHeap_Sort(Heap &H)// 利用三叉树形式的堆进行排序的算法
{
    for(i=H.length/3;i>0;i--)
        Heap_Adjust(H,i,H.length);
    for(i=H.length;i>1;i--)
    {
        H.r[1]<->H.r[i];
        Heap_Adjust(H,1,i-1);
    }
} //TriHeap_Sort

```

第 117 页 共 124 页

```

void Heap_Adjust(Heap &H,int s,int m)// 顺序表 H 中,H.r[s+1] 到 H.r[m] 已经是堆 ,把
H.r[s] 插入
并调整成堆
{
    rc=H.r[s];
    for(j=3*s-1;j<=m;j=3*j-1)
    {
        if(j<m&&H.r[j].key<H.r[j+1].key) j++;
        if(j<m&&H.r[j].key<H.r[j+1].key) j++;
        H.r[s]=H.r[j];
        s=j;
    }
    H.r[s]=rc;
} //Heap_Adjust

```

严蔚敏《数据结构习题集》解答

第 118 页 共 124 页

分析:本算法与课本上的堆排序算法相比,只有两处改动:1.建初始堆时,i 的上限从  $H.length/3$  开始(为什么?)2.调整堆的时候,要从结点的三个孩子结点中选择最大的那一个,最左边的孩子的序号的计算公式为  $j=3*s-1$ (为什么?)

10.36

void Merge\_Sort(int a[],int n)// 归并排序的非递归算法

```
{
    for(l=1;l<n;l*=2) // 为一趟归并段的段长
        for(i=0;(2*i-1)*l<n;i++) //i 为本趟的归并段序号
        {
            start1=2*i*l; // 求出待归并的两段的上下界
            end1=start1+l-1;
            start2=end1+1;
            end2=(start2+l-1)>(n-1)?(n-1):(start2+l-1); // 注意 end2 可能超出边界
            Merge(a,start1,end1,start2,end2); //归并
        }
}
```

//Merge\_Sort

void Merge(int a[],int s1,int e1,int s2,int e2)// 将有序子序列 a[s1]到 a[e1]和 a[s2]到 a[e2]归并为有序序列 a[s1]到 a[e2]

```
{
    int b[MAXSIZE]; // 设立辅助存储数组 b
    for(i=s1,j=s2,k=s1;i<=e1&&j<=e2;k++)
    {
        if(a[i]<a[j]) b[k]=a[i++];
        else b[k]=a[j++];
    }
    while(i<=e1) b[k++]=a[i++];
    while(j<=e2) b[k++]=a[j++]; // 归并到 b 中
    for(i=s1;i<=e2;i++) // 复制回去
        a[i]=b[i];
}
```

//Merge

10.37

void LinkedList\_Merge\_Sort1(LinkedList &L)// 链表结构上的归并排序非递归算法

```
{
    for(l=1;l<L.length;l*=2) // 为一趟归并段的段长
        for(p=L->next,e2=p;p->next;p=e2)
        {
            for(i=1,q=p;i<=l&&q->next;i++,q=q->next);
            e1=q;
            for(i=1;i<=l&&q->next;i++,q=q->next);
            e2=q; //求出两个待归并子序列的尾指针
            if(e1!=e2) LinkedList_Merge(L,p,e1,e2); // 归并
        }
}
```

//LinkedList\_Merge\_Sort1

void LinkedList\_Merge(LinkedList &L,LNode \*p,LNode \*e1,LNode \*e2)// 对链表上的子序列进

行归并,第一个子序列是从 p->next 到 e1,第二个是从 e1->next 到 e2

```
{
    q=p->next;r=e1->next; //q 和 r 为两个子序列的起始位置
    while(q!=e1->next&&r!=e2->next)
    {
        if(q->data<r->data) // 选择关键字较小的那个结点接在 p 的后面
        {
            p->next=q;p=q;
            q=q->next;
        }
        else
        {
            p->next=r;p=r;
            r=r->next;
        }
    }
} //while
while(q!=e1->next) // 接上剩余部分
{
    p->next=q;p=q;
    q=q->next;
}
while(r!=e2->next)
{
    p->next=r;p=r;
    r=r->next;
}
} //LinkedList_Merge
```

10.38

第 119 页 共 124 页

void LinkedList\_Merge\_Sort2(LinkedList &L) // 初始归并段为最大有序子序列的归并排序,采

用链表存储结构

```
{
    LNode *end[MAXSIZE]; // 设立一个数组来存储各有序子序列的尾指针
    for(p=L->next->next,i=0;p;p=p->next) // 求各有序子序列的尾指针
        if(!p->next||p->data>p->next->data) end[i++]=p;
    while(end[0]->next) // 当不止一个子序列时进行两两归并
    {
        j=0;k=0; //j: 当前子序列尾指针存储位置 ;k:归并后的子序列尾指针存储位置
        for(p=L->next,e2=p;p->next;p=e2) // 两两归并所有子序列
        {
            e1=end[j];e2=end[j+1]; // 确定两个子序列
            if(e1->next) LinkedList_Merge(L,p,e1,e2); // 归并
```

```

        end[k++]=e2; // 用新序列的尾指针取代原来的尾指针
        j+=2; // 转到后面两个子序列
    }

```

严蔚敏《数据结构习题集》解答

```

    }//while
} //LinkedList_Merge_Sort2

```

第 120 页 共 124 页

void LinkedList\_Merge(LinkedList &L,LNode \*p,LNode \*e1,LNode \*e2)// 对链表上的子序列进

行归并,第一个子序列是从 p->next 到 e1,第二个是从 e1->next 到 e2

```

{
    q=p->next;r=e1->next;
    while(q!=e1->next&& r!=e2->next)
    {
        if(q->data<r->data)
        {
            p->next=q;p=q;
            q=q->next;
        }
        else
        {
            p->next=r;p=r;
            r=r->next;
        }
    }
    }//while
    while(q!=e1->next)
    {
        p->next=q;p=q;
        q=q->next;
    }
    while(r!=e2->next)
    {
        p->next=r;p=r;
        r=r->next;
    }
}

```

//LinkedList\_Merge, 与上一题完全相同

10.39

void SL\_Merge(int a[ ],int l1,int l2)// 把长度分别为 l1,l2 且  $l1^2 < (l1+l2)$  的两个有序子序列归并

为有序序列

```

{
    start1=0;start2=l1; //分别表示序列 1 和序列 2 的剩余未归并部分的起始位置
    for(i=0;i<l1;i++) // 插入第 i 个元素
    {

```

```

        for(j=start2;j<l1+l2&& a[j]<a[start1+i];j++); // 寻找插入位置
        k=j-start2; //k 为要向右循环移动的位数
        RSh(a,start1,j-1,k); //将 a[start1]到 a[j-1]之间的子序列循环右移 k 位
        start1+=k+1;
        start2=j; //修改两序列尚未归并部分的起始位置
    }
} //SL_Merge

```

严蔚敏《数据结构习题集》解答

第 121 页 共 124 页

```

void RSh(int a[ ],int start,int end,int k) // 将 a[start]到 a[end]之间的子序列循环右移 k
位,算法原
理参见 5.18
{
    len=end-start+1;
    for(i=1;i<=k;i++)
        if(len%i==0&&k%i==0) p=i; // 求 len 和 k 的最大公约数 p
    for(i=0;i<p;i++) // 对 p 个循环链分别进行右移
    {
        j=start+i;l=start+(i+k)%len;temp=a[j];
        while(l!=start+i)
        {
            a[j]=temp;
            temp=a[l];
            a[l]=a[j];
            j=l;l=start+(j-start+k)%len; // 依次向右移
        }
        a[start+i]=temp;
    } //for
} //RSh

```

10.40

书后给出的解题思路在表述上存在问题 ,无法理解 .比如说 , "把第一个序列划分为两个子序列 ,使其中的第一个子序列含有  $s_1$  个记录 ,  $0 \leq s_1 < s$ , 第二个子序列有  $s$  个记录 ." 可是题目中并没有说明 ,第一个序列的长度  $< 2s$ . 请会做的朋友提供解法 .

10.41

```

void Hash_Sort(int a[ ]) // 对 1000 个关键字为四位整数的记录进行排序
{
    int b[10000];
    for(i=0;i<1000;i++) // 直接按关键字散列
    {
        for(j=a[i];b[j];j=(j+1)%10000);
        b[j]=a[i];
    }
    for(i=0,j=0;i<1000;j++) // 将散列收回 a 中

```



```

        if(c[j]<c[min]) min=j; // 求出最小记录的下标    min
    a[i]<->a[min]; // 与第 i 个记录交换
    c[min]=INFINITY; // 修改该记录的 c 值为无穷大以便下一次选取
}
} //Count_Sort
10.44

```

第 122 页 共 124 页

```

void Enum_Sort(int a[],int n)// 对关键字只能取 v 到 w 之间任意整数的序列进行排序
{
    int number[w+1],pos[w+1];
    for(i=0;i<n;i++) number[a[i]]++; // 计数
    for(pos[0]=0,i=1;i<n;i++)

```

严蔚敏《数据结构习题集》解答

第 123 页 共 124 页

pos[i]=pos[i-1]+num[i]; //pos 数组可以把关键字的值映射为元素在排好的序列中的位置

```

    for(i=0;i<n;i++) // 构造有序数组 c
        c[pos[a[i]]++]=a[i];
    for(i=0;i<n;i++)
        a[i]=c[i];
} //Enum_Sort

```

分析 :本算法参考了第五章三元组稀疏矩阵转置的算法思想 ,其中的 pos 数组和那里的 cpot 数组起的是类似的作用 .

10.45

```

typedef enum {0,1,2,3,4,5,6,7,8,9} digit; // 个位数类型
typedef digit[3] num; //3 位自然数类型 ,假设低位存储在低下标 ,高位存储在高下标
void Enum_Radix_Sort(num a[],int n)// 利用计数实现基数排序 ,其中关键字为 3 位自然数 ,共有 n 个自然数
{
    int number ,pos ;
    num c[MAXSIZE];
    for(j=0;j<3;j++) // 依次对个位 ,十位和百位排序
    {
        for(i=0;i<n;i++) number[a[i][j]]++; // 计数
        for(pos[0]=0,i=1;i<n;i++)
            pos[i]=pos[i-1]+num[i]; // 把关键字的值映射为元素在排好的序列中的位置
        for(i=0;i<n;i++) // 构造有序数组 c
            c[pos[a[i][j]]++]=a[i];
        for(i=0;i<n;i++)
            a[i]=c[i];
    } //for
} //Enum_Radix_Sort

```

分析 :计数排序是一种稳定的排序方法 .正因为如此 ,它能够被用来实现基数排序 .

10.46

```
typedef struct {  
    int key;  
    int pos;  
} Shadow; // 影子序列的记录类型  
void Shadow_Sort(Rectype b[ ], Rectype &a[ ], int n) // 对元素很大的记录序列 b 进行排序, 结果放入 a 中, 不移动元素  
{  
    Shadow d[MAXSIZE];  
    for(i=0; i<n; i++) // 生成影子序列  
    {  
        d[i].key=b[i].key;  
        d[i].pos=i;  
    }  
    for(i=n-1, change=1; i>1 && change; i--) // 对影子序列执行冒泡排序  
    {  
        change=0;  
        for(j=0; j<i; j++)  
            if(d[j].key>d[j+1].key)  
            {  
                d[j]<->d[j+1];  
                change=1;  
            }  
    }  
    //for  
    for(i=0; i<n; i++) // 按照影子序列里记录的原来位置复制原序列  
        a[i]=b[d[i].pos];  
} //Shadow_Sort
```

第 124 页 共 124 页