2.1. 一元稀疏多项式的求导算法

写出一元稀疏多项式的求导算法，用带表头结点的单链表存储该一元稀疏多项式，Lb 为头指针，用类 C 语言描述该求导算法，不另行开辟存储空间，删除无用结点，并分析算法的时间复杂度。该链表的数据结构如下：

```
typedef struct LNode{
    float    coe;  //系数
    int      exp;   //指数
    struct   LNode   *next;   //指针
} LNode , *LinkList ;
```

求导算法如下：

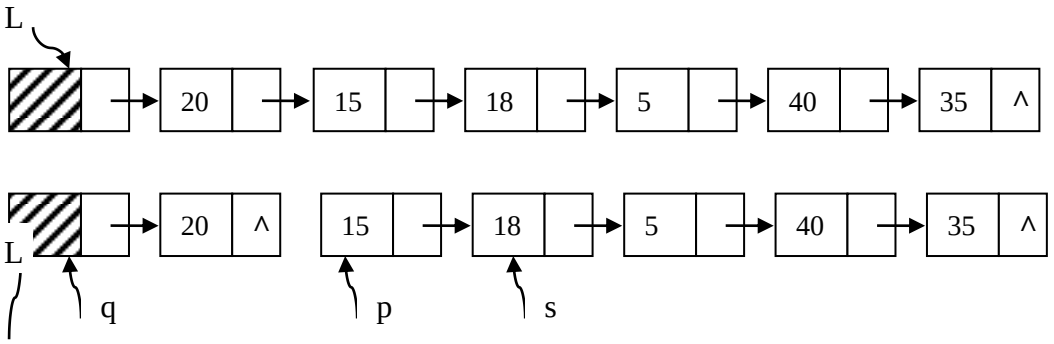
```
void   Differential(LinkList &Lb)
 { //求导算法，Lb 为链表头指针
 LinkList p, pre; //定义指针变量 p, pre
 pre=Lb; //初始化变量，pre 为 p 的前驱
 p=pre->next; // p 为当前待处理的结点
 while ( p ) //遍历链表
  {//逐个结点（数据项）求导处理
   if ( p->exp != 0 )//指数不等于零，非常数项
    {   p->coe = p->coe * p->exp ; //修改系数
        p->exp = p->exp – 1 ; //修改指数
        pre = pre->next ; //后移前驱指针 pre
    }
   else//指数等于零，常数项
    {   pre->next = p->next ; //逻辑删除常数项结点
        free ( p ); //物理删除常数项结点，即释放所占内存
    }
    p = pre->next ; //处理下一结点
   }
 }// Differential
```

时间复杂度为: O(n)

2.2. 单链表存储结构的排序算法

排序算法：将一组整数排序成非递减有序序列。用带头结点的单链表存储，L 为头指针，用类 C 语言写出该排序算法，不另行开辟存储空间，并分析算法的时间复杂度。该单链表的数据结构如下：

```
typedef struct LNode{
        int    data; //数据域
    struct    LNode    *next;    //指针域
        } LNode , *LinkList ;
void    Sort(LinkList    &L)//L 为链表头指针
{    //排序算法如下：将 L 排序成非递减单链表
    LinkList    q,p,s;    //定义变量
    if (L->next==NULL) { printf("空表\n");    return;}
//拆成 2 个链表，第 1 个链表只有头结点和首元结点，其余结点为第 2 个链表
    q=L;//q 为待处理结点的前驱
    p=q->next->next;// 第 2 个链表的头指针为 p
    q->next->next=NULL; // 第 1 个链表的头指针为 L
    //直接插入排序方法：第 2 个链表逐个结点插入到第 1 个链表
    while(p) //遍历链表
    { //比较数据域大小，定位合适的插入位置的前驱 q
        while(q->next && p->data >= q->next->data)    q=q->next;
        s=p->next; //保留下一个待处理结点
        p->next=q->next; //本条语句和下一条语句：p 结点插入到 q 结点的后面
        q->next=p;
        p=s; //处理下一结点
        q=L; //q 归位链表头结点 L
    }
}//sort
```



2.3. 设 H 为具有 n(n>0，n 很大且未知)个数据元素的单链表的头结点指针，试采用 C 语言编写一个程序，完成将单链表中第 n/2 个数据元素之后的全部数据元素倒置的功能。要求不另行开辟存储空间，算法的时间复杂度不超过 O(n)。

单链表结点的数据类型描述如下：

```
typedef struct Lnode {
  int data;    //数据元素为整数
  struct Lnode    *next;
}Lnode, *LinkList;

void ReverseN2(LinkList &H)
{//将单链表的正中间位置结点之后的全部结点倒置的功能
   LinkList p,q,s;
   p=H;   //初始化变量，p 指向头指针
   q=H;   //初始化变量，q 指向头指针
//定位链表正中间位置结点。p 结点走到表尾，q 结点在正中间位置
   while (p)
   {   if (p->next) //如果不是表尾结点
           p=p->next->next;   // p 指向下一结点的下一结点
       else//如果是表尾结点
           break;     //退出
       q=q->next;    //处理下一结点 q 指向下一结点
   }

   //拆成 2 个链表，第 1 个链表：头结点开始到正中间位置 q 结点
   //第 2 个链表：q 结点（不含 q）之后结点组成第 2 个链表
   p=q->next; // p 指向 q 结点的后继结点，p 为第 2 个链表的头指针
   q->next=NULL; //第 1 个链表的尾结点指针为 q
   //头插法（逆序），p 指向的第 2 个链表逐个逆序插入 q 结点的后面
   //实现倒置功能
   while (p)
   {   s= p->next; //保留下一个待处理结点
       p->next=q->next; //本条语句和下一条语句：p 结点插入到 q 结点的后面
       q->next=p;
       p=s; //处理下一结点
   }
}//ReverseN2
```

3.1．请写出如下递归程序的输出结果。

```
#include <stdio.h>
#include <stdlib.h>
int Fb(int x);
void main( )
{ int k=40, val;
   val= Fb(k);
  printf(" %d , %d 。" , val, k );
}

int Fb(int x)
{ int y;
  if (x <= 5 )   y = x / 5;     /* x/5 表示 x 整除 5*/
  else    y = Fb(x - 26) + x/3;
  printf("%d , ", x );
  printf("%d ; ", y );
  return ( y );
}
```

（1）主程序执行【int k=40, val; val= Fb(k);】后，调用递归函数 **Fb(40)**，子程序 **Fb(40)**入栈顺序为：

入栈顺序	堆栈		
	x	y	
3	**-12**	**Fb(-12)=-12/5=-2**	**←栈顶 Top**
2	**14**	**Fb(14)=Fb(14-26)+14/3= Fb(-12)+4**	
1	**40**	**Fb(40)=Fb(40-26)+40/3= Fb(14)+13**	**←栈底 Bottom**

即：

栈顶 Top→	**x=12，y=Fb(-12)=-12/5=-2**
	x=14，y=Fb(14)=Fb(14-26)+14/3= Fb(-12)+4
栈底 Bottom→	**x=40，y=Fb(40)=Fb(40-26)+40/3= Fb(14)+13**

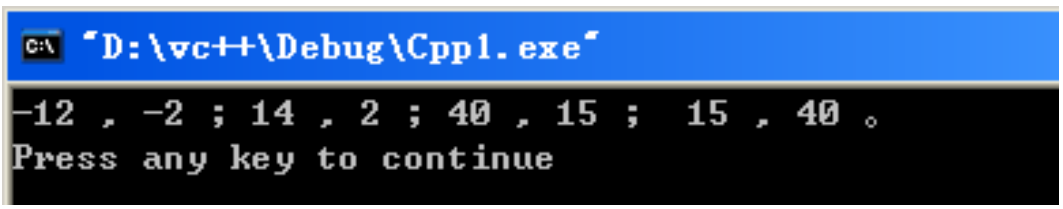
（2）子程序 **Fb( )**出栈顺序：

出栈次序	当前栈顶元素		输出数据	
	x	y	printf("%d , ", x);	printf("%d ; ", y);
1	**-12**	**Fb(-12)=-12/5=-2**	**-12，**	**-2；**
2	**14**	**Fb(14)= Fb(-12)+4=-2+4=2**	**14，**	**2；**
3	**40**	**Fb(40)= Fb(14)+13=2+13=15**	**40，**	**15；**

（3）主程序【**val= Fb(k);**】执行后【**printf(" %d , %d 。" , val, k);**】的输出结果：**15，40**。

因此，该程序执行结果为：**-12，-2；14，2；40，15；**15，40。

程序运行结果截图如下：



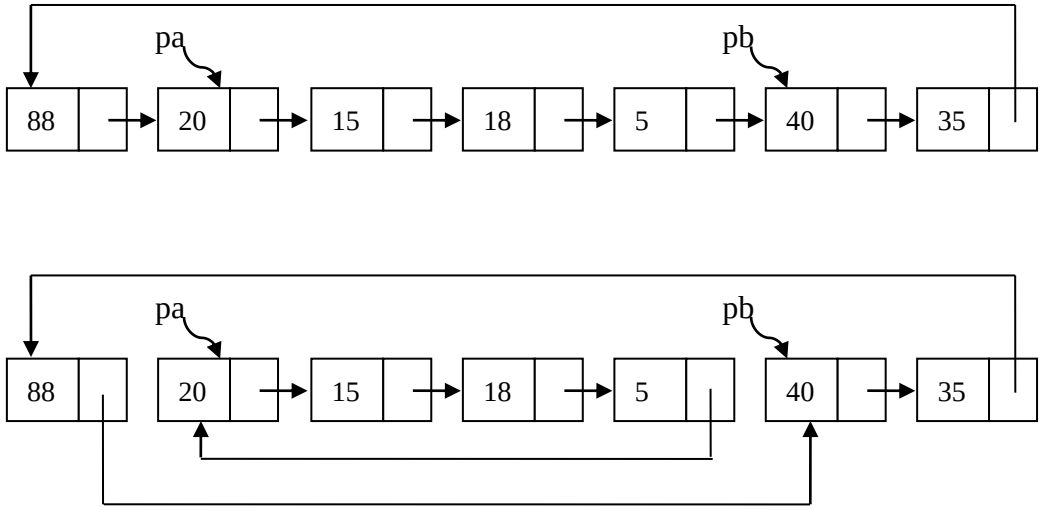
3.2. 简述下列算法的功能

```
void Split(Lnode *s , Lnode *q )
{
   Lnode *p;
   p=s;
   while ( p->next !=  q ) p = p->next;
   p->next = s;
}

void AtoBB(Lnode *pa , Lnode *pb )
{
   //pa和pb分别指向单循环链表( 结点数 > 1 )中的两个结点.
   split(pa,pb);
   split(pb,pa);
}
```

解：

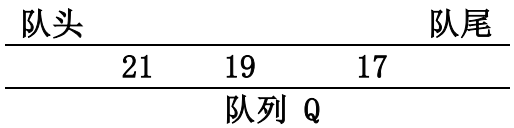
该算法的功能为：将原单循环链表从 pa 和 pb 所指向的结点断裂，即原来指向 pb 结点的链，转向指 pa，而原来指向 pa 结点的链，转向指 pb。这样就形成两个独立的单循环链表。



3.3．试读下列栈和队列操作的算法，请给出调用并执行 testit(3)后的所有输出结果。

```
//Stack为栈，Queue为队列
//InitStack(S)为构造一个空栈，InitQueue(Q)为构造一个空队列。
//EnQueue( Q , nb )表示入队列操作；DeQueue(Q)表示出队列操作。
void testit( int m )
{
   Stack S;
   Queue Q;
   int  na , nb , nm ;
   InitStack(S);   InitQueue(Q);
   na=12;   nb=21;   nm=m;
   while( na < nb )
   {
      Push( S ,na ); EnQueue( Q , nb );
      na++; nb -=2;  nm++;
   }
   printf("nm=%d,na=%d,nb=%d\n",nm,na,nb);
   while ( nm > 0 )
   {
      nm -= 2;
      na=POP(S);
      nb=DeQueue(Q) + na;
      printf("Out:%d\n",nb);
   }
}
```

解：



入栈/入队：

nm	na	nb
3	12	21
4	13	19
5	14	17
6	15	15

输出：nm=6, na=15, nb=15

出栈/出队：

nm	na	nb	输出
6 – 2 = 4	14	21 + 14 = 35	35
4 – 2 = 2	13	19 + 13 = 32	32
2 – 2 = 0	12	17 + 12 = 29	29

输出：Out:35

Out:32

Out:29

所以，输出结果为：

nm=6, na=15, nb=15

Out:35

Out:32

Out:29

4.1.1 模式匹配

设主串 S='aaabdaaabfaaabdaaabbdaaaabdaaabbda'，子串 T='aaabdaaabbda'，求解下列问题：

（1）求出模式 T 的 next[j]值；

（2）求出模式 T 的 nextval[j]值；

（3）在 S 中查找 T 至少需要几趟匹配？至少需要几次比较？模式匹配成功的位置序号？请给出详细的匹配过程。

提示：KMP 算法

$$\text{next}[j]=\begin{cases}0 & \text{当 } j=1 \text{ 时}\\ \max\{k \mid 1<k<j \text{ 且 } 'T_1\ldots T_{k-1}'='T_{j-(k-1)}\ldots T_{j-1}'\} & \\ 1 & \text{其他情况}\end{cases}$$

$$\text{nextval}[j]=\begin{cases}\text{next}[j] & \text{当 } T_j!=T_{\text{next}[j]}\\ \text{nextval}[\text{next}[j]] & \text{当 } T_j==T_{\text{next}[j]}\end{cases}$$

答：

j	1	2	3	4	5	6	7	8	9	10	11	12
模式串 T	*a*	*a*	a	b	d	a	a	a	b	b	d	a
Next[j]	0	1	2	3	1	1	2	3	4	5	1	1
NextVal[j]	0	0	0	3	1	0	0	0	3	5	1	0

匹配过程如下：

a a a b d a a a b f a a a b d a a a b b d a a a a b d a a a b b d a

a a a b d a a a b b …i=1→10, j=1→10, 比较 10 次。j=NextVal[10]=5

(a a a b)d …i=10→10, j=5→5, 比较 1 次。j=NextVal[5]=1

a …i=10→10, j=1→1, 比较 1 次。j=NextVal[1]=0，则 i++, j++

a a a b d a a a b b d a…i=11→23, j=1→13, 比较 12 次，匹配成功。

因此，至少需要 4 趟匹配，至少需要 10+1+1+12=24 次比较，模式匹配成功的位置序号为 i-T[0]=23-12=11。

4.1.2 模式匹配

设主串 S='ebababababcaababababcabadaaaac'，子串 T=' bababababcabad'，求解下列问题：
（1）求出模式串 T 的 Next[j]值；
（2）求出模式串 T 的 NextVal[j]值；
（3）请给出从主串的第 2 个字符开始的匹配过程，并回答在 S 中查找 T 至少需要几趟匹配？至少需要几次比较？模式匹配成功的位置序号？

提示：KMP 算法

$$\text{next}[j]=\begin{cases}0 & \text{当 } j=1 \text{ 时}\\ \max\{k \mid 1<k<j \text{ 且 } 'T_1\ldots T_{k-1}'='T_{j-(k-1)}\ldots T_{j-1}'\} & \\ 1 & \text{其他情况}\end{cases}$$

$$\text{nextval}[j]=\begin{cases}\text{next}[j] & \text{当 } T_j!=T_{\text{next}[j]}\\ \text{nextval}[\text{next}[j]] & \text{当 } T_j==T_{\text{next}[j]}\end{cases}$$

解：

j	1	2	3	4	5	6	7	8	9	10	11	12
模式串 T	b	a	b	a	b	a	b	c	a	b	a	d
Next[j]	0	1	1	2	3	4	5	6	1	1	2	3
NextVal[j]	0	1	0	1	0	1	0	6	1	0	1	3

从主串的第 **2** 个字符开始的匹配过程如下：

	i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
	主串 S	e	**b**	a	b	a	b	a	b	**a**	b	c	a	**a**	**b**	a	b	a	b	A	b	c	a	b	a	**d**	a	a	a	a	c
第一趟	模式串		**b**	a	b	a	b	a	b	**c**		比较 8 次，i=2→9，j=1→8。下一趟 j= NextVal[8]=6，i=9，即主串 S[9]和子串 T[6]比较																				
第二趟	模式串				(b | a | b | a | b)	**a**	b	c	a	**b**		比较 5 次，i=9→13，j=6→10。下一趟 j= NextVal[10]=0，则 i=i+1=13+1=14，j=j+1=0+1=1 即主串 S[14]和子串 T[1]比较																			
第三趟	模式串														**b**	a	b	a	b	a	b	c	a	b	a	**d**					
																匹配成功，比较 12 次，i=14→26，j=1→13，当 j=13 时匹配成功，返回模式匹配成功的位置序号：i－T 的串长=26-12=14，注意：最后一次（即 i=26, j=13 时）没有进行字符比较。																		

最少需要的趟数：3	最少需要的字符比较次数：8+5+12=25	模式匹配成功的位置序号：i-T[0]=26-12=14

4.2.设四维数组 B[1..3，2..8，0..5，1..8]以行主序顺序方法存储在一个连续的存储空间内,每一个数据元素占 2 个存储单元,且B[1,2,4,1]的存储地址是2000， 则B[2,3,4,5]的存储地址是 _____。

若为列主序存储，则[2,3,4,5]的存储地址是 _____。

解答：

行序存储，B[2,3,4,5]的存储地址是：

2000+((2-1)×(8-2+1)×(5-0+1)×(8-1+1)+(3-2)×(5-0+1)×(8-1+1)+(4-4) ×(8-1+1)+(5-1))×2

=2000+(336+48+0+4)×2=2776

列序存储，B[2,3,4,5]的存储地址是：

2000+((5-1)×(5-0+1)×(8-2+1)×(3-1+1)+(4-4)×(8-2+1)×(3-1+1)+(3-2) ×(3-1+1)+(2-1))×2

=2000+(504+0+3+1)×2=3016

4.3.已知广义表 A=((a, (b ,c)), (a, (b ,c),d)),则运算 head(head (tail (A)))的结果是_____ 。

解答:

head(head (tail (A)))

= head(head (((a, (b ,c),d))))

= head((a, (b ,c),d))

= a

4.4. 利用广义表的 Head(L)和 Tail(L)的运算，将元素 c 从广义表 L=(((a, b), e,(c, d)))中分离出来，其运算表达式为_____ 。

解答:

Head(Head(Tail(Tail(Head(Head(L))))))

=Head(Head(Tail(Tail(Head(((a,b),e,(c,d)))))))

= Head(Head(Tail(Tail(((a, b), e,(c, d))))))

= Head(Head(Tail((e,(c, d)))))

= Head(Head(((c, d))))

= Head((c, d))

= c

5.1. 某通信系统有八种字符： d、e、f、g、h、i、j，k，其权值分别为：0.24、0.10、0.03、0.15、0.09、0.06、0.29、0.04，请完成：

（1）构造 Huffman 树（要求所有结点左孩子的权值不大于右孩子的权值）；

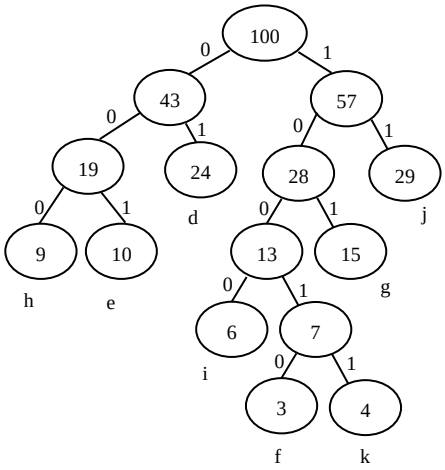
（2）据此设计出各个字符的 Huffman 编码；

（3）求出该 Huffman 树的带权路径长度 WPL；

（4）并译出下列报文：111000011011001010011。

解：

（1）Huffman 树：为表达方便，权值放大 100



（2）Huffman 编码

字符	Huffman 编码
d	01
e	001
f	10010
g	101
h	000
i	1000
j	11
k	10011

（3）求带权路径长度 WPL

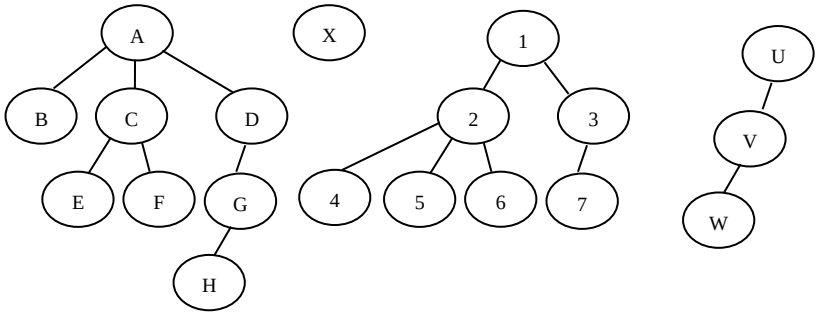
WPL=（（0.03+0.04）×5+0.06×4+（0.09+0.10+0.15）×3+（0.24+0.29）×2）=2.67

（4）译报文

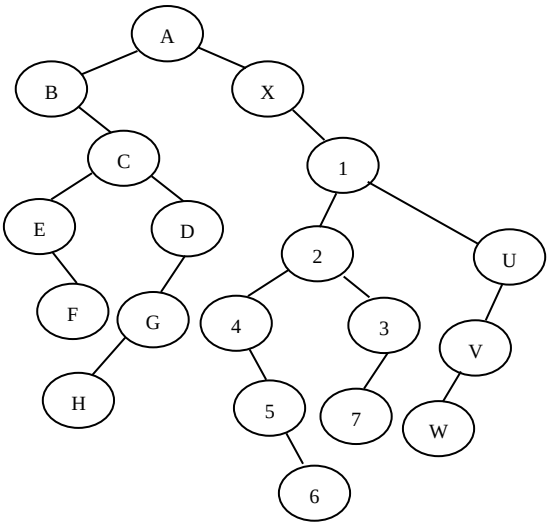
11	1000	01	101	10010	10011
j	i	d	g	f	k

报文译码为：

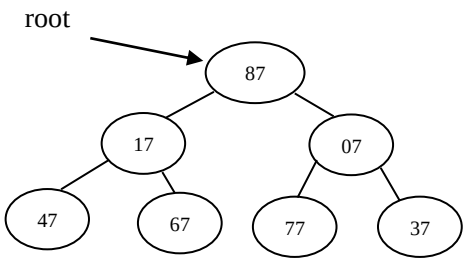
5.2.将下面的森林（F={T1，T2，T3，T4}）转换为对应的二叉树。



解：

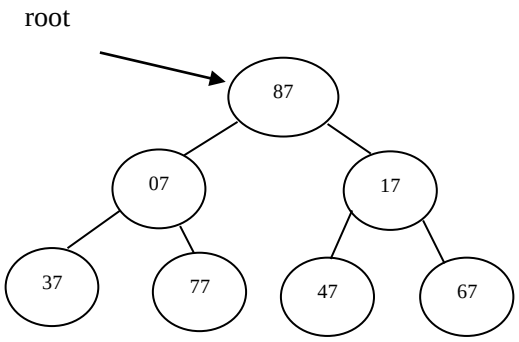


5.3. 以指向左侧二叉树的指针 root 作初始值，执行递归算法 ShiftTree(root)，请指出执行后的二叉树结构图。




```
ShiftTree( T )
{
      if (T == Null ) return( 0 );
      TL = T->Lchild;
      ShiftTree( TL);
      TR = T->Rchild;
      if ( ( TL <> Null ) && ( TR <> Null ) && ( TL->data > TR->data ) )
            {     T->Lchild = TR;
                  T->Rchild = TL;
            }
      ShiftTree( TR );
}
```

解：




```
0   ShiftTree( T )
1   {
2       if (T == Null ) return( 0 );
3       TL = T->Lchild;
4       ShiftTree( TL );
5       TR = T->Rchild;
6       if ( ( TL <> Null ) && ( TR <> Null ) && ( TL->data > TR->data ) )
7           {     T->Lchild = TR;
8                 T->Rchild = TL;
9           }
10      ShiftTree( TR );
11  }
```

5.4. 试证明：一棵非空的满 m 叉树上叶子结点数 n_0 和非叶子结点数 N 之间满足以下关系：$n_0 =（m - 1）*N + 1$

证明：

设分支总数为 B，结点总数为 M。

因为在满 m 叉树上，只存在度为 m 和度为 0 的结点，

所以，B = m*N

$M=n_0 + N$

又因为除了根结点外，每个结点有唯一的分支与之对应。

所以，M = B + 1 = m*N + 1

即有，$n_0 + N = m*N + 1$

也即，$n_0 =（m - 1）*N +1$

证毕。

总结：

1. 除了根结点外，每个结点有唯一的分支与之对应；
2. 满 m 叉树上，只存在度为 m 和度为 0 的结点。

5.5. 证明题

设结点 u 和结点 v 是树中的两个结点，且在对该树的先序遍历序列中 u 在 v 之前，而在其后序遍历序列中 u 在 v 之后，试证明结点 u 是结点 v 的祖先。

证明：用反证法证明本题：

假设 u 结点不是 v 结点的祖先结点，并设该树为 BT，其根结点为 r 结点。则 u 结点不在从 r 结点到 v 结点的路径上。

分两种情况讨论：

1. 若 u 结点是结点 v 的子树上的结点，则，在 BT 的先序遍历序列中，子树上的所有结点都在 v 结点之后，即 u 结点在 v 结点之后出现，故与原题条件矛盾，所以 u 结点不能是 v 的子树上的结点。

2. 若 u 结点不是结点 v 的子树上的结点，即 u 结点不为 v 结点的子孙结点,可设从 r 结点到 v 结点的路径序列为：r，r_1，r_2，…，r_k，v

即，r，r_1，r_2，…，r_k 是从 r 结点到 v 结点的路径上的结点，都是 v 结点的祖先结点。

则，u 结点只能在以 r ，r1，r2，…，rk 为根的，且不包含 v 结点作为子孙结点的子树中。

对 r，r_1，r_2，…，r_k 中的任意一个结点 x，

若 v 结点在 x 结点的子树 Xi 上，u 结点在 x 结点的子树 Xj 上，其中 i<j。于是，在对以 x 结点为根的树做先序遍历时，v 结点应在 u 结点之前出现（x 结点为根的先序遍历是 BT 的先序遍历序列的子序列），从而与原题的条件矛盾；

若 u 结点在 x 结点的子树 Xi 上，v 结点在 x 结点的子树 Xj 上，其中 i<j。于是，在对以 x 结点为根的树做后序遍历时，u 结点应在 v 结点之前出现（x 结点为根的后序遍历是 BT 的后序遍历序列的子序列），从而与原题的条件矛盾。

所以，u 结点不能是 r，r_1，r_2，…，r_k 以外的结点，只能是 r，r_1，r_2，…，r_k 中的某个结点，即 u 结点是 v 结点的祖先结点。证毕。

要点：1. 先序遍历和后序思想；

2. 结点的层次关系；

3. 证明思路清晰。

5.6．证明题

设结点 u 和结点 v 是树中的两个结点，且结点 u 是结点 v 的祖先。试证明在对该树的先序遍历序列中 u 在 v 之前，而在其后序遍历序列中 u 在 v 之后。

证明：

树的先序遍历算法：先访问树的根结点，然后依次先序遍历根的每棵子树。

树的后序遍历算法：先依次后序遍历根的每棵子树，然后访问树的根结点。

因为结点 u 是结点 v 的祖先，则以结点 u 为根的子树必包括结点 v，v 是 u 的子树。

根据树的先序遍历算法，当遍历到以结点 u 为根的子树时，第一个遍历的结点为 u，v 必然在 u 的后面，即对该树的先序遍历序列中 u 在 v 之前。

根据树的后序遍历算法，当遍历到以结点 u 为根的子树时，最后一个遍历的结点为 u，v 必然在 u 的前面，即对该树的后序遍历序列中 u 在 v 之后。

故命题得证。

要点：

1、说明树的先序遍历算法、后序遍历算法。

2、论证树的先序遍历序列中 u 在 v 之前。

3、论证树的后序遍历序列中 u 在 v 之后。

5.7．证明题

设一棵度为 k 的非空树上的叶子结点数为 n_0，度为 i 的结点数为 n_i（$1 \le i \le k$），试证明以下关系成立。

$$n_0 = 1 + \sum_{i=1}^{k}(i-1)* n_i$$

证明：

设分支总数为 B，结点总数为 N。

根据题意，有

$B= n_1 + 2*n_2 + \dots + k*n_k$

$N= n_0 + n_1 + n_2 + \dots + n_k$

又因为**除了根结点外，每个结点有唯一的分支与之对应**。

所以，$N = B + 1$

即有，$n_0 + n_1 + n_2 + \dots + n_k = n_1 + 2*n_2 + \dots + k*n_k + 1$

也即，

$$n_0 = 1 + \sum_{i=1}^{k}(i-1) * n_i$$

证毕。

要点：1. **除了根结点外，每个结点有唯一的分支与之对应**；

2. 结点总数 = 分支总数+1。

5.8．证明题

试证明：若一个具有 p 个结点、q 条边的非连通无向图是一个森林(p>q)，则该森林中必有 p-q 棵树。

证明：

由题目知，一个具有 p 个结点、q 条边的非连通无向图是一个森林(p>q)。

设该森林有 x 棵树。这些树的结点数分别为 n_1、n_2、n_3、……、n_x。

因为除了根结点以外，树的每个结点有唯一的分支（即边）与之对应，即结点数=边数+1。

所以这些树的边数分别为 n_1-1、n_2-1、n_3-1、……、n_x-1。

因为 $p=n_1+n_2+n_3+\ldots\ldots+n_x$

所以

$q= (n_1-1)+(n_2-1)+(n_3-1)+\ldots\ldots+(n_x-1)=(n_1+n_2+n_3+\ldots\ldots+n_x) – x=p – x,$

即 $x= p– q$

故该森林中必有 p-q 棵树。

证毕。

关键点：结点与分支（边）的关系；计算结点数与边数；推导过程。

5.9. 试用归纳法证明：高度为 h 的二叉树的结点总数不超过 2^h-1

证明：

对高度 h 采用第一归纳法。令 n 为二叉树的结点总数。

当 h=1 时，二叉树只含有根结点，所以，$n=2^1-1=1$ 成立。

假设 h=m-1 时（m>1），有 $n\leq 2^{m-1}-1$。

则当 h=m 时，由于高度为 m 的二叉树可在高度为 m-1 的二叉树上增加一层结点构成。

由于高度为 m-1 的二叉树的第 m-1 层上最多有 2^{m-2} 个结点，而每个结点最多引出两个分支，所以第 m 层上最多有 $2\times 2^{m-2}=2^{m-1}$ 个结点。

所以 $n\leq 2^{m-1}-1+2^{m-1}=2^m-1$ 结点。结论成立，即高度为 h 的二叉树的结点总数不超过 2^h-1。

证毕。

5.10. 对于那些所有非叶子结点（分支结点、非终端结点）均含有左右子树的二叉树 BT(即只有度为 2 和度为 0 的结点)：

（1）试问：具有 n 个叶子结点的这样的二叉树中共有多少个结点？

（2）试证明：$\sum_{i=1}^{n}2^{-(h_i-1)}=1$，其中 n 为叶子结点的个数，$h_i$ 表示第 i 个叶子结点所在的层次（设根结点所在层次为 1）

解答：

（1）设度为 2 的结点数为 n_2，由二叉树的性质 3（任何一棵二叉树的叶子结点数等于度为 2 的结点数+1），可得 $n=n_2+1$，则 BT 的结点总数为 $n+n_2=n+(n-1)=2n-1$。

（2）证明：

该问题的证明似乎可以采用归纳法证明，但由于本问题的问题规模（n，h_i）的规律很难掌握，所以归纳法证明非常困难。因为很难确定 n 个叶子结点分布在 BT 的哪个层次上，所以对 n 采用归纳法有难度，对 h_i 采用归纳法也受制于叶子结点数 n。下面采用计算的方法加以证明。

设叶子结点的最深层为 h。对出现在小于 h 层的每一个叶子结点，以该结点为根，向下补充结点，构造一棵满二叉树且使叶子结点出现在 BT 的第 h 层，且 BT 中不存在度为 1 的结点，所以可以得到一个深度为 h 的满二叉树 BT'。

根据满二叉树的性质，深度为 h 的满二叉树 BT'上共有 2^h-1 个结点，原 BT 的结点数为 2n-1，故得新补充的结点总数为$(2^h-1)-(2n-1)$。另外，以 BT 中每个原叶子结点为子树的根与它向下所补充的结点，一起也构成一棵深度为 k 的满二叉树，具有 2^k-1 个结点（每个原叶子结点向下所补充的结点数为 $2^k-1-1=2^k-2$，叶子结点在最深层 h 时，补充了 0 个结点），其中，深度 $k=h-h_i+1=h-(h_i-1)$。所以新补充的结点总数为：

$\sum_{i=1}^{n}(2^k-2)=\sum_{i=1}^{n}2^{h-(h_i-1)}-2n=(2^h-1)-(2n-1)$，整理得 $\sum_{i=1}^{n}2^{h-(h_i-1)}=2^h$，即

$$\sum_{i=1}^{n}2^{-(h_i-1)}=1$$

证毕。

5.11. 试证明：在任意非空二叉树 T 上，分支结点数 < 叶子结点数的充分必要条件是二叉树 T 上不存在度为 1 的结点。

证明：

在任意非空二叉树 T 上，设度为 2 的结点数是 n_2，度为 1 的结点数是 n_1，度为 0 的结点数（即叶子数）是 n_0，结点总数 n， 分支总数 B。所以分支结点数=n_2+n_1。

∵ 二叉树中结点总数 $n=n_0+n_1+n_2$(叶子数＋1 度结点数＋2 度结点数)

又∵二叉树中结点总数 n＝B+1 （ 分支总数＋根结点 ）

（因为除根结点外，每个结点必有一个直接前趋，即一个分支）

而分支总数 $B= n_1+2n_2$（1 度结点必有 1 个直接后继，2 度结点必有 2 个直接后继）

上面三式联立可得： $n_0+n_1+n_2= n_1+2n_2 +1$, 即 $n_0=n_2+1$

（1）必要条件（分支结点数 < 叶子结点数的必要条件是二叉树 T 上不存在度为 1 的结点），即证明：若分支结点数 < 叶子结点数，则二叉树 T 上不存在度为 1 的结点。

因为分支结点数 < 叶子结点数，即 $n_2+n_1 < n_0=n_2+1$，整理得 $n_1 < 1$，由于 n_1 为非负整数，故 $n_1 = 0$，即二叉树 T 上不存在度为 1 的结点。必要条件得证。

（2）充分条件（分支结点数 < 叶子结点数的充分条件是二叉树 T 上不存在度为 1 的结点），即证明：若二叉树 T 上不存在度为 1 的结点，则分支结点数 < 叶子结点数。

因为二叉树 T 上不存在度为 1 的结点，即 $n_1 = 0$，由于 n_1 为非负整数，

得 $n_1 < 1$，两边加 n_2，得 $n_2+n_1 < n_2+1$，因为 $n_0=n_2+1$，故 $n_2+n_1 < n_0$，即分支结点数 < 叶子结点数。

充分条件得证。

故命题得证。

5.12. 试证明：若森林 F 中有 n 个非终端结点，B 是由 F 变换得到的二叉树，则 B 中右指针域为空的结点有 n+1 个。

证明：

本题思路：右指针域为空的个数 =空指针域总数 - 左指针域为空的个数

（1）题目非终端结点数为 n，那么假设森林总结点数为 m，终端结点（即叶子结点）数为 m-n，指针域总数就是 2*m。

（2）除根结点外，每个结点直接都被其父结点指向，使用的指针域为 m-1 个，所以空指针域总数为 2*m -（m-1） = m+1

（3）m-n 个终端结点转化为二叉树后，都没有左孩子，左指针域就为空，即左指针域为空的个数是 m-n。因为森林转二叉树后，左孩子是第一个孩子，右孩子是兄弟，所以终端结点转化后一定无左孩子，右孩子可能有也可能没有，即转化后有两种可能：a）仍为叶子结点，b）不再是叶子结点，但只有右孩子。

所以，右指针域为空的个数=空指针域总数 - 左指针域为空的个数 =m + 1 -（m - n） = n + 1。

命题得证。

7.1.1. 哈希表问题

哈希函数 Hash (Key)=Key MOD 13，处理冲突函数 Hi=(Hash(key)+di) mod m (m 为哈希表长度，d_i=1,2,3,4,5,……)。

给定关键字序列：14，02，24，29，01，33，79，41，53，46，68，90。

试在 S.elem[0..15] 存储空间上，解答如下问题：

（1）构造如上给定关键字序列的哈希表，统计每个关键字的查找次数

（2）计算其查找成功时的平均查找长度 ASL

（3）计算其查找不成功时的平均查找长度 ASL

7.1.1 解：处理冲突函数 Hi=(Hash(key)+di) mod m (m 为哈希表长度 16，d_i=1,2,3,……)

（1）构造哈希表及每个关键字的查找次数

哈希表 S.elem[0..15]

地址	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
关键字		14	02	29	01	79	41	33	53	46	68	24	90			
查找成功时的比较次数		1	1	1	4	5	5	1	8	3	8	1	1			

（2）计算关键字查找成功时的平均查找长度 ASL=（Σ各关键字查找成功时的比较次数）/（关键字的个数）

查找成功时的 ASL=（1+1+1+4+5+5+1+8+3+8+1+1）/ 12 = 39/12 =13/4= 3.25

（3）计算关键字查找不成功时的平均查找长度 ASL=（Σ各地址查找不成功时的比较次数）/（不成功地址个数）

根据哈希函数 Hash (Key)=Key MOD 13，则第一次计算的哈希地址值只可能是 0~12，这些哈希地址查找不成功的比较次数为它到第一个没有存储关键字的地址的比较次数。因此查找不成功的比较次数如下表所示：

地址	0	1	2	3	4	5	6	7	8	9	10	11	12
查找不成功时的比较次数	1	13	12	11	10	9	8	7	6	5	4	3	2

查找不成功时的 ASL=（1+13+12+11+10+9+8+7+6+5+4+3+2）/13 = 91/13=7

7.1.2 哈希表问题

哈希函数 Hash (Key)=Key MOD 13，处理冲突函数 Hi=(Hash(key)+di) mod m（若 Hi<0，则 Hi=Hi+m），m 为哈希表长度 16，d_i=1，-1，2，-2，3，-3，4，-4，……）。

给定关键字序列：14，02，24，29，01，33，79，41，53，46，68，90。

试在 S.elem[0..15] 存储空间上，解答如下问题：

（1）构造如上给定关键字序列的哈希表，统计每个关键字的查找次数

（2）计算其查找成功时的平均查找长度 ASL

（3）计算其查找不成功时的平均查找长度 ASL

7.1.2. 解：处理冲突函数 Hi=(Hash(key)+di) mod m（若 Hi<0，则 Hi=Hi+m），m 为哈希表长度 16，d_i=1，-1，2，-2，3，-3，……

（1）构造哈希表及每个关键字的查找次数。给定关键字序列为 14，02，24，29，01，33，79，41，53，46，68，90。

哈希表 S.elem[0..15]={01,14,02,29,41,68,#,33,46,#,#,24,90,#,53,79}

地址	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
关键字	01	14	02	29	41	68		33	46			24	90		53	79
查找成功时的比较次数	3	1	1	1	4	4		1	2			1	1		7	5

（2）计算关键字查找成功时的平均查找长度 ASL=（∑各关键字查找成功时的比较次数）/（关键字的个数）

查找成功时的 ASL=（3+1+1+1+4+4+1+2+1+1+7+5）/ 12 = 31/12 ≈ 2.58

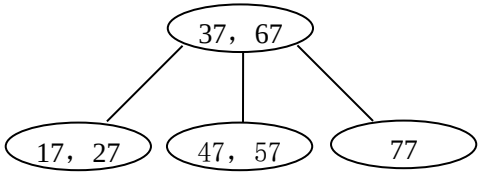
（3）计算关键字查找不成功时的平均查找长度 ASL=（∑各地址查找不成功时的比较次数）/（不成功地址个数）

根据哈希函数 Hash (Key)=Key MOD 13，则第一次计算的哈希地址值只可能是 0~12，这些哈希地址查找不成功的比较次数为它到第一个没有存储关键字的地址的比较次数。因此查找不成功的比较次数如下表所示：

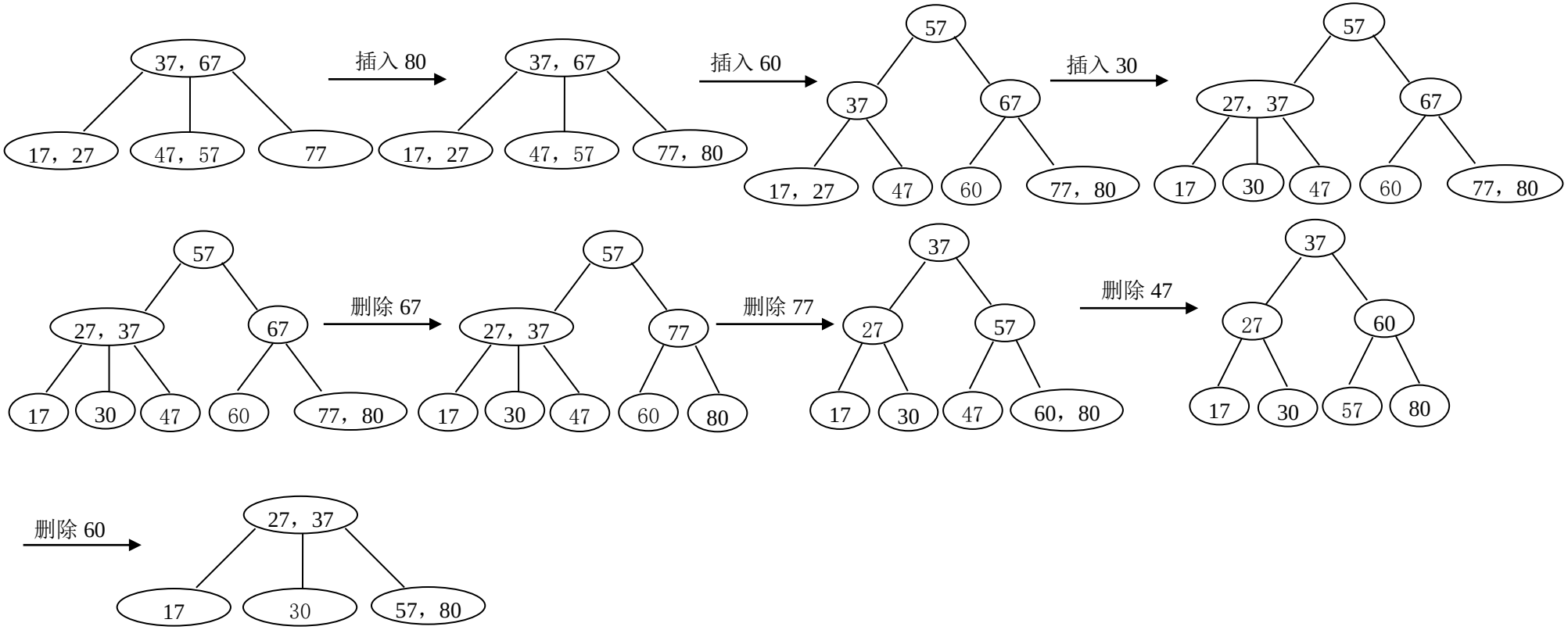
地址	0	1	2	3	4	5	6	7	8	9	10	11	12
查找不成功时的比较次数	7	9	8	6	4	2	1	3	2	1	1	3	2

查找不成功时的 ASL=（7+9+8+6+4+2+1+3+2+1+1+3+2）/13 = 49/13 ≈ 3.77

7.2. 请在下面的 3 阶（2-3）B_树上依次插入关键字 80、60、30，然后再依次删除关键字 67、77、47、60。试画出每次操作后的 B_树结构。



7.2. 解：


37，67
17，27
47，57
77
插入 80
37，67
17，27
47，57
77，80
插入 60
57
37
67
17，27
47
60
77，80
插入 30
57
27，37
67
17
30
47
60
77，80
57
27，37
67
17
30
47
60
77，80
删除 67
57
27，37
77
17
30
47
60
80
删除 77
37
27
57
17
30
47
60，80
删除 47
37
27
60
17
30
57
80
删除 60
27，37
17
30
57，80

7.3. 平衡二叉树

已知关键字序列为｛47, 57, 87, 77, 67, 27, 07, 97, 57, 37, 17｝，其中，$\underline{x}$ 表示值为 x 的另一关键字。按如下要求完成本题。

（1）针对给定关键字序列的不同排列，所构造出的不同形态的二叉排序树中，在最好和最坏情况下，该二叉排序树的高度各是多少？

（2）根据给定的关键字序列，构造一棵平衡二叉排序树。

（3）在等概率的情况下，计算查找成功时该平衡二叉排序树的 $ASL_{成功}$。

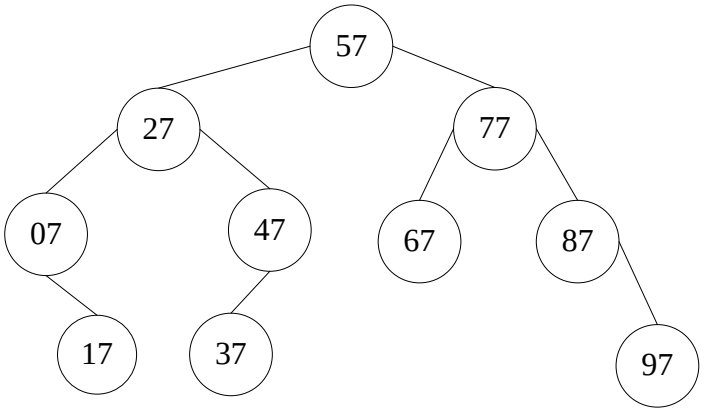
（4）在等概率的情况下，计算查找不成功时该平衡二叉排序树的 $ASL_{不成功}$。

解：

（1）最好情况下的二叉排序树的高度为：4，最坏情况下的二叉排序树的高度为：10

注意：二叉排序树最好情况下的高度为平衡二叉树的高度，最坏情况下的高度为不同的关键字个数。

（2）构造一棵平衡二叉排序树



详细构造过程如下页：

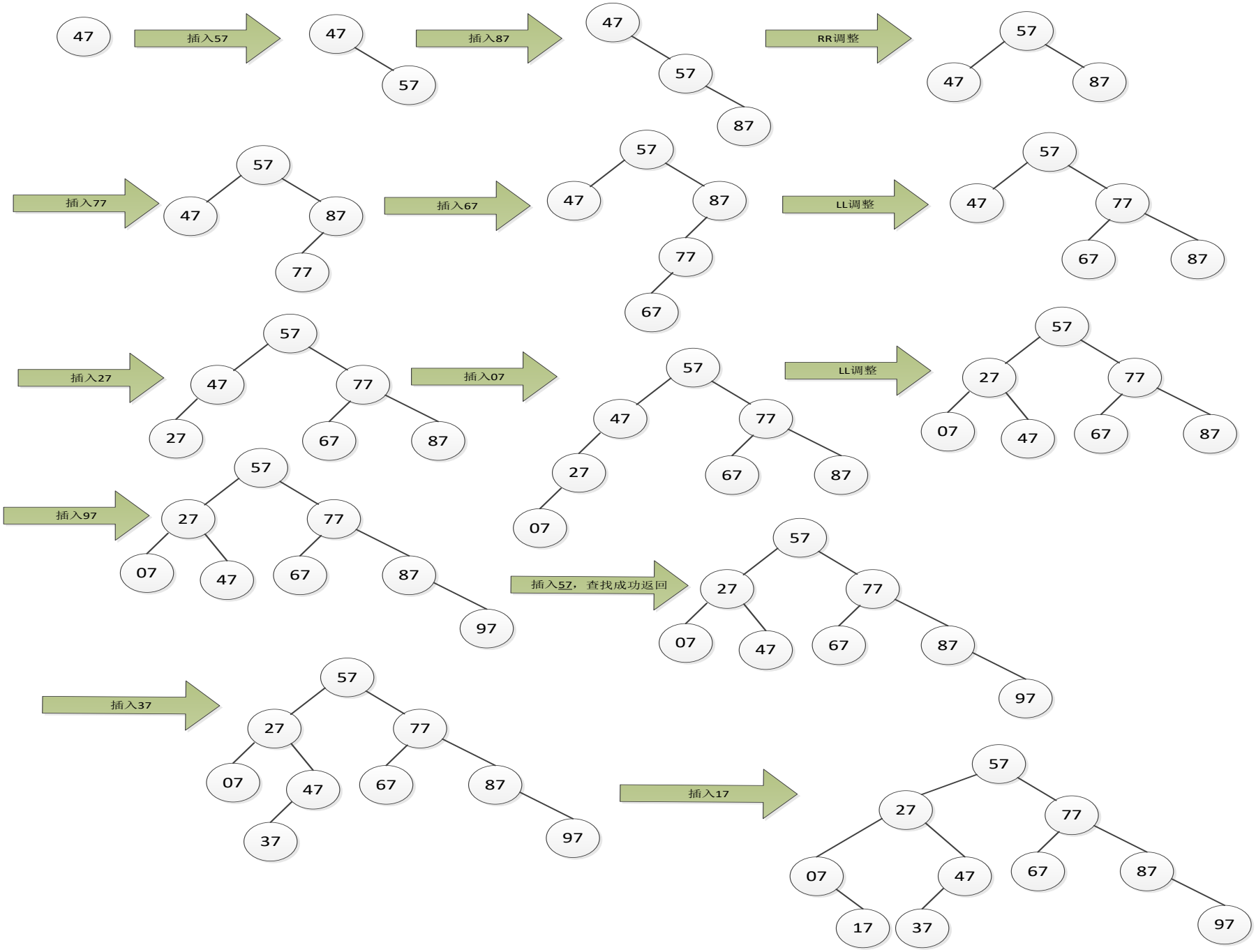
（3）查找成功时的平均查找长度

$ASL_{成功} = (3\times4+4\times3+2\times2+1\times1)/10=2.9$

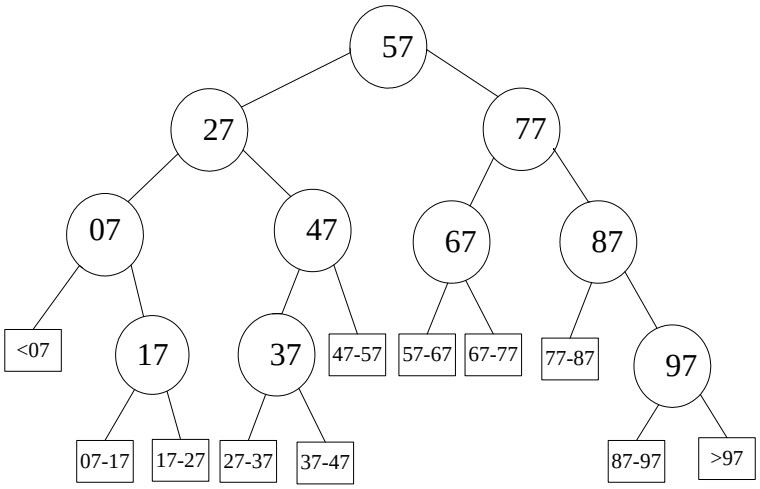
（4）查找不成功时的平均查找长度

$ASL_{不成功} = (5\times3+6\times4)/11=39/11$

平衡二叉树构造过程：关键字序列为｛47, 57, 87, 77, 67, 27, 07, 97, 57, 37, 17｝，构造一棵平衡二叉树。



7.4. 求折半查找判定树的平均查找长度



根据上图的折半查找判定树，解答如下问题：

（1）在等概率的情况下，计算查找成功时该判定树的平均查找长度 ASL。

（2）在等概率的情况下，计算查找不成功时该判定树的平均查找长度 ASL。

解：

折半查找判定树满足以下两点：

1. 折半查找判定树是一棵二叉排序树，即每个结点的值均大于其左子树上所有结点的值，小于其右子树上所有结点的值；

2. 折半查找判定树中的结点（圆形结点）都是查找成功的情况，将每个结点的空指针指向一个实际上并不存在的结点——称为外结点，所有外结点即是查找不成功的情况（方框形状的结点）。如果有序表的长度为 n，则外结点一定有 n+1 个。

（1）在等概率的情况下，查找成功时该判定树的 ASL。

在折半查找判定树中，某结点所在的层数即是查找该结点的比较次数，整个判定树代表的有序表的平均查找长度即为查找每个结点的比较次数之和除以有序表的长度。

ASL=(1×1+2×2+3×4+4×3)/10=29/10

（2）在等概率的情况下，查找不成功时该判定树的 ASL。

在折半查找判定树中，查找不成功时的比较次数即是查找相应外结点时与内结点的比较次数。整个判定树代表的有序表在查找失败时的平均查找长度即为查找每个外结点的比较次数之和除以外结点的个数。

ASL=(3×5+4×6)/11=39/11

7.5. 试证明：在由 n(n>0)个结点构成的有序表中进行折半查找，其最坏情况下与关键字比较的次数不超过由 n 个结点所构成的完全二叉树的深度。

证明：

根据完全二叉树的性质 4，可以知道：

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

考虑具有 n 个关键字的有序表 S[1..n]，对 S 采用折半查找方法查找表中的元素是，首先查找的是 S[(1+n)/2]。

当查找不成功时，继续在左或右边的一半中继续查找。如此重复，直到查找成功或查找不成功时，查找过程结束。

当查找成功时，查找过程结束在一个内部结点上；

当查找不成功时，查找过程结束在一个外部结点上。

根据上述查找过程，可以得到一棵具有 n 个结点的判定树。

设判定树的高度为 h，则外部结点全部出现在第 h 层和第 h+1 层上

考虑最坏的情况，查找不成功时需要比较的最多次数 h

由于**具有 n 个结点的二叉判定树的深度为$\lfloor \log_2 n \rfloor+1$，即 h=$\lfloor \log_2 n \rfloor+1$**。

所以，最坏情况下，与关键字比较的次数不超过由 n 个结点所构成的完全二叉树的深度。

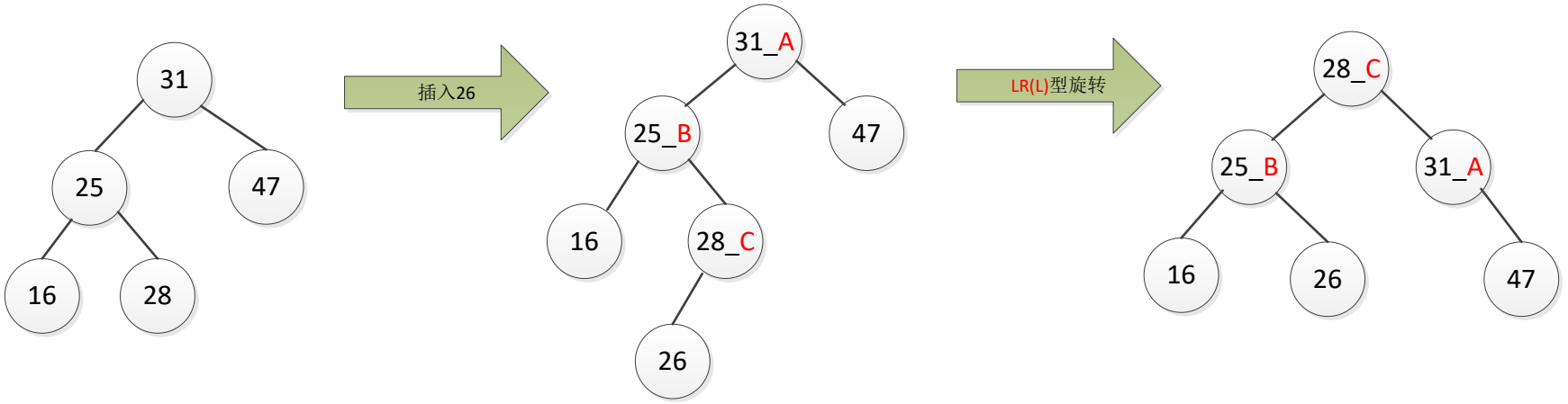
证毕。

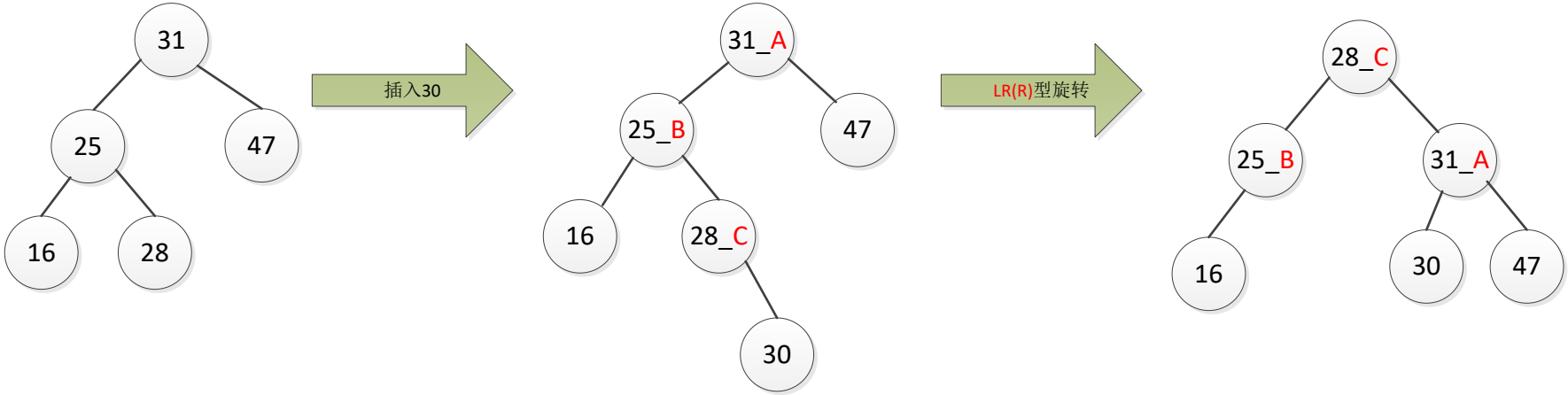
要点：1. **具有 n 个结点的完全二叉树的深度为$\lfloor \log_2 n \rfloor+1$**；

2. **具有 n 个结点的二叉判定树的深度为$\lfloor \log_2 n \rfloor+1$**。

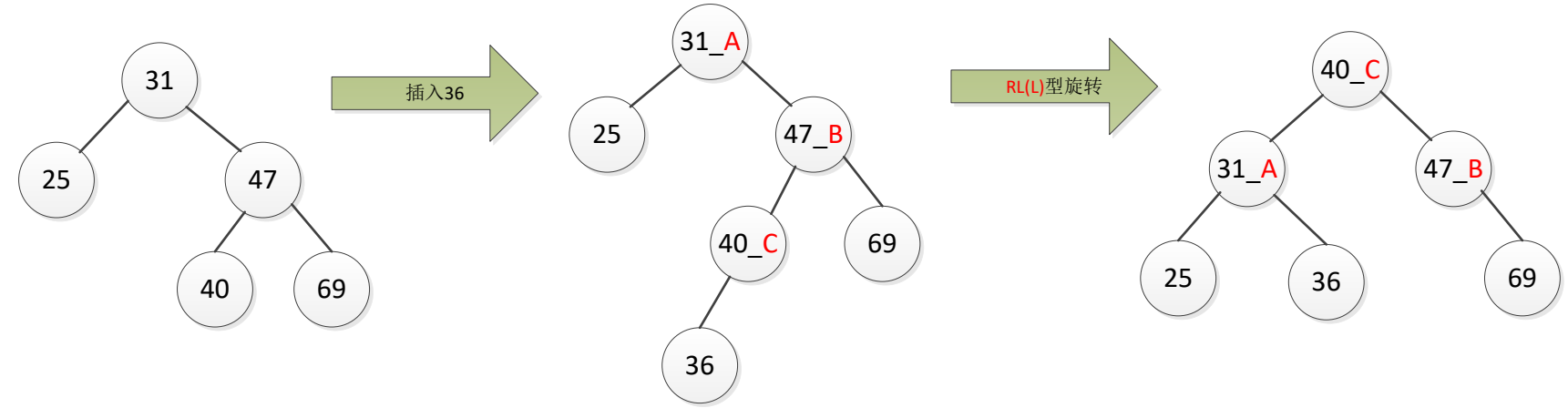
7.6 平衡二叉树（教材例题第 207 页~209 页）

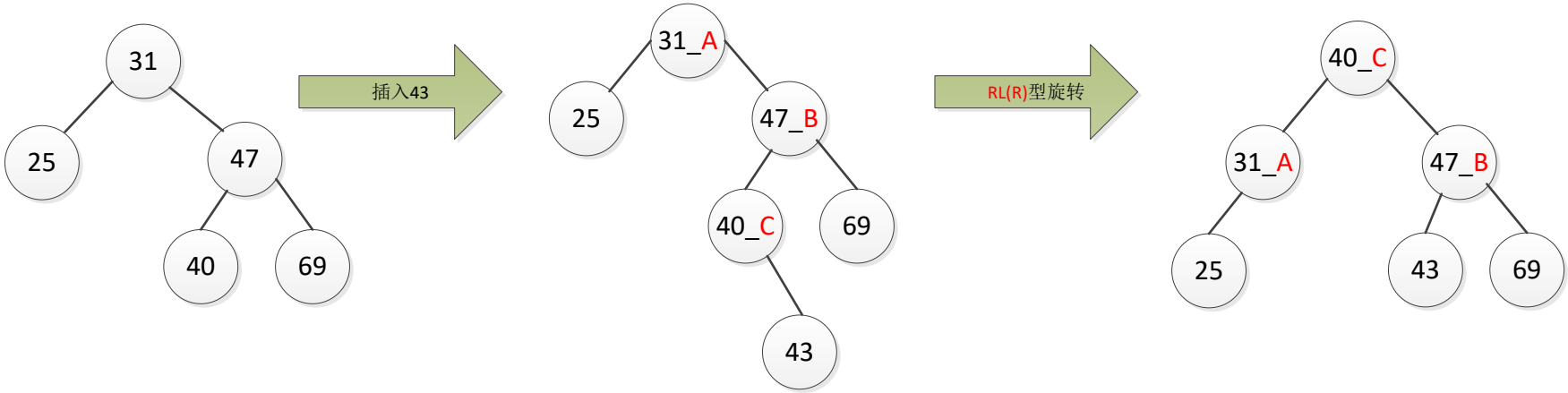
（1）LR 型旋转（LR(L)型、LR(R)型）（教材第 208 页图 7.18）





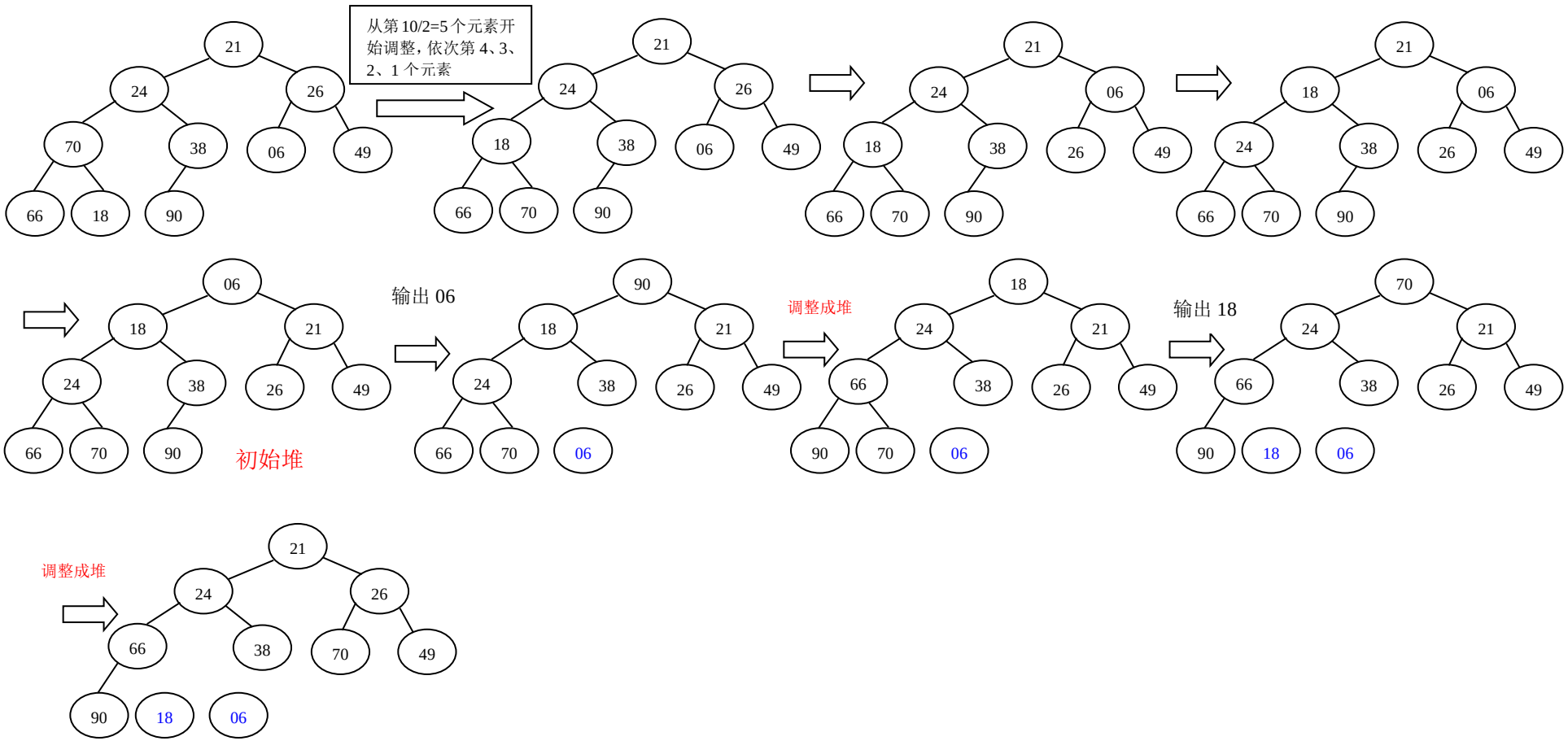
（2）RL 型旋转（RL(L)型、RL(R)型）（教材第 209 页图 7.20）





8.1. 将下列给定的关键字序列（21，24， 26， 70， 38， 06， 49， 66， 18， 90）调整成一个堆，使其满足 $K_i \leqslant K_{2i}$ 且 $K_i \leqslant K_{2i+1}$ ，并画出从初始堆到输出关键字 18 后的过程中形成的所有堆，并给出堆的完全二叉树的顺序存储表示形式。

解：



给出堆的完全二叉树的顺序存储表示形式											
初始堆	**06**	**18**	**21**	**24**	**38**	**26**	**49**	**66**	**70**	**90**	
输出第 **1** 个元素 **06** 后的堆	**18**	**24**	**21**	**66**	**38**	**26**	**49**	**90**	**70**	**06**	
输出第 **2** 个元素 **18** 后的堆	**21**	**24**	**26**	**66**	**38**	**70**	**49**	**90**	**18**	**06**	本题已完成
输出第 **3** 个元素 **21** 后的堆	**24**	**38**	**26**	**66**	**90**	**70**	**49**	**21**	**18**	**06**	继续排序
输出第 **4** 个元素 **24** 后的堆	**26**	**38**	**49**	**66**	**90**	**70**	**24**	**21**	**18**	**06**	
输出第 **5** 个元素 **26** 后的堆	**38**	**66**	**49**	**70**	**90**	**26**	**24**	**21**	**18**	**06**	
输出第 **6** 个元素 **38** 后的堆	**49**	**66**	**90**	**70**	**38**	**26**	**24**	**21**	**18**	**06**	
输出第 **7** 个元素 **49** 后的堆	**66**	**70**	**90**	**49**	**38**	**26**	**24**	**21**	**18**	**06**	
输出第 **8** 个元素 **66** 后的堆	**70**	**90**	**66**	**49**	**38**	**26**	**24**	**21**	**18**	**06**	
输出第 **n-1=9** 个元素 **70** 后	**90**	**70**	**66**	**49**	**38**	**26**	**24**	**21**	**18**	**06**	排序全部结束