数据结构

第9章　查找

主讲教师：祝建华

查找表：由同一类型的数据元素（记录）组成的集合。

记作：$ST=\{a_1, a_2, \ldots, a_n\}$

学生成绩表

序号	学　号	姓　名	性别	数学	外语
1	200041	刘大海	男	80	75
2	200042	王　伟	男	90	83
3	200046	吴晓英	女	82	88
4	200048	王　伟	女	80	90
n	……	…..	…	….	…
	数据项1（主关键字）	数据项2			数据项5

- 关键字：　可以标识一个记录的数据项
- 主关键字：　可以唯一地标识一个记录的数据项
- 次关键字：　可以识别若干记录的数据项

查找表的操作：

生成查找表

查找元素(记录)x在是否在表ST中

查找元素(记录)x的属性

插入新元素(记录)x

删除元素(记录)x

……

➢查找：根据给定的某个关键字值，在查找表中确定一个其关键字等于给定值的记录或数据元素。

设k为给定的一个关键字值，R[1..n]为n个记录的表，若存在R[i].key=k, 1≤i≤n, 称查找成功；否则称查找失败。

➢静态查找：查询某个特定的元素，检查某个特定的数据元素的属性，不插入新元素或删除元素(记录)。

➢ 动态查找：在查找过程中，同时插入查找表中不存在的数据元素(记录)。

- 查找表的类型及其查找方法
  - (1) 静态查找表
    - 顺序表，用顺序查找法
    - 线性链表，用顺序查找法
    - 有序的顺序表，用：折半查找法；
      **斐班那契查找法；插值查找法；
    - 索引顺序表/分块表，用分块查找法。
  - (2) 动态查找表
    - 二叉排序树，平衡二叉树（AVL树）
    - **B树，B+树，键树
  - (3) 哈希(Hash)表

➢平均查找长度： 查找一个记录时比较关键字次数的平均值。

$$ASL=\sum_{i=1}^{n} P_iC_i$$

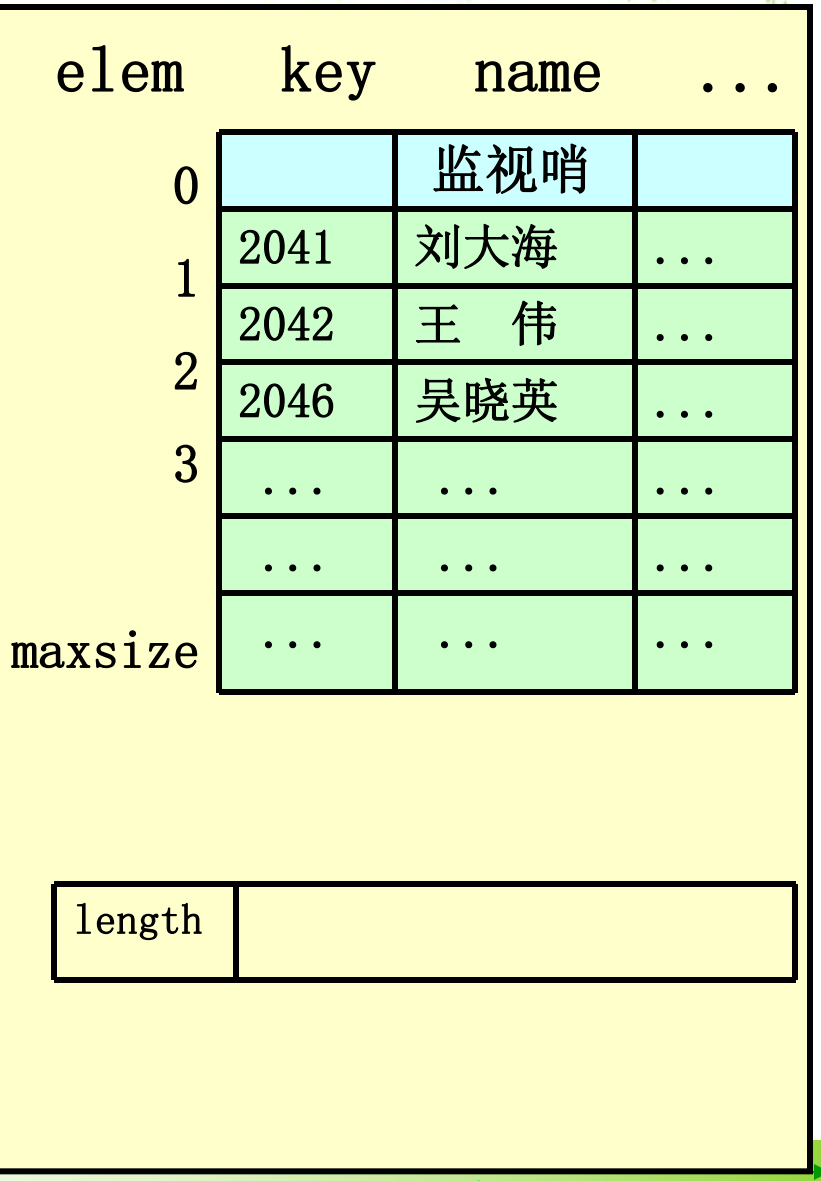
P_i --- 查找r[i]的概率

C_i --- 查找r[i]所需比较关键字的次数

9.1 静态查找表

9.1.1 顺序表与顺序查找法

1. 顺序表的描述

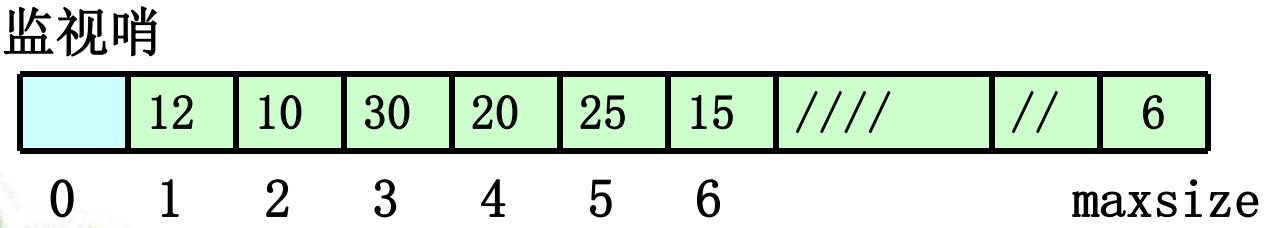


例1 元素类型为记录(结构)

```
#define maxsize 100                    //表长100
typedef int keytype;
typedef struct
 { keytype key ;                       //关键字类型
   char name[6];                       //姓名
   ......                              //其它
 } ElemType;
typedef struct
 { ElemType elem[maxsize+1]; //maxsize+1个记录,
                             //elem[0]为监视哨
   int length;
  } SSTable;
 SSTable ST1,ST2;
```

例2 元素类型为整型

```
#define maxsize 100 //表长100
typedef int ElemType;
typedef struct
 {
  ElemType elem[maxsize+1]; //maxsize+1个记录,
                            //elem[0]为监视哨
  int length;
  } SSTable;
SSTable ST1,ST2;
```



2. 顺序查找法(sequential search)算法设计

算法1：假定不使用监视哨elem[0]

基本思想：

将关键字k依次与记录的关键字：

elem [n].key, elem[n-1].key，..., elem[1].key 比较。

如果找到一个记录elem[i],有：

elem[i].key＝k (1≤i≤n)，

则查找成功，停止比较,返回记录的下标i；

否则,查找失败,返回0。

输入：查找表ST，查找条件（关键字）k
输出：成功时：记录序号，失败时：0

```
int sequsearch(SSTable ST,keytype k)
{
  int i=ST.length;                    //从第n个记录开始查找
  while (i>=1 && k!=ST.elem[i].key)
       i--;                           //继续扫描
  if (i)
     printf(" success\n" );           //查找成功
  else   printf(" fail\n" );          //查找失败
  return i;                           //返回记录的下标i
 }
```

算法2：假定使用监视哨elem[0]

基本思想：

先将关键字k存入elem[0].key,再将k依次与elem[n].key, ...,elem[1].key,elem[0].key进行比较。

必然存在一个i (0≤i≤n)满足：
k=elem [i].key。

如果 i>0,
　　则查找成功；
否则,
　　查找失败。

输入：查找表ST，查找条件（关键字）k
输出：成功时：记录序号，失败时：0

```
int sequsearch(SSTable  ST,keytype k)
{
  int i=ST.length;                    //从第n个记录开始查找
  ST.elem[0].key=k;                   //k填入ST.elem[0].key
  while( k!=ST.elem[i].key )
    i-- ;                             //继续扫描
  return i;                           //返回记录的下标i
 }
```

3 查找算法性能分析：

对n个记录的表，所需比较关键字的次数

- 若查找成功：最少为1，最多为n

  假定每个记录的查找概率相等，

  即 P1 = P2 = ... = Pn =1/n

$$ASL = \sum_{i=1}^{n} P_i C_i = \frac{1}{n} \sum_{i=1}^{n} C_i$$

$$= \frac{1}{n}(n + (n-1) + \ldots + 1)$$

$$=(n+1)/2$$

- 若查找失败：使用监视哨elem[0]，为 n+1
  不使用监视哨elem[0]，为 n

假定查找成功和失败的机会相同，对每个记录的查找概率相等，即：

$$P_i = \frac{1}{n}$$

在使用监视哨的情况下：

$$ASL = \frac{1}{2}ASL_{成功} + \frac{1}{2}ASL_{失败} = \frac{n+1}{4} + \frac{n+1}{2} = \frac{3n+3}{4}$$

9.1.2 有序的顺序表的查找与折半查找法

1. 有序表

elem[1].key≤elem[2].key≤...≤elem[n].key

2. 折半查找(binary search,对半查找，二分查找)

假定 k=10

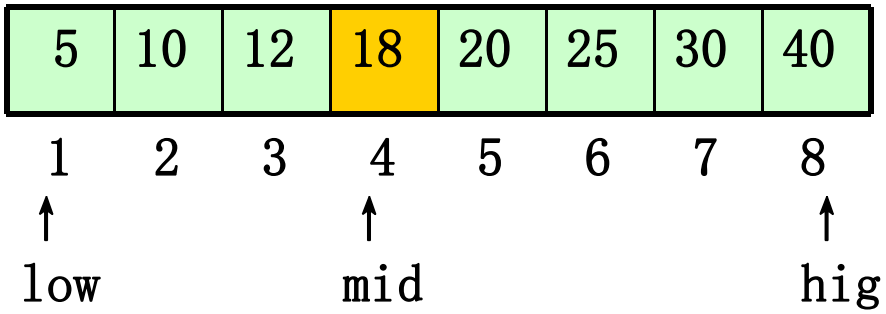
5	10	12	18	20	25	30	40
1	2	3	4	5	6	7	8
↑ low			↑ mid				↑ hig

low=1,hig=8
mid=(low+hig)/2=4

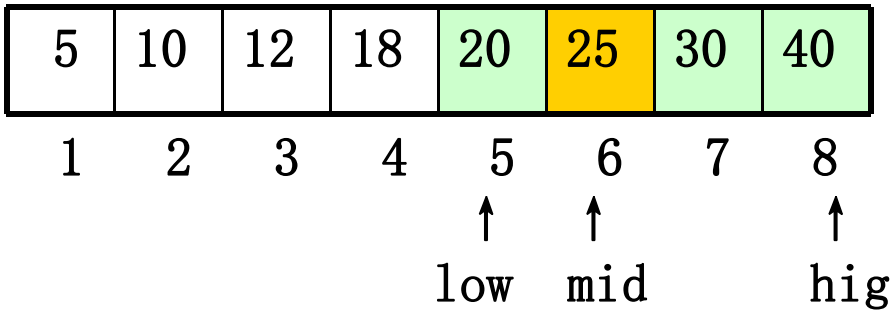
5	10	12	18	20	25	30	40
1	2	3	4	5	6	7	8
↑ low	↑ mid	↑ hig					

low=1,hig=3
mid=(low+hig)/2=2

假定k=40

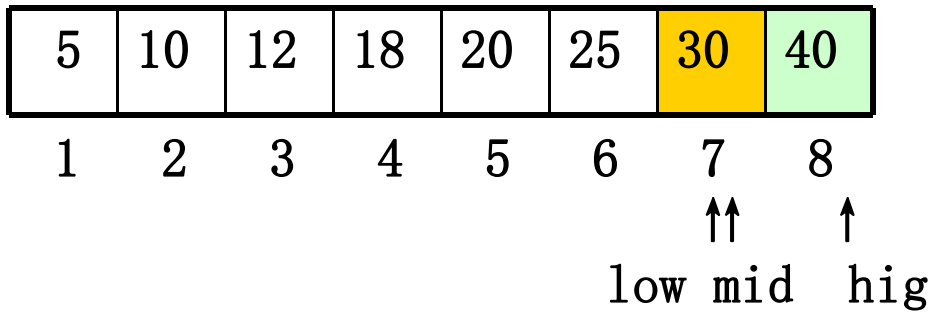


low=1,hig=8
mid=(low+hig)/2=4

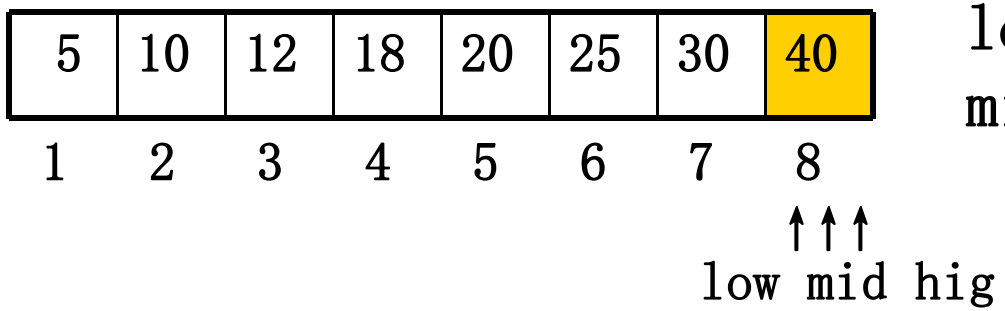


low=5,hig=8
mid=(low+hig)/2=6

假定 k=40（续）

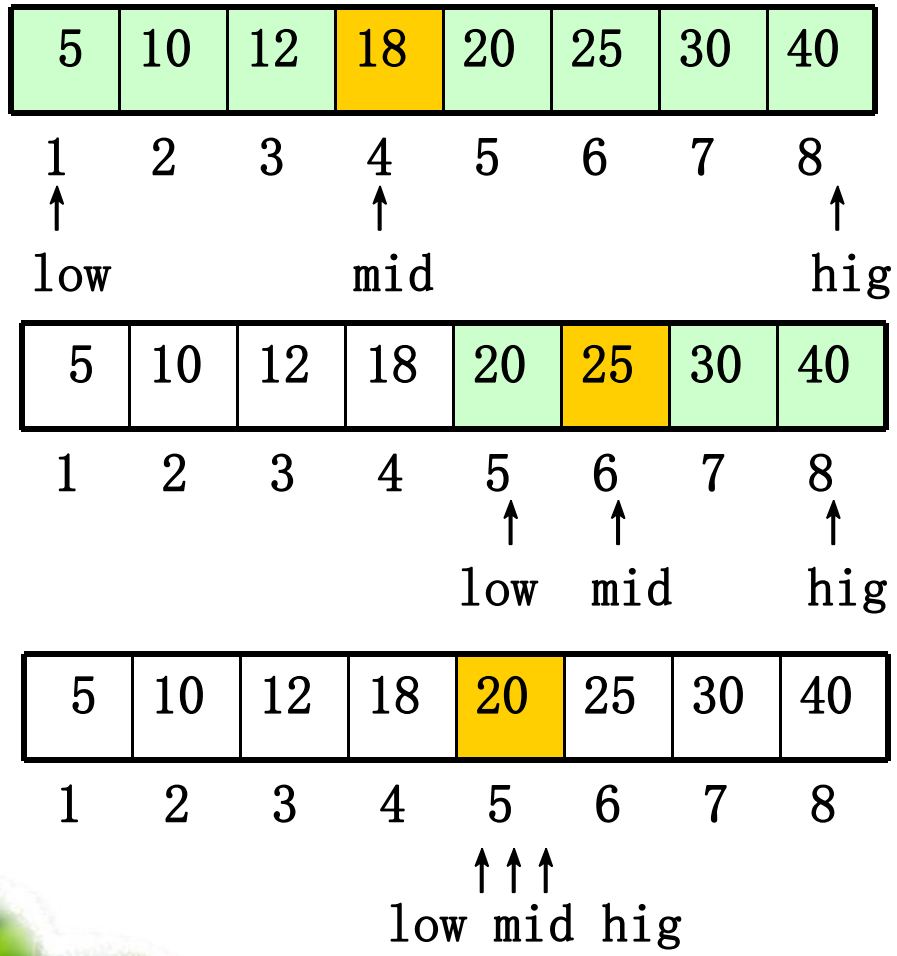


low=7,hig=8
mid=(low+hig)/2=7



low=8,hig=8
mid=(low+hig)/2=8

假定 k=22

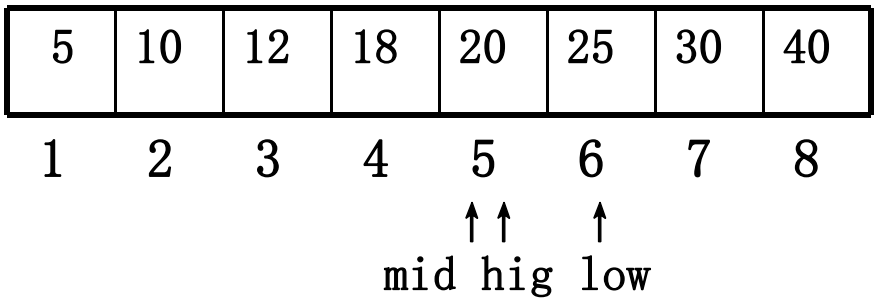


low=1,hig=8
mid=(low+hig)/2=4

low=5,hig=8
mid=(low+hig)/2=6

low=5,hig=5
mid=(low+hig)/2=5

假定 k=22（续）



mid=5
low=6,hig=5

hig<low,查找失败

3 折半查找算法1

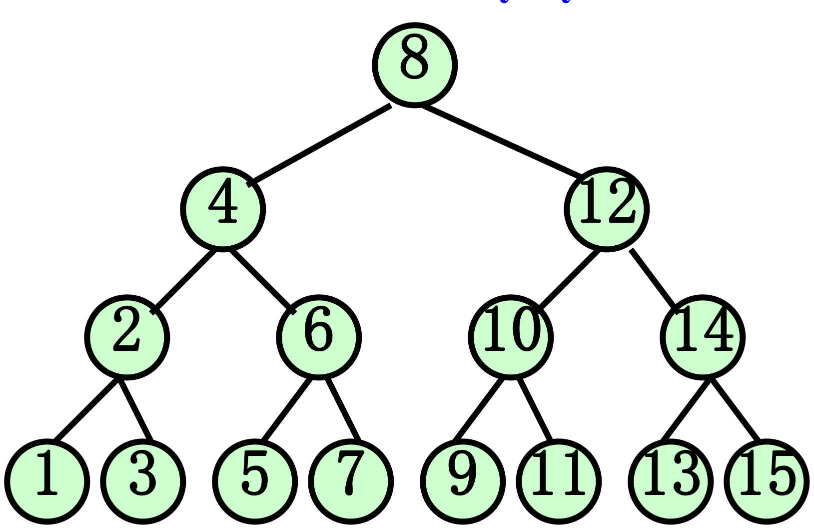
```
int binsrch(SSTable  ST,keytype k)
{ int low,mid,hig;
  low=1;
  hig=ST.length;
  while (low<=hig)
     {
      mid=(low+hig)/2;        //计算中间记录的地址
      if (k<ST.elem[mid].key)
           hig=mid-1;          //查左子表
      else if (k==ST.elem[mid].key)
           break;              //查找成功,退出循环
           else low=mid+1;     //查右子表
     }
```

```
if (ST.elem[mid].key==k)          //查找成功
      {
        printf("success\n");
        return mid;
       }
 else                             //查找失败
      {
        printf("fail\n");
        return 0;
       }
}
```

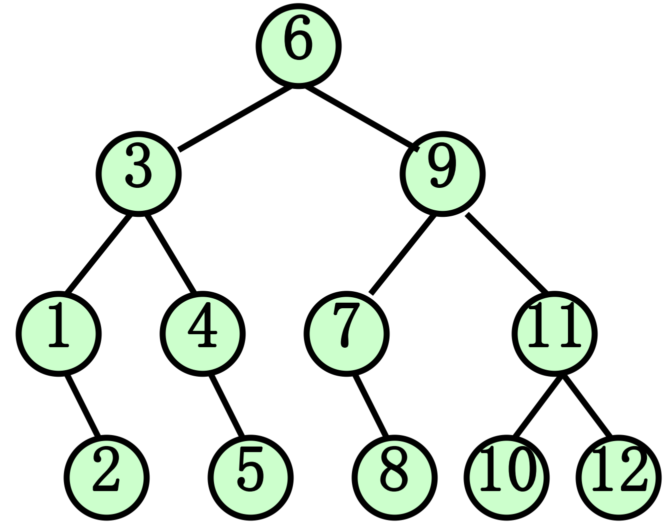
折半查找算法2

```
int binsrch(SSTable  ST,keytype k)
{ int low,mid,hig;
  low=1;  hig=ST.length;
  while (low<=high)
   { mid=(low+high)/2;
     if (k<ST.elem [mid].key)
          hig=mid-1;                    //查左子表
     else if (k==ST.elem [mid].key)
               return mid;              //查找成功，返回mid
          else low=mid+1;               //查右子表
   }
  return 0 ;                            //查找失败，返回0
}
```

4. 判定树（描述折半查找过程的二叉树）



n=15, 满二叉树



n=12, 非满二叉树

- 结点内的数据表示数据元素的序号
- 根结点表示首先要和关键字k进行比较的数据元素的序号（如8），比较相等时，查找成功，否则，当k小于根结点对应元素的关键字时，下步就和左子结点（如序号4）对应元素的关键字比较，否则，下步就和右子结点（如序号12）对应元素的关键字比较。

- 若 $n = 2^k - 1$，则判定树为满二叉树，其深度为 $k=\log_2(n+1)$

  假定$P_i=1/n(i=1,2,\ldots,n)$， 比较关键字的次数：

  (1)最少 $C_{min}=1$

  (2)最多 $C_{max}=\log_2(n+1)$

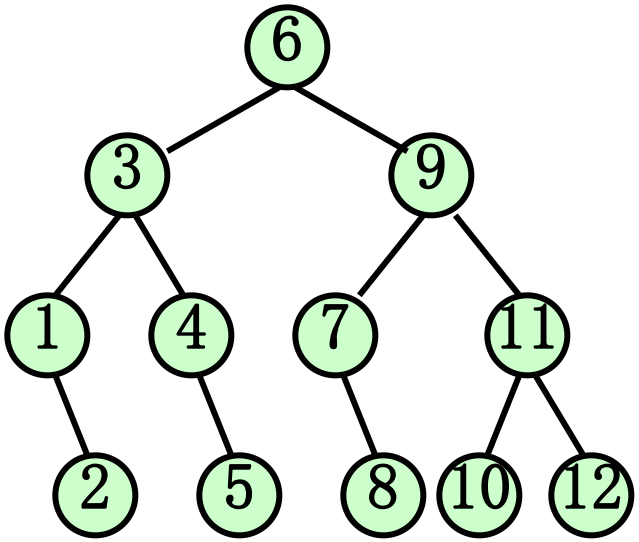
  (3)$ASL=\frac{n+1}{n}\log_2(n+1)-1$ （按满二叉树）

  设n=15 $ASL=\frac{15+1}{15}\log_2(15+1)-1=\frac{16}{15}*4-1\approx 3.3$

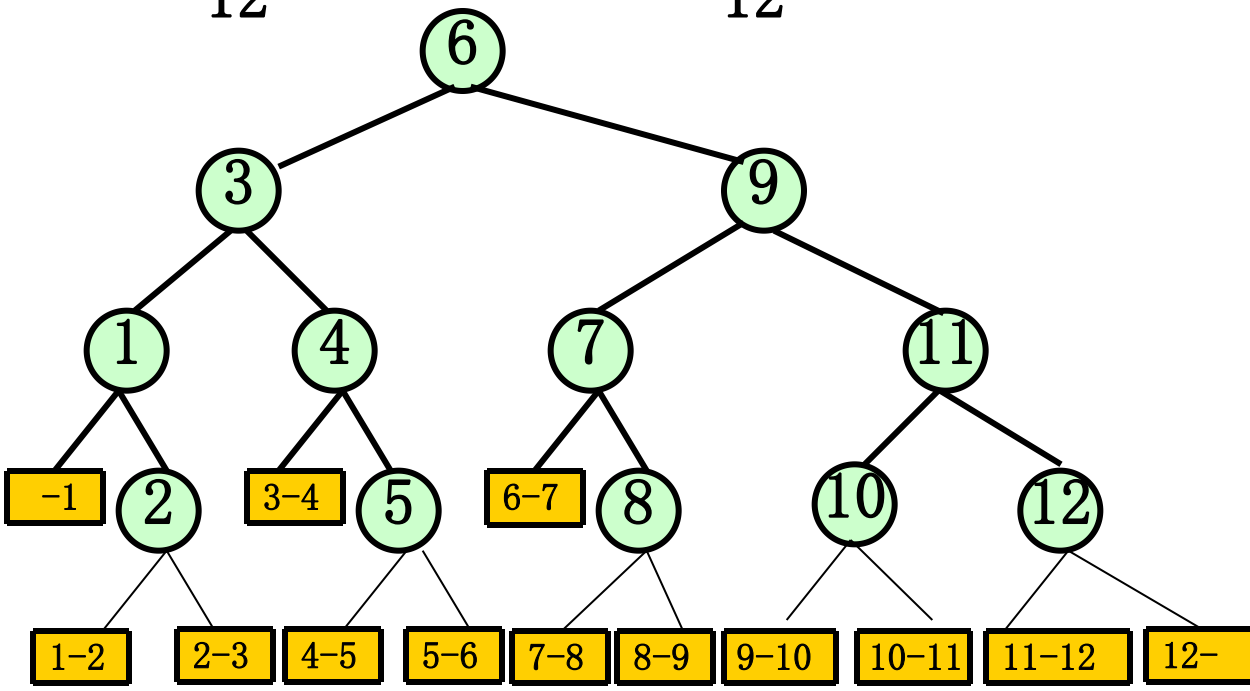
对任意的n

$$ASL \approx \frac{n+1}{n}\log_2(n+1)-1 = O(\log_2 n)$$

设n=12，$ASL = \frac{1+2+2+3+3+3+3+4+4+4+4+4}{12} = \frac{37}{12} \approx 3.1$

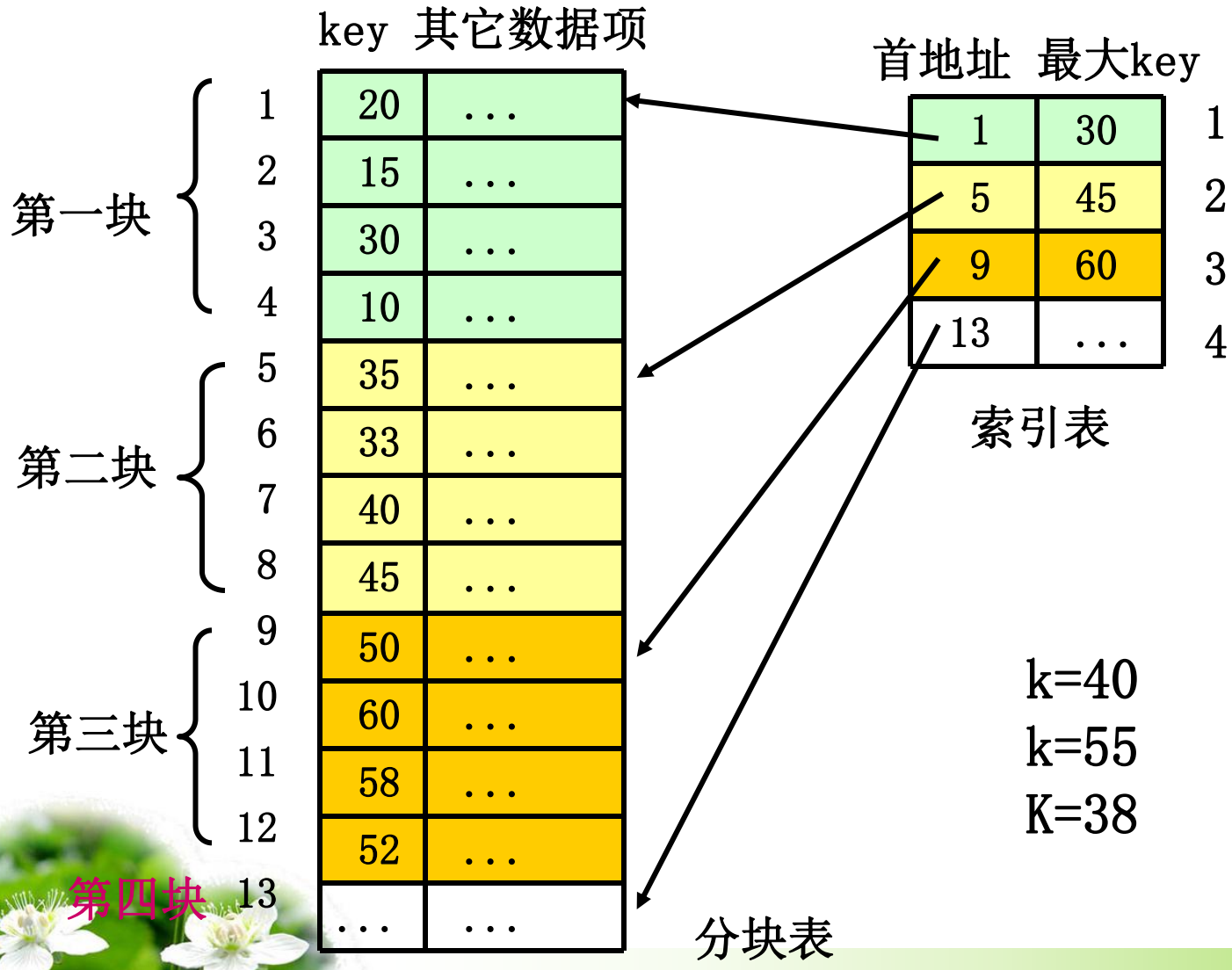


n=12，判定树



n=11，加外部结点的判定树

9.1.3 索引顺序表(分块表)与分块查找法



- 条件
  (1)分块表"按块有序"，索引表"按key有序"
  (2)设n个记录分为b个块，每块的记录数s=n/b
- 查找方法与ASL
  (1)顺序查找(或折半查找)索引表
  确定k值所在的块号或块的首地址

$$ASL(1)=Lb=\frac{b+1}{2}$$

  (2)在某一块中顺序查找

$$ASL(2)=Lw=\frac{s+1}{2}$$

- $ASL=Lb+Lw=\frac{b+1}{2}+\frac{s+1}{2}=\frac{1}{2}(b+s)+1=\frac{1}{2}(\frac{n}{s}+s)+1$
- 最佳分块

$s=\sqrt{n}$　　$b=\sqrt{n}$　　$ASL_{min}=\sqrt{n}+1=O(\sqrt{n})$

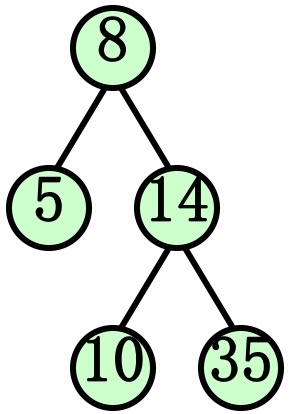
9.2 动态查找表

1. 二叉排序树(二叉查找树)

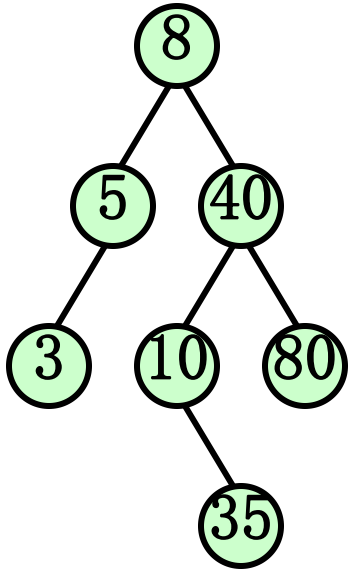
二叉排序树的定义

如果二叉树的任一结点大于其非空左子树的所有结点，而小于其非空右子树的所有结点，则这棵二叉树称为二叉排序树。

对一棵二叉排序树进行中序遍历，所得的结点序列一定是递增有序的。

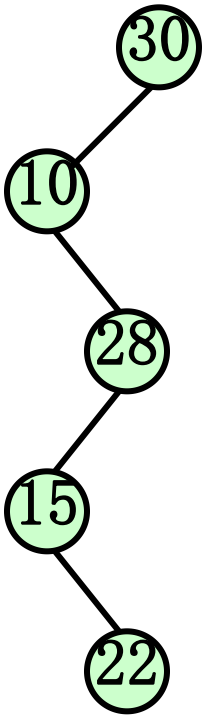


LDR：5, 8, 10, 14, 35

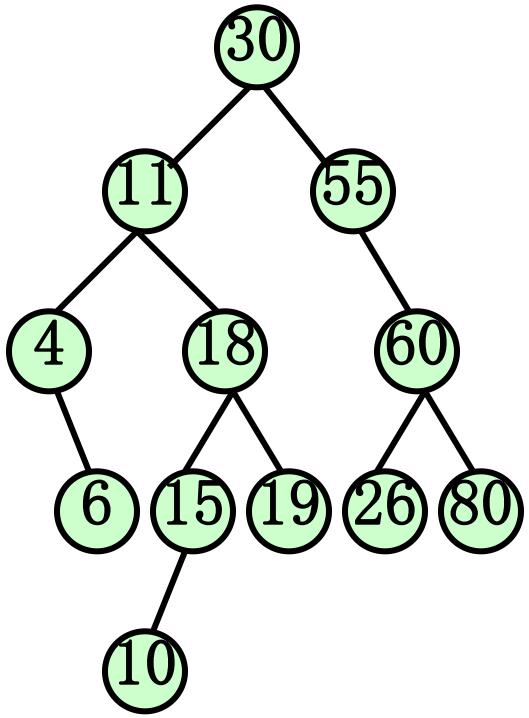


LDR：3, 5, 8, 10, 35, 40, 80

下列二叉树是否为二叉排序树？



T1

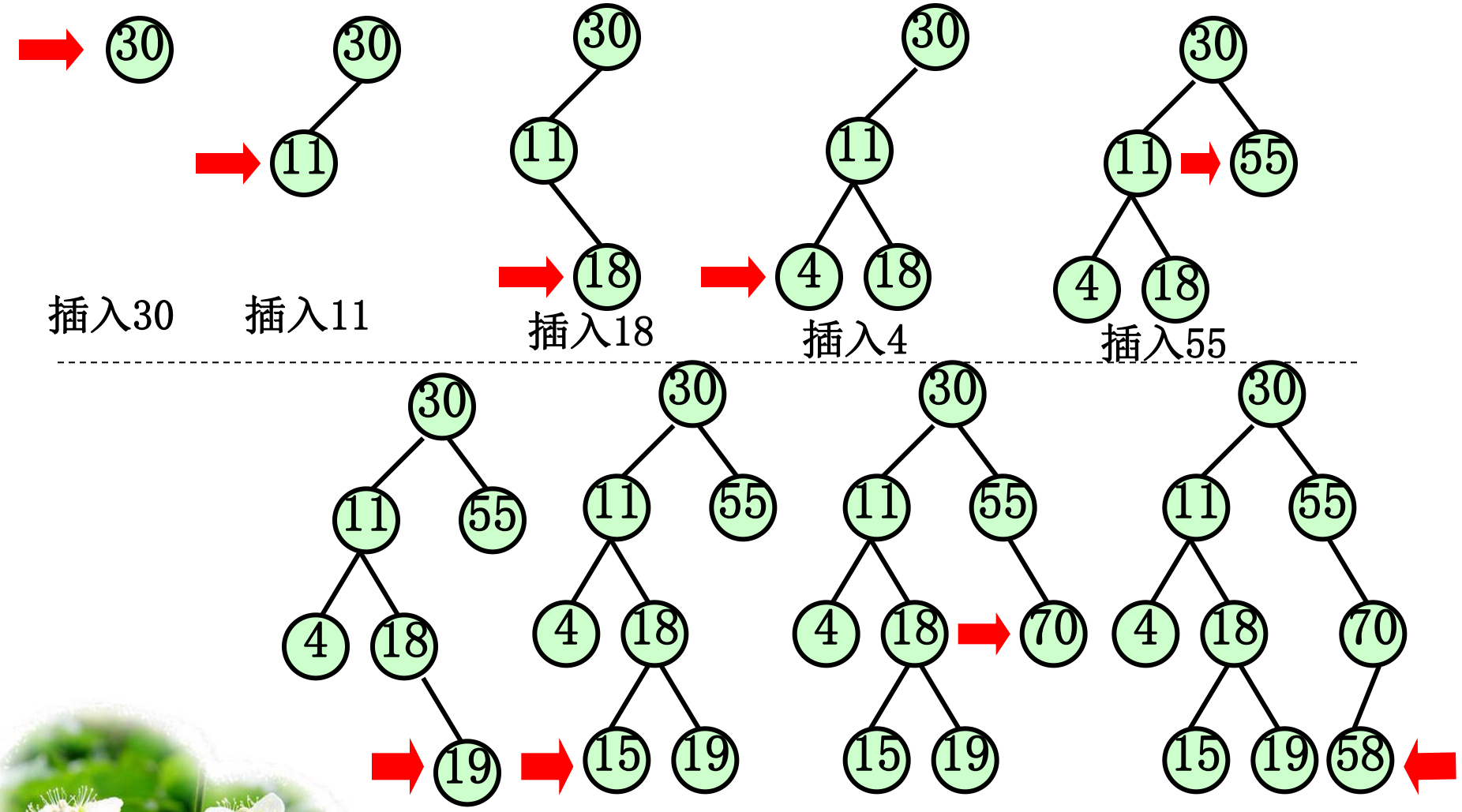


T2

T3

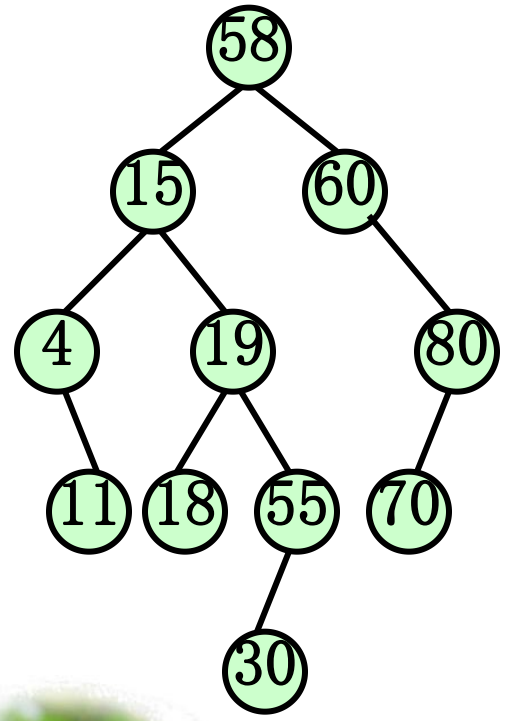
2 二叉排序树的生成

设输入序列为：30, 11, 18, 4, 55, 19, 15, 70, 58



课堂练习：

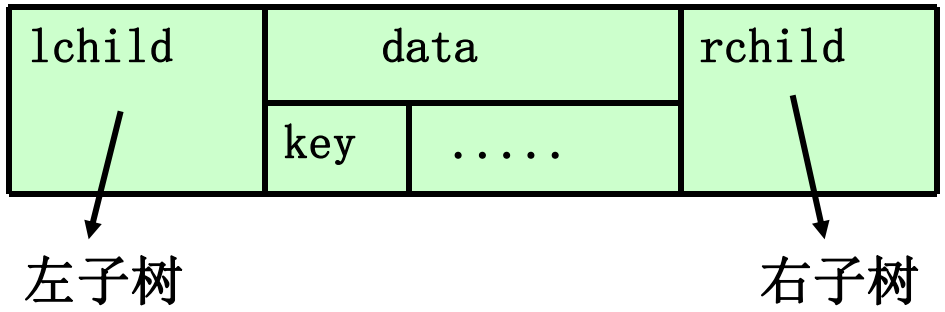
设输入关键字序列为:58,60,15,80,19,55,4,18,70,11,30,生成二叉排序树,试画出二叉排序树；假定查找每个结点(关键字)的概率相同，计算查找成功时的平均查找长度ASL。



$$ASL=\frac{1+2+2+3+3+3+4+4+4+4+5}{11}=\frac{35}{11}\approx 3.18$$

3 二叉排序树的存储结构

结点形式：



结点类型定义：

```
struct  node
{ struct
  { int key ;                          //关键字
    .....                              //其它数据项
   } data ;
   struct node *lchild,*rchild ;  //左右子树的指针
} *root,*t;
```

4 二叉排序树的查找算法

（返回值　失败：NULL　成功：非NULL，结点指针）

1）递归算法

```
struct node *search_tr(struct node *t keytype k)
{ if (t==NULL) return NULL;          //查找失败
  else
    if (k==t->data.key)
          return t;  //查找成功
    else
      if (k<t->data.key)
        return search_tr(t->lchild,k); //查左子树
      else
        return search_tr(t->rchild,k); //查右子树
}
```

2）非递归算法

```
struct node *search_tree(struct node *t,keytype k)
  {
    while (t!=NULL)
      if (k==t->data.key)
          return t;              //查找成功
      else
       if (k<t->data.key)
            t=t->lchild;         //查左子树
       else
            t=t->rchild;         //查右子树
    return t;                    //查找失败
   }
```

5 插入1个元素到二叉排序树的算法

1）递归算法

```
int intree(struct node &*t,ElemType x)
{  //t是指向二叉树根指针的指针
  if (t==NULL){
   t=(struct node *)malloc(sizeof(struct node));
   t->data=x; t->lchild=t->rchild=NULL; //生成叶子结点x
   return TRUE;  }
  if(x.key<t->data.key)
     return intree(t->lchild,x);        //插入左子树
  else if(x.key>t->data.key)
          return intree(t->rchild,x); //插入右子树
       else return FALSE;
}
```

2）非递归算法

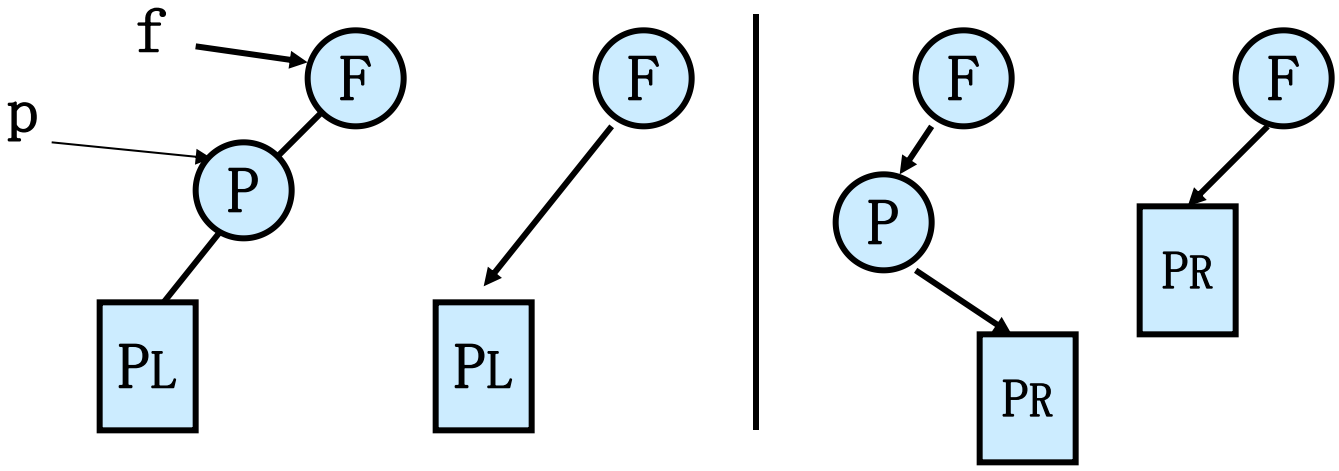
```
int intree(struct node &*T,ElemType x){
p=T; q=NULL;
while (p!=NULL)
    if(x.key<p->data.key)
          q=p,p=p->lchild;              //考虑插入左子树
    else if(x.key>p->data.key)
            q=p, p=p->rchild;           //考虑插入右子树
          else return FALSE;            //有该结点，不插入
s=(struct node *)malloc(sizeof(struct node));
s->data=x; s->lchild=t->rchild=NULL; //生成叶子结点x
if (q==NULL) T=s;
else  if(x.key<q->data.key)  q->lchild=s;
      else q->rchild=s;
       return TRUE; }
```

6 如何删除二叉排序树中的一个结点

设在二叉排序树上被删除结点为 *p ，其双亲结点为*f，且不失一般性，令 *p 是 *f 的左孩子。讨论三种情况：

(1) 若*p结点为叶子结点，则只需修改双亲结点的指针即可。

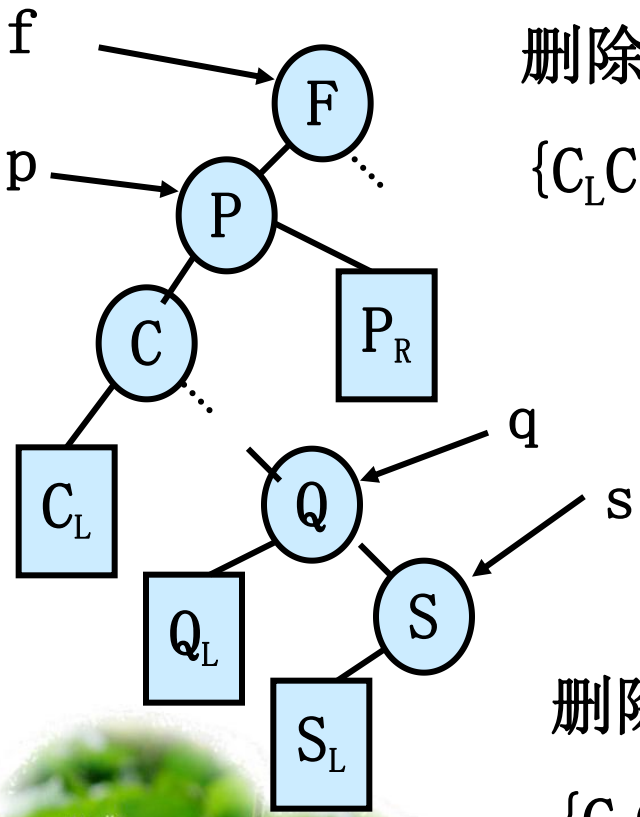
(2) 若*p结点只有左子树PL或只有右子树PR，则令PL或PR为双亲结点为*f的左子树即可。



(3) 若*p结点的左子树PL和右子树PR均不空，方法一：

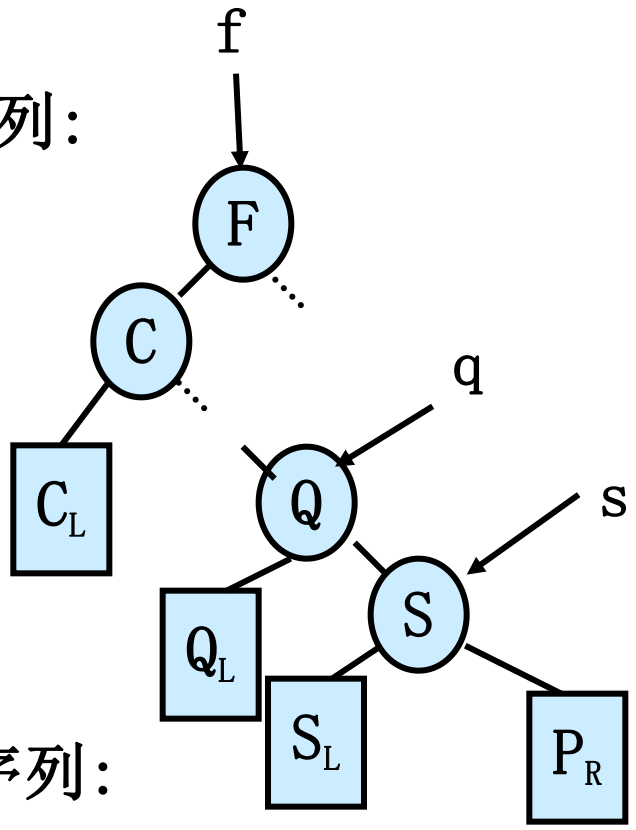
a) *p左子树作为*f的左子树

b) *p右子树作为S的右子树。



删除结点P之前的中序序列：

$\{C_LC \cdots Q_LQS_LSPP_RF \cdots\}$



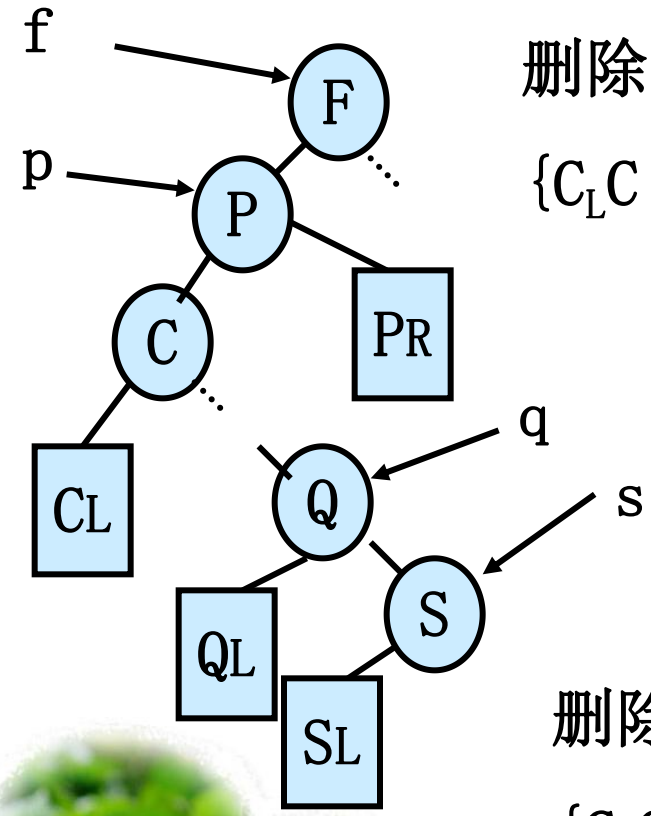
删除结点P之后的中序序列：

$\{C_LC \cdots Q_LQS_LSP_RF \cdots\}$

3) 若*p结点的左子树PL和右子树PR均不空，方法二：

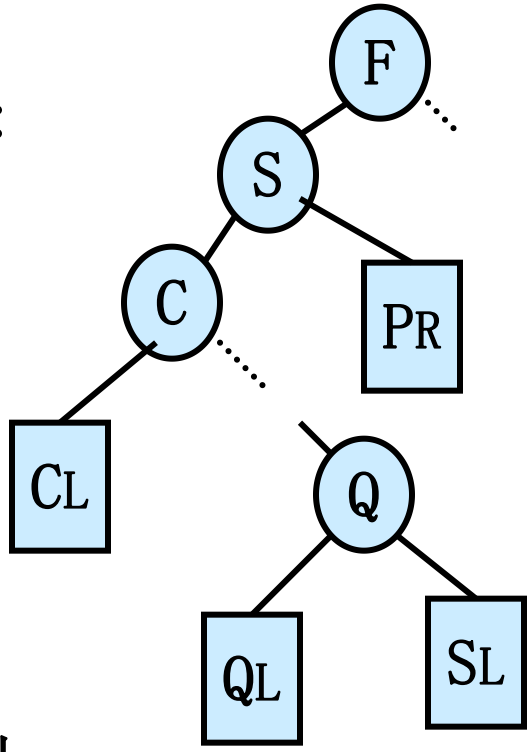
a) 求*p结点的前驱*s，前驱*s结点的双亲*q

b) 用*s替代*p，SL为*q的右子树。



删除结点P之前的中序序列：

$\{C_L C \cdots Q_L Q S_L S P P_R F \cdots\}$



删除结点P之后的中序序列：

$\{C_L C \cdots Q_L Q S_L S P_R F \cdots\}$

删除二叉排序树结点p的算法(方法二)

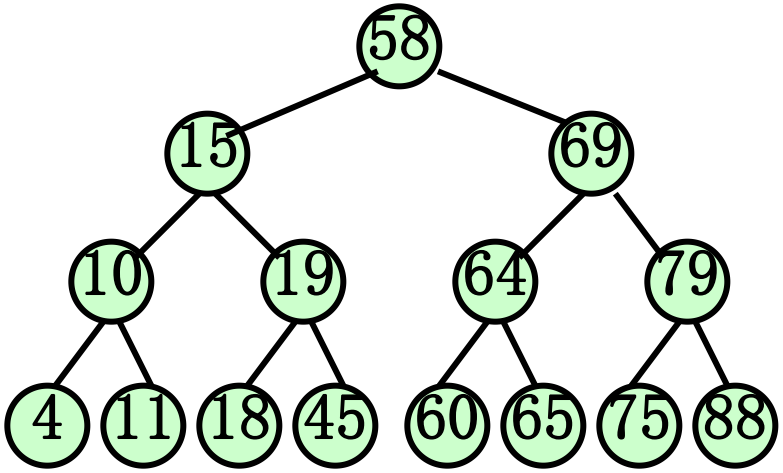
```
int Delete (BiTree &p) {
 if (!p->rchild)          //P结点为叶子或右子树PR为空
   { q = p; p = p->lchild;  free(q);}
 else if (!p->lchild)  //P结点的左子树PL为空
      {q = p; p = p->rchild;  free(q);}
      else                    //结点P的左,右子树均不空
       { q = p; s = p->lchild;
         while (s->rchild)
           {//求结点p中序遍历的前驱s,结点s的双亲q
            q = s;  s = s->rchild;}
         p->data = s->data;     //删除p转变成删除s
         if ( q != p)
            q->rchild = s->lchild;
          else                  //p的左孩子只有左子树
            q->lchild = s->lchild;
```

最好情况(为满二叉树)

$$ASL=\frac{n+1}{n}\log_2(n+1)-1 = O(\log n)$$

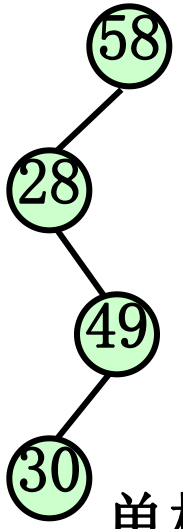
最坏情况(为单枝树)： ASL=(1+2+...+n)/n=(n+1)/2

平均值： ASL≈O(log n)



满二叉树

ASL=(15+1)/15*log2(15+1)-1≈3.3

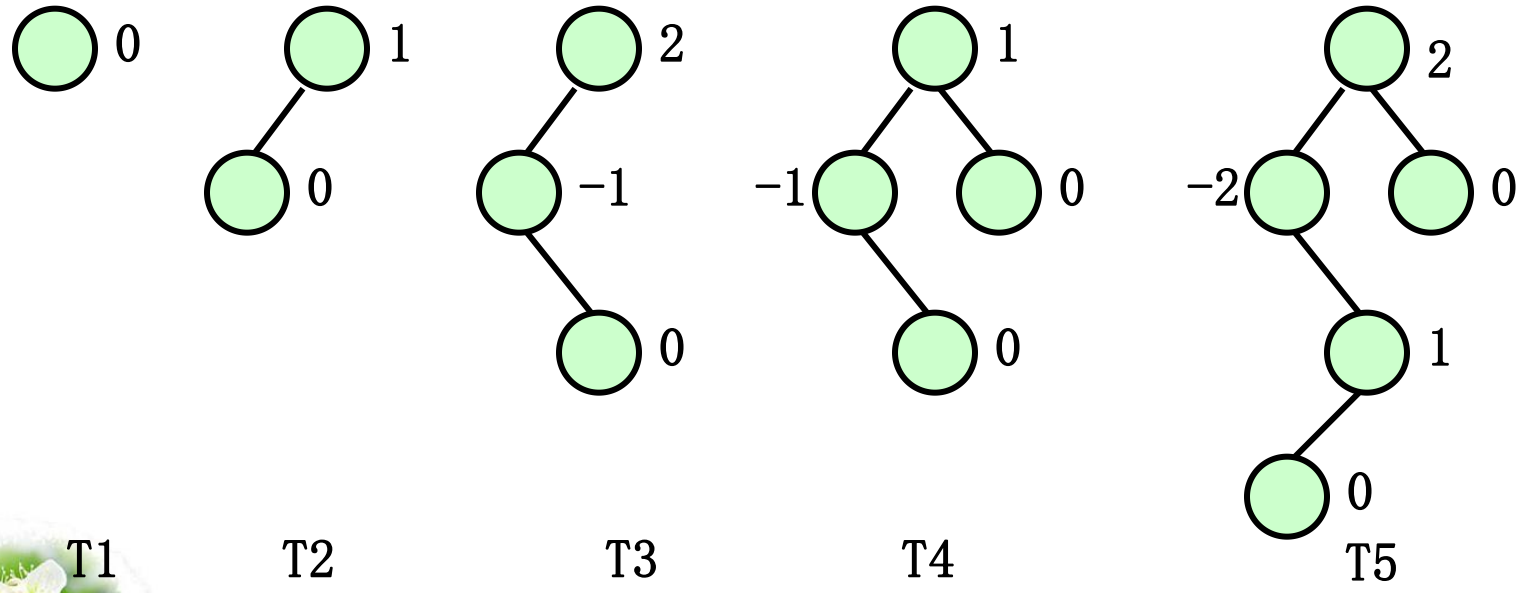


单枝树

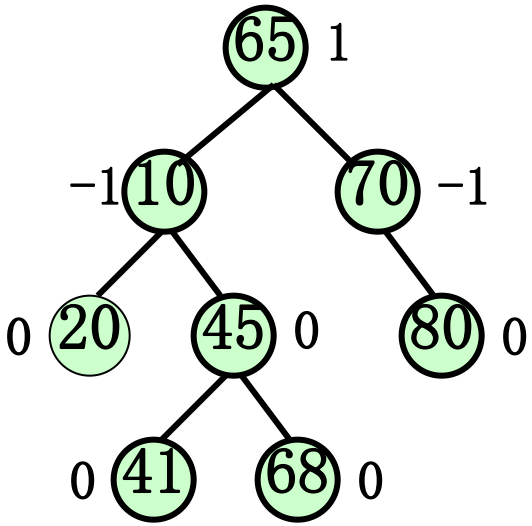
ASL=(1+2+3+4)/4=2.5

8.平衡二叉树(高度平衡二叉树)

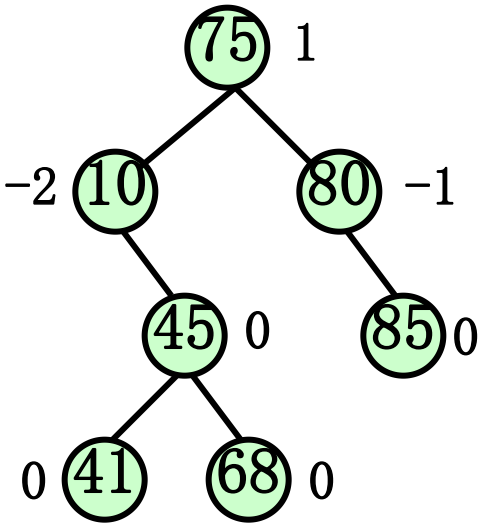
- AVL树：由G.M.Adelson-Velskii和E.M.Landis提出。
- 结点的平衡因子：结点的左右子树的深度之差。
- 平衡二叉树：任意结点的平衡因子的绝对值小于等于1的二叉树。



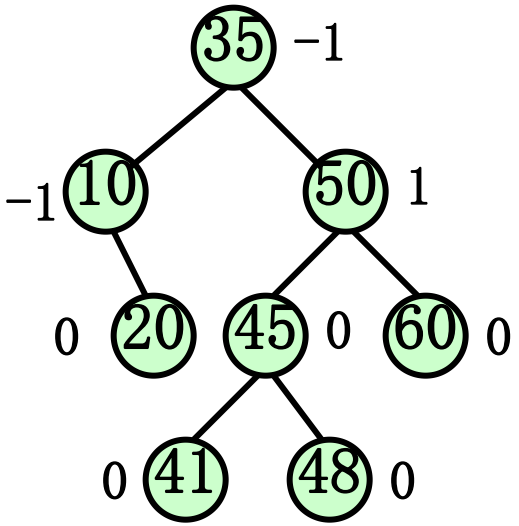
平衡二叉树、二叉排序树、平衡二叉排序树的区别：



平衡二叉树



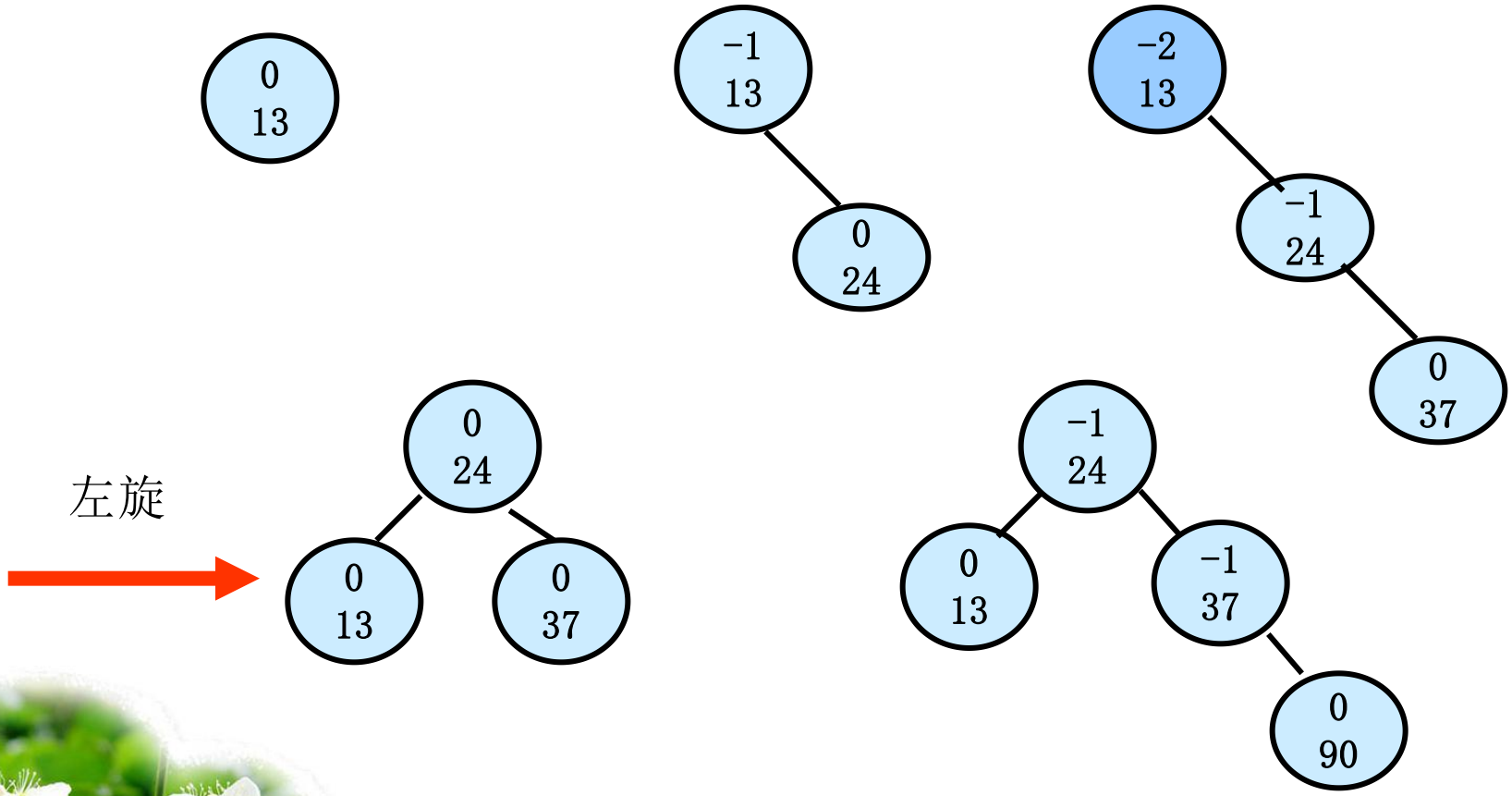
二叉排序树



平衡二叉排序树

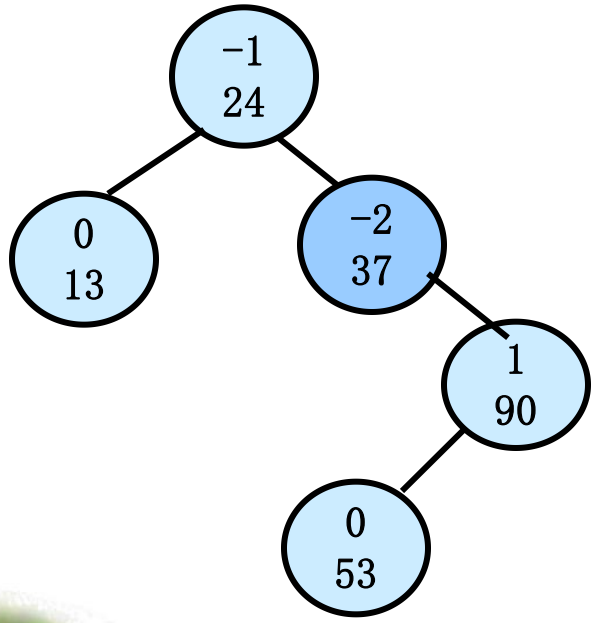
构造平衡二叉排序树

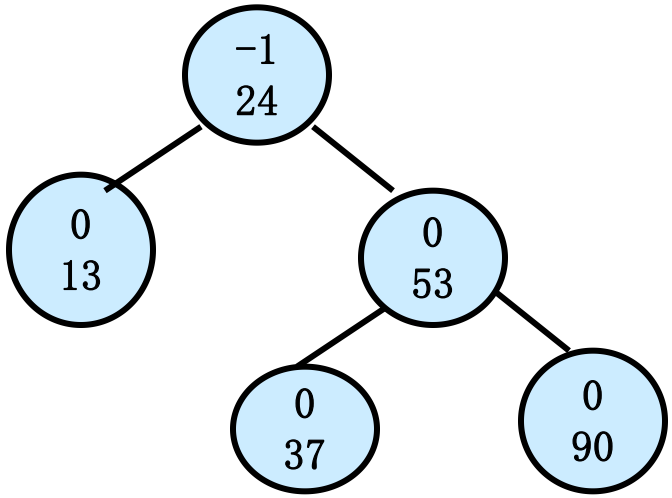
设关键字序列为{13,24,37,90,53},构造平衡二叉排序树



构造平衡二叉排序树

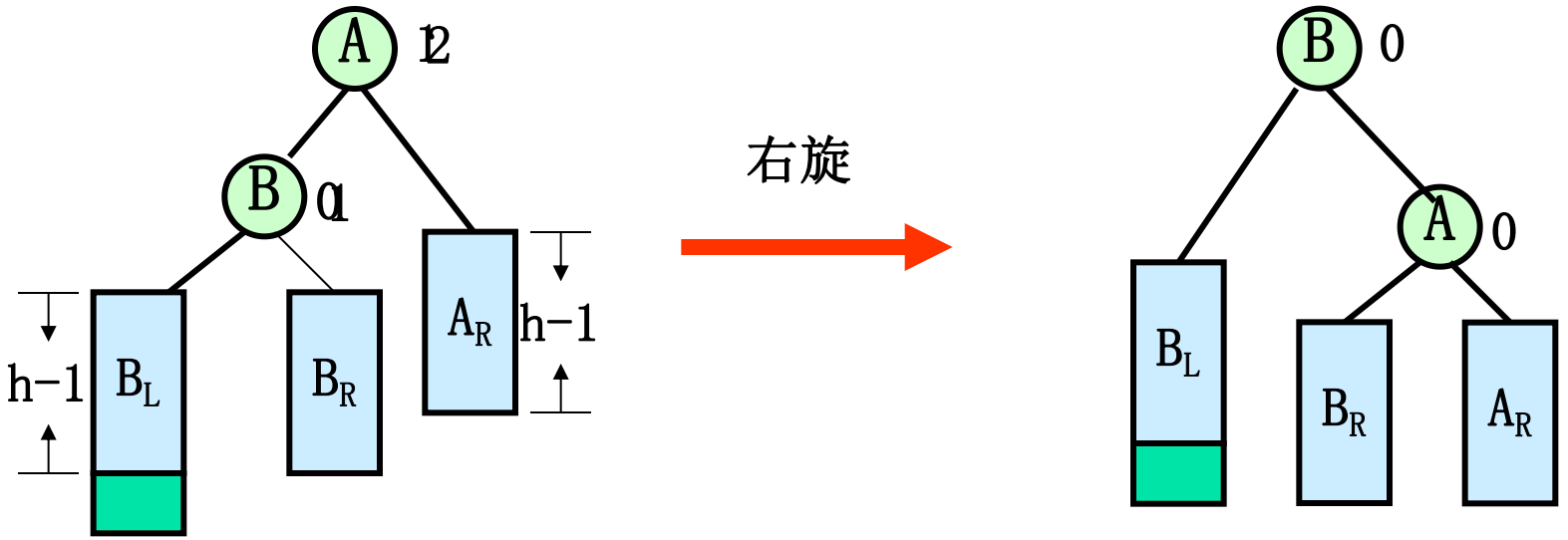
设关键字序列为{13, 24, 37, 90, 53}，构造平衡二叉排序树



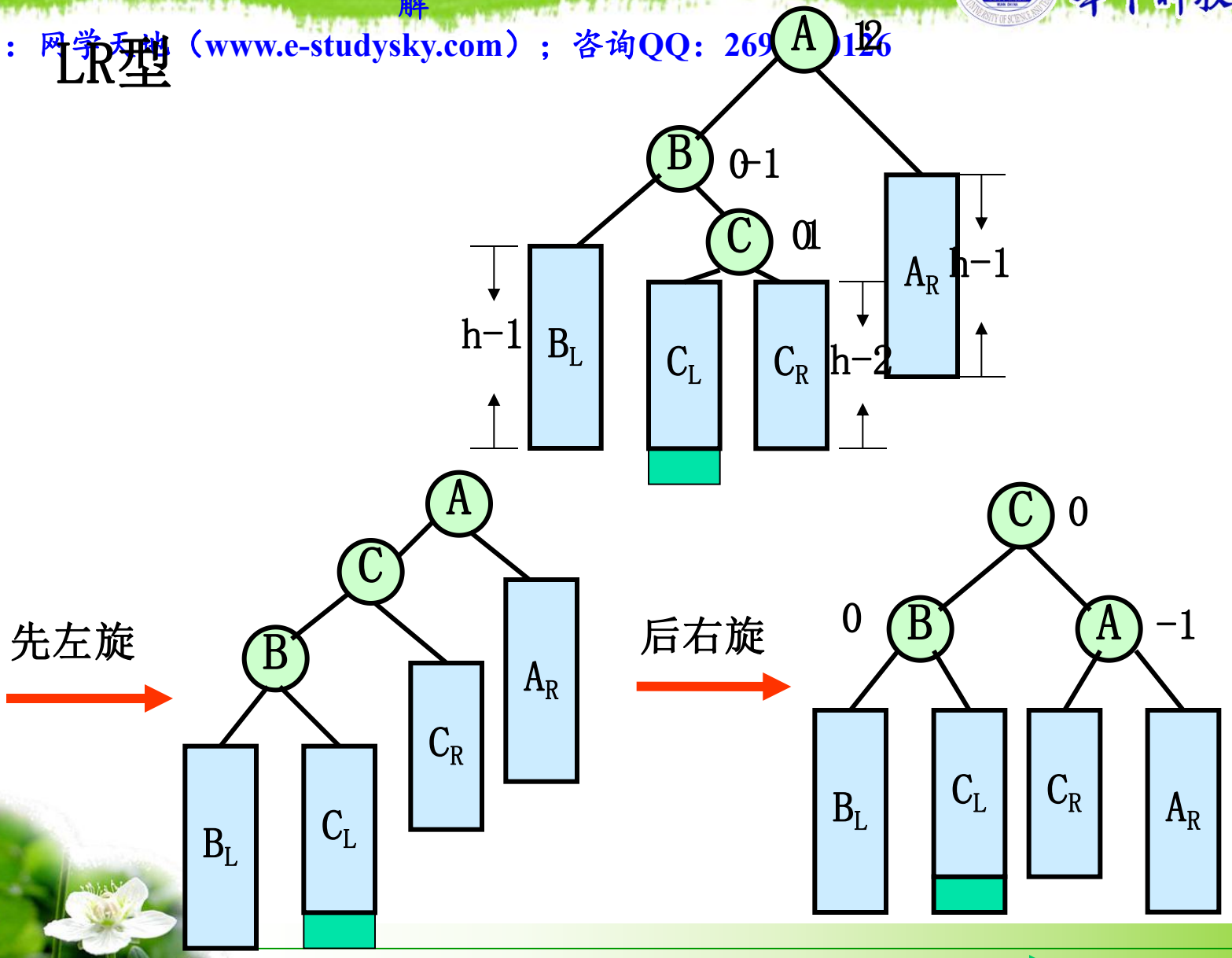


平衡二叉排序树的平衡旋转方法：

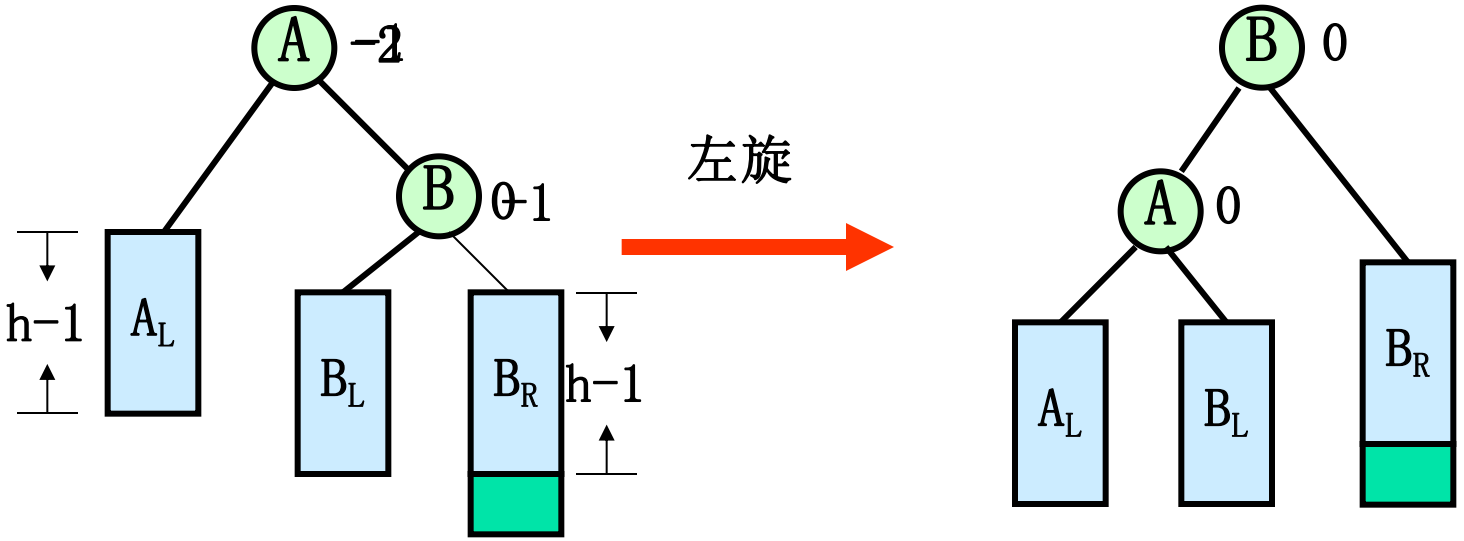
（a） LL型



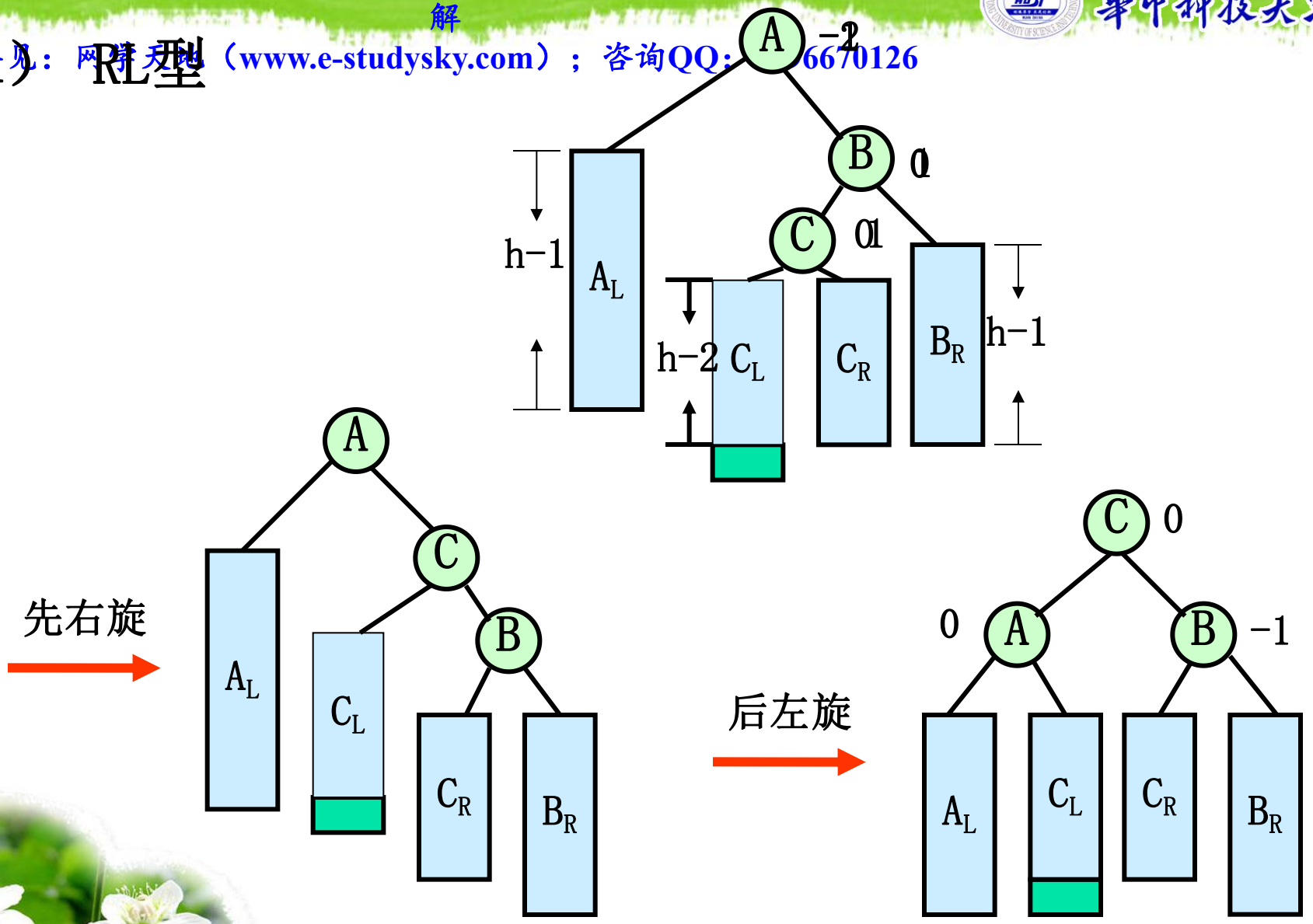
(b) LR型



（c） RR型



（d）RL型



9.2.2 B-树和 B+ 树（自学部分）

前面的查找称为内部查找，适合规模比较小的文件。如以平衡二叉树作为磁盘文件的索引组织时，若以结点为内、外存交换单位，则查找到需要的关键字，平均对磁盘进行 log n次访问。

j	p_0	k_1	p_1	k_2	…				k_j	p_j

j： 关键字的个数　　p_i：孩子指针　　k_1：关键字

转到HASH

一、B-树定义

一棵m阶B-树，或为空树，或为满足下列特性的m叉树：

（1）树中每个结点至多有m棵子树；

（2）若根结点不是叶子结点，则至少有两棵子树；

（3）除根之外的所有非终端结点至少有⌈m/2⌉ 棵子树；

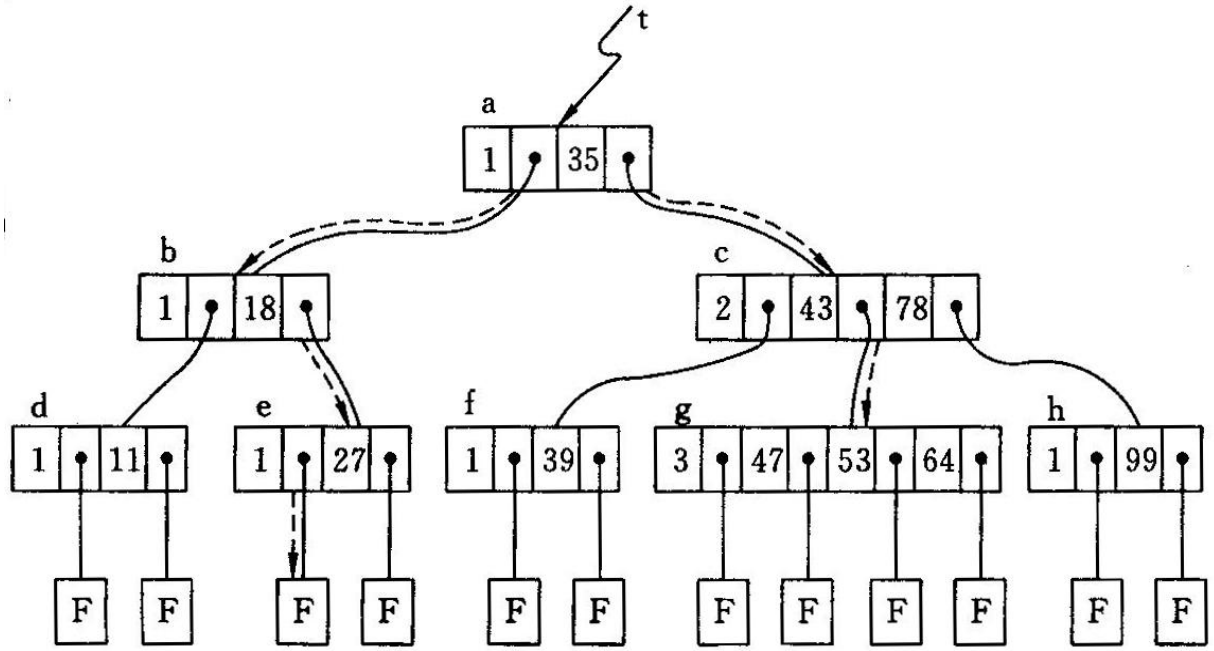
（4）所有的非终端结点中包含下列信息数据

$$(n, A_0, K_1, A_1, K_2, A_2, \cdots K_n, A_n)$$

其中：$K_i (i=1, \cdots, n)$为关键字，且$K_i < K_{i+1}$；　$A_i (i=1, \cdots, n)$为指向子树根结点的指针，且指针A_{i-1}所指子树中所有结点的关键字均小于K_i，A_n所指子树中所有结点的关键字均大于K_n，n（⌈m/2⌉ -1 ≤ n ≤ m-1）为关键字的个数（或为子树个数）。

（5）所有的叶子结点都出现在同一层次上，并且不带信息（可以看作是外部结点或查找失败的结点，实际上这些结点不存在，指向这些结点的指针为空）。

例如：m = 4 阶 B_ 树。除根结点和叶子结点之外，每个结点的儿子个数至少为⌈ m / 2 ⌉= 2 个；结点的关键字个数至少为 1 。该 B_ 树的深度为 4。叶子结点都在第 4 层上。



第 1 层

第 2 层

第 3 层(L层)

第 4 层(L+1层)

查找关键字47的过程

查找关键字23的过程

二、B-树上的查找过程

在B-树上进行查找的过程是一个顺指针查找结点和在结点的关键字中进行查找交叉进行的过程。

由于B-树主要用作文件的索引，因此它的查找涉及外存的存取，在此略去外存的读写，只作示意性的描述。

在B-树中查找元素x时，只须从根页起，每次把一个待查页从二级存储器调入内存（通常根页是常驻内存的），然后在该页中查找x0，若找到了元素x0，则检索成功，否则按下述方式继续查找，如果

（1）ki＜x＜ki+1(1≤i＜n)准备查找Ai页。

（2）x＜k1则准备查找A0页。

（3）x＞kn则准备查找An页。

如果已遇到空页，则检索失败，说明x不在B-树中；否则重复上述（1）~（3）。

因为在内存中查找所需时间把比页调入内存的时间要少很多，所以一般的B-树m值比树高要大很多。一般地，元素在页内是顺序存储（或采用二叉排序树形式），而检索算法用简单的顺序检索算法（m较小时）或者采用折半查找（m较大时）。m的实用值大约在100到500之间。

三、B-树查找分析

设关键字的总数为N，求m阶B_树的最大层次L

层次	结点数（最少）
1	1
2	2
3	$2(m/2)$
4	$2(m/2)^2$
L	$2(m/2)^{L-2}$
L+1	$2(m/2)^{L-1}$

而 L+1层的结点为叶子结点。若m阶B_树关键字的总数为N，则叶子结点即查找不成功的结点为N+1，故

$$N+1>=2(m/2)^{L-1}$$

即：

$$L<=\log\ m/2=\log((N+1)/2)+1$$

四、B-树的插入和删除

在B-树中插入新元素x，首先用检索算法查找插入位置，检索过程将终止于某一空树A_i，这时把x插在其父页（一定是叶页）的第i个位置。若插入x之后使该页上溢（页长＞m-1），需把该页分成两页，中间的那个元素递归地插入上一层页中（若m是偶数，分页时会使两页元素差1，但通常m取奇数）。递归地插入会一直波及到根页。当根页上溢时，把根页一分为二，并将中间元素上移，而产生仅含一个元素的新根页。

B-树的插入

例如，在下图1中插入25后其变化图

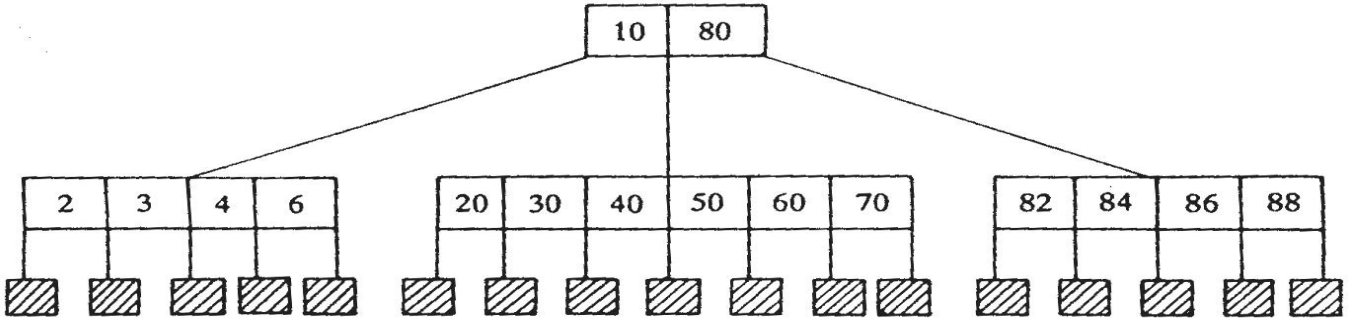
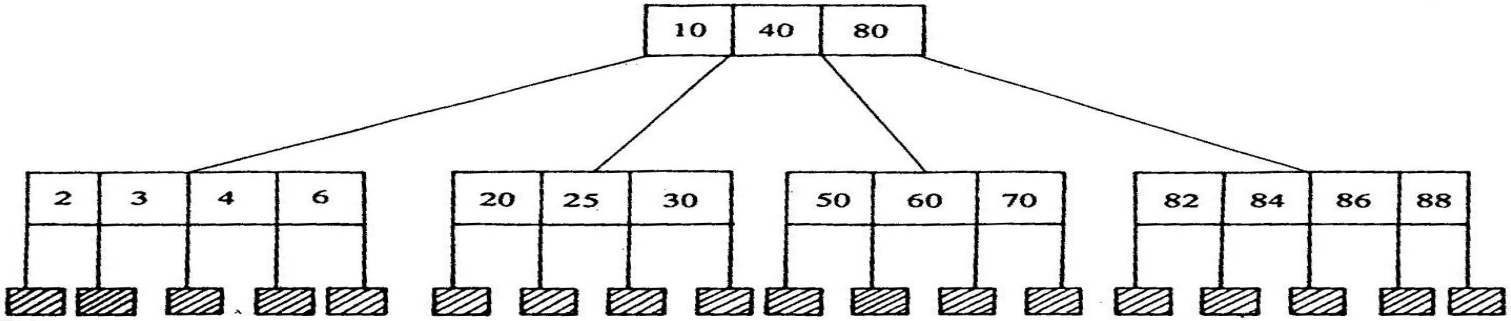


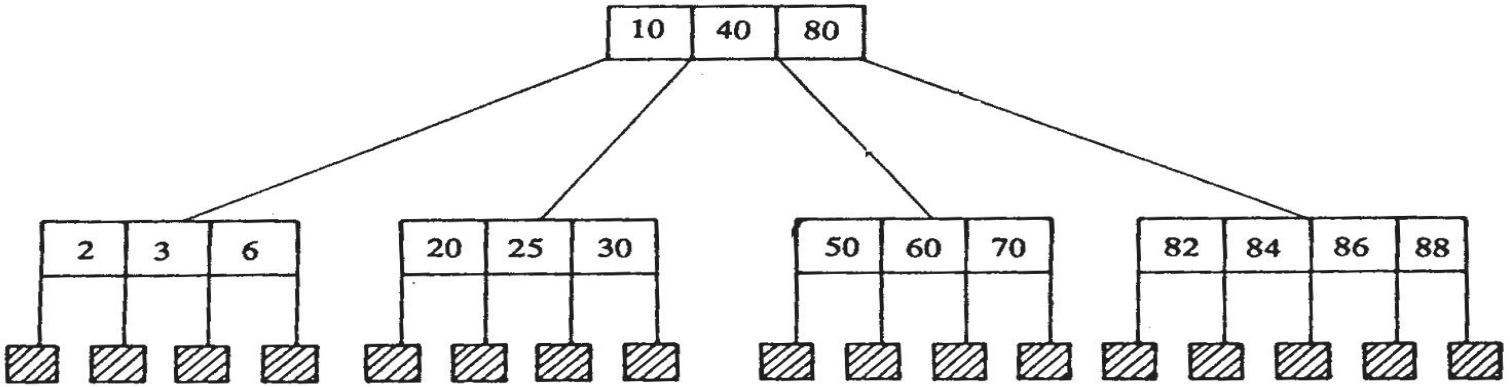
图1　一棵7 阶B-树

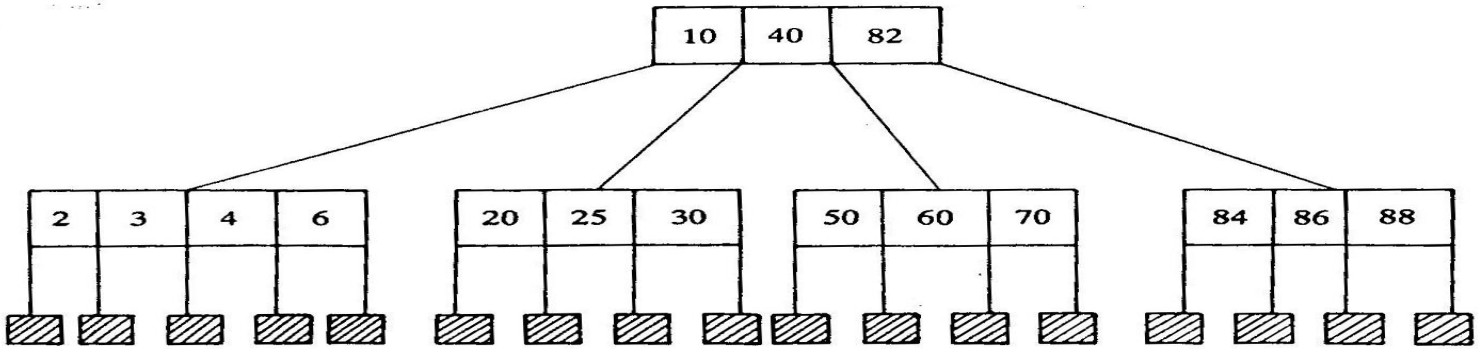


B-树的删除

删除运算分两种情形实现：
(1) 被删元素为叶页的元素，直接作删除；
(2) 被删元素是非叶页的元素，这时需用该元素左子树下最大元素（或右子树下最小元素）与被删元素作交换，最后，待被删元素落在叶页上后再作删除。

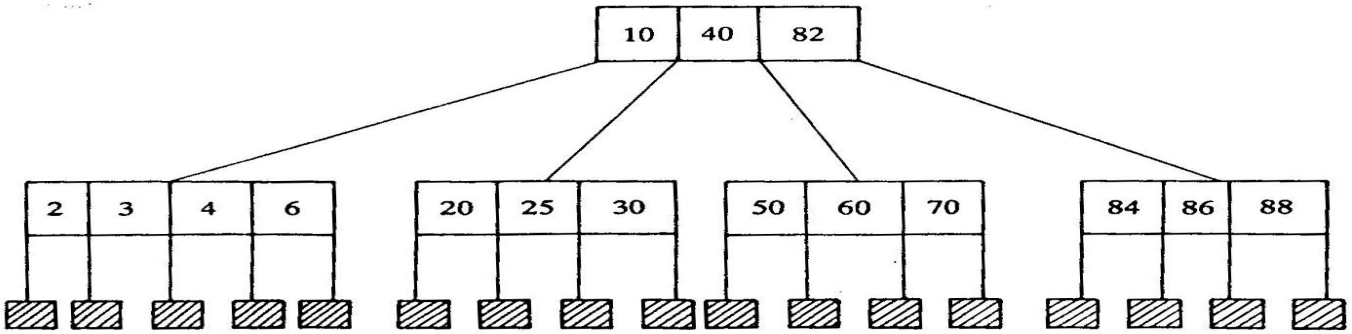
例如图中删除元素80后，用80的右子树最小元素82替代80再作删除，删除后的7阶的B-树为下图所示。



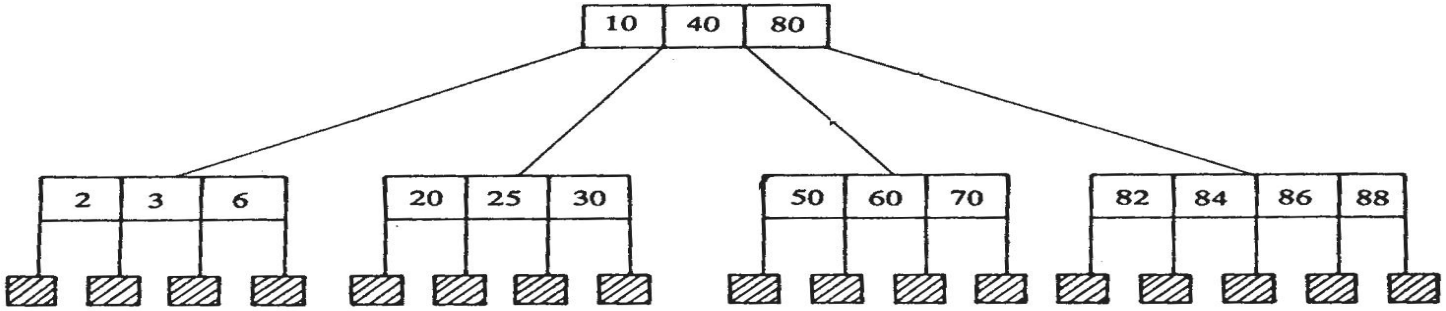


以下讨论被删元素落在叶页上的m阶的B-树中的删除运算，m阶的B-树中元素的删除可以归纳为下述三种情形：

（1）被删元素所在叶页的元素个数≥m/2，则只需直接在该页中删除该元素。如在图中删除元素4后，B-树的变化情况。



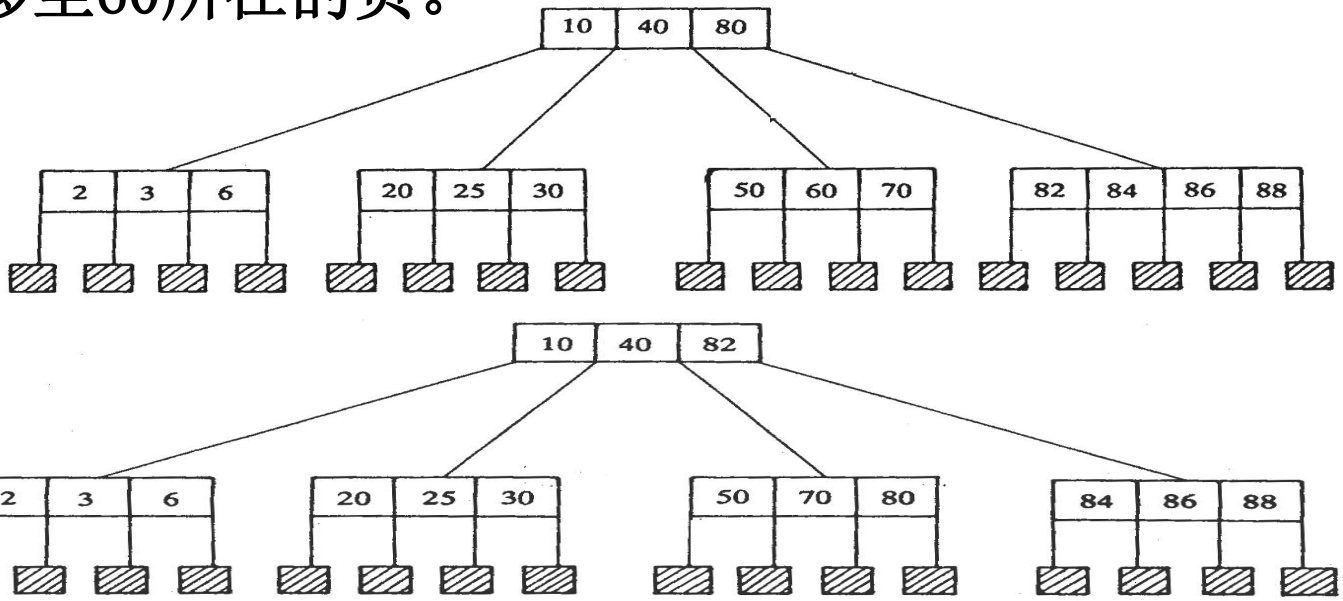
删除前



删除后

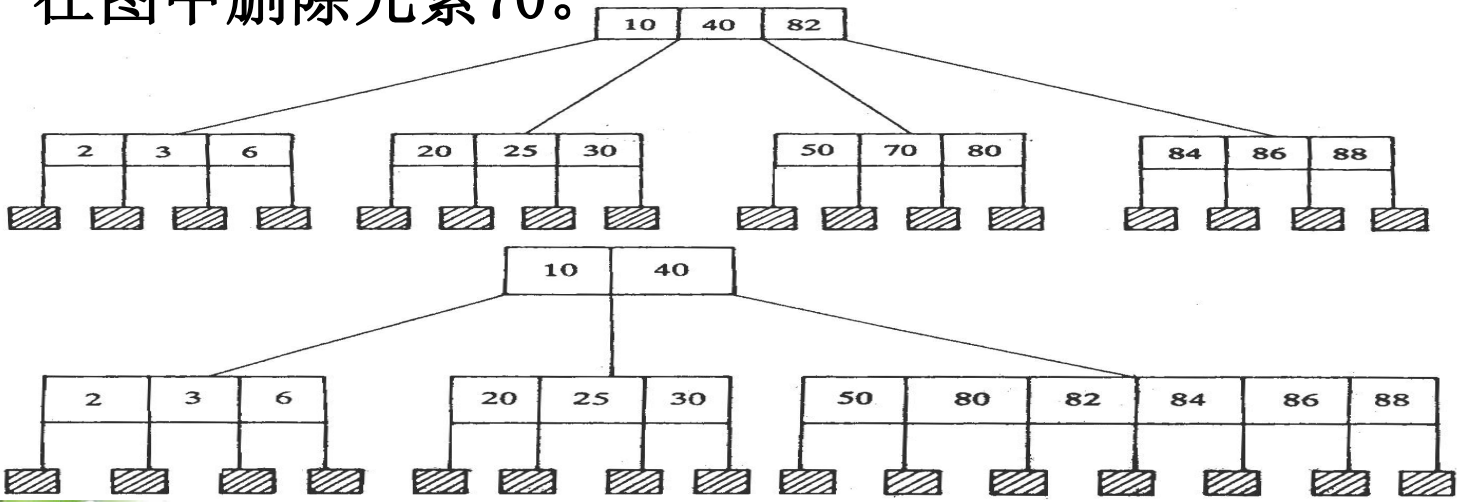
（2）被删元素所在叶页中的元素个数=m/2-1，而与该结点相邻的右兄弟或左兄弟页中元素个数大于m/2-1，则需将其兄弟结点中的最小或最大的关键码（元素）上移至其父页中，而将父页中小于或大于该上移元素的元素下移至被删元素所在页中。

例如在图中删除元素60，82上移至根页，根页中的元素80下移至60所在的页。



（3）被删元素所在页及其相邻的页中元素的数目均等于m/2-1。假设该元素所在页有右兄弟，则将该元素所在页删除后剩下的元素和右兄弟页的元素连同父页中位于被删元素所在页和右兄弟页中的元素一起合并构成新页，这时，如果父页中的一个元素下移后可能引起下溢（即元素个数＜m/2-1，则需采用（1）~（3）的办法再递归地调整。

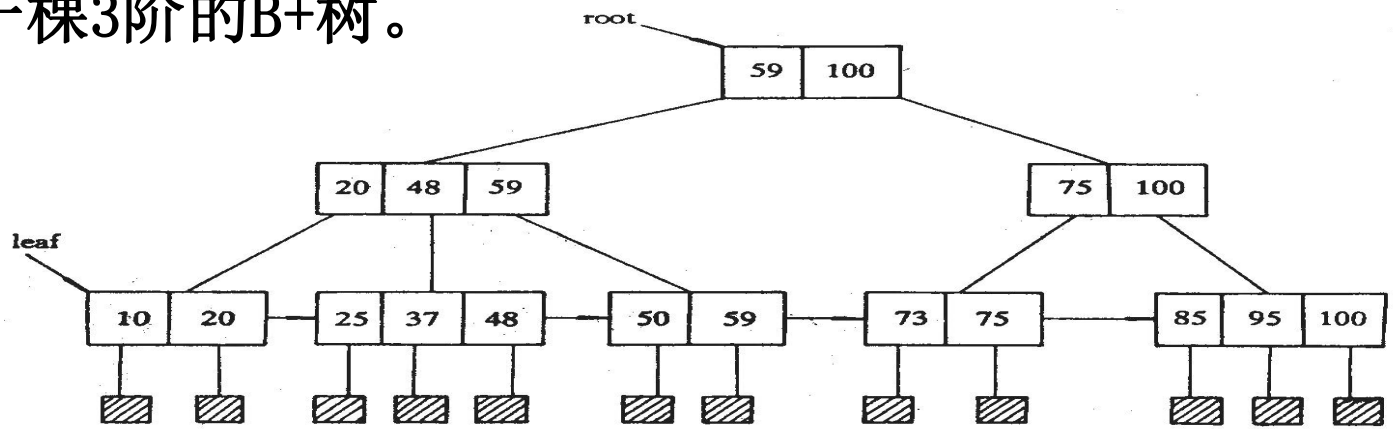
例如，在图中删除元素70。



B+树

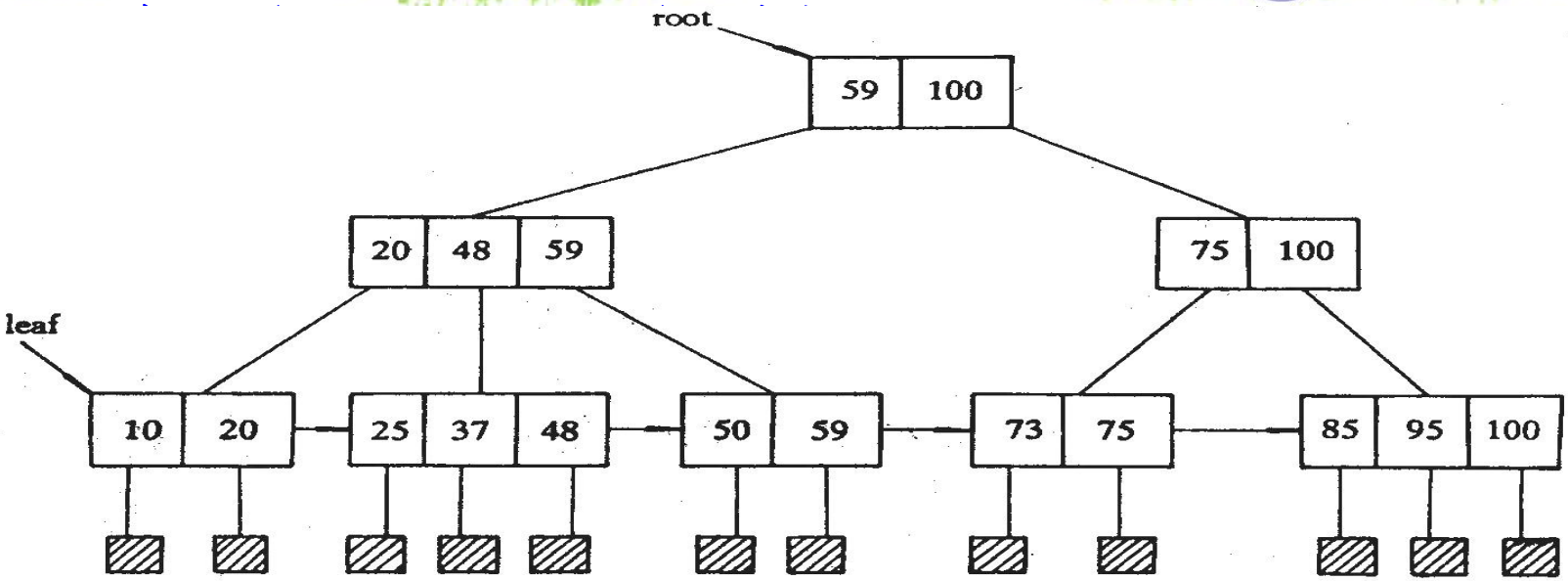
考虑B-树的一种变形，该B-树中所有的信息仅存于B树的叶页上，而非叶页仅含有便于查找的辅助信息，如本子树中结点关键码最小（或最大）值，这样的B-树被称B+树.

例 一棵3阶的B+树。



通常在树上有两个头指针，一个指向根页，另一个指向关键码最小的叶页。因此，可以对树进行两种查找运算。一种是从最小关键码起顺序查找，另一种是从根结点开始进行查找。

详



通常在树上有两个头指针，一个指向根页，另一个指向关键码最小的叶页。因此，可以对树进行两种查找运算。一种是从最小关键码起顺序查找，另一种是从根结点开始进行查找。

静态和动态查找表查找方法

静态查找表和动态查找表通过比较关键字进行查找：

（1）顺序表，对数据元素的存储一般有两种形式：

(a) 是按到来次序连续存放，查找时顺序比较查找；

(b) 按关键字的相对关系整理后以递增或递减形式连续存放，则查找时使用顺序法或二分法比较查找。

（2）二叉排序树，从根开始进行比较查找。

不足：查找时无法根据关键字的值估计数据元素可能在的位置。

9.3 哈希(Hash)表和哈希法

存储数据元素时，利用一个Hash函数根据数据元素的关键字计算出该数据元素的存储位置，查找时，也是根据给定的数据元素的关键字计算出该数据元素可能存储位置，这样一来，存储和查找的效率相当高，

哈希表也称为散列表，其数据元素的存储一般是不连续的。 通过Hash函数计算出的地址称为哈希地址或散列地址。

9.3.1 根据设定的哈希函数H(k)和处理冲突的方法

Hash函数实现的是将一组关键字映象到一个有限的地址区间上，理想的情况是不同的关键字得到不同的地址。

设K1和K2为关键字， 若K1≠K2,H(K1)=H(K2),则称
K1,K2为同义词,K2与K1为发生了冲突

例 设 H(k)=k%17

k1=5　　k2=22

∵ H(5)=5%17=5　　H(22)=22%17=5

H(5)=H(22)=5

∴ 5和22是同义词,5和22发生冲突

9.3.2 构造哈希函数的方法

1. 直接定址法

取关键字或关键字的某个线性函数值为哈希地址

H(key)=key

H(key)=a.key+b

例1 人口统计表

序号(地址)	年 龄	人 数(万)
1	1	10.5
2	2	12.6
3	3	11.0
4	4	20.8
	...	...
150	150	...

key

H(key)=key = 地址

H(年龄) = 年龄

例2 学生成绩表

序号(地址)	学　号	姓　名	性别	数学	外语
1	200041	刘大海	男	80	75
2	200042	王　伟	男	90	83
3	200043	吴晓英	女	82	88
4	200044	王　伟	女	80	90
	……	…..	…	..	..
n	……	…..	…	..	..

key

H(key) = key - 200040 = 地址

H(学号)= 学号- 200040

例3 标识符表

序号	标识符(key)
1	ABC
2	
3	CAD
4	DAT
5	
6	FOX
25	YAB
26	ZOO

H(key)=key的第一个字母在字母表中的序号

key	=ABC	CAD	DAT	FOX	YAB	ZOO
H(key)=	1	3	4	6	25	26

2.除留余数法

设哈希表HT[0..m-1]的表长为m，哈希地址为key除以p所的的余数：

H(key)=key MOD p　　//PASCAL语言

H(key)=key % p　　//C语言

其中：MOD,%为“取模”或“求余”

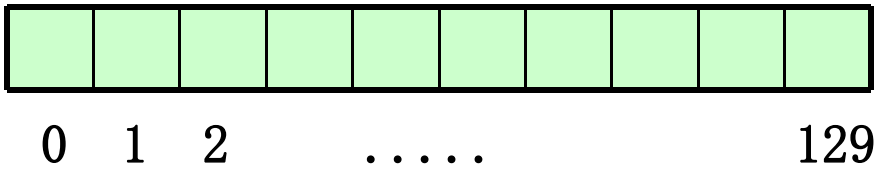
p<=m ，p为接近m的质数(素数)，如：

3,5,7,11,13,17,19,23,29,31,37.....

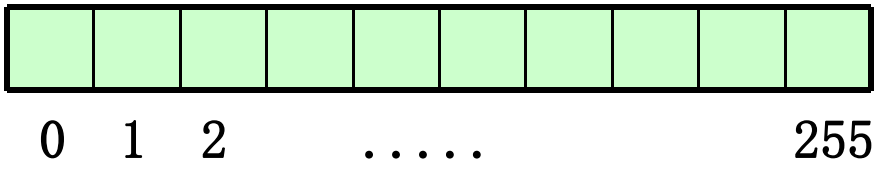
或不包含小于20的质因子的合数，如：

713(=23*31)

例1　设 m=130, 取p=127,
H(key)=key % 127

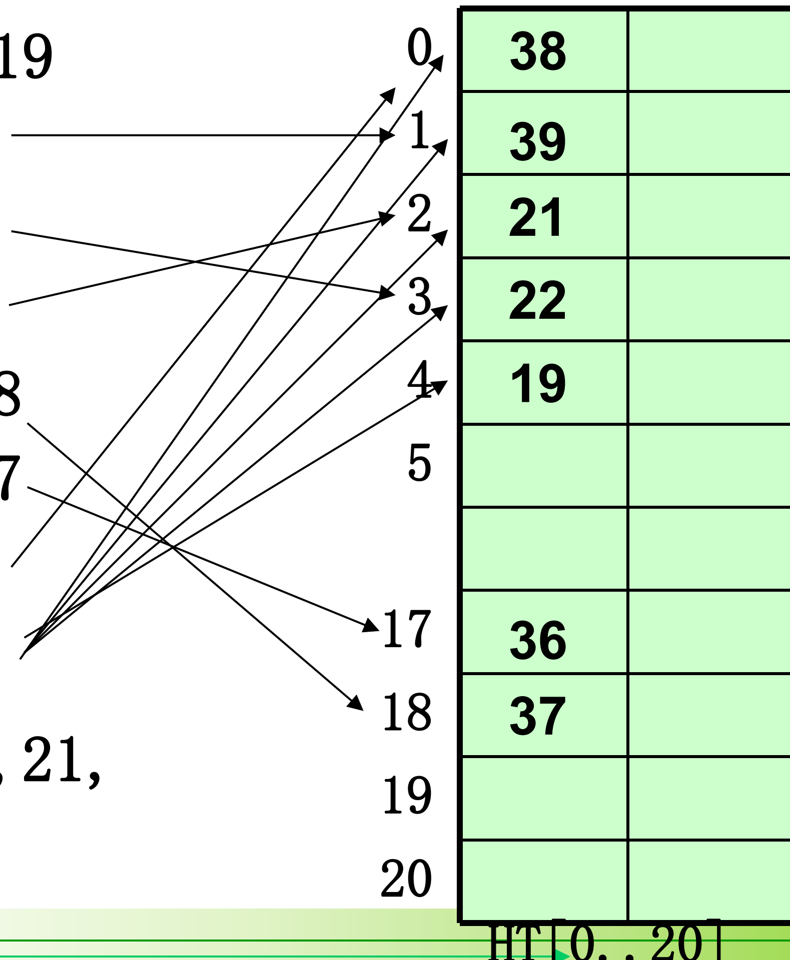


例2　设 m=256　取 p=251
H(key)=key % 251



例 设哈希表的地址范围为0～20,哈希函数为：H(K)=K % 19
输入关键字序列：39,22,21,37,36,38,19,解决冲突的方法为线性探测再散列(哈希),构造哈希表HT[0..20]。

关键字K	H(K)=K%19
39	39%19=1
22	22%19=3
21	21%19=2
37	37%19=18
36	36%19=17
38	38%19=0
19	19%19=0



19与38冲突,再与39,21,22冲突,存入HT[4]

例 设哈希表的地址范围为0～20，哈希函数为：H(K)=K % 19
输入关键字序列：39, 22, 21, 37, 36, 38, 19, 解决冲突的方法为线性探测再散列(哈希)，构造哈希表HT[0..20]。

关键字K	H(K)=K%19
39	39%19=1
22	22%19=3
21	21%19=2
37	37%19=18
36	36%19=17
38	38%19=0
19	19%19=0

19与38冲突，再与39, 21, 22冲突，存入HT[4]

19

0	38	
1	39	
2	21	
3	22	
4		
5		
17	36	
18	37	
19		
20		

HT[0..20]

再输入17, 56, 55

17%19=17

17与36冲突, 再与37冲突, 存入HT[19]。

56%19=18

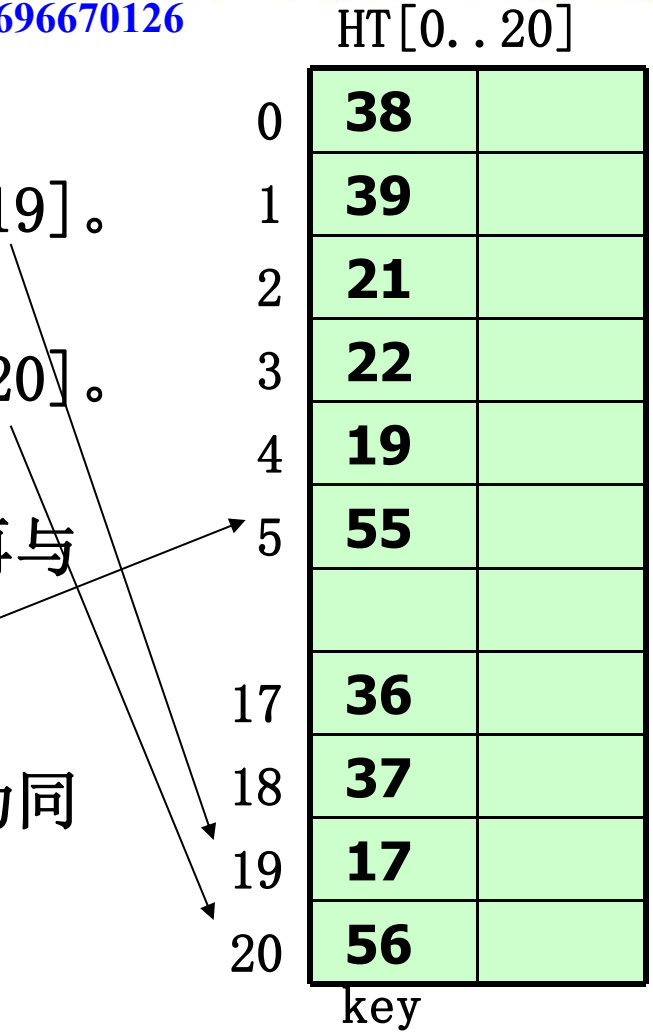
56与37冲突, 再与17冲突, 存入HT[20]。

55%19=17

55与36冲突, 再与37, 17, 56冲突, 再与38, 39, 21, 22, 19冲突, 存入HT[5]。

对于H(k)=k % 19，所有的19a+b为同义词, 0≤b≤19

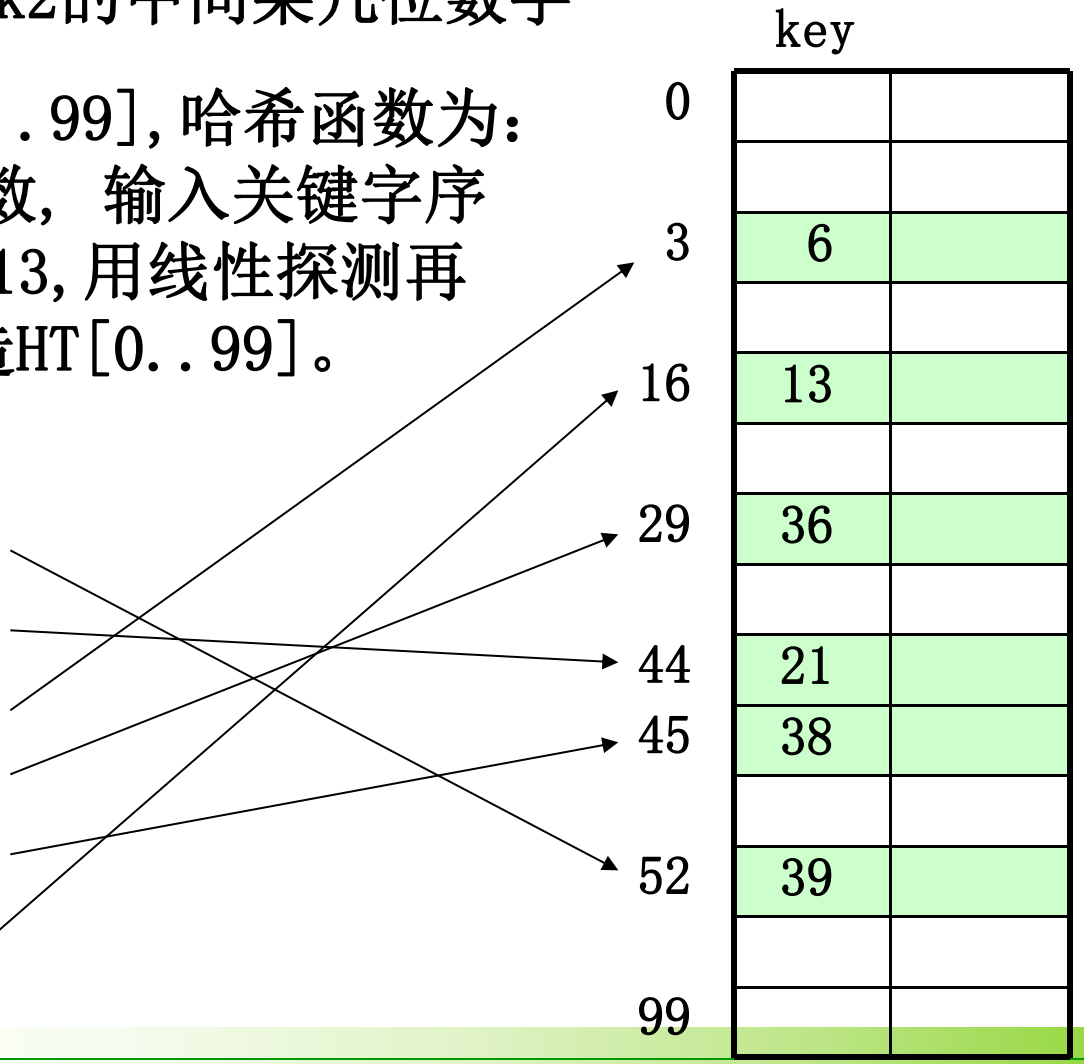
如：5, 24, 43, 62, 81, ...



3. 平方取中法——取关键字平方后的中间某几位为哈希地址，即：　　H(k)=取k2的中间某几位数字

例. 设哈希表为HT[0..99]，哈希函数为：H(K)=取k^2的中间2位数，输入关键字序列：39，21，6，36，38，13，用线性探测再散列法解决冲突，构造HT[0..99]。

K	k^2	H(K)
39	1521	52
21	0441	44
6	0036	03
36	1296	29
38	1444	44
13	0169	16



4.折叠法

将关键字分割成位数相同的几部分，然后取这几部分的叠加和作为哈希地址。

(1)边界折叠法

设表地址范围为0～999

k1=056439527

650 + 439 + 725 =1814

H(k1)=814

k2=123486790

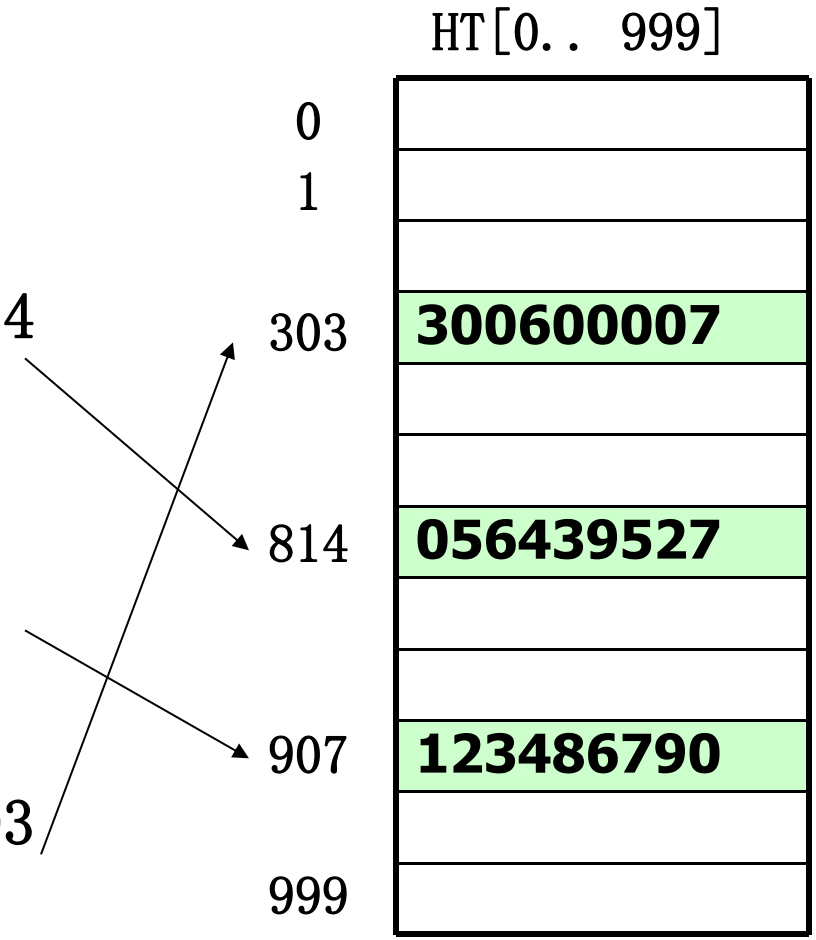
321 + 486 + 097 =907

H(k2)=907

k3=300600007

003 + 600 + 700 =1303

H(k2)=303



4. 折叠法

将关键字分割成位数相同的几部分，然后取这几部分的叠加和作为哈希地址。

(2) 移位折叠法(移位法)

设表地址范围为0～999

k1=056439527

056 + 439 + 527 =1022 → 22

H(k1)=022

k2=123486790

123 + 486 + 790 =1399 → 399

H(k2)=399

k3=300600007

300 + 600 + 007 =907 → 907

H(k2)=907

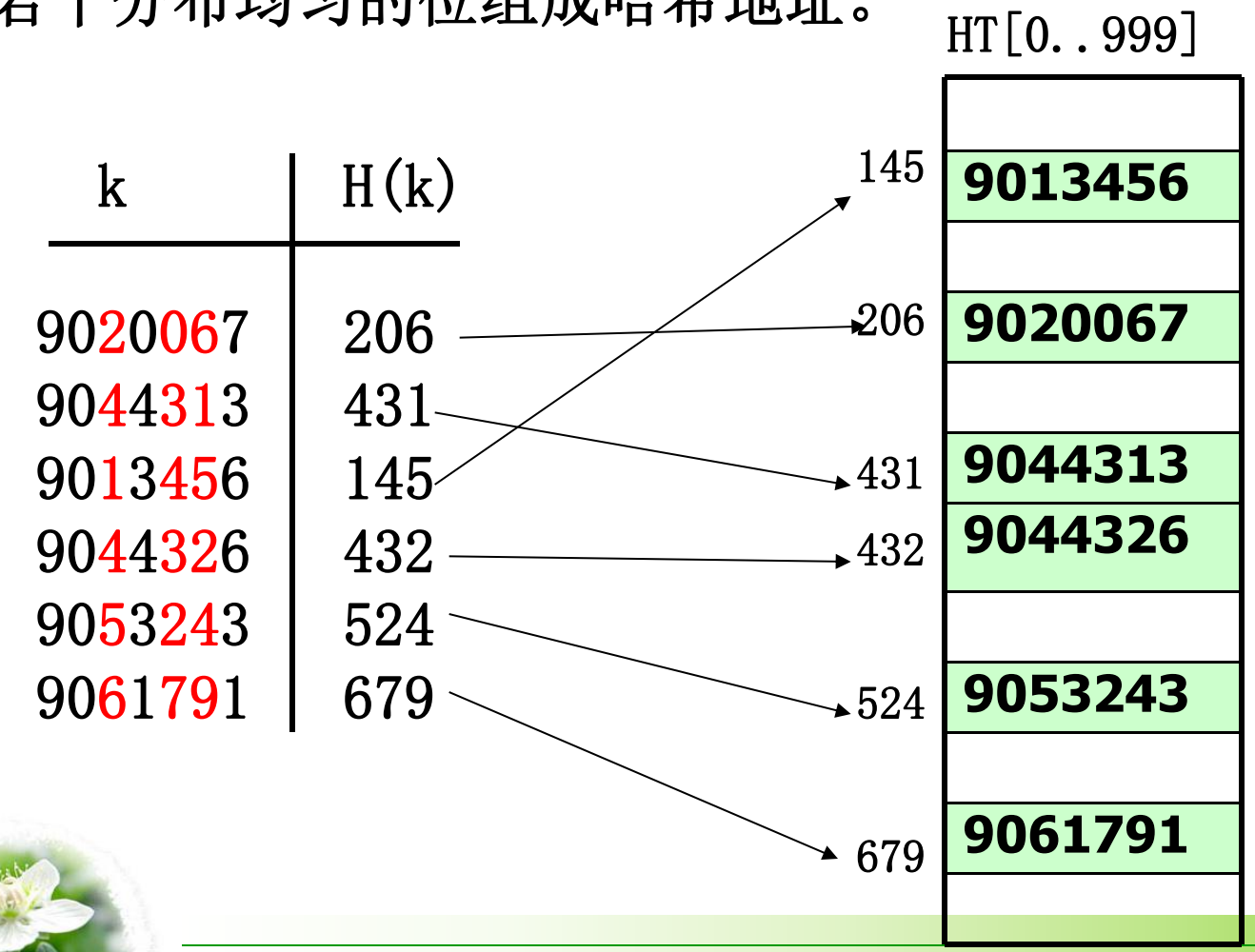
HT[0.. 999]

0	
1	
22	056439527
399	123486790
907	300600007
999	

5.数字分析法

设哈希表中可能出现的关键字都是事先知道的,则可取关键字的若干分布均匀的位组成哈希地址。

k	H(k)
9020067	206
9044313	431
9013456	145
9044326	432
9053243	524
9061791	679



6.随机数法

H(key)=random(key)

random(key)为产生伪随机数的函数

7.灵活构造哈希函数

例.设哈希表为HT[0..40],哈希函数为：

H(K)=取k^2的中间2位数*40/99

其中40/99将其00～99压缩到00～40之内，

输入关键字序列：39,21,6,36,38,13,

用线性探测再散列法解决冲突。

K	k^2	H(K)
39	1521	52*40/99=21
21	0441	44*40/99=17
6	0036	03*40/99=1
77	5929	92*40/99=37
38	1444	44*40/99=17
13	0169	16*40/99=6

key	
0	
1	6
3	
6	13
17	21
18	38
21	39
37	77
40	

9.3.3 如何解决冲突

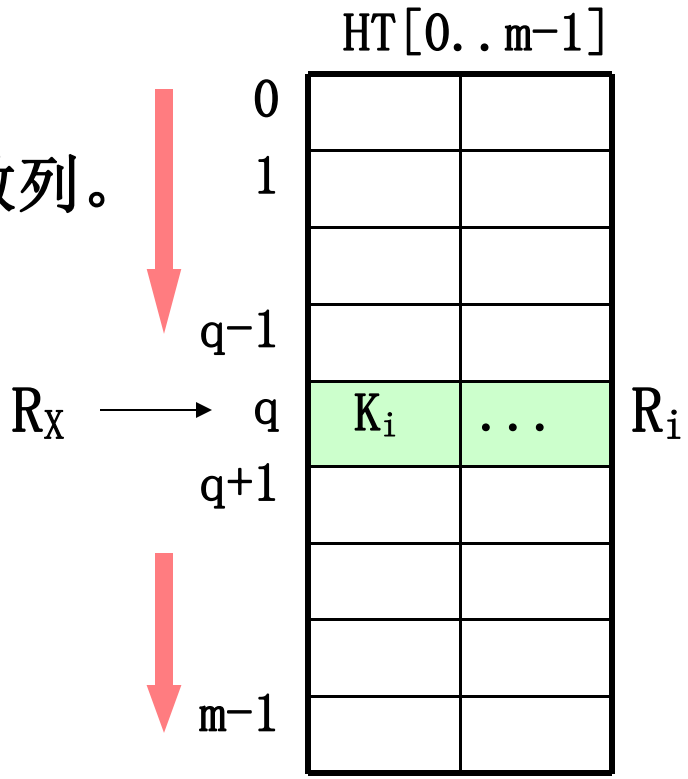
1. 开放地址法(开式寻址法)

假定记录R_i, R_X的关键字K_i, K_X为同义词，散列地址为q, R_i已存入HT[0..m-1]中的HT[q]中，则R_X 存入HT中的某个空位上。依次在地址：

$$q+1, q+2, \ldots, m-1, 0, 1, \ldots, q-1$$

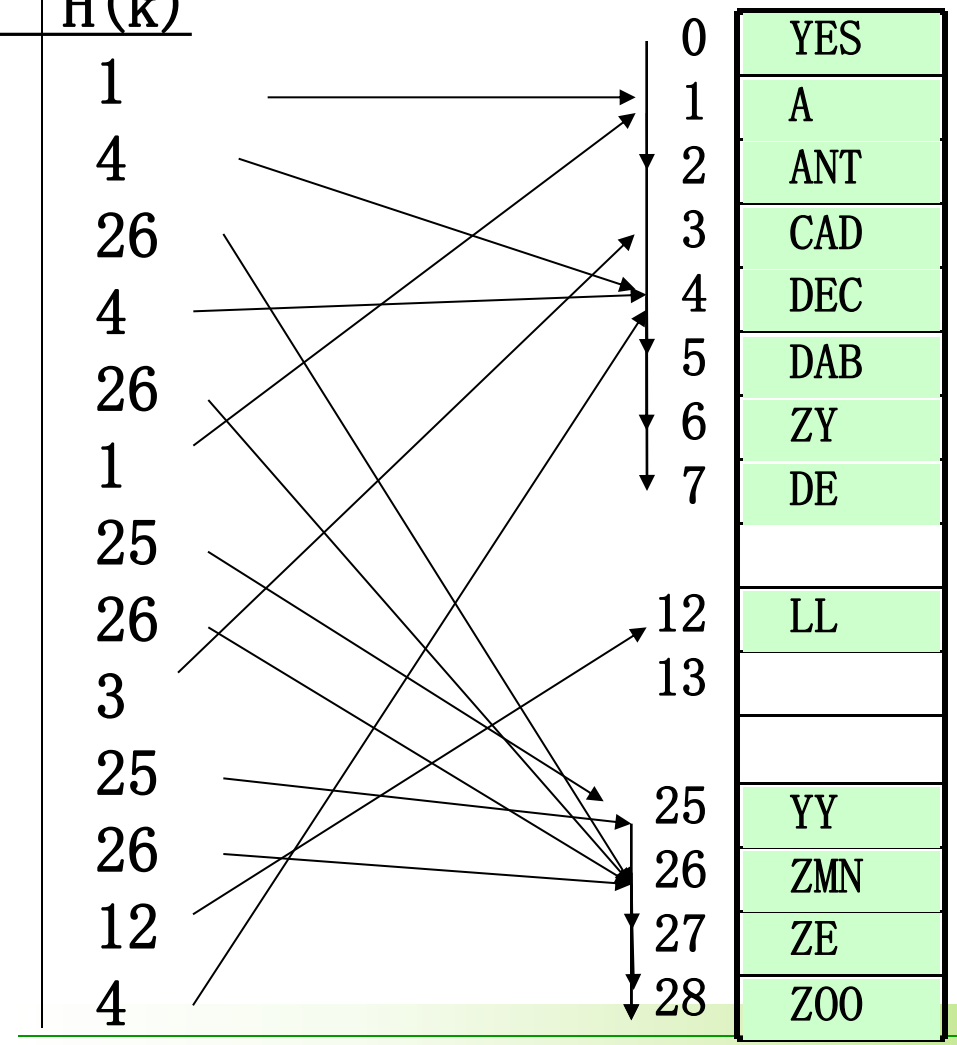
中寻找一个空位，叫做线性探测再散列。

(1) 线性探测再散列



课堂练习：设 H(k)=k的首字母在字母表中的序号，用线性探测再散列法解决冲突，依次用下列关键字，造哈希表 HT[0..28]。

	k	H(k)
1	A	1
2	DEC	4
3	ZMN	26
4	DAB	4
5	ZE	26
6	ANT	1
7	YY	25
8	ZOO	26
9	CAD	3
10	YES	25
11	ZY	26
12	LL	12
13	DE	4

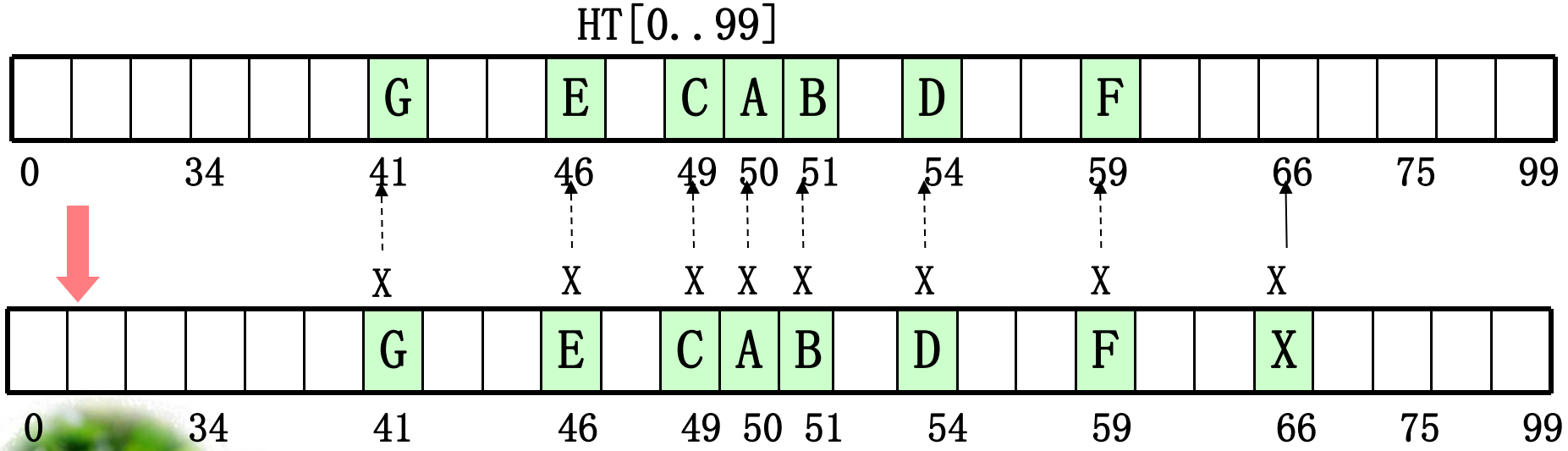


HT[0..28]

(2) 二次探测再散列

假定记录R_i和R_j的关键字K_i和K_j为同义词，散列地址为q，R_i已存入HT[0..m-1]中的HT[q]中。若依次在地址$q+1^2, q-1^2, q+2^2, q-2^2, \ldots, q+i^2, q-i^2, \ldots$中寻找一个空位，叫做二次探测再散列。

例：设记录X和A为同义词，散列地址为50，二次探测再散列的地址序列为：1, 49, 54, 46, 59, 41, 66, 34, 75, ...

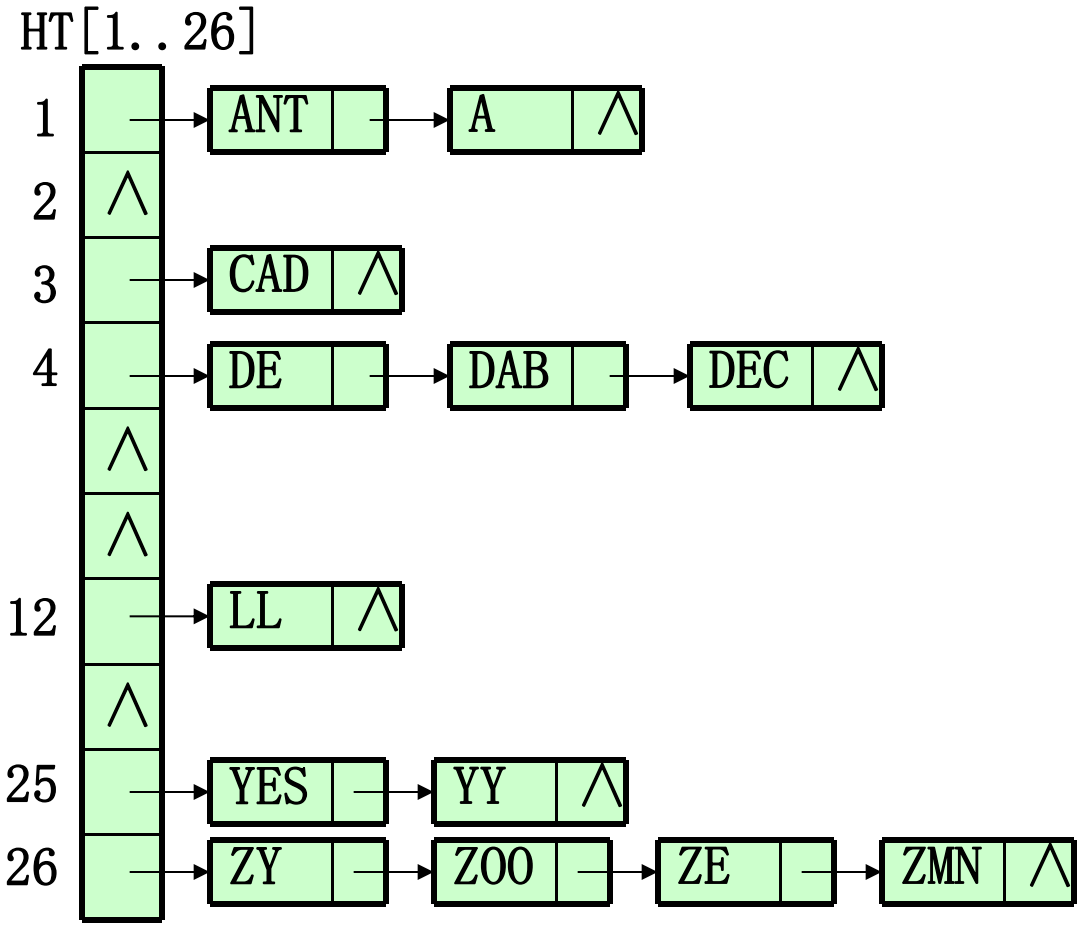


2. 链地址法

将关键字为同义词的所有记录存入同一链表中。

例　设H(k)=k的首字母在字母表中的序号，用下列关键字造哈希表 HT[1..26]

	k	H(k)
1	A	1
2	DEC	4
3	ZMN	26
4	DAB	4
5	ZE	26
6	ANT	1
7	YY	25
8	ZOO	26
9	CAD	3
10	YES	25
11	ZY	26
12	LL	12
13	DE	4

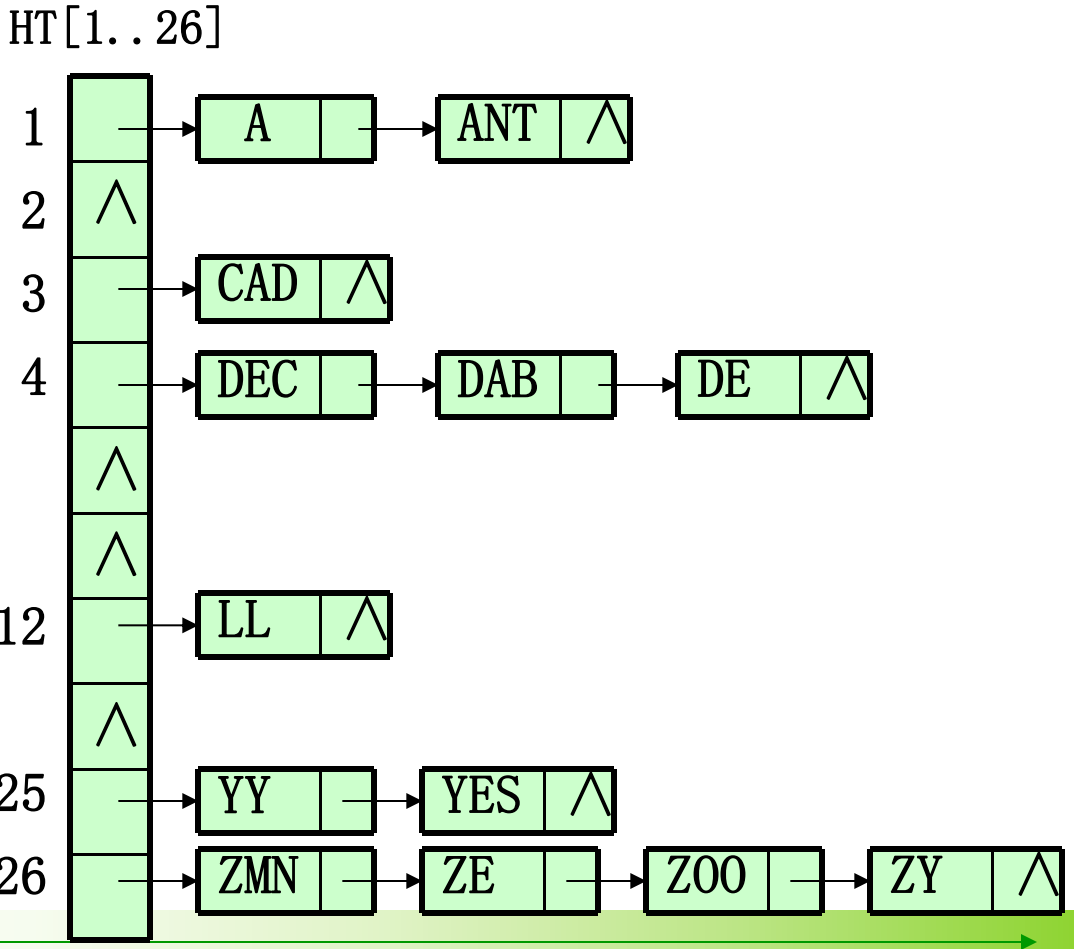


2 链地址法

将关键字为同义的所有记录存入同一链表中。

例　设H(k)=k的首字母在字母表中的序号，用下列关键字造哈希表 HT[1..26]

	k	H(k)
1	A	1
2	DEC	4
3	ZMN	26
4	DAB	4
5	ZE	26
6	ANT	1
7	YY	25
8	ZOO	26
9	CAD	3
10	YES	25
11	ZY	26
12	LL	12
13	DE	4



3. 建立公共溢出区

将发生冲突的所有记录存入一个公共溢出表OT[0..v]中。

例　设H(k)=k的首字母在字母表中的序号,用下列关键字生成基本表HT[1..26]和溢出表OT[0..50]

序号	关键字	H（K）
1	A	1
2	DEC	4
3	ZMN	26
4	DAB	4
5	ZE	26
6	ANT	1
7	YY	25
8	ZOO	26
9	CAD	3
10	YES	25
11	ZY	26
12	LL	12
13	DE	4

HT[1..26]

1	A
2	
3	CAD
4	DEC
12	LL
25	YY
26	ZMN

OT[0..50]

1	DAB
2	ZE
3	ANT
4	ZOO
5	YES
6	ZY
7	DE
50	

4. 再哈希法

发生冲突时，使用下一个哈希函数计算哈希地址：

j1=H1(K)； j2=H2(K)； j3=H3(K)；.....

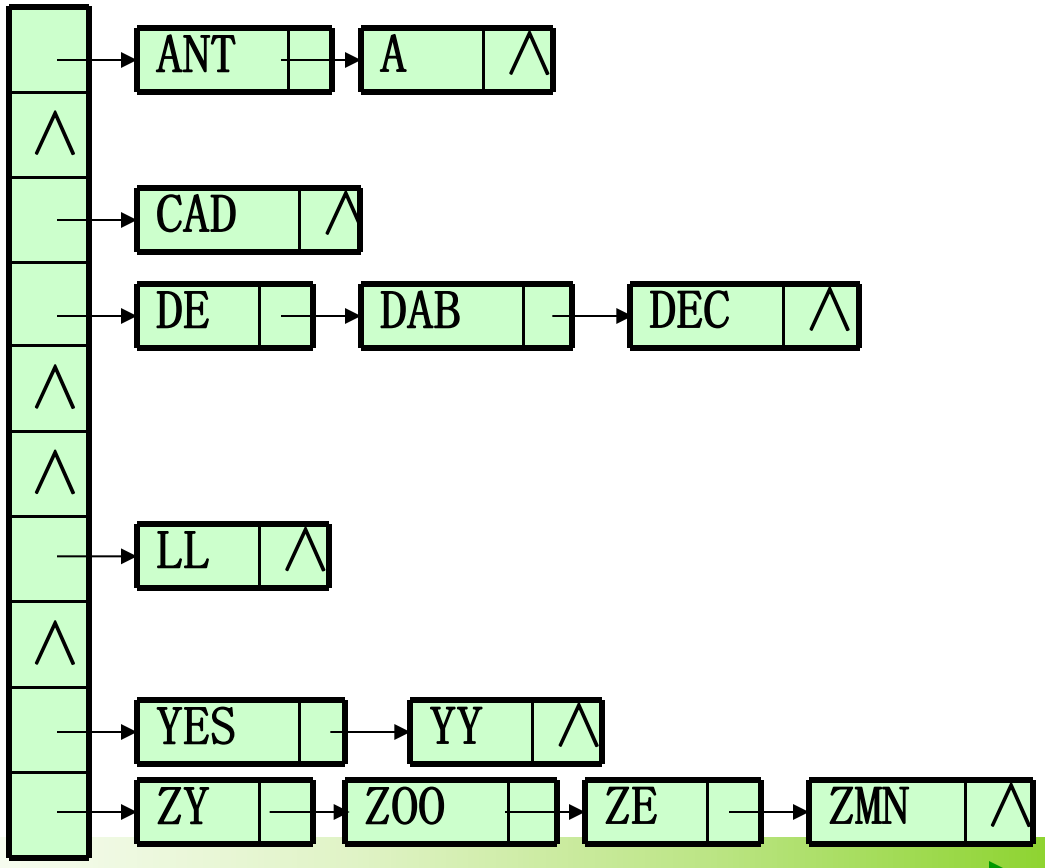
9.3.4 哈希表的查找及其分析

例1(链地址法)假定每个记录的查找概率相等，查找成功时的平均查找长度：

ASL=(1+2+1+1+2+3+1+1+2+1+2+3+4)/13

= 24/13

≈1.85



例2（线性探测再散列）假定每个记录的查找概率相等，查找成功时的平均查找长度.

ASL=34/13 ≈2.62

关键字	H（K）	比较次数
YES	25	5
A	1	1
ANT	1	2
CAD	3	1
DEC	4	1
DAB	4	2
ZY	26	10
DE	4	4
LL	12	1
YY	25	1
ZMN	26	1
ZE	26	2
ZOO	26	3

HT[0..28]

0	YES
1	A
2	ANT
3	CAD
4	DEC
5	DAB
6	ZY
7	DE
12	LL
13	
25	YY
26	ZMN
27	ZE
28	ZOO

例2(续) (线性探测再散列)查找不成功时的平均查找长度.需统计不成功时比较次数

ASL=52/26

H(k)	比较次数	H(k)	比较次数
1	7	14	0
2	6	15	0
3	5	16	0
4	4	17	0
5	3	18	0
6	2	19	0
7	1	20	0
8	0	21	0
9	0	22	0
10	0	23	0
11	0	24	0
12	1	25	12
13	0	26	11

HT[0..28]

0	YES
1	A
2	ANT
3	CAD
4	DEC
5	DAB
6	ZY
7	DE
12	LL
13	
25	YY
26	ZMN
27	ZE
28	ZOO

一般情况：平均查找长度依赖于哈希表的装填因子：

α =(表中填入记录数)/(哈希表的长度)

解决冲突的方法	平均查找长度	
	查找成功	查找失败
线性探测再散列	$\frac{1}{2}(1+\frac{1}{1-\alpha})$	$\frac{1}{2}[1+\frac{1}{(1-\alpha)^2}]$
二次探测再散列	$\frac{\ln(1-\alpha)}{\alpha}$	$\frac{1}{1-\alpha}$
链地址	$1+\frac{\alpha}{2}$	$\alpha+e^{-\alpha}$