

第1章 絮论

习题

1. 简述下列概念:数据、数据元素、数据项、数据对象、数据结构、逻辑结构、存储结构、抽象数据类型。
2. 试举一个数据结构得例子,叙述其逻辑结构与存储结构两方面得含义与相互关系。
3. 简述逻辑结构得四种基本关系并画出它们得关系图。
4. 存储结构由哪两种基本得存储方法实现?
5. 选择题
 - (1)在数据结构中,从逻辑上可以把数据结构分成()。
A.动态结构与静态结构 B. 紧凑结构与非紧凑结构
C. 线性结构与非线性结构 D. 内部结构与外部结构
 - (2)与数据元素本身得形式、内容、相对位置、个数无关得就是数据得()。
A. 存储结构 B. 存储实现
C. 逻辑结构 D. 运算实现
 - (3)通常要求同一逻辑结构中得所有数据元素具有相同得特性,这意味着()。
A. 数据具有同一特点
B. 不仅数据元素所包含得数据项得个数要相同,而且对应数据项得类型要一致
C. 每个数据元素都一样
D. 数据元素所包含得数据项得个数要相等
 - (4)以下说法正确得就是()。
A.数据元素就是数据得最小单位
B.数据项就是数据得基本单位
C. 数据结构就是带有结构得各数据项得集合
D. 一些表面上很不相同得数据可以有相同得逻辑结构
 - (5)以下与数据得存储结构无关得术语就是()。
A. 顺序队列 B. 链表 C. 有序表 D. 链栈
 - (6)以下数据结构中,()就是非线性数据结构
A. 树 B. 字符串 C. 队 D. 栈
6. 试分析下面各程序段得时间复杂度。
 - (1)

```
x = 90; y = 1 0 0;
while (y > 0)
    if (x > 100)
        { x = x - 10; y--; }
    else x++;
}
```
 - (2)

```
for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
        a[i][j] = 0;
```
 - (3)

```
s = 0;
```

```

f o r  i=0; i < n; i++)
    for (j= 0 ;j< n ; j++)
        s+= B [i] [j] ;
    sum=s;
(4) i = 1 ;
    whil e (i<=n)
        i =i* 3 ;
(5) x=0;
    f o r (i=1; i<n; i++)
        f or (j=1; j <=n- i ; j++)
            x++;
(6)x=n; / /n>1
    y =0;
    wh i l e (x≥(y+ 1 ) * (y+1))
        y++;
(1)O (1)
(2) O (m*n)
(3)O (n2)
(4 ) O (log3n)
(5) 因为 x++共执行了 n-1+n-2+……+1= n (n-1)/2, 所以执行时间为 O(n2)
(6)O()

```

第 2 章 线性表

1.选择题

(1) 一个向量第一个元素得存储地址就是 100, 每个元素得长度为 2, 则第 5 个元素得地址就是()。

- A.110 B. 108 C. 100 D. 120

(2) 在 n 个结点得顺序表中, 算法得时间复杂度就是 O(1)得操作就是()。

- A.访问第 i 个结点($1 \leq i \leq n$)与求第 i 个结点得直接前驱 ($2 \leq i \leq n$)
- 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

(3) 向一个有 127 个元素得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____得元素个数为 ()。

- A.8 B. 63.5 C.63 D. 7

(4)链接存储得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得指针
- 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得指针
- D. 分两部分, 一部分存放结点值, 另一部分存放结点所占单元数

(5)线性表若采用链式存储结构时, 要求内存中可用存储单元得地址 () .

A. 必须就是连续得 B. 部分地址必须就是连续得

C. 一定就是不连续得 D. 连续或不连续都可以

(6) 线性表 L 在 () 情况下适用于使用链式结构实现。

A. 需经常修改 L 中得结点值 B. 需不断对 L 进行删除插入

C. L 中含有大量得结点 D. L 中结点结构复杂

(7) 单链表得存储密度()。

A. 大于 1 B. 等于 1 C. 小于 1 D. 不能确定

(8) 将两个各有 n 个元素得有序表归并成一个有序表，其最少得比较次数就是()。

A. n B. 2n-1 C. 2n D. n-1

(9) 在一个长度为 n 得顺序表中，在第 i 个元素 ($1 \leq i \leq n+1$) 之前插入一个新元素时须向后移动 () 个元素。

A. n-i B. n-i+1 C. n-i-1 D. i

(10) 线性表 L = (a₁, a₂, ..., a_n)，下列说法正确得就是()。

A. 每个元素都有一个直接前驱与一个直接后继

B. 线性表中至少有一个元素

C. 表中诸元素得排列必须就是由小到大或由大到小

D. 除第一个与最后一个元素外，其余每个元素都有一个且仅有一个直接前驱与直接后继。

(11) 若指定有 n 个元素得向量，则建立一个有序单链表得时间复杂性得量级就是()。

A. O(1) B. O(n) C. O(n²) D. O(n log₂n)

(12) 以下说法错误得就是()。

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

B. 顺序存储得线性表可以随机存取

C. 由于顺序存储要求连续得存储区域，所以在存储管理上不够灵活

D. 线性表得链式存储结构优于顺序存储结构

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

A. s->next=p+1; p->next=s;

B. (*p)->next=s; (*s)->next=(*p); next->next;

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

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

(14) 在双向链表存储结构中，删除 p 所指得结点时须修改指针()。

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

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

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

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

(15) 在双向循环链表中，在 p 指针所指得结点后插入 q 所指向得新结点，其修改指针得操作就是()。

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

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

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

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

= q;

2. 算法设计题

(1) 将两个递增得有序链表合并为一个递增得有序链表。要求结果链表仍使用原来两个链表得存储空间，不另外占用其它得存储空间。表中不允许有重复得数据。

```
void MergeList_L(LinkList &La, LinkList &Lb, LinkList &Lc){  
    pa=La->next; pb=Lb->next;  
    Lc=pc=La; //用 La 得头结点作为 Lc 得头结点  
    while (pa && pb){  
        if (pa->data < pb->data) { pc->next=pa; pc=pa; pa=pa->next; }  
        else if (pa->data > pb->data) { pc->next=pb; pc=pb; pb=pb->next; }  
        else { // 相等时取 La 得元素, 删除 Lb 得元素  
            pc->next=pa; pc=pa; pa=pa->next;  
            q=pb->next; delete pb; pb=q; }  
    }  
    pc->next=pa ? pa: pb; // 插入剩余段  
    delete Lb; //释放 Lb 得头结点}
```

(2) 将两个非递减得有序链表合并为一个非递增得有序链表。要求结果链表仍使用原来两个链表得存储空间，不另外占用其它得存储空间。表中允许有重复得数据。

```
void union (LinkList & La, LinkList& Lb, LinkList& Lc, ) {  
    pa = La->next; pb = Lb->next; // 初始化  
    Lc=pc=La; //用 La 得头结点作为 Lc 得头结点  
    Lc->next = NULL;  
    while ( pa || pb ) {  
        if ( !pa ) { q = pb; pb = pb->next; }  
        else if ( !pb ) { q = pa; pa = pa->next; }  
        else if (pa->data <= pb->data ) { q = pa; pa = pa->next; }  
        else { q = pb; pb = pb->next; }  
        q->next = Lc->next; Lc->next = q; // 插入  
    }  
    delete Lb; //释放 Lb 得头结点}
```

(3) 已知两个链表 A 与 B 分别表示两个集合，其元素递增排列。请设计算法求出 A 与 B 得交集，并存放于 A 链表中。

```
void Mix(LinkList & La, LinkList& Lb, LinkList & Lc, ) {  
    pa=la->next; pb=lb->next; //设工作指针 pa 与 pb;  
    Lc=pc=La; //用 La 得头结点作为 Lc 得头结点  
    while (pa && pb)  
        if (pa->data==pb->data) //交集并入结果表中.  
            { pc->next=pa; pc=pa; pa=pa->next;  
            u=pb; pb=pb->next; delete u; }  
        else if (pa->data < pb->data) { u=pa; pa=pa->next; delete u; }  
        else { u=pb; pb=pb->next; delete u; }  
    while (pa) { u=pa; pa=pa->next; delete u; } // 释放结点空间
```

```

while( p b) {u=p b; p b=p b-> n e xt; d e l e t e u;} //释放结点空间
pc-> n e xt=n u l l ;//置链表尾标记。
d e l e t e L b; //注：本算法中也可对B表不作释放空间得处理

```

(4) 已知两个链表 A 与 B 分别表示两个集合, 其元素递增排列。请设计算法求出两个集合 A 与 B 得差集(即仅由在 A 中出现而不在 B 中出现得元素所构成得集合), 并以同样得形式存储, 同时返回该集合得元素个数。

```

v o i d D i f f e r e n c e (L i n k L i s t A , B , *n)
//A 与 B 均就是带头结点得递增有序得单链表, 分别存储了一个集合, 本算法求两集合得差集, 存储于单链表 A 中, *n 就是结果集合中元素个数, 调用时为 0
{ p =A->next; //p 与 q 分别就是链表 A 与 B 得工作指针。
q =B->next; p r e =A; //p r e 为 A 中 p 所指结点得前驱结点得指针。
w h i l e(p!=n u l l && q !=n u l l )
    i f (p->d a t a <q-> d a t a ) {p r e =p;p=p->next; *n++;} // A 链表中当前结点指
针后移。
    e l s e i f (p-> d a t a >q-> d a t a ) q=q->next; //B 链表中当前结点指针后移。
    e l s e {p r e -> n e xt=p-> n e xt; //处理 A, B 中元素值相同得结点, 应
删除。
        u=p; p=p->next; d e l e t e u;} //删除结点

```

(5) 设计算法将一个带头结点得单链表 A 分解为两个具有相同结构得链表 B、C, 其中 B 表得结点为 A 表中值小于零得结点, 而 C 表得结点为 A 表中值大于零得结点(链表 A 得元素类型为整型, 要求 B、C 表利用 A 表得结点)。

(6) 设计一个算法, 通过一趟遍历在单链表中确定值最大得结点。

```

E l e m T y p e M a x (L i n k L i s t L )
    i f (L->n e xt==N U L L ) r e t u r n N U L L ;
    pmax=L-> n e xt; //假定第一个结点中数据具有最大值
    p=L-> n e xt-> n e xt;
    w h i l e(p != N U L L ) { //如果下一个结点存在
        i f (p->d a t a > pmax->d a t a ) pmax=p;
        p=p-> n e xt;
    }
    r e t u r n pma x->d a t a ;

```

(7) 设计一个算法, 通过遍历一趟, 将链表中所有结点得链接方向逆转, 仍利用原表得存储空间。

```

v o i d i n v e r s e (L i n k L i s t & L ) {
    // 逆置带头结点得单链表 L
    p=L-> n e xt; L-> n e xt=N U L L ;
    w h i l e ( p ) {
        q=p-> n e xt; // q 指向*p 得后继
        p-> n e xt=L-> n e xt;
        L-> n e xt=p; // *p 插入在头结点之后
        p = q;
    }
}

```

(8) 设计一个算法, 删除递增有序链表中值大于 $\min k$ 且小于 $\max k$ 得所有元素(m i n

k 与 maxk 就是给定得两个参数，其值可以与表中得元素相同，也可以不同)。

```
void delete (LinkList &L, int mink, int maxk) {
    p=L->next; //首元结点
    while (p && p->data<=mink)
        { pre=p; p=p->next; } //查找第一个值≤mink 得结点
    if (p) {
        while (p && p->data<=maxk) p=p->next;
        // 查找第一个值 ≥maxk 得结点
        q=pre->next; pre->next=p; // 修改指针
        while (q!=p)
            { s=q->next; delete q; q=s; } // 释放结点空间
    } //if
}
```

(9) 已知 p 指向双向循环链表中得一个结点, 其结点结构为 data、prior、next 三个域, 写出算法 change (p), 交换 p 所指向得结点与它得前缀结点得顺序。

知道双向循环链表中得一个结点, 与前驱交换涉及到四个结点 (p 结点, 前驱结点, 前驱得前驱结点, 后继结点) 六条链。

```
void Exchange(LinkList p)
//p 就是双向循环链表中得一个结点, 本算法将 p 所指结点与其前驱结点交换。
{ q=p->llink;
    q->llink->rlink=p; // p 得前驱得前驱之后继为 p
    p->llink=q->llink; // p 得前驱指向其前驱得前驱。
    q->rlink=p->rlink; // p 得前驱得后继为 p 得后继。
    q->llink=p; // p 与其前驱交换
    p->rlink->llink=q; // p 得后继得前驱指向原 p 得前驱
    p->rlink=q; // p 得后继指向其原来得前驱
} //算法 exchange 结束。
```

(10)已知长度为 n 得线性表 A 采用顺序存储结构, 请写一时间复杂度为 O (n)、空间复杂度为 O (1) 得算法, 该算法删除线性表中所有值为 item 得数据元素。

[题目分析] 在顺序存储得线性表上删除元素, 通常要涉及到一系列元素得移动 (删第 i 个元素, 第 i+1 至第 n 个元素要依次前移)。本题要求删除线性表中所有值为 item 得数据元素, 并未要求元素间得相对位置不变。因此可以考虑设头尾两个指针 (i=1, j=n), 从两端向中间移动, 凡遇到值 item 得数据元素时, 直接将右端元素左移至值为 item 得数据元素位置。

```
void Delete (ElementType A[], int n)
// A 就是有 n 个元素得一维数组, 本算法删除 A 中所有值为 item 得元素。
{i=1; j=n; // 设置数组低、高端指针(下标)。
while (i<j)
    {while (i<j && A[i]!=item) i++; //若值不为 item, 左移指
    针。
        if (i<j) while (i<j && A[j]==item) j--; //若右端元素值为 i t e m,
    指针左移
        if (i<j) A[i++]=A[j--];
    }
}
```

[算法讨论] 因元素只扫描一趟，算法时间复杂度为 $O(n)$.删除元素未使用其它辅助空间，最后线性表中得元素个数就是 j 。

第3章 栈与队列

习题

1. 选择题

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

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

1

(2) 若已知一个栈得入栈序列就是 1, 2, 3, ..., n, 其输出序列为 p1, p2, p3, ..., pn, 若 $p_1=n$, 则 p_i 为()。

- A. i B. n-i C. n-i+1

D. 不确定

(3)数组 Q[n] 用来表示一个循环队列,f 为当前队列头元素得前一位置, r 为队尾元素得位置, 假定队列中元素得个数小于 n, 计算队列中元素个数得公式为 ()。

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

(4) 链式栈结点为: (data, link), top 指向栈顶、若想摘除栈顶结点, 并将删除结点得值保存到 x 中, 则应执行操作 ()。

- A. $x=top->data; top=top->link;$ B. $top=top->link; x=top->data;$
C. $x=top; top=top->link;$ D. $x=top->link;$

(5) 设有一个递归算法如下

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

则计算 fact (n) 需要调用该函数得次数为 ()。

- A. n+1 B. n-1 C. n D. n+

2

(6) 栈在 () 中有所应用.

- A. 递归调用 B. 函数调用 C. 表达式求值 D. 前三个选项都有

(7)为解决计算机主机与打印机间速度不匹配问题,通常设一个打印数据缓冲区.主机将要输出得数据依次写入该缓冲区, 而打印机则依次从该缓冲区中取出数据.该缓冲区得逻辑结构应该就是 ()。

- A. 队列 B. 栈 C. 线性表 D. 有序表

(8) 设栈 S 与队列 Q 得初始状态为空, 元素 e1, e2, e3, e4, e5 与 e6 依次进入栈 S, 一个元素出栈后即进入 Q, 若 6 个元素出队得序列就是 e2, e4, e3, e6, e5 与 e1, 则栈 S 得容量至少应该就是 ()。

- A. 2 B. 3 C. 4 D. 6

(9) 在一个具有 n 个单元得顺序栈中, 假设以地址高端作为栈底, 以 top 作为栈顶指

针，则当作进栈处理时，top得变化为（）。

- A. top不变 B. top=0 C. top-- D. top++

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

- A. 线性表得顺序存储结构 B. 队列
C. 线性表得链式存储结构 D. 栈

(11) 用链接方式存储得队列，在进行删除运算时（）。

- A、仅修改头指针 B、仅修改尾指针
C、头、尾指针都要修改 D、头、尾指针可能都要修改

(12) 循环队列存储在数组 A[0.., m] 中，则入队时得操作为（）。

- A、rear=rear+1 B、rear=(rear+1)%(m-1)
C、rear=(rear+1)%m D、rear=(rear+1)%(m+1)

(13) 最大容量为 n 得循环队列，队尾指针就是 rear，队头就是 front，则队空得条件就是（）。

- A、(rear+1)%n==front B、rear==front
C、rear+1==front D、(rear-1)%n==front

(14) 栈与队列得共同点就是（）。

- A、都就是先进先出 B、都就是先进后出
C、只允许在端点处插入与删除元素 D、没有共同点

(15) 一个递归算法必须包括（）。

- A、递归部分 B、终止条件与递归部分
C、迭代部分 D、终止条件与迭代部分

(2) 回文就是指正读反读均相同得字符序列，如“ab b a”与“a b d ba”均就是回文，但“g o o d”不就是回文。试写一个算法判定给定得字符向量是否为回文。（提示：将一半字符入栈）

```
根据提示，算法可设计为：▲ //以下为顺序栈得存储结构定义▲ #define StackSize 100 //假定
预分配得栈空间最多为 100 个元素▲ type def char DataT ype; //假定栈元素得数据类型为字符
type def struct {▲ DataT ype data[StackSize]; ▲ int top; ▲ } SeqStack; ▲ int
IsHuiwen ( char *t)▲ { //判断 t 字符向量是否为回文，若就是，返回 1，否则返回 0
    SeqStack s;
    int i, len;
    char temp;
    InitStack (&s);
    len=strlen(t); //求向量长度
    for ( i=0; i<len/2; i++) //将一半字符入栈
        Push( &s, t [i]);
    while ( !EmptyStack ( &s))▲ { // 每弹出一个字符与相应字符比较
        Pop(&s, temp);
        if ( t[i] != temp)
            return 0;
    }
    return 1;
}
```

```

temp=Pop (&s) ;

if ( temp!=s[i] )   return 0 ; // 不等则返回 0

else i++;

}

return 1 ; // 比较完毕均相等则返回 1
}

```

(3) 设从键盘输入一整数得序列: $a_1, a_2, a_3, \dots, a_n$, 试编写算法实现:用栈结构存储输入得整数,当 $a_i \neq -1$ 时, 将 a_i 进栈;当 $a_i=-1$ 时, 输出栈顶整数并出栈。算法应对异常情况(入栈满等) 给出相应得信息。

```

#define maxsize 栈空间容量

void InOutS(int s[maxsize])
//s 就是元素为整数得栈, 本算法进行入栈与退栈操作。
{int top=0; //top 为栈顶指针, 定义 top=0 时为栈空。
for (i=1; i<=n; i++) //n 个整数序列作处理。
{scanf(" %d", &x); //从键盘读入整数序列。
if (x!= -1) // 读入得整数不等于-1 时入栈。
if (top==maxsize-1) {printf(" 栈满\n"); exit(0); } else s[++top]=x;
//x 入栈。
else //读入得整数等于-1 时退栈。
{if (top==0) {printf(" 栈空\n"); exit(0); } else printf(" 出
栈元素就是%d\n", s[top-1]);
}
}
//算法结束。
}

```

(4)从键盘上输入一个后缀表达式, 试编写算法计算表达式得值. 规定: 逆波兰表达式得长度不超过一行, 以\$符作为输入结束, 操作数之间用空格分隔, 操作符只可能有+、-、*、/四种运算。例如: 2 3 4 3 4 + 2 * \$。

[题目分析] 逆波兰表达式(即后缀表达式)求值规则如下:设立运算数栈OPND, 对表达式从左到右扫描(读入), 当表达式中扫描到数时, 压入OPND栈。当扫描到运算符时, 从OPND退出两个数, 进行相应运算, 结果再压入OPND栈。这个过程一直进行到读出表达式结束符\$, 这时OPND栈中只有一个数, 就是结果。

```

float expr()
// 从键盘输入逆波兰表达式, 以 '$' 表示输入结束, 本算法求逆波兰式表达式得值。
{float OPND[30]; // OPND 就是操作数栈。
init(OPND); //两栈初始化。
float num=0.0; //数字初始化。
scanf(" %c", &x); // x 就是字符型变量。
while(x!='$')
{switch
{case '0'<=x<='9': while((x=='0' && x<='9')||x=='.' )
// 拼数
if (x=='.') //处理整数
num=num*10+(ord(x)-ord('0'));
scanf(" %c", &x);
}
}
}

```

```

    e l s e           // 处理小数部分。
        ( sca l e=1 0 、 0; s c a n f ( “%c” , &x) ;
        w h i l e (x>=' 0' &&x<=' 9' )
        { n u m=n u m+ ( o r d ( x ) -o r d ( ‘ 0’ ) /
s c a l e;
        s c a l e =s c a l e*1 0 ;   s c a n f ( “%
c ” , &x); }
    } / / e l s e
        p u s h ( O P N D , n u m) ; n u m=0, 0; / /

```

数压入栈，下个数初始化

```

c a s e   x = ‘ ’ : b r e a k ;      // 遇空格，继续读下一个字符。
c a s e   x=‘ +’ : p u s h ( O P N D , p o p ( O P N D ) +p o p ( O P N D )); b r e a k ;
c a s e   x = ‘ —’ : x 1=p o p ( O P N D ) ; x 2=p o p ( O P N D ); p u s h ( O P N D , x 2
-x 1 ) ; b r e a k ;
c a s e   x = ‘ *’ : p u s h ( O P N D , p o p ( O P N D ) *p o p ( O P N D )); b r e a k ;
c a s e   x = ‘ /’ : x 1=p o p ( O P N D ) ; x 2=p o p ( O P N D ); p u s h ( O P N D , x 2/x 1 ) ; b r e
a k ;
d e f a u l t:           // 其它符号不作处理。
} / / 结束 s w i t c h
s c a n f ( “%c” , &x); // 读入表达式中下一个字符。
} / / 结束 w h i l e ( x ! = ‘ $’ )
p r i n t f ( “后缀表达式得值为%f” , p o p ( O P N D ));
} / / 算法结束。

```

[算法讨论] 假设输入得后缀表达式就是正确得，未作错误检查。算法中拼数部分就是核心。若遇到大于等于‘0’且小于等于‘9’得字符，认为就是数。这种字符得序号减去字符‘0’得序号得出数。对于整数，每读入一个数字字符，前面得到得部分数要乘上10再加新读入得数得到新得部分数。当读到小数点，认为数得整数部分已完，要接着处理小数部分。小数部分得数要除以10(或10得幂数)变成十分位，百分位，千分位数等等，与前面部分数相加。在拼数过程中，若遇非数字字符，表示数已拼完，将数压入栈中，并且将变量num恢复为0，准备下一个数。这时对新读入得字符进入‘+’、‘-’、‘*’、‘/’及空格得判断，因此在结束处理数字字符得case后，不能加入break语句。

(5)假设以I与O分别表示入栈与出栈操作。栈得初态与终态均为空，入栈与出栈得操作序列可表示为仅由I与O组成得序列，称可以操作得序列为合法序列，否则称为非法序列。

①下面所示得序列中哪些就是合法得？

A、 I O I I O I O O B、 I O O I O I I O C、 I I I O I O I O D、 I I
I O O I O O

②通过对①得分析，写出一个算法，判定所给得操作序列就是否合法。若合法，返回true，否则返回false(假定被判定得操作序列已存入一维数组中)。

①A与D就是合法序列，B与C就是非法序列。

②设被判定得操作序列已存入一维数组A中。

```

i n t J u d g e ( c h a r  A [ ] )
// 判断字符数组A中得输入输出序列就是否就是合法序列。如就是，返回t r u e ,
否则返回f a l s e .

```

```

    i=0;                                // i 为下标。
    j=k=0;                                // j 与 k 分别为 I 与字母 O 得得个数。
while (A[i] != '\0') // 当未到字符数组尾就作。
    {
        switch(A[i])
        {
            case 'I' : j++; break; // 入栈次数增 1。
            case 'O' : k++; if(k>j) {printf ("序列非法\n"); exit(0);}
        }
        i++; // 不论 A[i] 就是 'I' 或 'O', 指针 i 均后移。}
        if (j!=k) {printf ("序列非法\n"); return (false);}
        else {printf ("序列合法\n"); return (true);}
    } // 算法结束。

```

[算法讨论] 在入栈出栈序列(即由 'I' 与 'O' 组成得字符串)得任一位置, 入栈次数 ('I' 得个数) 都必须大于等于出栈次数 (即 'O' 得个数), 否则视作非法序列, 立即给出信息, 退出算法。整个序列 (即读到字符数组中字符串得结束标记 '\0'), 入栈次数必须等于出栈次数 (题目中要求栈得初态与终态都为空), 否则视为非法序列。

(6) 假设以带头结点得循环链表表示队列, 并且只设一个指针指向队尾元素站点(注意不设头指针), 试编写相应得置空队、判队空、入队与出队等算法。

算法如下:

```

// 先定义链队结构:

type def struct queuenode {
    Datatype data;
    struct queuenode *next; } QueueNode; // 以上就是结点类型得定义

typedef struct {
    queuenode *rear; } LinkQueue; // 只设一个指向队尾元素得指针 (1) 置空队 void
InitQueue (LinkQueue *Q) { // 置空队: 就就是使头结点成为队尾元素 QueueNode *
s; Q->rear = Q->rear->next; // 将队尾指针指向头结点

while (Q->rear != Q->rear->next) // 当队列非空, 将队中元素逐个出队
{s=Q->rear->next; Q->rear->next=s->next; free(s); } //
/ 回收结点空间
}

```

```

(2) 判队空 int EmptyQueue (LinkQueue *Q) { // 判队空
// 当头结点得 next 指针指向自己时为空队 return Q->rear->next->next==Q->re
ar->next;
}

```

} (3) 入队

```
void EnQueue ( LinkQueue *Q, DataType x)
{ // 入队
    // 也就是在尾结点处插入元素
    QueueNode *p= (QueueNode *) malloc (sizeof (QueueNode));
    p->data= x;
    p->next= Q->rear->next; // 初始化新结点并链入
    Q->rear->next=p; // 将尾指针移至新结点
}

// (4) 出队
DataType DeQueue ( LinkQueue *Q) // 出队, 把头结点之后得元素摘下
{
    QueueNode *p; // 头结点
    if (EmptyQueue (Q)) // 队列空
        Error ("Queue underflow");
    p= Q->rear->next->next; // p 指向将要摘下得结点
    x= p->data; // 保存结点中数据
    if (p== Q->rear) // 当队列中只有一个结点时, p 结点出队后, 要将队尾指针指向头结点
        Q->rear = Q->rear->next; // Q->rear->next=p->next;
    else // 摘下结点 p
        free (p); // 释放被删结点
    return x;
}
```

(7) 假设以数组 $Q[m]$ 存放循环队列中得元素, 同时设置一个标志 tag , 以 $tag == 0$ 与 $tag == 1$ 来区别在队头指针($front$)与队尾指针($rear$)相等时, 队列状态为“空”还是“满”。试编写与此结构相应得插入($enqueue$)与删除($dequeue$)算法。

【解答】

循环队列类定义

```
#include <assert.h>
template <class Type> class Queue { // 循环队列得类定义
public:
    Queue ( int =10 );
    ~Queue ( ) { delete [] Q; }
    void EnQueue ( Type & item );
    Type DeQueue ( );
    Type GetFront ( );
    void MakeEmpty ( ); // 置空队列
    int IsEmpty ( ) const { return front == rear && tag == 0; } // 判队列空否
    int IsFull ( ) const { return front == rear && tag == 1; } // 判队列满否
private:
    int rear, front, tag; // 队尾指针、队头指针与队满标志
```

```

    * Type * Q; //存放队列元素得数组
    int m; //队列最大可容纳元素个数
}

构造函数
template <class Type>
Queue<Type>:: Queue( int sz ) : rear( 0 ), front( 0 ), tag( 0 ), m( sz ) {
//建立一个最大具有m个元素得空队列。
    Q = new Type [m]; //创建队列空间
    assert( Q != 0 ); //断言：动态存储分配成功与否
}

插入函数
template <class Type>
void Queue<Type>:: EnQueue( Type &item ) {
    assert( ! IsFull() ); //判队列就是否不满，满则出错处理
    rear = ( rear + 1 ) % m; //队尾位置进1，队尾指针指示实际队尾位置
    Q[rear] = item; //进队列
    tag = 1; //标志改1,表示队列不空
}

删除函数
template <class Type>
Type Queue<Type>:: DeQueue() {
    assert( ! IsEmpty() ); //判断队列就是否不空，空则出错处理
    front = ( front + 1 ) % m; //队头位置进1，队头指针指示实际队头得前一位置
    tag = 0; //标志改0，表示栈不满
    return Q[front]; //返回原队头元素得值
}

读取队头元素函数
template <class Type>
Type Queue<Type>:: GetFront() {
    assert( ! IsEmpty() ); //判断队列就是否不空，空则出错处理
    return Q[ ( front + 1 ) % m ]; //返回队头元素得值
}

(8) 如果允许在循环队列得两端都可以进行插入与删除操作。要求：
① 写出循环队列得类型定义；
② 写出“从队尾删除”与“从队头插入”得算法。
[题目分析] 用一维数组 v[0..M-1] 实现循环队列，其中 M 就是队列长度。设队头指针 front 与队尾指针 rear，约定 front 指向队头元素得前一位置，rear 指向队尾元素。定义 front=rear 时为队空，(rear+1)%M=front 为队满。约定队头端入队向下标小得方向发展，队尾端入队向下标大得方向发展。
(1) #define M 队列可能达到得最大长度
typedef struct {
    elemtp data [M];
    int front, rear;
} cqueue;

```

```

(2) elemtp delqueue (cycqueue Q)
// Q 就是如上定义得循环队列, 本算法实现从队尾删除, 若删除成功, 返回被删除元素,
否则给出出错信息。
{ if (Q, front==Q, rear) {printf("队列空"); exit(0);}
  Q, rear=(Q, rear+1)%M; //修改队尾指针。
  return (Q, data[(Q, rear+1+M)%M]); //返回出队元素。
} // 从队尾删除算法结束

void enqueue (cycqueue Q, elemtp x)
// Q 就是顺序存储得循环队列, 本算法实现“从队头插入”元素 x。
{if (Q, rear == (Q, front-1+M)%M) {printf("队满"); exit(0);}
  Q, data [Q, front] =x; //x 入队列
  Q, front= (Q, front-1+M) % M; // 修改队头指针。
} // 结束从队头插入算法。

```

(9) 已知 Ackerman 函数定义如下:

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

① 写出计算 Ack (m, n) 得递归算法, 并根据此算法给出出 Ack (2, 1) 得计算过程。

② 写出计算 Ack (m, n) 得非递归算法。

```

int Ack (int m, n)
{ if (m==0) return (n+1);
  else if (m!=0&&n==0) return (Ack(m-1, 1));
  else return (Ack(m-1, Ack(m, m-1)));
} //算法结束

```

(1) Ack (2, 1) 得计算过程

```

Ack (2, 1) =Ack (1, Ack(2, 0)) //因 m<>0, n>0 而得
=Ack (1, Ack(1, 1)) //因 m<>0, n=0 而得
=Ack (1, Ack (0, Ack(1, 0))) // 因 m<>0, n <> 0 而得

```

得

```

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

```

(2) int Ackerman (int m, int n)

```

{int akm [M] [N]; int i, j;
for (j=0; j<N; j++) akm [0][j] =j+1;

```

```

for ( i = 1; i < m; i++)
    { akm[i][0] = a km[i-1][1];
      for ( j=1; j < N; j++)
          akm [i] [j] = a km [i-1][a km [i][j-1]];
    }
return ( akm[m] [n]);
} //算法结束

```

(10) 已知 f 为单链表得表头指针, 链表中存储得都就是整型数据,试写出实现下列运算得递归算法:。

- ① 求链表中得最大整数;
- ② 求链表得结点个数;
- ③ 求所有整数得平均值。

```

#include <iostream.h> //定义在头文件"Re curv e List.h" 中
class List;
class ListNode { ... }; //链表结点类
friend class List;
private:
    int data; //结点数据
    ListNode *link; //结点指针
    ListNode ( const int item ) : data ( item ), link ( NULL ) { } //构造函数
};

class List { ... }; //链表类
private:
    ListNode *first, *current;
    int Max ( ListNode *f );
    int Num ( ListNode *f );
    float Avg ( ListNode *f, int& n );
public:
    List () : first ( NULL ), current ( NULL ) { } //构造函数
    ~List () { ... }; //析构函数
    ListNode* NewNode ( const int item ); //创建链表结点, 其值为 item
    void NewList ( const int retValue ); //建立链表, 以输入 retValue 结束
    void PrintList (); //输出链表所有结点数据
    int GetMax () { return Max ( first ); } //求链表所有数据得最大值
    int GetNum () { return Num ( first ); } //求链表中数据个数
    float GetAvg () { return Avg ( first ); } //求链表所有数据得平均
    ...
};

ListNode* List::NewNode ( const int item ) { //创建新链表结点
    ListNode *newnode = new ListNode ( item );
    return newnode;
}

void List::NewList ( const int retValue ) { //建立链表, 以输入 retValue
    ...
}

```

```

if (rst == NULL) { int val; ListNode *q;
    cout << "Input your data:\n"; cin >> value; //提示
    if (cin >> value) { //输入有效
        q = NewNode (value); //建立包含 value 的新结点
        if (first == NULL) first = current = q; //空表时，新结点成为链表第一个结点
        else { current->link = q; current = q; } //非空表时，新结点链入链尾
        cin >> value; //再输入
    }
    current->link = NULL; //链尾封闭
}
void List::PrintList () { //输出链表
    cout << "\nThe List is :\n";
    ListNode *p = first;
    while (p != NULL) { cout << p->data << ' '; p = p->link; }
    cout << '\n';
}

int List::Max (ListNode *f) { //递归算法：求链表中得最大值
    if (f->link == NULL) return f->data; //递归结束条件
    int temp = Max (f->link); //在当前结点得后继链表中求最大值
    if (f->data > temp) return f->data; //如果当前结点得值还要大，返回当前检点值
    else return temp; //否则返回后继链表中得最大值
}

int List::Num (ListNode *f) { //递归算法：求链表中结点个数
    if (f == NULL) return 0; //空表，返回 0
    return 1 + Num (f->link); //否则，返回后继链表结点个数加 1
}

float List::Avg (ListNode *f, int& n) { //递归算法：求链表中所有元素得平均值
    if (f->link == NULL) //链表中只有一个结点，递归结束条件
        { n = 1; return (float) (f->data); }
    else { float Sum = Avg (f->link, n)*n; n++; return (f->data + Sum) / n; }
}

#include "RecurveList.h" //定义在主文件中
int main (int argc, char* argv[]) {
    List test; int finish;
    cout << "Input build list end data: ";
    cin >> finish; //输入建表结束标志数据
    test, NewList (finish); //建立链表
    test.PrintList (); //打印链表
    cout << "\nThe Max is : " << test.GetMax ();
    cout << "\nThe Num is: " << test.GetNum ();
}

```

```

    >>> cout << "\nThe Ave is : " << test.GetAve() << '\n';
    >>> print( "Hello World! \n" );
    >>> return 0;
}

```

第4章 串、数组与广义表

习题

1. 选择题

- (1) 串就是一种特殊得线性表, 其特殊性体现在()。
- A. 可以顺序存储 B. 数据元素就是一个字符
 C. 可以链式存储 D. 数据元素可以就是多个字符若
- (2) 串下面关于串得得叙述中,() 就是不正确得?
- A. 串就是字符得有限序列 B. 空串就是由空格构成得串
 C. 模式匹配就是串得一种重要运算 D. 串既可以采用顺序存储, 也可以采用链式存储
- (3) 串 “ababaaababaa”得 next 数组为()。
- A. 9 B. 2 C. 0 4 5 6 D. 4 5
- (4) 串 “a b a b a a b a b” 得 nextv a l 为 ()。
- A. 010104101 B. 0 1 0 1 0 2 1 0 1 C. 0 1 0 1 0 0 0 1 1 D. 010
 1 0 1 0 1 1
- (5) 串得长度就是指()。
- A. 串中所含不同字母得个数 B. 串中所含字符得个数
 C. 串中所含不同字符得个数 D. 串中所含非空格字符得个数
- (6) 假设以行序为主序存储二维数组 A=array[1 .. 100, 1 .. 100], 设每个数据元素占 2 个存储单元, 基址址为 10, 则 LOC [5,5]= ()。
- A. 808 B. 818 C. 10 10 D. 10
 2 0
- (7) 设有数组 A [i, j] , 数组得每个元素长度为 3 字节, i 得值为 1 到 8, j 得值为 1 到 10, 数组从内存首地址 BA 开始顺序存放, 当用以列为主存放时, 元素 A[5, 8] 得存储首地址为()。
- A. BA+141 B. BA+180 C. BA+222 D. BA+
 2 2 5
- (8) 设有一个 10 阶得对称矩阵 A, 采用压缩存储方式, 以行序为主存储, a_{11} 为第一元素, 其存储地址为 1, 每个元素占一个地址空间, 则 a_{85} 得地址为 ()。
- A. 13 B. 33 C. 18 D. 40
- (9) 若对 n 阶对称矩阵 A 以行序为主序方式将其下三角形得元素 (包括主对角线上所有元素) 依次存放于一维数组 B [1 .. (n(n+1)) / 2] 中, 则在 B 中确定 a_{ij} ($i < j$) 得位置 k 得关系为 ()。
- A. $i * (i-1) / 2 + j$ B. $j * (j-1) / 2 + i$ C. $i * (i+1) / 2 + j$ D. $j * (j + 1) / 2 + i$

(10) A [N, N]就是对称矩阵, 将下面三角(包括对角线)以行序存储到一维数组 T[N (N+1)/2] 中, 则对任一上三角元素 a[i] [j] 对应 T [k] 得下标 k 就是 ()。

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

(11) 设二维数组 A[1,, m, 1,, n] (即 m 行 n 列)按行存储在数组 B [1,, m*n] 中, 则二维数组元素 A[i, j] 在一维数组 B 中得下标为 ()。

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

(12) 数组 A [0,, 4, -1,, -3, 5,, 7] 中含有元素得个数()。

- A. 5 5 B. 45 C. 36 D. 16

(13) 广义表 A=(a,b,(c,d), (e, (f , g))), 则 Head(Tail(Head(Tail(Tail(A)))) 得值为 ()。

- A. (g) B. (d) C. c D. d

(14) 广义表 ((a,b,c, d)) 得表头就是 (), 表尾就是()。

- A. a B. () C. (a, b, c,d) D. (b, c, d)

(15) 设广义表 L= ((a, b,c)), 则 L 得长度与深度分别为()。

- A. 1 与 1 B. 1 与 3 C. 1 与 2 D. 2 与 3

(1) 已知模式串 t= ‘ab ca a bb ab c a b’ 写出用 KMP 法求得得每个字符对应得 next 与 nextval 函数值。

模式串 t 得 next 与 nextval 值如下:

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

(2) 设目标为 t= “a b ca a b b a b c a b a acbabc a ”, 模式为 p= “a b cab a a”

- ① 计算模式 p 得 nextval 函数值;
 ② 不写出算法, 只画出利用 KMP 算法进行模式匹配时每一趟得匹配过程。

① p 得 nextval 函数值为。(p 得 next 函数值为)。

② 利用 KMP(改进得 nextval) 算法, 每趟匹配过程如下:

第一趟匹配: a b caabbabc a b a acb a c ba
 ab c a b (i =5, j =5)

第二趟匹配: abc a a b ba b c abaac b a cba
 a bc (i =7, j =3)

第三趟匹配: a b caa b ba b cabaacb a cba
 a (i =7, j =1)

第四趟匹配: a b c a abb a b c abaac b a cba
 (成功) abcabaa (i =15, j =8)

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

- ① 存放该数组所需多少单元?
- ② 存放数组第 4 列所有元素至少需多少单元?
- ③ 数组按行存放时, 元素 A[7, 4] 得起始地址就是多少?
- ④ 数组按列存放时, 元素 A[4, 7] 得起始地址就是多少?

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

- (1) 242 (2) 22 (3) s+182 (4) s+142

(4) 请将香蕉 b anan a 用工具 H()—He a d(), T()—Tai l () 从 L 中取出。

L= (ap p le, (orange, (strawberry, (banana)), p ea c h), pear)

H(H (T (H (T (H (T (L)))))))

(5) 写一个算法统计在输入字符串中各个不同字符出现得频度并将结果存入文件 (字符串中得合法字符为 A-Z 这 26 个字母与 0—9 这 10 个数字)。

```
void Count ()
//统计输入字符串中数字字符与字母字符得个数。
{ int i, num[36];
char ch;
for (i=0; i < 36; i++) num [i] = 0; // 初始化
while ((ch=getchar ()) != '#') // '#' 表示输入字符串结束.
    if ('0' <= ch <= '9') { i = ch - 48; num [i]++; } // 数字字符
    else if ('A' <= ch <= 'Z') { i = ch - 65 + 10; num [i]++; } // 字母字符
for (i=0; i < 10; i++) // 输出数字字符得个数
    printf ("数字%d 得个数=%d\n", i, num [i]);
for (i=10; i < 36; i++) // 求出字母字符得个数
    printf ("字母字符%c 得个数=%d\n", i + 55, num [i]);
}// 算法结束.
```

(6) 写一个递归算法来实现字符串逆序存储, 要求不另设串存储空间。

[题目分析] 实现字符串得逆置并不难, 但本题 “要求不另设串存储空间” 来实现字符串逆序存储, 即第一个输入得字符最后存储, 最后输入得字符先存储, 使用递归可容易做到。

```
void InvertStore (char A[])
//字符串逆序存储得递归算法。
{ char ch;
static int i = 0; //需要使用静态变量
scanf ("%c", &ch);
if (ch != ' ') //规定' '就是字符串输入结束标志
    {InvertStore (A);
    A[i++] = ch;//字符串逆序存储
    }
A[i] = '\0'; // 字符串结尾标记
}// 结束算法 InvertStore.
```

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

[题目分析] 本题就是字符串得插入问题, 要求在字符串 s 得 pos 位置, 插入字符串 t。

首先应查找字符串 s 得 pos 位置, 将第 pos 个字符到字符串 s 尾得子串向后移动字符串 t 得长度, 然后将字符串 t 复制到字符串 s 得第 pos 位置后。

对插入位置 pos 要验证其合法性, 小于 1 或大于串 s 得长度均为非法, 因题目假设给字符串 s 得空间足够大, 故对插入不必判溢出。

```
void insert(char *s, char *t, int pos)
// 将字符串 t 插入字符串 s 得第 pos 个位置。
{int i=1, x=0; char *p=s, *q=t; // p, q 分别为字符串 s 与 t
得工作指针
    if(pos < 1) {printf("pos 参数位置非法\n"); exit(0);}
    while(*p != '\0' && i < pos) {p++; i++;} //查 pos 位置
        // 若 pos 小于串 s 长度, 则查到 pos 位置时, i=pos。
        if (*p == ' ') {printf("%d 位置大于字符串 s 的长度", pos);
exit(0);}
    else //查找字符串得尾
        while(*p != '\0') {p++; i++;} //查到尾时, i 为字符 '\0' 得下标, p 也指向 '\0'.
    while (*q != '\0') {q++; x++;} //查找字符串 t 得长度 x, 循环结束时 q 指向 '\0'.
    for (j=i; j>=pos; j--) {*(p+x)=*p; p--;} //串 s 得 pos 后得子串右移, 空出串 t 得位置。
    q--; //指针 q 回退到串 t 得最后一个字符
    for (j=1; j<=x; j++) *p--=*q--; //将 t 串插入到 s 得 pos 位
置上
```

[算法讨论] 串 s 得结束标记 ('\0') 也后移了, 而串 t 得结尾标记不应插入到 s 中。

(8)已知字符串 S1 中存放一段英文, 写出算法 format(s1, s2, s3, n), 将其按给定得长度 n 格式化成两端对齐得字符串 S2, 其多余得字符送 S3。

[题目分析] 本题要求字符串 s1 拆分成字符串 s2 与字符串 s3, 要求字符串 s2 “按给定长度 n 格式化成两端对齐得字符串”, 即长度为 n 且首尾字符不得为空格字符。算法从左到右扫描字符串 s1, 找到第一个非空格字符, 计数到 n, 第 n 个拷入字符串 s2 得字符不得为空格, 然后将余下字符复制到字符串 s3 中。

```
void format(char *s1, *s2, *s3)
//将字符串 s1 拆分成字符串 s2 与字符串 s3, 要求字符串 s2 就是长 n 且两端对齐
{char *p=s1, *q=s2;
int i=0;
while (*p == ' ' && *p == '\0') p++; //滤掉 s1 左端空格
if (*p == '\0') {printf("字符串 s1 为空串或空格串\n"); exit(0);}
while (*p != '\0' && i < n) {*q=*p; q++; p++; i++;} //字符串
s1 向字符串 s2 中复制
if (*p == '\0') {printf("字符串 s1 没有%d 个有效字符\n", n); exit(0);}
if (*(--q) == ' ') //若最后一个字符为空格, 则需向后找到第一个非空格
字符
    {p--; //p 指针也后退
```

```

    wh i l e (*p==' ' && * p !=' \0' ) p++; //往后查找一个非空格字符
作串 s2 得尾字符
    i f (*p=='\0' ) {pr i nt f (" s1 串没有%d 个两端对齐得字符串\n" , n);
ex i t(0) ;}
    * q= * p; // 字符串 s2 最后一个非空字符
    * (++q) =' \0'; //置 s2 字符串结束标记
    }
    * q=s 3; p++; //将 s1 串其余部分送字符串 s3。
    w h i l e (*p !=' \0') {*q= * p; q++; p++;}
    * q=' \0'; //置串 s3 结束标记
}

```

(9) 设二维数组 $a[1..m, 1..n]$ 含有 $m*n$ 个整数。

- ① 写一个算法判断 a 中所有元素是否互不相同？输出相关信息(y e s/ n o)；
- ② 试分析算法得时间复杂度。

[题目分析] 判断二维数组中元素是否互不相同，只有逐个比较，找到一对相等得元素，就可结论为不就是互不相同。如何达到每个元素同其它元素比较一次且只一次？在当前行，每个元素要同本行后面得元素比较一次（下面第一个循环控制变量 p 得 for 循环），然后同第 $i+1$ 行及以后各行元素比较一次，这就是循环控制变量 k 与 p 得二层 for 循环。

```

int JudgE qua l ( i ng a[m] [n], i nt m, n)
//判断二维数组中所有元素是否互不相同，如就是，返回 1；否则，返回 0。
{for (i=0; i <m; i++)
    for (j=0; j <n-1; j++)
        ( f or ( p=j+1; p <n; p++) //与同行其它元素比较
            i f (a[ i ] [j] ==a [i] [p] ) {pr i nt f (" n o " ); re t u rn(0); }
        //只要有一个相同得，就结论不就是互不相同
        for (k=i+1;k<m; k++) //与第 i+1 行及以后元素比较
            for (p=0; p <n;p++)
                if(a[ i ] [j]==a[k] [p] ) {pr i nt f (" no " ); re t u rn(0) ; }
    } / / for(j=0; j <n-1; j++)
pr i nt f(y e s " ); re t u rn(1); //元素互不相同
} // 算法 Ju dg EQu a l 结束

```

(2) 二维数组中得每一个元素同其它元素都比较一次，数组中共 $m*n$ 个元素，第 1 个元素同其它 $m*n - 1$ 个元素比较，第 2 个元素同其它 $m*n - 2$ 个元素比较，……，第 $m*n - 1$ 个元素同最后一个元素($m*n$)比较一次，所以在元素互不相等时总得比较次数为 $(m*n - 1) + (m*n - 2) + \dots + 2 + 1 = (m*n)(m*n - 1)/2$ 。在有相同元素时，可能第一次比较就相同，也可能最后一次比较时相同，设在 $(m*n - 1)$ 个位置上均可能相同，这时得平均比较次数约为 $(m*n)(m*n - 1)/4$ ，总得时间复杂度就是 $O(n^4)$ 。

(10) 设任意 n 个整数存放于数组 $A(1..n)$ 中，试编写算法，将所有正数排在所有负数前面（要求算法复杂性为 $O(n)$ ）。

[题目分析] 本题属于排序问题，只就是排出正负，不排出大小。可在数组首尾设两个指针 i 与 j ， i 自小至大搜索到负数停止， j 自大至小搜索到正数停止。然后 i 与 j 所指数据交换，继续以上过程，直到 $i=j$ 为止。

```

void Arrang e (int A[], int n)
//n 个整数存于数组 A 中，本算法将数组中所有正数排在所有负数得前面

```

```

{int i=0, j=n-1, x; //用类 C 编写, 数组下标从 0 开始
while (i < j)
    {while (i < j && A[i] > 0) i++;
     while (i < j && A[j] < 0) j--;
     if (i < j) {x=A[i]; A[i]=A[j]; A[j]=x;} //交换 A[i] 与
A[j]
    }
} //算法 Ar r ange 结束、

```

[算法讨论] 对数组中元素各比较一次, 比较次数为 n 。最佳情况 (已排好, 正数在前, 负数在后) 不发生交换, 最差情况 (负数均在正数前面) 发生 $n/2$ 次交换。用类 c 编写, 数组界偶就是 $0.., n-1$ 。空间复杂度为 $O(1)$ 、

第 5 章 树与二叉树

1. 选择题

- (1) 把一棵树转换为二叉树后, 这棵二叉树得形态就是()。

A. 唯一得	B. 有多种
C. 有多种, 但根结点都没有左孩子	D. 有多种, 但根结点都没有右孩子
- (2) 由 3 个结点可以构造出多少种不同得二叉树? ()

A. 2	B. 3	C. 4	D. 5
------	------	------	------
- (3)一棵完全二叉树上有 1001 个结点, 其中叶子结点得个数就是 ()。

A. 250	B. 500	C. 254	D. 501
--------	--------	--------	--------
- (4) 一个具有 1025 个结点得二叉树得高 h 为 ()。

A. 11	B. 10	C. 11 至 1025 之间	D. 10 至 1024 之间
-------	-------	-----------------	-----------------
- (5)深度为 h 得满 m 叉树得第 k 层有 () 个结点。 $(1 \leq k \leq h)$

A. m^{k-1}	B. $m^k - 1$	C. m^{h-1}	D. $m^h - 1$
--------------	--------------	--------------	--------------
- (6) 利用二叉链表存储树, 则根结点得右指针就是()。

A. 指向最左孩子	B. 指向最右孩子	C. 空	D. 非空
-----------	-----------	------	-------
- (7) 对二叉树得结点从 1 开始进行连续编号, 要求每个结点得编号大于其左、右孩子得编号, 同一结点得左右孩子中, 其左孩子得编号小于其右孩子得编号, 可采用()遍历实现编号。

A. 先序	B. 中序	C. 后序	D. 从根开始按层次遍历
-------	-------	-------	--------------
- (8)若二叉树采用二叉链表存储结构, 要交换其所有分支结点左、右子树得位置, 利用()遍历方法最合适。

A. 前序	B. 中序	C. 后序	D. 按层次
-------	-------	-------	--------
- (9) 在下列存储形式中, () 不就是树得存储形式?

A. 双亲表示法	B. 孩子链表表示法	C. 孩子兄弟表示法	D. 顺序存储表示法
----------	------------	------------	------------
- (10) 一棵非空得二叉树得先序遍历序列与后序遍历序列正好相反, 则该二叉树一定满足()。

A. 所有得结点均无左孩子	B. 所有得结点均无右孩子
---------------	---------------

- C. 只有一个叶子结点 D. 就是任意一棵二叉树
- (11) 某二叉树得前序序列与后序序列正好相反，则该二叉树一定就是（ ）得二叉树。
- A. 空或只有一个结点 B. 任一结点无左子树
C. 高度等于其结点数 D. 任一结点无右子树
- (12) 若 X 就是二叉中序线索树中一个有左孩子得结点，且 X 不为根，则 X 得前驱为（ ）。
- A. X 得双亲 B. X 得右子树中最左得结点
C. X 得左子树中最右结点 D. X 得左子树中最右叶结点
- (13) 引入二叉线索树得目得就是（ ）。
- A. 加快查找结点得前驱或后继得速度 B. 为了能在二叉树中方便得进行插入与删除
C. 为了能方便得找到双亲 D. 使二叉树得遍历结果唯一
- (14) 线索二叉树就是一种（ ）结构。
- A. 逻辑 B. 逻辑与存储 C. 物理 D. 线性
- (15) 设 F 就是一个森林，B 就是由 F 变换得得二叉树。若 F 中有 n 个非终端结点，则 B 中右指针域为空得结点有（ ）个。
- A. n-1 B. n C. n+1 D. n + 2

2. 应用题

(1) 试找出满足下列条件得二叉树

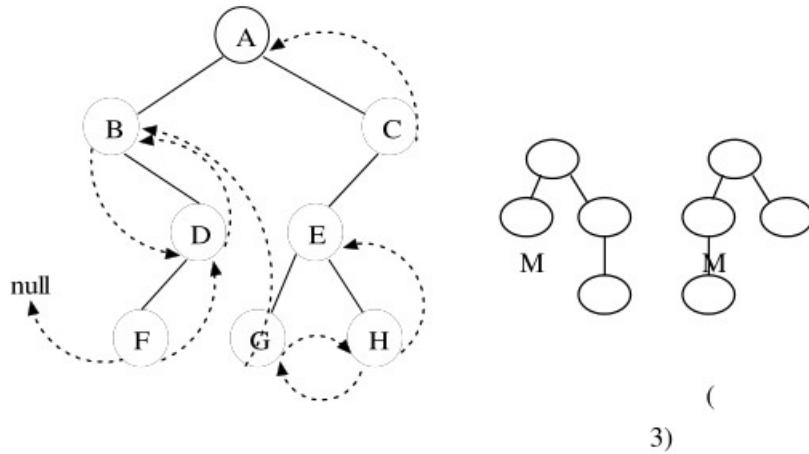
- ① 先序序列与后序序列相同 ② 中序序列与后序序列相同
 ③ 先序序列与中序序列相同 ④ 中序序列与层次遍历序列相同

先序遍历二叉树得顺序就是“根—左子树—右子树”，中序遍历“左子树—根—右子树”，后序遍历顺序就是：“左子树—右子树—根”，根据以上原则，本题解答如下：

- (1) 若先序序列与后序序列相同，则或为空树，或为只有根结点得二叉树。
 (2) 若中序序列与后序序列相同，则或为空树，或为任一结点至多只有左子树得二叉树。
 (3) 若先序序列与中序序列相同，则或为空树，或为任一结点至多只有右子树得二叉树。
 (4) 若中序序列与层次遍历序列相同，则或为空树，或为任一结点至多只有右子树得二叉树。

(2) 设一棵二叉树得先序序列： A B D F C E G H ， 中序序列： B F D A G E H C

- ① 画出这棵二叉树。
 ② 画出这棵二叉树得后序线索树。
 ③ 将这棵二叉树转换成对应得树(或森林)。



(1)

(2)

(3) 假设用于通信得电文仅由 8 个字母组成, 字母在电文中出现得频率分别为 0、07, 0、19, 0、02, 0、06, 0、32, 0、03, 0、21, 0、10。

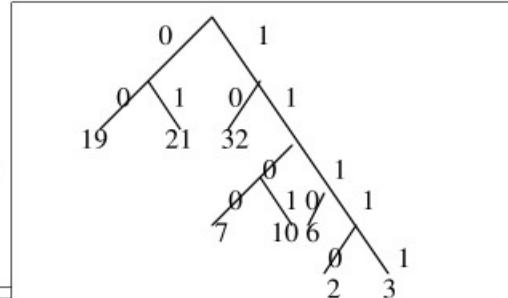
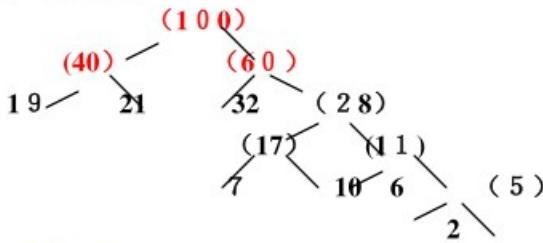
- ① 试为这 8 个字母设计赫夫曼编码。
- ② 试设计另一种由二进制表示得等长编码方案。
- ③ 对于上述实例, 比较两种方案得优缺点.

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: $\{(2, 3), 6\}, (7, 10)\}, \dots$

19, 21, 32



方案比较:

字母 编号	对应 编码	出现 频率	2 + 0, 1 = 2, 6, 1 2+0, 21+	1child	2child	3child
1	1100	0.07				
2	00	0.19				
3	11110	0.02				
4	1110	0.06				
5	10	0.32				
初态:	11111	0.03				
7	we i ght	parent 0, 21				
18	3101	0, 10				
2	12	0				
3	7	0				
4	weight	parent 0, 21				
5	2	0				
6 1	3	0.08				
7 2	4	0.12				
8 3	7	0.10				
9 4	4	0.09				
10 5	2	0.08				
11 6	8	0.10				
12 7	11	0.11				
13 8	5	0.09				
9	9	0.11				
10	15	0.12				
11	20	0.13				
12	27	0.13				

字母 编号	对应 编码	出现 频率	、 3)
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	

1	4 7	0	11	12
3				

终态

3。算法设计题

以二叉链表作为二叉树

得存储结构,编写以下算法:

(1) 统计二叉树得叶结点个数。

```
int LeafNodeCount(BiT ree T)
{
    if(T==NULL)
        return 0; //如果就是空树, 则叶
子结点个数为0
    else if(T->l child==NULL&&T->r child==NULL)
        return 1; //判断该结点就是否就
是叶子结点(左孩子右孩子都为空), 若就是则返回1
    else
        return LeafNodeCount(T->l ch i ld)+LeafNodeCount(T->r ch i ld);
}
```

(2) 判别两棵树就是否相等。

(3)交换二叉树每个结点得左孩子与右孩子。

```
void ChangeLR (BiTree &T)
{
    BiTree temp;
    if(T->lch i ld==NULL&&T->rch i ld==NULL)
        return;
    else
    {
        temp = T->lch i ld;
        T->lch i ld = T->rch i ld;
        T->rch i ld = temp;
    }
    ChangeLR (T->lch i ld);
    ChangeLR (T->rch i ld);
}
```

(4) 设计二叉树得双序遍历算法(双序遍历就是指对于二叉树得每一个结点来说, 先访问这个结点, 再按双序遍历它得左子树, 然后再一次访问这个结点, 接下来按双序遍历它得右子树) .

```
void DoubleTraverse (BiTree T)
{
    if(T == NULL)
        return;
    else if(T->lch i ld==NULL&&T->rch i ld==NULL)
        cout << T->data;
    else
    {
        cout << T->data;
```

```

    =>DoubleTraverse(T->lchild);
    > cout<<T->data;
    =>DoubleTraverse(T->rchild);
}
}

```

(5) 计算二叉树最大得宽度 (二叉树得最大宽度就是指二叉树所有层中结点个数得最大值)。

[题目分析] 求二叉树高度得算法见上题。求最大宽度可采用层次遍历得方法, 记下各层结点数, 每层遍历完毕, 若结点数大于原先最大宽度, 则修改最大宽度。

```

int Width(BiTTree bt) //求二叉树bt得最大宽度
{
    if (bt==NULL) return (0); //空二叉树宽度为0
    else
        {BiTree Q[]; //Q就是队列, 元素为二叉树结点指针, 容量足够大
         front=1; rear=1; last=1; //front队头指针, rear队尾指针, last同层最右结点在队列中得位置
         temp=0; maxw=0; //temp记局部宽度, maxw记最大宽度
         Q[rear]=bt; //根结点入队列
         while(front<last)
             {p=Q[front++]; temp++; //同层元素数加1
              if(p->lchild!=NULL) Q[++rear]=p->lchild; //左子女入队
              if(p->rchild!=NULL) Q[++rear]=p->rchild; //右子女入队
              if(front>last) //一层结束,
                  {last=rear;
                   if(temp>maxw) maxw=temp; //last指向层最右元素, 更新当前最大宽度
                   temp=0;
                 } //if
            } //while
            return (maxw);
        } //结束width

```

(6) 用按层次顺序遍历二叉树得方法, 统计树中具有度为1得结点数目。

```

int Level(BiTTree bt) //层次遍历二叉树, 并统计度为1得结点得个数
{
    int num=0; //num统计度为1得结点得个数
    if(bt) {QueueInit(Q); QueueIn(Q, bt); //Q就是以二叉树结点指针为元素得队列
             while(!QueueEmpty(Q))
                 {p=QueueOut(Q); printf(p->data); //出队, 访问结点
                  if(p->lchild && !p->rchild || !p->lchild&& p->rchild) num++; //度为1得结点
                  if(p->lchild) QueueIn(Q, p->lchild); //非空左子女入队
                  if(p->rchild) QueueIn(Q, p->rchild); //非空右子女入队
                }
            }
    return num;
}

```

```

        }
    } //if (bt)
return (num); //返回度为1得结点得个数
(7) 求任意二叉树中第一条最长得路径长度, 并输出此路径上各结点得值。
[题目分析] 因为后序遍历栈中保留当前结点得祖先得信息, 用一变量保存栈得最高栈顶指针, 每当退栈时, 栈顶指针高于保存最高栈顶指针得值时, 则将该栈倒入辅助栈中, 辅助栈始终保存最长路径长度上得结点, 直至后序遍历完毕, 则辅助栈中内容即为所求。
void LongestPath(BiTree bt) //求二叉树中得第一条最长路径长度
{BiTree p=bt, l[], s[]; //l, s就是栈, 元素就是二叉树结点指针, l中保留当前最长路径中得结点
int i, top=0, tag[], longest=0;
while(p || top>0)
{ while (p) {s[++top]=p; tag[top]=0; p=p->Lc;} //沿左分枝向下
if (tag[top]==1) //当前结点得右分枝已遍历
{if (!s[top]->Lc && !s[top]->Rc) //只有到叶子结点时, 才查瞧路径长度
if (top>longest) {for (i=1;i<=top;i++) l[i]=s[i]; longest=top; top--;}
//保留当前最长路径到l栈, 记住最高栈顶指针, 退栈
}
else if (top>0) {tag[top]=1; p=s[top]; Rc;} //沿右子分枝向下
}//while (p != null || top>0)
} //结束LongestPath

```

(8)输出二叉树中从每个叶子结点到根结点得路径。

[题目分析] 采用先序遍历得递归方法, 当找到叶子结点*b 时, 由于*b 叶子结点尚未添加到path 中, 因此在输出路径时还需输出 b->data 值。对应得递归算法如下:

```

void AllPath(BTNode *b, ElemenType path[], int pathlen)
{
int i;
if (b!=NULL)
{
if ((b->lchild==NULL && b->rchild==NULL) /*b 为叶子结点
{
cout << " " << b->data << " 到根结点路径:" << b->
data;
for (i=pathlen-1;i>=0; i--)
cout << endl;
}
else
{
path [pathlen]=b->data; //将当前结点放入路径中
pathlen++; // 路径长度增1
AllPath(b->lchild, path, pathlen); // 递归扫描左子树
}
}

```

```

    AllPath(b->rchild, path, pathlen);
    //递归扫描右子树
    pathlen--;
    //恢复环境
}
}
}

```

第6章 图

1.选择题

- (1) 在一个图中,所有顶点得度数之与等于图得边数得()倍。
 A. 1/2 B. 1 C. 2 D. 4
- (2) 在一个有向图中, 所有顶点得入度之与等于所有顶点得出度之与得()倍。
 A. 1/2 B. 1 C. 2 D. 4
- (3) 具有 n 个顶点得有向图最多有()条边。
 A. n B. n(n-1) C. n(n+1) D. n^2
- (4) n 个顶点得连通图用邻接距阵表示时,该距阵至少有()个非零元素。
 A. n B. 2(n-1) C. $n/2$ D. n^2
- (5) G 就是一个非连通无向图,共有 28 条边,则该图至少有()个顶点。
 A. 7 B. 8 C. 9 D. 10
- (6) 若从无向图得任意一个顶点出发进行一次深度优先搜索可以访问图中所有得顶点,则该图一定就是()图。
 A. 非连通 B. 连通 C. 强连通 D. 有向
- (7) 下面()算法适合构造一个稠密图 G 得最小生成树。
 A. Prim 算法 B. Kruskal 算法 C. Floyd 算法 D. Dijkstra 算法
- (8) 用邻接表表示图进行广度优先遍历时, 通常借助()来实现算法。
 A. 栈 B. 队列 C. 树 D. 图
- (9) 用邻接表表示图进行深度优先遍历时, 通常借助()来实现算法。
 A. 栈 B. 队列 C. 树 D. 图
- (10) 深度优先遍历类似于二叉树得()。
 A. 先序遍历 B. 中序遍历 C. 后序遍历 D. 层次遍历
- (11) 广度优先遍历类似于二叉树得()。
 A. 先序遍历 B. 中序遍历 C. 后序遍历 D. 层次遍历
- (12) 图得 BFS 生成树得树高比 DFS 生成树得树高()。
 A. 小 B. 相等 C. 小或相等 D. 大或相等
- (13) 已知图得邻接矩阵如图 6、25 所示, 则从顶点 0 出发按深度优先遍历得结果就是()。
 A. 0 2 4 3 1 5 6
 B. 0 1 3 6 5 4 2
 C. 0 1 3 4 2 5 6
 D. 0 3 6 1 5 4 2
- 图 6、25 邻接矩阵
- (14) 已知图得邻接表如图 6、26 所示, 则从顶点 0 出发按广度优先遍历得结果就是()。

()按深度优先遍历得结果就是().

图 6、26 邻接表

(15)下面()方法可以判断出一个有向图

A. 深度优先遍历 B. 拓扑排序

A. 0 1 3 2

B. 0 2 3 1

C. 0 3 2 1

D. 0 1 2 3

C. 水取矩阵增广

D. 水大矩阵增广

2. 应用题

(1) 已知如图 6、27 所示得有向图,请给出:

- ① 每个顶点得入度与出度;
- ② 邻接矩阵;
- ③ 邻接表;
- ④ 逆邻接表。

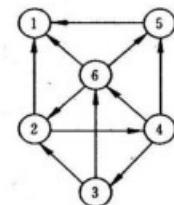


图 6、27 有

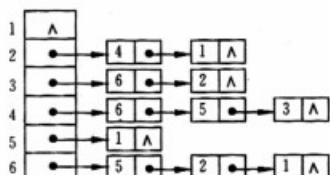
(1)

顶点	1	2	3	4	5	6
入度	3	2	1	1	2	2
出度	0	2	2	3	1	3

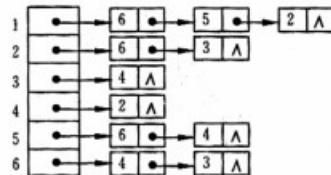
(2) 邻接矩阵

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

(3) 邻接表



(4) 逆邻接表



(2) 已知如图 6、28 所示得无向网,请给出:

- ① 邻接矩阵;
- ② 邻接表;
- ③ 最小生成树

$$\begin{bmatrix} \infty & 4 & 3 & \infty & \infty & \infty & \infty & \infty \\ 4 & \infty & 5 & 5 & 9 & \infty & \infty & \infty \\ 3 & 5 & \infty & 5 & \infty & \infty & \infty & 5 \\ \infty & 5 & 5 & \infty & 7 & 6 & 5 & 4 \\ \infty & 9 & \infty & 7 & \infty & 3 & \infty & \infty \\ \infty & \infty & \infty & 6 & 3 & \infty & 2 & \infty \\ \infty & \infty & \infty & 5 & \infty & 2 & \infty & 6 \\ \infty & \infty & 5 & 4 & \infty & \infty & 6 & \infty \end{bmatrix}$$

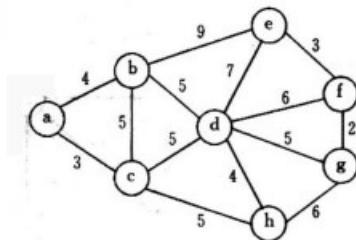
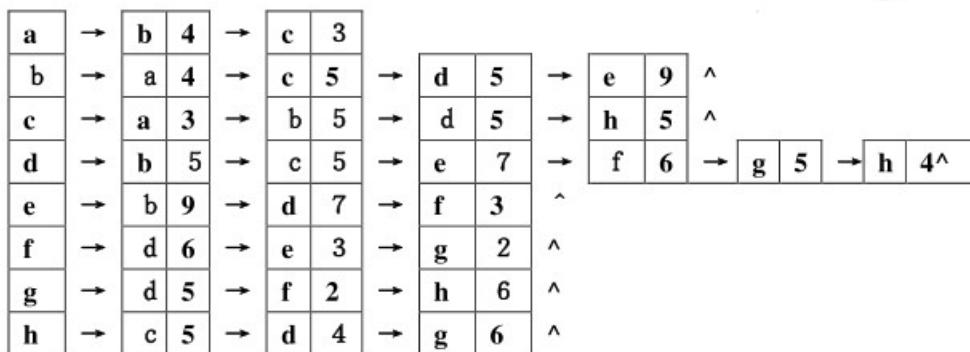
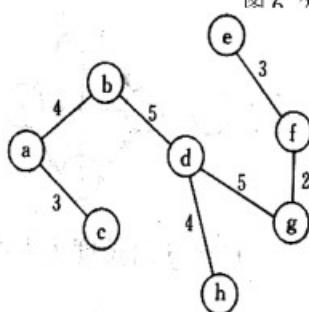


图 6.28 无

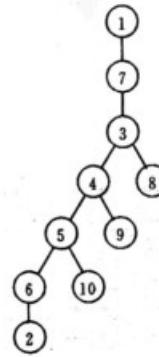


(3) 已知图得邻接矩阵如 6、29 所示。试分别画出自顶点 1 出发进行遍历所得得深度优先生成树和广度优先生成树。

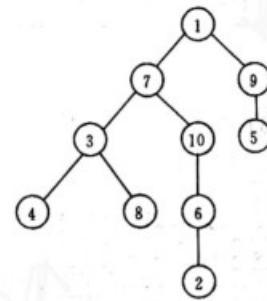
图 6.29 邻接矩

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

深度优先生成树



广度优先生成树



(4) 有向网如图 6、30 所示, 试用迪杰斯特拉算法求出从顶点 a 到其他各顶点间得最短路径, 完成表 6、9。

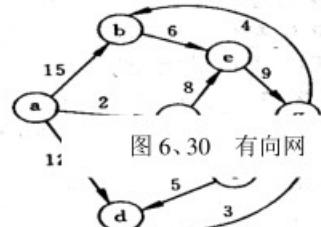


图 6、30 有向网

D 终 点	i = 1	i=2	i=3	i=4	i=5	i=6
b	15 (a, b)	15 (a, b)	15 (a, b)	15 (a, b)	15 (a, b)	15 (a, b)
c	2 (a, c)					
d	12 (a, d)	12 (a, d)	11 (a, c, f, d)	11 (a, c, f, d)		
e	∞	10 (a, c, e)	10 (a, c, e)			
f	∞	6 (a, c, f)				
g	∞	∞	16 (a, c, f, g)	16 (a, c, f, g)	14 (a, c, f, d, g)	
S 终 点集	{a, c}	{a, c, f}	{a, c, f, e}	{a, c, f, e, d}	{a, c, f, e, d, g}	{a, c, f, e, d, g, b}

(5) 试对图 6、31 所示得 AOE 一网:

- 求这个工程最早可能在什么时间结束;
- 求每个活动得最早开始时间与最迟开始时间;
- 确定哪些活动就是关键活动

【解答】按拓扑有序得顺序计算各个顶点得最早可能升 VI。然后再计算各个活动得最早可能开始时间 l 与最迟允许开始时间 t , 根据 $t - l = ?$

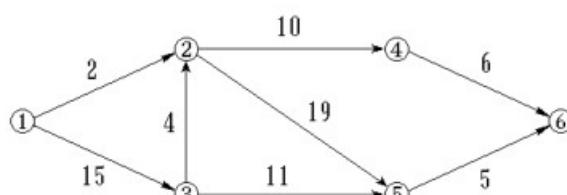


图 6、31 AOE- 网开始时间

来确定关键活动，从而确定关键路径。

	1	2	3	4	5	6		
Ve	0	19	15	29	38	43		
VI	0	19	15	37	38	43		
	< 1, 2>	< 1 , 3>	< 3 , 2>	< 2 , 4>	< 2, 5>	< 3 , 5>	< 4 , 6>	< 5, 6 >
e	0	0	15	19	19	15	29	38
l	17	0	15	27	19	27	37	38
-e	17	0	0	8	0	12	8	0

•此工程最早完成时间为43。关键路径为<1, 3> <3, 2> <2, 5> <5, 6>

3. 算法设计题

(1) 分别以邻接矩阵与邻接表作为存储结构,实现以下图得基本操作:

- ① 增添一个新顶点 v, InsertVex(G, v);
- ② 删除顶点 v 及其相关得边, DeleteVex(G, v);
- ③ 增加一条边<v,w>, InsertArc(G, v, w);
- ④ 删除一条边<v,w>, DeleteArc(G, v, w)。

```
//本题中得图 G 均为有向无权图, 其余情况容易由此写出
Status InsertVex(MGraph &G, char v)//在邻接矩阵表示得图 G 上插入顶点 v {if (G->vexnum+1)>MAX_VERTEX_NUM return INFESIBLE; *G->vexs[+G->vexnum]=v; return OK; } //InsertVex
Status InsertArc(MGraph &G, char v, char w) //在邻接矩阵表示得图 G 上插入边(v, w)
{if ((i=LocateVex(G,v))<0) return ERROR; if((j=LocateVex(G,w))<0)
return ERROR;
if (i==j) return ERROR; if (!G->arcs[j].adj)
{*G->arcs[j].adj=1; *G->arcnum++; }
return OK;
} //InsertArc
Status DeleteVex (MGraph &G, char v) // 在邻接矩阵表示得图 G 上删除顶点 v
{int 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[m]=G->arcs[n];
G->arcs[m]=G->arcs[n]; //将边得关系随之交换
}
G->arcs[m][m].adj=0;
```

```

G、 v e xnum——;
return OK; } // Delete_Vex
分析:如果不把待删除顶点交换到最后一个顶点得话, 算法将会比较复杂,而伴随着大量元素得移动, 时间复杂度也会大大增加. Status Delete_Arc(MGraph &G, char v,char w) // 在邻接矩阵表示得图 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[j] , adj)
{G、 arcs [j] , adj=0; G、 arcnum--;
}
return OK}//Delete_Arc
//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) //在邻接表表示得图 G 上插入边( v,w )
{
if( ( i =LocateVex ( G,v )) <0 ) return ERROR;
if( (j=LocateVex ( G, w)) <0) return ERROR; Ap=( ArcNode*) malloc (size
of (ArcNode));
p->adjvex=j; p->nextarc=NULL;
if (!G、 vertices、 firstarc) G、 vertices、 firstarc=p;
else {
for ( q=G、 vertices、 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

```

(2) 一个连通图采用邻接表作为存储结构,设计一个算法, 实现从顶点 v 出发得深度优先遍历得非递归过程。

数据结构考研指导 232 页 7. 3. 7

(3)设计一个算法, 求图 G 中距离顶点 v 得最短路径长度最大得一个顶点, 设 v 可达其余各个顶点.

数据结构考研指导 232 页 7. 3. 8

(4) 试基于图得深度优先搜索策略写一算法, 判别以邻接表方式存储得有向图中就是否存在由顶点 v_i 到顶点 v_j 得路径 ($i \neq j$)。

解 1:

```

int visited [MAXSIZE]; // 指示顶点就是在当前路径上
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
就是是否有路径, 就是则返回 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 下游得顶点到 j 有路径
    } //for
} //else
//exist_path_DFS

```

解 2：(以上算法似乎有问题：如果不存在路径，则原程序不能返回 0。我得解决方式就是在原程序得中引入一变量 level 来控制递归进行得层数。具体得方法我在程序中用红色标记出来了。)

```

int visited[MAXSIZE]; // 指示顶点是否在当前路径上
int level=1; // 递归进行得层数
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
    就是否有路径，就是则返回 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, level++)
        { level++;
            k=p->adjvex;
            if(!visited[k]&& exist_path(k, j)) return 1; //i 下游得顶点到 j
        } //for
    } //else
    if (level==1) return 0;
} //exist_path_DFS

```

(5) 采用邻接表存储结构，编写一个算法，判别无向图中任意给定得两个顶点之间就是否存在一条长度为为 k 得简单路径。

(注 1：一条路径为简单路径指得就是其顶点序列中不含有重现得顶点。

注 2：此题可参见严题集 P207—208 中有关按“路径”遍历得算法基本框架。)

```

int visited[MAXSIZE];
int exist_path_len(ALGraph G, int i, int j, int k) //判断邻接表方
式存储得有向图 G
    得顶点 i 到 j 就是否存在长度为 k 得简单路径
{
{
    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, 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得结点出现在另一条路径中
} //else
return 0; //没找到
} //exist_path_len

```

第7章 查找

1.选择题

- (1) 对 n 个元素得表做顺序查找时, 若查找每个元素得概率相同, 则平均查找长度为()。
- A. $(n-1)/2$ B. $n/2$ C. $(n+1)/2$ D. n
- (2) 适用于折半查找得表得存储方式及元素排列要求为()。
- A. 链接方式存储, 元素无序 B. 链接方式存储, 元素有序
 C. 顺序方式存储, 元素无序 D. 顺序方式存储, 元素有序
- (3) 当在一个有序得顺序表上查找一个数据时, 既可用折半查找, 也可用顺序查找, 但前者比后者得查找速度()。
- A. 必定快 B. 不一定
 C. 在大部分情况下要快 D. 取决于表递增还是递减
- (4) 折半查找有序表 (4, 6, 10, 12, 20, 30, 50, 70, 88, 100). 若查找表中元素 58, 则它将依次与表中() 比较大小, 查找结果就是失败。
- A. 20, 70, 30, 50 B. 30, 88, 70, 50
 C. 20, 50 D. 30, 88, 50
- (5) 对 22 个记录得有序表作折半查找, 当查找失败时, 至少需要比较()次关键字。
- A. 3 B. 4 C. 5 D. 6
- (6) 折半搜索与二叉排序树得时间性能()。
- A. 相同 B. 完全不同
 C. 有时不相同 D. 数量级都是 $O(\log_2 n)$
- (7) 分别以下列序列构造二叉排序树, 与用其它三个序列所构造得结果不同得就是()。
- A. (100, 80, 90, 60, 120, 110, 130)
 B. (100, 120, 110, 130, 80, 60, 90)
 C. (100, 60, 80, 90, 120, 110, 130)
 D. (100, 80, 60, 90, 120, 130, 110)
- (8) 在平衡二叉树中插入一个结点后造成了不平衡, 设最低得不平衡结点为 A, 并已知 A 得左孩子得平衡因子为 0 右孩子得平衡因子为 1, 则应作()型调整以使其平衡。
- A. LL B. LR C. RL D. RR
- (9) 下列关于 m 阶 B—树得说法错误得就是()。
- A. 根结点至多有 m 棵子树
 B. 所有叶子都在同一层次上
 C. 非叶结点至少有 $m/2$ (m 为偶数) 或 $m/2+1$ (m 为奇数) 棵子树
 D. 根结点中得数据就是有序得
- (10) 下面关于 B—与 B+树得叙述中, 不正确得就是()。
- A. B-树与 B+树都就是平衡得多叉树 B. B—树与 B+树都可用于文件得索引结构

C. B—树与 B+树都能有效地支持顺序检索 D. B-树与 B+树都能有效地支持随机检索

(11) m 阶 B—树就是一棵()。

A. m 叉排序树

B. m 叉平衡排序树

C. m—1 叉平衡排序树

D. m+1 叉平衡排序树

(12) 下面关于哈希查找得说法, 正确得就是()。

A. 哈希函数构造得越复杂越好, 因为这样随机性好, 冲突小

B. 除留余数法就是所有哈希函数中最好得

C. 不存在特别好与坏得哈希函数, 要视情况而定

D. 哈希表得平均查找长度有时也与记录总数有关

(13) 下面关于哈希查找得说法, 不正确得就是()。

A. 采用链地址法处理冲突时, 查找一个元素得时间就是相同得

B. 采用链地址法处理冲突时, 若插入规定总就是在链首, 则插入任一个元素得时间就是相同得

C. 用链地址法处理冲突, 不会引起二次聚集现象

D. 用链地址法处理冲突, 适合表长不确定得情况

(14) 设哈希表长为 14, 哈希函数就是 $H(key)=key \% 11$, 表中已有数据得关键字为 15, 38, 61, 84 共四个, 现要将关键字为 49 得元素加到表中, 用二次探测法解决冲突, 则放入得位置就是()。

A. 8

B. 3

C. 5

D. 9

(15) 采用线性探测法处理冲突, 可能要探测多个位置, 在查找成功得情况下, 所探测得这些位置上得关键字()。

A. 不一定都就是同义词

B. 一定都就是同义词

C. 一定都不就是同义词

D. 都相同

2. 应用题

(1) 假定对有序表: (3, 4, 5, 7, 24, 30, 42, 54, 63, 72, 87, 95) 进行折半查找, 试回答下列问题:

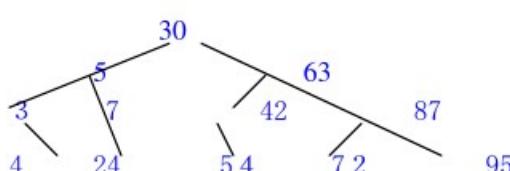
① 画出描述折半查找过程得判定树;

② 若查找元素 54, 需依次与哪些元素比较?

③ 若查找元素 90, 需依次与哪些元素比较?

④ 假定每个元素得查找概率相等, 求查找成功时得平均查找长度。

① 先画出判定树如下 (注: $mid = \lfloor (1+12)/2 \rfloor = 6$) :



② 查找元素 54, 需依次与 30, 63, 42, 54 元素比较;

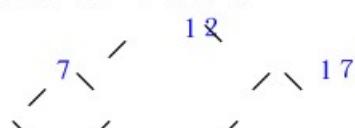
③ 查找元素 90, 需依次与 30, 63, 87, 95 元素比较;

④ 求 ASL 之前, 需要统计每个元素得查找次数。判定树得前 3 层共查找 $1+2\times 2+4\times 3=17$ 次;

但最后一层未满, 不能用 8×4 , 只能用 $5\times 4=20$ 次,

所以 $ASL = 1/12(17+20) = 37/12 \approx 3.08$

(2) 在一棵空得二叉排序树中依次插入关键字序列为 12, 7, 17, 11, 16, 2, 13, 9, 21, 4, 请画出所得到得二叉排序树。



2	11	16	2 1
4	9	13	

验算方法：用中序遍历应得到排序结果：2, 4, 7, 9, 11, 12, 13, 16, 17, 21

(3) 已知如下所示长度为 12 得表：(Jan, Feb, Mar, Apr, May, June, July, Aug, Sep, Oct, Nov, Dec)

① 试按表中元素得顺序依次插入一棵初始为空得二叉排序树,画出插入完成之后得二叉排序树,并求其在等概率得情况下查找成功得平均查找长度。

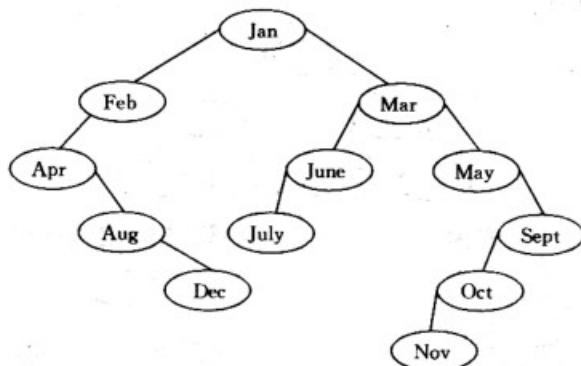
② 若对表中元素先进行排序构成有序表,求在等概率得情况下对此有序表进行折半查找时查找成功得平均查找长度。

③ 按表中元素顺序构造一棵平衡二叉排序树，并求其在等概率得情况下查找成功得平均查找长度。

解：

(1) 求得的二叉排序树如下图所示,在等概率情况下查找成功的平均查找长度为

$$ASL_{\text{succ}} = \frac{1}{12}(1 \times 1 + 2 \times 2 + 3 \times 3 + 4 \times 3 + 5 \times 2 + 6 \times 1) = \frac{42}{12}$$



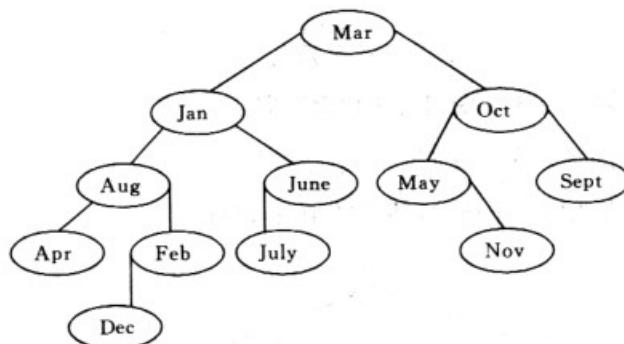
(2) 经排序后的表及在折半查找时找到表中元素所经比较的次数对照如下:

Apr	Aug	Dec	Feb	Jan	July	June	Mar	May	Nov	Oct	Sept
3	4	2	3	4	1	3	4	2	4	3	4

等概率情况下查找成功时的平均查找长度为

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

(3) 按教科书 9.2.1 节所述求得的平衡二叉树为

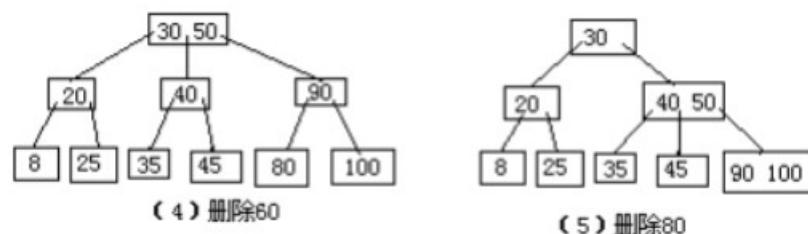
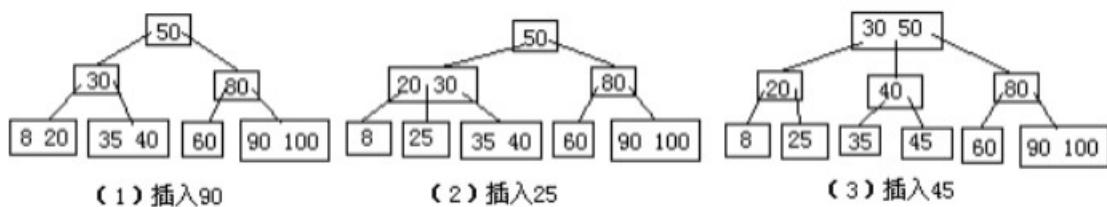


它在等概率情况下的平均查找长度为

$$ASL = \frac{1}{12}(1 \times 1 + 2 \times 2 + 3 \times 4 + 4 \times 4 + 5 \times 1) = \frac{38}{12}$$

(4) 对下面得 3 阶 B-树, 依次执行下列操作, 画出各步操作得结果。

- ① 插入 90 ② 插入 25 ③ 插入 45 ④ 删除 60



(5) 设哈希表得地址范围为 0~17, 哈希函数为: $H(\text{key}) = \text{key} \% 16$ 。用线性探测法处理冲突, 输入关键字序列:(10, 24, 32, 17, 31, 30, 46, 47, 40, 63, 49), 构造哈希表, 试回答下列问题:

- ① 画出哈希表得示意图;

- ② 若查找关键字 63，需要依次与哪些关键字进行比较？
- ③ 若查找关键字 60，需要依次与哪些关键字比较？
- ④ 假定每个关键字得查找概率相等，求查找成功时得平均查找长度。

①画表如下：

															16	1 7
3	1	6	49				24	4								
2	7	3														

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

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

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

④ 对于黑色数据元素，各比较 1 次；共 6 次；

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

所以 $ASL = 1/11 (6+2+3 \times 3+6) = 23/11$

⑥ 设有一组关键字 (9, 01, 23, 14, 55, 20, 84, 27)，采用哈希函数： $H(key) = key \% 7$ ，表长为 10，用开放地址法得二次探测法处理冲突。要求：对该关键字序列构造哈希表，并计算查找成功得平均查找长度。

散列地址	0	1	2	3	4	5	6	7	8	9
关键字	14							20		
比较次数	1	1	1	2	3	4	1	2		

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

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

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

⑦ 设哈希函数 $H(K) = 3 K \bmod 11$ ，哈希地址空间为 $0 \sim 10$ ，对关键字序列 (32, 13, 49, 24, 38, 21, 4, 12)，按下述两种解决冲突得方法构造哈希表，并分别求出等概率下查找成功时与查找失败时得平均查找长度 ASL_{succ} 与 ASL_{unsucc} 。

① 线性探测法；

② 链地址法。

①

散列地址										0
关键字		4		12	49	38	13	24	32	21
比较次数		1		1	1	2	1	2	1	2

$ASL_{succ} = (1+1+1+2+1+2+1+2)/8 = 11/8$

$ASL_{unsucc} = (1+2+1+2+3+1+3+1+3+1+1)/11 = 40/11$

②

$ASL_{succ} = (1*5+2*3)/8 = 11/8$

$ASL_{unsucc} = (1+2+1+2+3+1+3+1+3+1+1)/11 = 19/11$

⑤ 设哈希表得地址范围为 $0 \sim 17$ ，哈希函数为： $H(key) = key \% 16$ 。用线性探测法处理冲突，输入关键字序列：(10, 24, 32, 17, 31, 30, 46, 47, 40, 63, 49)，构造哈希表，试回答下列问题：

- ① 画出哈希表得示意图；
- ② 若查找关键字 63，需要依次与哪些关键字进行比较？
- ③ 若查找关键字 60，需要依次与哪些关键字比较？
- ④ 假定每个关键字得查找概率相等，求查找成功时得平均查找长度。

解：（1）画表如下：

															16	17
32	1	6	4					2	4							
	7	3	9					4								

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

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

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

(4) 对于黑色数据元素，各比较 1 次；共 6 次；

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

所以 $ASL=1/11(6+2+3\times3+6)=23/11$

(6) 设有一组关键字 (9, 01, 23, 14, 55, 20, 84, 27)，采用哈希函数： $H(key) = key \% 7$ ，表长为 10，用开放地址法得二次探测法处理冲突。要求：对该关键字序列构造哈希表，并计算查找成功得平均查找长度。

散列地址	0	1	2	3	4	5	6	7	8	9
关键字	14							20		
比较次数	1	1	1	2		4	1	2		
					3					

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

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

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

(7) 设哈希函数 $H(K) = 3 \mod 11$ ，哈希地址空间为 0~10，对该关键字序列 (32, 13, 49, 24, 38, 21, 4, 12)，按下述两种解决冲突得方法构造哈希表，并分别求出等概率下查找成功时与查找失败时得平均查找长度 ASL_{succ} 与 ASL_{unsucc} 。

- ① 线性探测法；
- ② 链地址法。

散列地址									0		
关键字		4		12	49	38	13	24	32	21	
比较次数		1		1	1	2	1	2	1	2	

$$ASL_{succ} = (1+1+1+2+1+2+1+2)/8 = 11/8$$

$$ASL_{unsucc} = (1+2+1+8+7+6+5+4+3+2+1)/11 = 40/11$$

第 8 章 排序

1. 选择题

- (1) 从未排序序列中依次取出元素与已排序序列中得元素进行比较，将其放入已排序序列

得正确位置上得方法,这种排序方法称为()。

- A. 归并排序 B. 冒泡排序 C. 插入排序 D. 选择排序

(2) 从未排序序列中挑选元素,并将其依次放入已排序序列(初始时为空)得一端得方法,称为()。

- A. 归并排序 B. 冒泡排序 C. 插入排序 D. 选择排序

(3) 对 n 个不同得关键字由小到大进行冒泡排序,在下列()情况下比较得次数最多.

- A. 从小到大排列好得 B. 从大到小排列好得
C. 元素无序 D. 元素基本有序

(4) 对 n 个不同得排序码进行冒泡排序,在元素无序得情况下比较得次数最多为()。

- A. $n+1$ B. n C. $n-1$ D. n

(n-1)/2

(5) 快速排序在下列()情况下最易发挥其长处。

- A. 被排序得数据中含有多个相同排序码
B. 被排序得数据已基本有序
C. 被排序得数据完全无序
D. 被排序得数据中得最大值与最小值相差悬殊

(6) 对 n 个关键字作快速排序,在最坏情况下,算法得时间复杂度就是()。

- A. $O(n)$ B. $O(n^2)$ C. $O(n \log_2 n)$
D. $O(n^3)$

(7) 若一组记录得排序码为(46, 79, 56, 38, 40, 84),则利用快速排序得方法,以第一个记录为基准得到得一次划分结果为()。

- A. 38, 40, 46, 56, 79, 84 B. 40, 38, 46, 79, 56, 84
C. 40, 38, 46, 56, 79, 84 D. 40, 38, 46, 84, 56, 79

(8) 下列关键字序列中,()就是堆。

- A. 16, 72, 31, 23, 94, 53 B. 94, 23, 31, 72, 16, 53
C. 16, 53, 23, 94, 31, 72 D. 16, 23, 53, 31, 94, 72

(9) 堆就是一种()排序。

- A. 插入 B. 选择 C. 交换 D. 归并

(10) 堆得形状就是一棵()。

- A. 二叉排序树 B. 满二叉树 C. 完全二叉树 D. 平衡二叉树

(11) 若一组记录得排序码为(46, 79, 56, 38, 40, 84),则利用堆排序得方法建立得初始堆为()。

- A. 79, 46, 56, 38, 40, 84 B. 84, 79, 56, 38, 40, 46
C. 84, 79, 56, 46, 40, 38 D. 84, 56, 79, 40, 46, 38

(12) 下述几种排序方法中,要求内存最大得就是()。

- A. 希尔排序 B. 快速排序 C. 归并排序 D. 堆排序

(13) 下述几种排序方法中,()就是稳定得排序方法。

- A. 希尔排序 B. 快速排序 C. 归并排序 D. 堆排序

(14) 数据表中有 10000 个元素,如果仅要求求出其中最大得 10 个元素,则采用()算法最节省时间.

A.冒泡排序

B。快速排序

C.简单选择排序

D. 堆排序

(15) 下列排序算法中,()不能保证每趟排序至少能将一个元素放到其最终得位置上。

A.希尔排序

B.快速排序

C. 冒泡排序

D。堆排序

2. 应用题

(1) 设待排序得关键字序列为{12,2, 16, 30, 28, 10, 16*, 20, 6, 18}, 试分别写出使用以下排序方法, 每趟排序结束后关键字序列得状态.

① 直接插入排序

② 折半插入排序

③ 希尔排序(增量选取 5, 3, 1)

④ 冒泡排序

⑤ 快速排序

⑥ 简单选择排序

⑦ 堆排序

⑧ 二路归并排序

①直接插入排序

[2 12] 16 30 28 10 16* 20 6 18
[2 12 16] 30 28 10 16* 20 6 18
[2 12 16 30] 28 10 16* 20 6 18
[2 12 16 28 30] 10 16* 20 6 18
[2 10 12 16 28 30] 16* 20 6 18
[2 10 12 16 16* 28 30] 20 6 18
[2 10 12 16 16* 20 28 30] 6 18
[2 6 10 12 16 16* 20 28 30] 18
[2 6 10 12 16 16* 18 20 28 30]

② 折半插入排序 排序过程同①

③ 希尔排序 (增量选取 5,3,1)

10 2 16 6 18 12 16* 20 30 28 (增量选取 5)
6 2 12 10 18 16 16* 20 30 28 (增量选取 3)
2 6 10 12 16 16* 18 20 28 30 (增量选取 1)

④ 冒泡排序

2 12 16 28 10 16* 20 6 18 [30]
2 12 16 10 16* 20 6 18 [28 30]
2 12 10 16 16* 6 18 [20 28 30]
2 10 12 16 6 16* [18 20 28 30]
2 10 12 6 12 [16 16* 18 20 28 30]
2 6 10 12 [16 16* 18 20 28 30]
2 6 10 12 [16 16* 18 20 28 30]

⑤ 快速排序

12 [6 2 10] 12 [28 30 16* 20 16 18]
6 [2] 6 [10] 12 [28 30 16* 20 16 18]
28 2 6 10 12 [18 16 16* 20] 28 [30]
18 2 6 10 12 [16* 16] 18 [20] 28 30

16* 2 6 10 12 16* [16] 18 20 28 30

左子序列递归深度为1,右子序列递归深度为3

⑥ 简单选择排序

```

2      [12 16 30 28 10 16* 20 6 18]
2 6    [16 30 28 10 16* 20 12 18]
2 6 10 [30 28 16 16* 20 12 18]
2 6     10 12 16 [28 16 16* 20 30 18]
2 6     10 12 16 [28 16* 20 30 18]
2 6     10 12 16 16* [28 20 30 18]
2 6     10 12 16 16* 18 [20 30 28]
2 6     10 12 16 16* 18 20 [28 30]
2 6     10 12 16 16* 18 20 28 [30]

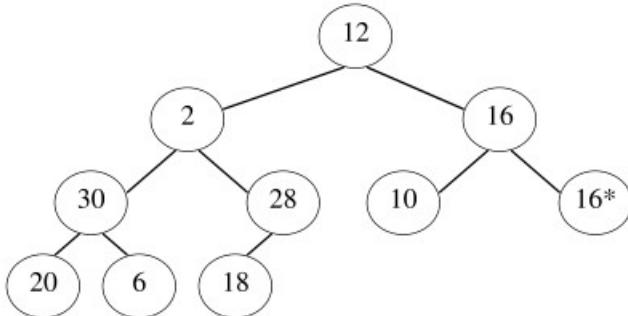
```

⑧ 二路归并排序

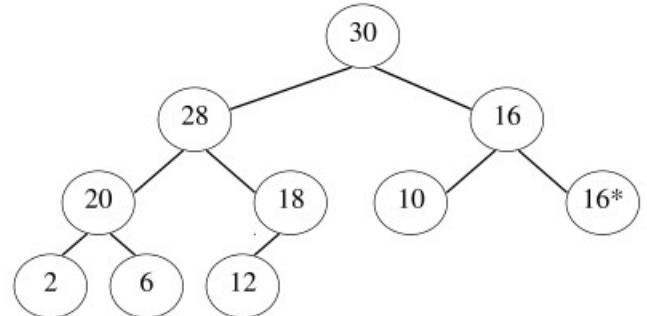
$$\begin{array}{cccccc}
 \underline{2} & \underline{12} & \underline{16} & \underline{30} & \underline{10} & \underline{28} \\
 \underline{2} & \underline{1} & \underline{2} & \underline{1} & \underline{6} & \underline{3} & \underline{0} \\
 \underline{2} & \underline{10} & \underline{1} & \underline{2} & \underline{1} & \underline{6} & \underline{16} \\
 \underline{\underline{2}} & \underline{\underline{6}} & \underline{\underline{10}} & \underline{\underline{12}} & \underline{\underline{16}} & \underline{\underline{16}} & \underline{\underline{30}}
 \end{array}$$

⑦ 堆排序

第一步, 形成初始大根堆 (详细过程略), 第二步做堆排序。

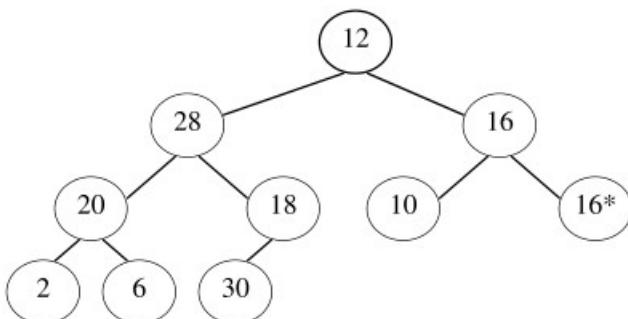


初始排序 不就是大根堆

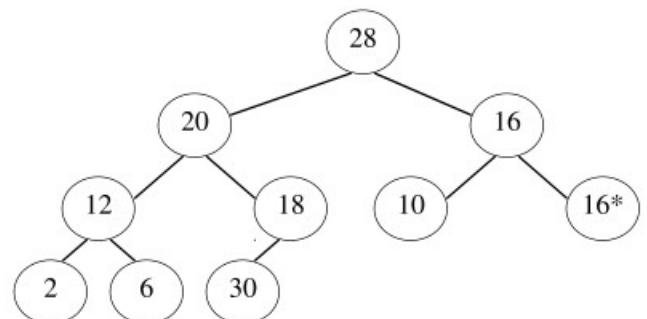


根堆

形成初始大

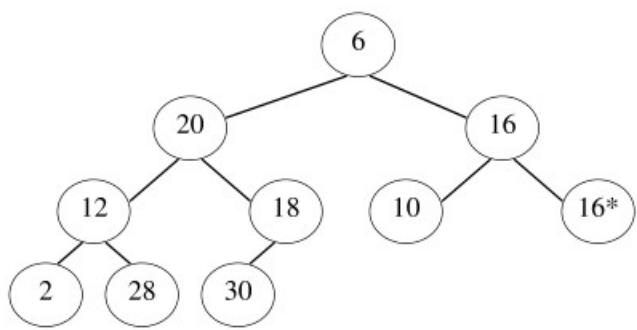


交换 1 与 10 对象

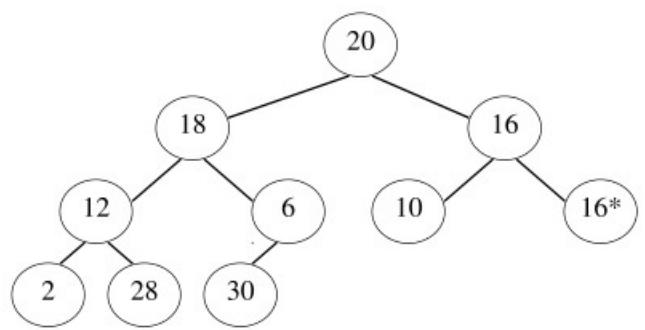


堆

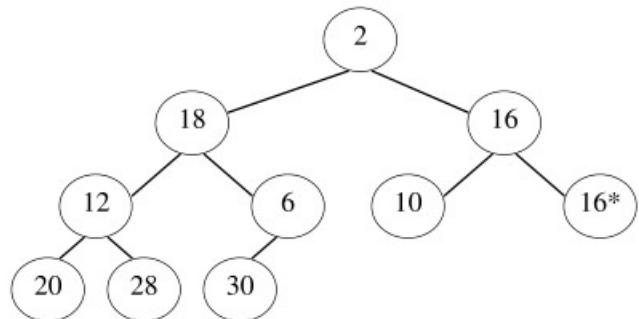
从 1 到 9 重新形成



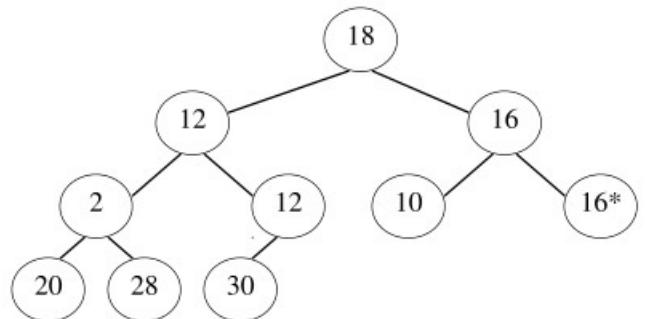
交换 1 与 9 对象
形成堆



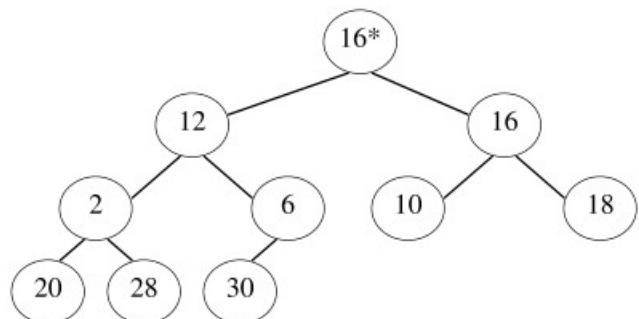
从 1 到 8 重新



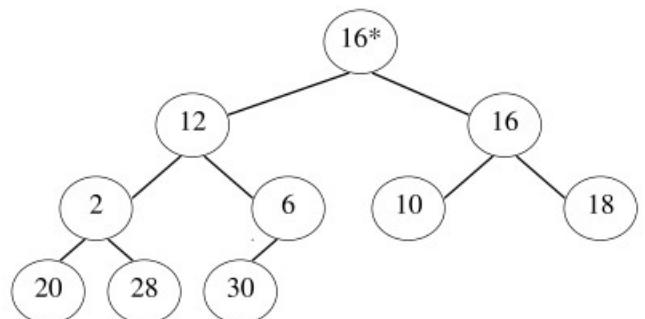
交换 1 与 8 对象



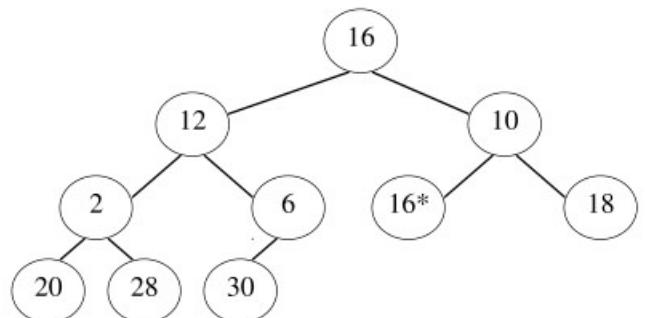
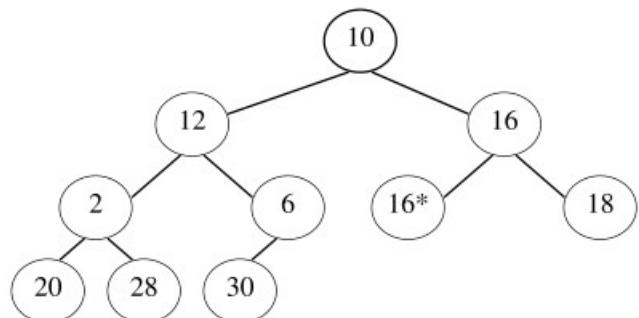
从 1 到 7 重新形成堆



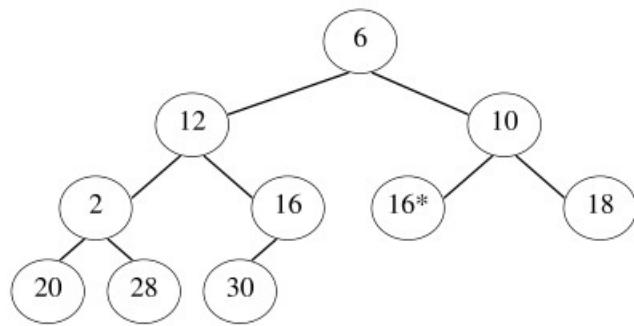
交换 1 与 7 对象



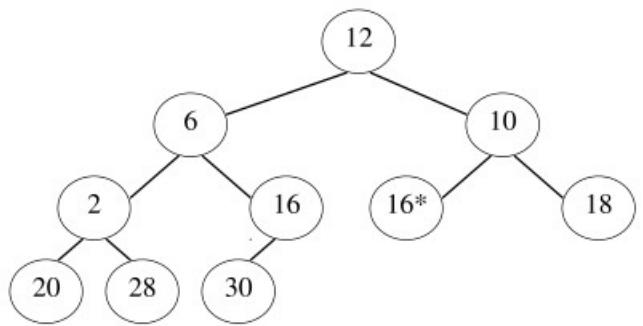
从 1 到 6 重新形成堆



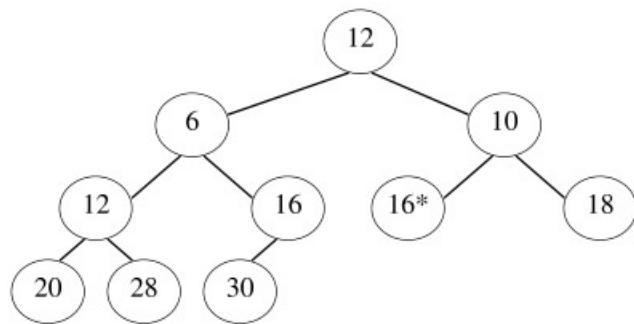
交换 1 与 6 对象



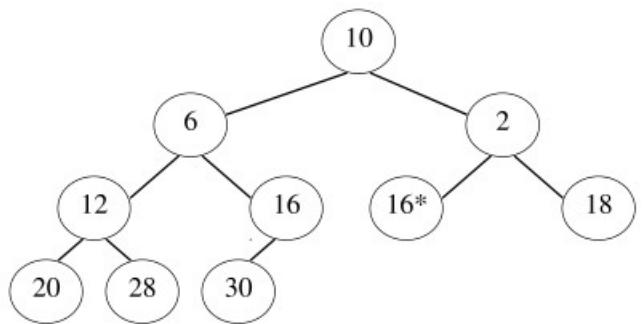
从 1 到 5 重新形成堆



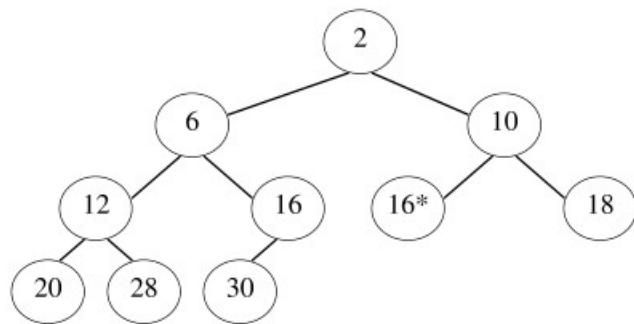
交换 1 与 5 对象



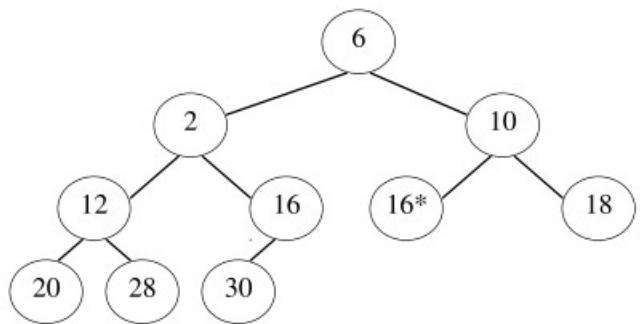
从 1 到 4 重新形成堆



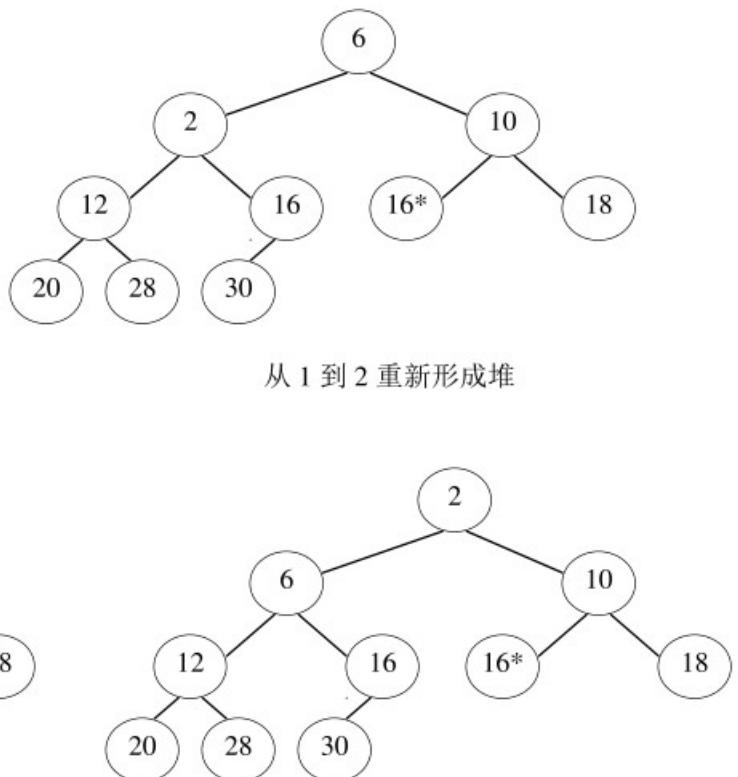
交换 1 与 4 对象



从 1 到 3 重新形成堆



交换 1 与 3 对象



交换 1 与 2 对象

得到结果

(2) 给出如下关键字序列 { 3 21, 156, 57, 46, 28, 7, 331, 33, 34, 63 }, 试按链式基数排序方法,列出每一趟分配与收集得过程。

3。算法设计题

(1)试以单链表为存储结构,实现简单选择排序算法。

```
void ListSelectSort(LinkedList head)
//本算法一趟找出一个关键字最小得结点,其数据与当前结点进行交换;若要交换指针,则
须记下
//当前结点与最小结点得前驱指针
p = head->next;
while (p != null)
    {q=p->next; r=p;      // 设 r 就是指向关键字最小得结点得指针
     while (q!=null)
        {if (q->data < r->data)   r=q;
         q=q->next;
     }
     if (r != p)   r->data -->p->data;
     p=p->next;
    }
```

(2) 有 n 个记录存储在带头结点得双向链表中,现用双向冒泡排序法对其按上升序进行排序,请写出这种排序得算法。(注: 双向冒泡排序即相邻两趟排序向相反方向冒泡)。

```
typedef struct node
{ ElemType data;
    struct node *prior, *next;
}node, *DLinkedList;
void TwoWayBubbleSort(DLinkedList la)
//对存储在带头结点得双向链表 la 中得元素进行双向起泡排序。
{int exchange=1; // 设标记
 DLinkedList p, temp, tail;
 head = la           //双向链表头, 算法过程中就是向下起泡得开始
结点
 tail=null;          //双向链表尾, 算法过程中就是向上起泡得开始结点
 while (exchange)
 {p=head->next;      // p 就是工作指针, 指向当前结点
  exchange=0;          //假定本趟无交换
  while (p->next!=tail) // 向下(右)起泡,一趟有一最大元素沉底
    if (p->data > p->next->data) //交换两结点指针, 涉及 6 条链
      {temp=p->next; exchange=1; //有交换
       p->next=temp->next;temp->next->prior=p //先将结点
从链表上摘下
       temp->next=p; p->prior->next=temp; // 将 temp
插到 p 结点前
       temp->prior=p->prior; p->prior=temp;
     }
```

```

    e l s e   p=p->next; //无交换，指针后移
        t a i l=p; //准备向上起泡
        p =tail-> p r i o r;
    w h i l e   (e x c h a n g e && p ->p r i o r!=he a d ) //向上(左)起泡,一趟有一最小元素冒出
        i f   (p->data<p-> p r i o r->d a t a) //交换两结点指针,涉及 6 条链
            { t e m p =p-> prior;   ex c h a n g e=1; //有交换
                p-> p r i o r=temp-> p r i o r; temp-> prior->nex t =p; //先将temp 结点从链表上摘下
                t e m p->prior=p;   p-> ne x t ->p r i o r=t e m p; //将temp 插到 p 结点后(右)
                t e m p-> next=p->next;   p->next=t e m p;
            }
            e l s e   p=p-> prior; // 无交换，指针前移
            he a d=p; //准备向下起泡
        } // w h i l e   (ex c h a n g e)
    } //算法结束

```

(3) 设有序放置得 n 个桶,每个桶中装有一粒砾石,每粒砾石得颜色就是红,白,蓝之一。要求重新安排这些砾石,使得所有红色砾石在前,所有白色砾石居中,所有蓝色砾石居后,重新安排时对每粒砾石得颜色只能瞧一次,并且只允许交换操作来调整砾石得位置.

[题目分析]利用快速排序思想解决。由于要求“对每粒砾石得颜色只能瞧一次”,设 3 个指针 i , j 与 k , 分别指向红色、白色砾石得后一位置与待处理得当前元素。从 $k=n$ 开始,从右向左搜索,若该元素就是兰色,则元素不动,指针左移(即 $k-1$);若当前元素就是红色砾石,分 $i>j$ (这时尚没有白色砾石) 与 $i<j$ 两种情况。前一情况执行第 i 个元素与第 k 个元素交换,之后 $i+1$;后一情况, i 所指得元素已处理过(白色), j 所指得元素尚未处理,应先将 i 与 j 所指元素交换,再将 i 与 k 所指元素交换.对当前元素就是白色砾石得情况,也可类似处理。

为方便处理,将三种砾石得颜色用整数 1、2 与 3 表示。

```

v o i d Q k S o r t (r e c t y p e r [ ], i n t  n )
// r 为含有 n 个元素得线性表,元素就是具有红、白与兰色得砾石,用顺序存储结构存储,
//本算法对其排序,使所有红色砾石在前,白色居中, 兰色在最后。
{ i n t  i=1, j=1, k=n, temp;
w h i l e  (k !=j)
    { w h i l e  (r [k]、key==3) k--; // 当前元素就是兰色砾石, 指针左移
        i f   (r [k]、key==1) // 当前元素就是红色砾石
            i f  (i>=j) { t e m p =r [k]; r [k] =r [i]; r [i]=t e m p; i++; }
            //左侧只有红色砾石,交换 r [k] 与 r [i]
        e l s e      { t e m p =r [j]; r [j]=r [i]; r [i]=t e m p; j++; }
            //左侧已有红色与白色砾石,先交换白色砾石到位
            t e m p =r [k]; r [k] =r [i]; r [i]=t e m p; i++;
            //白色砾石(i 所指) 与待定砾石(j 所指)
        } //再交换 r [k] 与 r [i],使红色砾石入位。
    i f   (r [k]、key==2)

```

```

    i f  ( i <=j) { temp=r[k]; r[k] =r [ j]; r [ j] =temp; j++; }
        // 左侧已有白色砾石, 交换 r [k] 与 r [ j ]
    e l s e      { temp=r[k]; r[k] = r [ i]; r[i]=temp; j=i+1; }
        //i、 j 分别指向红、白色砾石得后一位置
} //while
i f  ( r [k]==2) j++; /* 处理最后一粒砾石
e l s e i f (r [k]==1) { temp=r[j]; r[ j ]=r[i]; r [ i] =temp; i ++; j++; }
//最后红、白、兰色砾石得个数分别为: i—1; j—i; n—j+1
} //结束 QkS o r 算法

```

[算法讨论]若将 j (上面指向白色)瞧作工作指针, 将 r[1,, j—1] 作为红色, r [j,, k-1] 为白色, r [k,, n] 为兰色。从 j=1 开始查瞧, 若 r [j] 为白色, 则 j=j+1; 若 r [j] 为红色, 则交换 r [j] 与 r [i], 且 j=j+1, i=i+1; 若 r [j] 为兰色, 则交换 r [j] 与 r [k]; k=k-1。算法进行到 j> k 为止。

算法片段如下:

```

i nt i=1 , j=1, k=n;
w h i l e (j<=k)
    i f (r [j] ==1) //当前元素就是红色
        { t e m p=r [i] ; r [i] =r[j]; r [j]=t e m p; i ++; j++; }
    e l s e i f (r [j]==2) j++; //当前元素就是白色
        e l s e // ( r [j] ==3 当前元素就是兰色
            { t e m p=r [j]; r [j]=r[k]; r [k] =temp; k--; }

```

对比两种算法, 可以瞧出, 正确选择变量(指针)得重要性。

(4) 编写算法, 对 n 个关键字取整数值得记录序列进行整理, 以使所有关键字为负值得记录排在关键字为非负值得记录之前, 要求:

- ① 采用顺序存储结构, 至多使用一个记录得辅助存储空间;
- ② 算法得时间复杂度为 O(n)。

(5) 借助于快速排序得算法思想, 在一组无序得记录中查找给定关键字值等于 key 得记录。设此组记录存放于数组 r [l,, n] 中。若查找成功, 则输出该记录在 r 数组中得位置及其值, 否则显示 “not find” 信息。请简要说明算法思想并编写算法。

[题目分析]把待查记录瞧作枢轴, 先由后向前依次比较, 若小于枢轴, 则从前向后, 直到查找成功返回其位置或失败返回 0 为止。

```

i nt i ndex (R e c T y p e R [], i nt l,h, datatype key )
{
    i nt i =l, j=h;
    w h i l e (i < j)
        { w h i l e ( i <= j && R [j] >key) j--;
            i f (R [j] < key) r e t u r n j;
            w h i l e ( i <= j && R [i] < key) i++;
            i f (R [i] < key) r e t u r n i;
        }
        p r i n t f ("Not f i n d " ) ; r e t u r n 0;
} / / i ndex

```

(6) 有一种简单得排序算法, 叫做计数排序。这种排序算法对一个待排序得表进行排序, 并将排序结果存放到另一个新得表中。必须注意得就是, 表中所有待排序得关键字互不相同, 计数排序算法针对表中得每个记录, 扫描待排序得表一趟, 统计表中有多少个记录得关键

字比该记录得关键字小.假设针对某一个记录,统计出得计数值为 c , 那么, 这个记录在新得有序表中得合适得存放位置即为 c .

- ① 给出适用于计数排序得顺序表定义;
- ② 编写实现计数排序得算法;
- ③ 对于有 n 个记录得表,关键字比较次数就是多少?
- ④ 与简单选择排序相比较,这种方法就是否更好?为什么?

```
typ ede f struct
{
    int key; datat ype info} RecType
void CountSort(RecType a[], b [], int n) //计数排序算法, 将 a 中
记录排序放入 b 中
{ for(i=0; i < n; i++) // 对每一个元素
    { for (j=0, cnt=0; j < n; j++)
        if(a [j] < key < a[i]、key) cnt++; //统计关键字比它小得元素个数
        b[c nt]=a [i];
    }
} //Count_Sort
```

(3) 对于有 n 个记录得表, 关键码比较 n^2 次。
(4) 简单选择排序算法比本算法好. 简单选择排序比较次数就是 $n(n-1)/2$, 且只用一个交换记录得空间; 而这种方法比较次数就是 n^2 , 且需要另一数组空间。

[算法讨论] 因题目要求“针对表中得每个记录, 扫描待排序得表一趟”, 所以比较次数就是 n^2 次。若限制“对任意两个记录之间应该只进行一次比较”, 则可把以上算法中得比较语句改为:

```
for (i=0;i<n; i++) a [i], co unt=0; //各元素再增加一个计数域, 初始化为
0
for (i=0; i < n; i++)
    for (j =i+1; j < n ;j++)
        if (a[i] < key < a[j]、key) a[j], co unt++; else ea[i], co unt++;
```