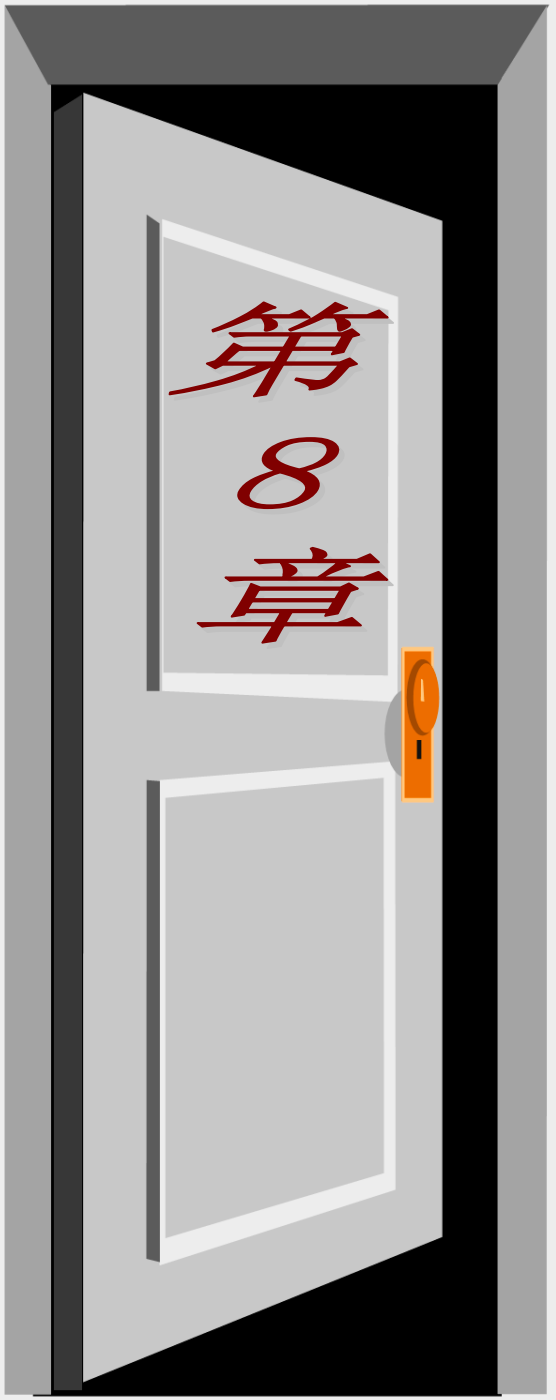
尚德机构

数据结构

主讲：王老师

学习是一种信仰！ IN LEARING WE TRUST



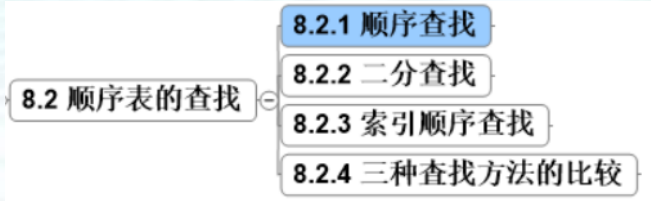
查 找

- 查找同排序一样，也有内查找和外查找之分：若整个查找过程都在内存中进行，则称之为内查找；反之，称为外查找。

- 由于查找运算的主要操作是关键字的比较，因此，通常把查找过程中的平均比较次数（也称为平均查找长度）作为衡量一个查找算法效率优劣的标准。

- 平均查找长度（Average Search Length，ASL)的计算公式定义为

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

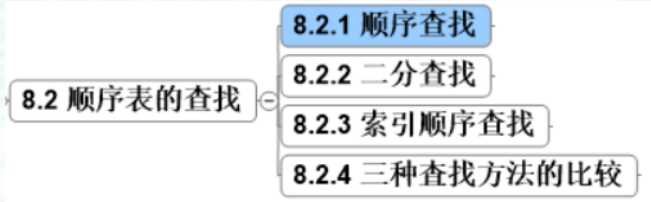


顺序表是指线性表的顺序存储结构。

假定在本章的讨论中，顺序表采用一维数组来表示，其元素类型为NodeType，它含有关键字key域和其他数据域data，key域的类型假定用标识符KeyType (int)表示，具体表的类型定义如下：

```
typedef struct {
    KeyType key;
    InfoType data;
}NodeType;
typedef NodeType SeqList [n+1] ;      //0 号单元用作哨兵
```

在顺序表上的查找方法有多种，这里只介绍最常用的、最主要的两种方法，即顺序查找和二分查找。



顺序查找（Sequential Search)又称线性查找，是一种最简单和最基本的查找方法。其基本思想是：从表的一端开始，顺序扫描线性表，依次把扫描到的记录关键字与给定的值k相比较，若某个记录的关键字等于k，则表明查找成功，返回该记录所在的下标。 若直到所有记录都比较完，仍未找到关键字与k相等的记录，则表明查找失败，返回0值。顺序查找的算法描述如下：

```
int SeqSearch (SeqList R,KeyType k,int n)
{  //R[0]作为哨兵，用R[0] .key==k作为循环下界的终结条件
    R[0] .key=k;                          //设置哨兵
    i=n;                                  //从后向前扫描
    while (R[i].key!=k)
        i--
    return i;                        //返回其下标，若找不到，返回0
}
```

算法分析

成功查找：$ASL=\sum_{i=0}^{n-1} P_i C_i$（设每个记录的查找概率相等）

$=1/n \sum_{i=0}^{n-1} (n-i+1)$

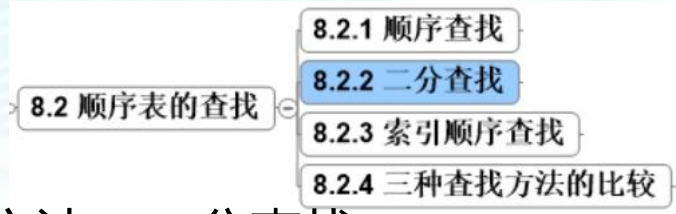
$=(n+1)/2$

不成功查找：$ASL=n+1$

▲顺序查找优点：简单，对表的结构无任何要求，顺序存储、链式存储均可；

▲顺序查找缺点：效率低。

8.2.2二、二分查找



- 二分查找（Binary Search)又称折半查找，是一种效率较高的查找方法。二分查找要求查找对象的线性表必须是顺序存储结构的有序表（不妨设递增有序)。
- 二分查找的过程是：首先将待查的k值和有序表R[1...n]的中间位置mid上的记录的关键字进行比较：
  - 若相等，则查找成功，返回该记录的下标mid；
  - 若R[mid].key>k，则k在左子表R[1...mid-1]中，接着再在左子表中进行二分查找即可；
  - 若 R[mid].key<k，则说明待查记录在右子表R[mid+1..n]中，接着只要在右子表中进行二分查找即可。

这样，经过一次关键字的比较，就可缩小一半的查找空间，如此进行下去， 直到找到关键字为k的记录或者当前查找区间为空时（即查找失败）为止。二分查找的过程是递归的。


```
int BinSearch(SeqList R,KeyType k,int low,int high)
{   //在区间R[low..high]内进行二分递归，查找关键字值等于k的记录
    //low的初始值为1,high的初始值为n
    int mid;
    if (low<=high) {
        mid= (low+high) /2;
        if (R[mid] .key==k) return mid;                  //查找成功，返回其下标
        if (R[mid] .key>k)
              return BinSearch(R,k,low,mid-1);           //在左子表中继续查找
        else
              return BinSearch(R,k,mid+1, high);         //在右子表中继续查找
    }
    else
        return 0;                                        //查找失败，返回0值
}
```

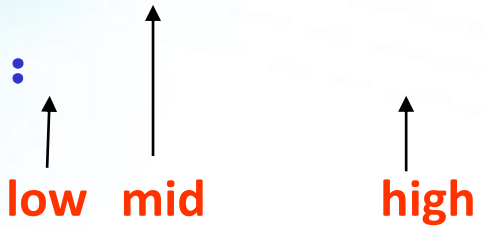
二分查找算法也可以用非递归方法来实现，其算法描述如下：

```
int BinSearch(SeqList R, KeyType k,int n)
{
    int low=1, mid, high=n;                    //初始化上下界
    while(low<=high) {
        mid=(low+high)/2;
        if(R[mid].key==k)
            return mid;                        //查找成功，返回其下标
        if(R[mid].key>k)                       //修改上界
            high=mid-1;                        //修改下界
        else low=mid+1;
    }
    return 0;                                  //查找失败，返回0值
}
```

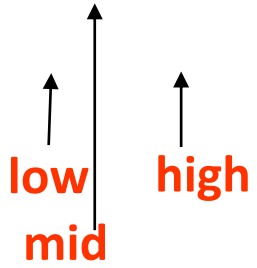
例：（在下列有序顺序表中查找K=18）

1	2	3	4	5	6	7	8	9
15，	17，	18，	22，	35，	51，	60，	88，	93
low				mid				high

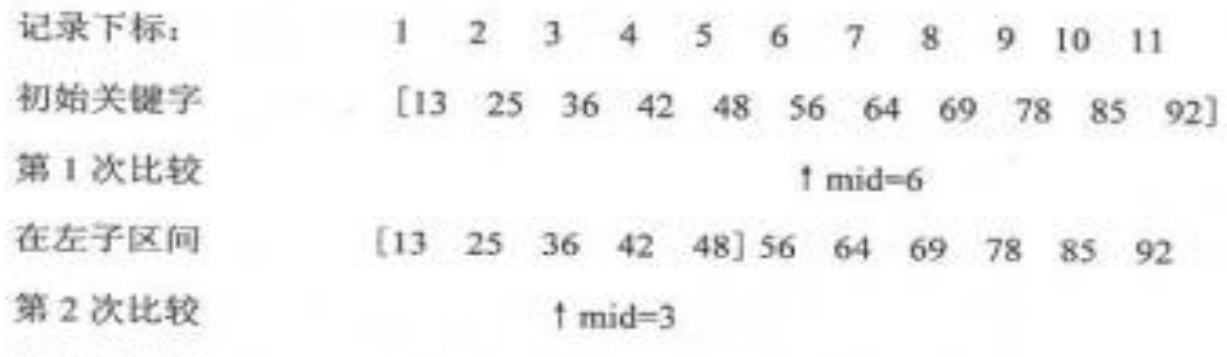
因为18<35，所以：



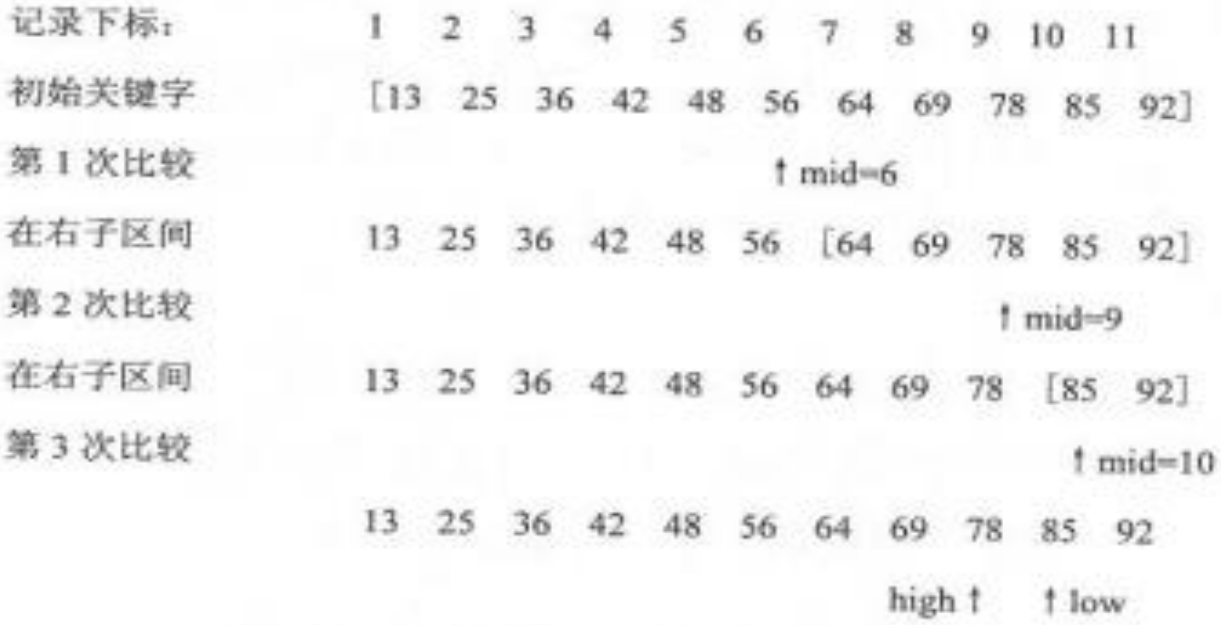
因为18>17，所以：



因为18=18，所以：查找成功，回送结果为mid=3



（a）查找 k=42 的过程（三次比较后查找成功）



（b）查找 k=80 的过程（三次比较后查找失败）

图 8.1　二分查找查找成功和查找失败的查找过程示意图

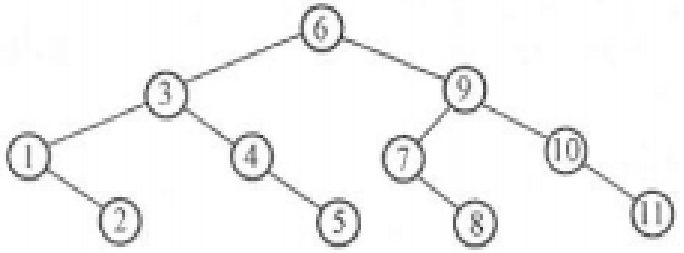


图 8.2　长度为 11 的有序表二分查找判定树

查找成功时二分查找的平均长度为

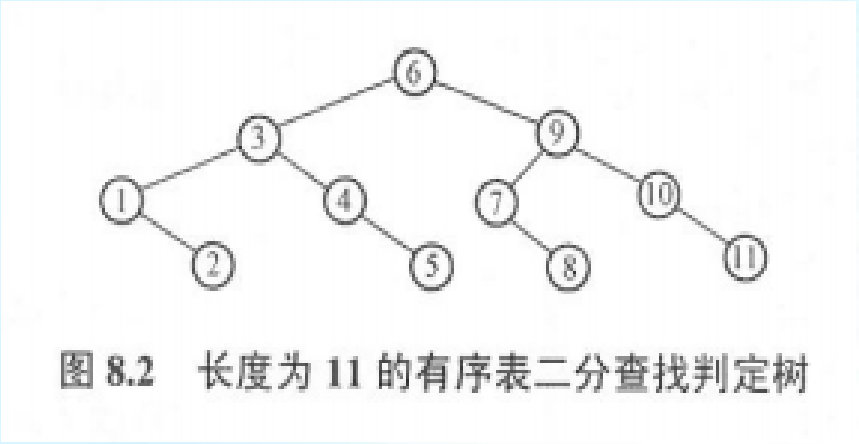


图 8.2　长度为 11 的有序表二分查找判定树

查找成功时二分查找的平均长度为：

$$ASL=\sum_{j=0}^{n}p_ic_i=\frac{1}{n}\sum_{j=0}^{h}j\bullet 2^{j-1}=\frac{n+1}{n}\log_2(n+1)-1$$

当n很大很大时，可用近似公式 $ASL=\log_2(n+1)-1$ 来表示二分查找成功时的平均查找长度。二分查找失败时，所需要比较的关键字个数不超过判定树的深度。

【例8.2】对19个记录的有序表进行二分查找，试画出描述二分查找过程的二叉树， 并计算在每个记录的查找概率相同的情况下的平均查找长度。

分析：二分查找过程的二叉树如图8.3所示。

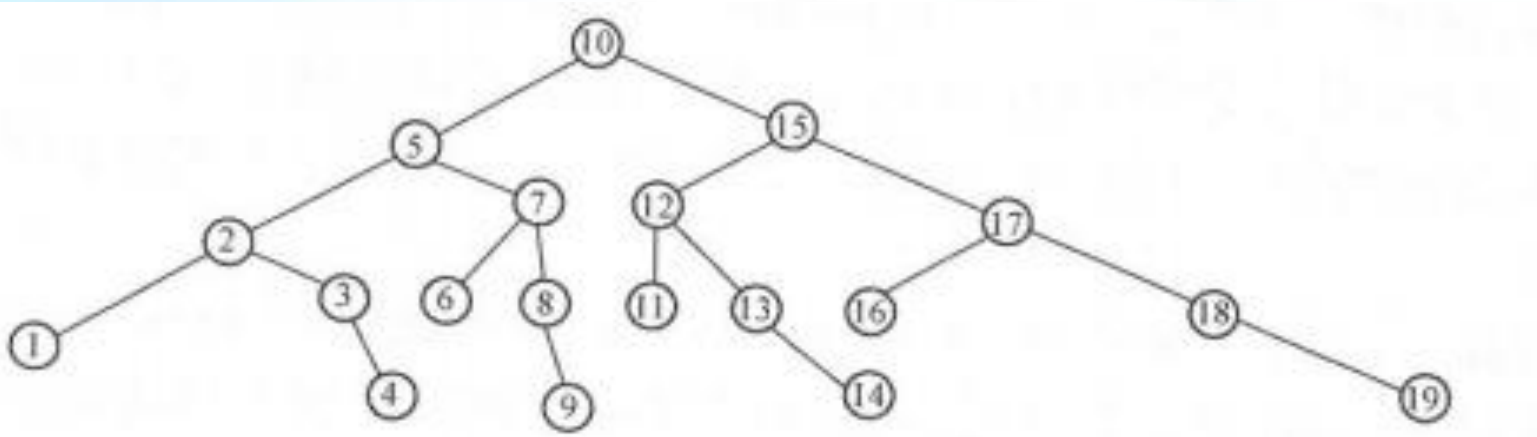


图 8.3　19 个结点的二叉树判定树

在二叉判定树上表示的结点所在的层数（深度）就是查找该结点所需要比较的次数，因此，其平均查找长度（平均比较次数）：

$$ASL = (1+2\times2+3\times4+4\times8+5\times4)/19 = 69/19 \approx 3.4.2$$

真题演练

对有序表(1,9,12,41,62,77,82,95,100)采用二分查找方法查找值82，查找过程中关键字的比较次数是（　）。

A:1

B:2

C:4

D:7

真题演练

对有序表(1,9,12,41,62,77,82,95,100)采用二分查找方法查找值82，查找过程中关键字的比较次数是（　）。

A:1

B:2

C:4

D:7

答案：B

真题演练

对含有16个元素的有序表进行二分查找，关键字比较次数最多是（ ）。

A:3

B:4

C:5

D:6

真题演练

对含有16个元素的有序表进行二分查找，关键字比较次数最多是（ ）。

A:3

B:4

C:5

D:6

答案：C

8.2.3三、索引顺序查找

- 索引顺序查找又称分块查找，是一种介于顺序查找和二分查找之间的查找方法。
- 分块查找的基本思想是：首先查找索引表，可用二分查找或顺序查找，然后在确定的块中进行顺序查找。由于分块查找实际上是两次查找过程，因此整个查找过程的平均查找长度，是两次查找的平均查找长度之和。

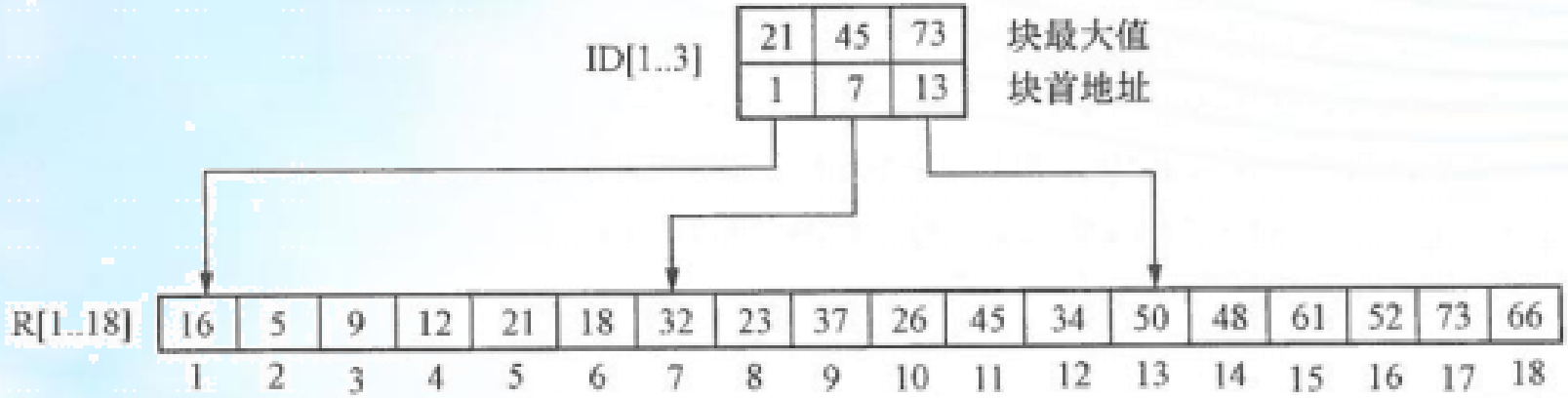


图 8.4　分块有序表及其索引表的存储表示

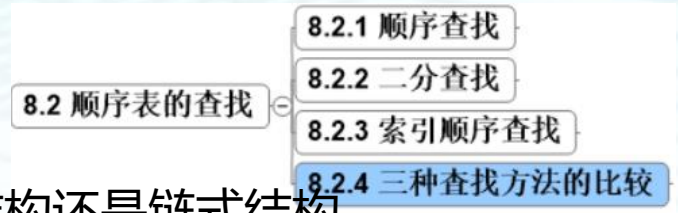
查找块有两种方法，一种是二分查找，若按此方法来确定块，则分块查找的平均查找长度为：

$$ASL_{blk} = ASL_{bin} + ASL_{seq} = \log(b+1) - 1 + (s+1)/2 \approx \log(n/s+1) + s/2$$

另一种是顺序查找，此时的分块查找的平均查找长度为：

$$ASL_{blk} = (b+1)/2 + (s+1)/2 = (s^2 + 2s + n)/(2s)$$

8.2.4四、三种查找方法的比较



- 顺序查找的优点是算法简单，且对表的存储结构无任何要求，无论是顺序结构还是链式结构，也无论结点关键字是有序还是无序，都适应顺序查找；其缺点是当n较大时，其查找成功的平均查找长度约为表长的一半（n+1）/2，查找失败则需要比较n+1次，查找效率低。
- 二分查找的速度快，效率高，查找成功的平均查找长度约为$\log_2(n+1)-1$，但是它要求表以**顺序存储表示并且是按关键字有序**，使用高效率的排序方法也要花费$O(n\log_2 n)$的时间。另外，当对表结点进行插入或删除时，需要移动大量的元素，所以二分查找适用于表不易变动且又经常查找的情况。
- 分块查找的优点是，在表中插入或删除一个记录时，只要找到该记录所属的块，就可以在该块内进行插入或删除运算。因为块内记录是无序的，所以插入或删除比较容易，无需移动大量记录。分块查找的主要缺点是需要增加一个辅助数组的存储空间和将初始表块排序的运算，它也不适宜用链式存储结构。
- 此外，根据平均查找长度，不难得到以上顺序查找、二分查找和分块查找三种查找算法的时间复杂度分别为：$O(n)$、$O(\log_2 n)$和$O(\sqrt{n})$。

真题演练

查找较快，且插入和删除操作也比较方便的查找方法是（　）。

A:分块查找

B:二分查找

C:顺序查找

D:折半查找

真题演练

查找较快，且插入和删除操作也比较方便的查找方法是（　）。

A:分块查找

B:二分查找

C:顺序查找

D:折半查找

答案：A

查找方法	**优点**	缺点
顺序查找	算法简单。 适用顺序结构和链式结构，结点关键字有序或无序都可。	查找效率低。
二分查找（或折半查找）	速度快，效率高。	只适用顺序存储且按关键字有序。 表结点进行插入或删除时，需要移动大量的元素。
分块查找	**插入或删除比较容易，无需移动大量记录。**	需要增加一个辅助数组的存储空间和将初始表块排序的运算。 只适用顺序存储。

真题演练

设有序表为{1,3,9,12,32,41,45,62,75,77,82}，采用二分查找法查找关键字75，查找过程中关键字之间的比较次数是（　）。

A:1

B:2

C:3

D:4

真题演练

设有序表为{1,3,9,12,32,41,45,62,75,77,82}，采用二分查找法查找关键字75，查找过程中关键字之间的比较次数是（　）。

A:1

B:2

C:3

D:4

答案：B

真题演练

下列线性表中，能使用二分查找的是（　）。

A:顺序存储(2,12,5,6,9,3,89,34,25)

B:链式存储(2,12,5,6,9,3,89,34,25)

C:顺序存储(2,3,5,6,9,12,25,34,89)

D:链式存储(2,3,5,6,9,12,25,34,89)

真题演练

下列线性表中，能使用二分查找的是（　）。

A:顺序存储(2,12,5,6,9,3,89,34,25)

B:链式存储(2,12,5,6,9,3,89,34,25)

C:顺序存储(2,3,5,6,9,12,25,34,89)

D:链式存储(2,3,5,6,9,12,25,34,89)

答案：C

顺序查找	适用顺序结构和链式结构，结点关键字有序或无序都可。
二分查找（或折半查找）	只适用**顺序存储**且按关键字**有序**。
分块查找	只适用顺序存储。

真题演练

线性表采用顺序存储或链式存储，对其进行查找的方法应是（　）。

A:顺序查找

B:二分查找

C:散列查找

D:索引查找

真题演练

线性表采用顺序存储或链式存储，对其进行查找的方法应是（　）。

A:顺序查找

B:二分查找

C:散列查找

D:索引查找

答案：A

二叉排序树(Binary Sort Tree, BST)又称二叉查找树，是一种特殊的二叉树，它或者是一棵空树，或者是具有下列性质的二叉树：

（1）若它的右子树非空，则右子树上所有结点的值均大于根结点的值。

（2）若它的左子树非空，则左子树上所有结点的值均小于根结点的值。

（3）左、右子树本身又各是一棵二叉排序树。

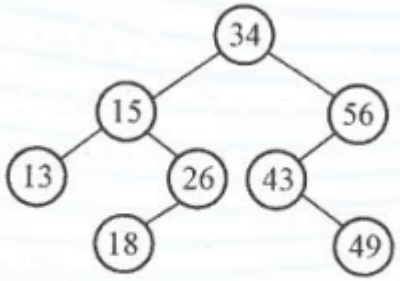


图 8.5 一棵二叉排序树

从上述性质可推出二叉排序树的另一个重要性质，即按中序遍历二叉排序树所得到 的遍历序列是一个递增有序序列。例如，图8.5所示就是一棵二叉排序树，树中每个结点的关键字都大于它左子树中所有结点的关键字，而小于它右子树中所有结点的关键字。 若对其进行中序遍历，得到的遍历序列为：13, 15, 18, 26, 34, 43, 49, 56。可见， 此序列是一个有序序列。

已知输入关键字序列为（35, 26, 53, 18, 32, 65），生成二叉排序树的过程如图8.6所示。

图 8.6　二叉排序树的构造过程

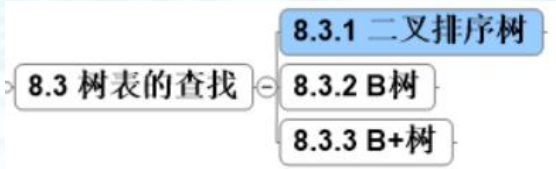
二叉排序树的平均查找长度为：　$ASL = \sum_{i=1}^{6} p_i c_i = (1+2\times2+3\times3)/6 \approx 2.3$

若输入关键字序列为（18，26，32，35，53，65），则生成的二叉排序树如图8.7所示。从上例可以看到，同样的一组关键字序列，由于其输入顺序不同，所得到的二叉排序树也有所不同。上面生成的两棵二叉排序树，一棵的深度是3，而另一棵的深度则为6，因此，含有n个结点的二叉排序树不是唯一的。

图 8.7　有序关键字的二叉排序树

一般情况下，构造二叉排序树的真正目的并不是为了排序，而是为了更好地查找。因此，通常称二叉排序树为二叉查找树。

$$ASL = \sum_{i=1}^{6} p_i c_i = (1+2+3+4+5+6)/6 = 3.5$$

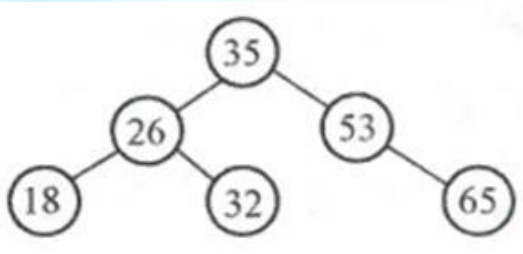


2. 二叉排序树上的查找

二叉排序树可看成一个有序表，所以在二叉排序树上查找与二分查找类似，也是一个逐步缩小查找范围的过程。根据二叉排序树的定义，查找其关键字等于给定值key的元素的过程为：若二叉排序树为空，则表明查找失败，应返回空指针。否则，若给定值key等于根结点的关键字，则表明查找成功，返回当前根结点指针；若给定值key小于根结点的关键字，则继续在根结点的左子树中查找，若给定值key大于根结点的关键字，则继续在根结点的右子树中查找。显然，这是一个递归的查找过程，其递归算法描述如下：

```
BSTNode * SearchBST(BSTree T, KeyType x)
{  //在二叉排序树上查找关键字值为X的结点
    if (T==NULL || T->key==x)
        return T;
    if(x<T->key)
        return SearchBST(T->lchild, x);
    else
        return SearchBST(T->rchild, x);
}
```

在二叉排序树上进行查找的过程中，给定值key与树中结点比较的次数最少为一次（即根结点就是待查的结点），最多为树的深度，所以平均查找次数要小于树的深度。若查找成功，则是从根结点出发走了一条从根到待查结点的路径；若查找不成功，则是从根结点出发走了一条从根结点到某个叶子的路径。同二分查找类似，与给定值的比较次数不会超过树的深度。若二叉排序树是一棵理想的平衡树或接近理想的平衡树，如图8.6 (g) 所示，则进行查找的时间复杂度为$O(\log_2 n)$;若退化为一棵单支树，如图8.8所示，则其查找的时间复杂度为$O(n)$。对于一般情况，其时间复杂度应为$O(\log_2 n)$。由此可知， 在二叉排序树上查找比在线性表上进行顺序查找的时间复杂度O(n)要好得多，这正是构造二叉排序树的主要目的之一。



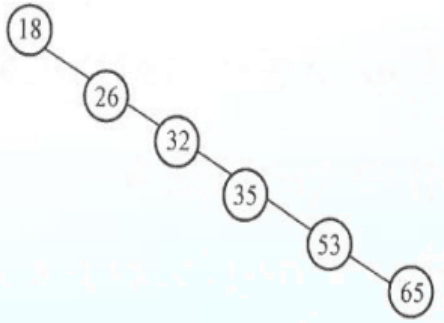
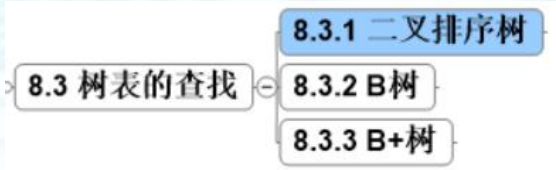


图 8.7　有序关键字的二叉排序树



3. 二叉排序树上的删除

从BST树上删除一个结点，仍然要保证删除后满足BST的性质。设被删除结点为p， 其父结点为f，如图8.8 (a)所示的BST树。具体删除情况分析如下：

（1）若p是叶子结点：直接删除p，如图8.8 (b)所示。

（2）若p只有一棵子树（左子树或右子树），直接用p的左子树（或右子树）取代p的位置而成为f的一棵子树。即原来p是f的左子树，则p的子树成为f的左子树；原来 p是f的右子树，则p的子树成为f的右子树，如图8.8 (c)所示。

（3）若p既有左子树又有右子树，处理方法有以下两种，可以任选其中一种。

①用p的直接前驱结点代替p，即从p的左子树中选择值最大的结点s放在p的位置 (用结点s的内容替换结点p内容)，然后删除结点s。s是p的左子树中最右边的结点且没有右子树，对s的删除同（2），如图8.8 (d)所示。

②用p的直接后继结点代替p，即从p的右子树中选择值最小的结点s放在p的位置（用结点s的内容替换结点p的内容），然后删除结点s。s是p的右子树中的最左边的结点且没有左子树，对s的删除同（2）。例如，对图8.8 (a)所示的二叉排序树，删除结点8后所得的结果如图8.8 (e)所示。

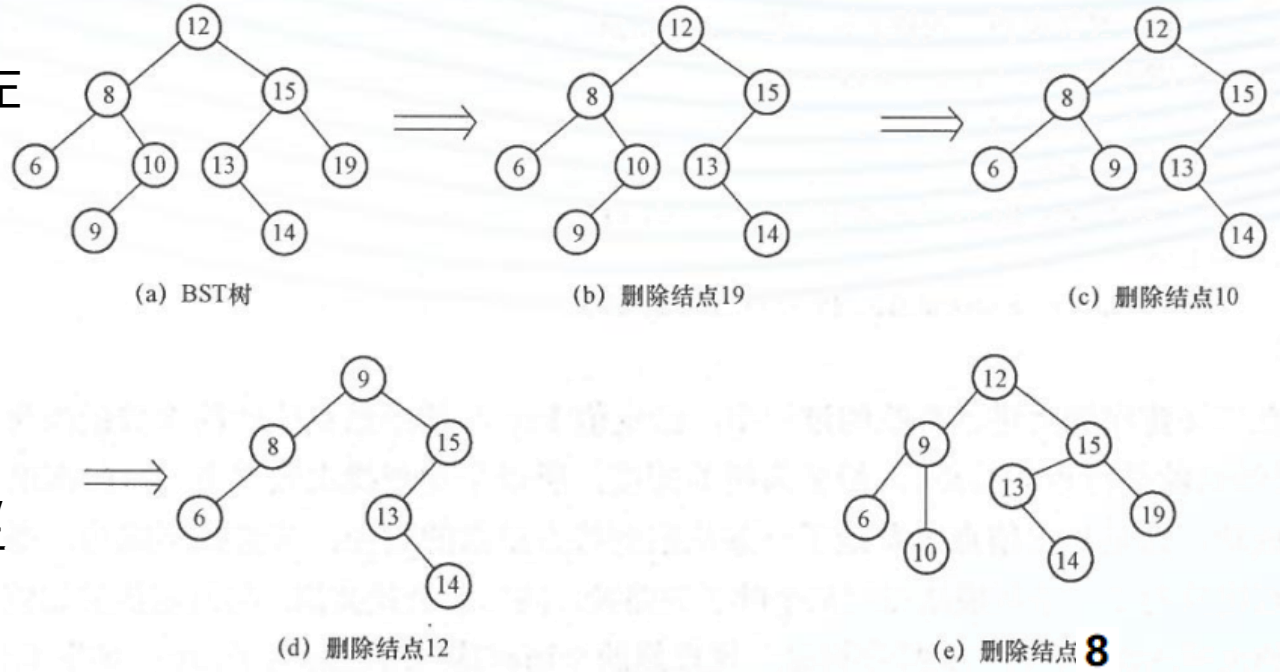


图 8.8 BST 树的结点删除情况

真题演练

将下列数据依次插入到初始为空的二叉排序树中，能得到高度最小的二叉排序树的序列是（　）。

A:2,4,7,5,8,10

B:5,1,2,6,3,4

C:6,4,1,8,10,5

D:9,7,2,1,4,0

真题演练

将下列数据依次插入到初始为空的二叉排序树中，能得到高度最小的二叉排序树的序列是（　）。

A:2,4,7,5,8,10

B:5,1,2,6,3,4

C:6,4,1,8,10,5

D:9,7,2,1,4,0

答案：C

真题演练

一棵二叉排序树中，关键字n所在结点是关键字m所在结点的孩子，则（　）。

A:n一定大于m

B:n一定小于m

C:n一定等于m

D:n与m的大小关系不确定

真题演练

一棵二叉排序树中，关键字n所在结点是关键字m所在结点的孩子，则（　）。

A:n一定大于m

B:n一定小于m

C:n一定等于m

D:n与m的大小关系不确定

答案：D

真题演练

分别用以下序列生成二叉排序树，其中三个序列生成的二叉排序树是相同的，不同的序列是（　）。

A:(4,1,2,3,5)

B:(4,2,3,1,5)

C:(4,5,2,1,3)

D:(4,2,1,5,3)

真题演练

分别用以下序列生成二叉排序树，其中三个序列生成的二叉排序树是相同的，不同的序列是（　）。

A:(4,1,2,3,5)

B:(4,2,3,1,5)

C:(4,5,2,1,3)

D:(4,2,1,5,3)

答案：A

真题演练

下列选项中，其平均查找性能与基于二叉排序树的查找相当的是（　）。

A:二分查找

B:顺序查找

C:分块查找

D:索引顺序查找

真题演练

下列选项中，其平均查找性能与基于二叉排序树的查找相当的是（ ）。

A:二分查找

B:顺序查找

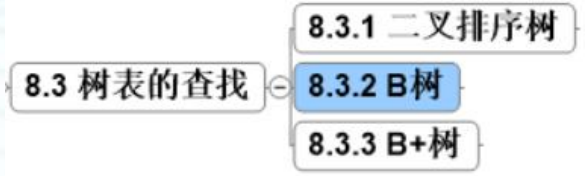
C:分块查找

D:索引顺序查找

答案：A

8.3.2二、B树

B树是一种平衡的多路查找树，它在文件系统中非常有用。



1．B 树的定义

一棵 m（$m\geqslant 3$）阶的 B 树，或为空树，或为满足下列性质的 m 叉树：

（1）每个结点至少包含下列信息域：

$$(n,p_0,k_1,p_1,k_2,\cdots,k_n,p_n)$$

其中，n 为关键字的个数；k_i（$1\leqslant i\leqslant n$）为关键字，且 $k_i<k_{i+1}$（$1\leqslant i\leqslant n-1$）；$p_i$（$0\leqslant i\leqslant n$）为指向子树根结点的指针，且 p_i 所指向子树中所有结点的关键字均小于 k_{i+1}，p_n 所指子树中所有结点关键字均大于 k_n；

（2）树中每个结点至多有 m 棵子树。

（3）若树为非空，则根结点至少有 1 个关键字，至多有 $m-1$ 个关键字。因此，若根结点不是叶子，则它至少有两棵子树。

（4）所有的叶结点都在同一层上，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，指向它们的指针均为空），叶子的层数为树的高度 h。

（5）每个非根结点中所包含的关键字个数满足：$\lceil m/2\rceil-1\leqslant n\leqslant m-1$。因为每个内部结点的度数正好是关键字总数加 1，所以，除根结点之外的所有非终端结点（非叶子结点的最下层的结点称为终端结点）至少有$\lceil m/2\rceil$棵子树，至多有 m 棵子树。

例如，如图 8.10 所示为一棵 4 阶的 B 树，其深度为 3。当然，同二叉排序树一样，关键字插入的次序不同，将可能生成不同结构的 B 树。该树共三层，所有叶子结点均在第三层上。在一棵 4 阶的 B 树中，每个结点的关键字个数最少为$\lceil m/2\rceil-1=\lceil 4/2\rceil-1=1$，最多为 $m-1=4-1=3$；每个结点的子树数目最少为$\lceil m/2\rceil=\lceil 4/2\rceil=2$，最多为 $m=4$。

8.3.2二、B树

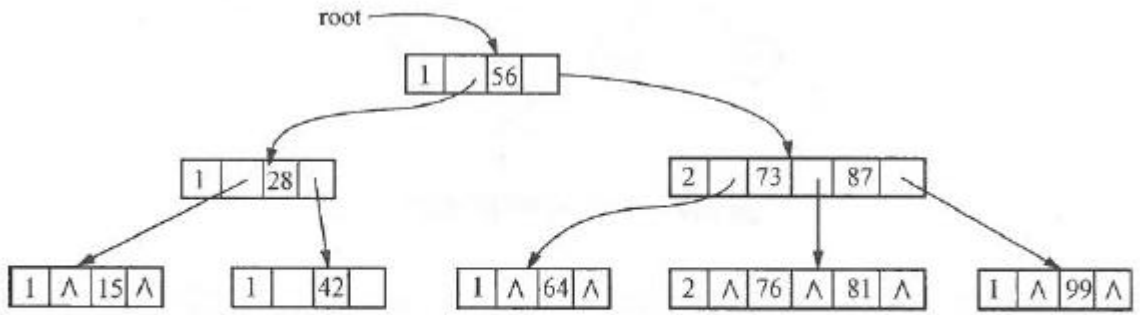


图 8.10　一棵 4 阶的 B 树

B 树的结点类型定义如下：

```
#define m 10                    // m 为 B 树的阶,结点中关键字最多可有 m-1 个
typedef struct node {
    int     keynum;             //结点中关键字个数,即结点的大小
    KeyType key[m];             //关键字向量,key[0]不用
    struct * parent;            //指向双亲结点
    struct node *ptr[m];        //子树指针向量
} BTNode;
typedef BTNode *BTree;
```

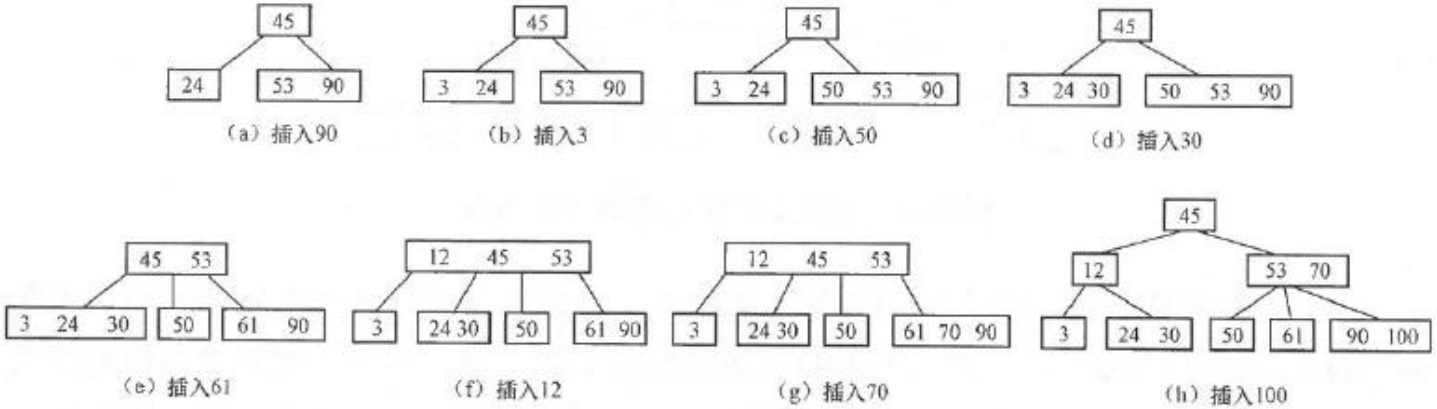


图 8.11　B 树生成过程图

真题演练

下列叙述中，不符合m阶B树定义的是（　）。

A:根结点可以只有一个关键字

B:所有叶结点都必须在同一层上

C:每个结点内最多有m棵子树

D:每个结点内最多有m个关键字

真题演练

下列叙述中，不符合m阶B树定义的是（　）。

A:根结点可以只有一个关键字

B:所有叶结点都必须在同一层上

C:每个结点内最多有m棵子树

D:每个结点内最多有m个关键字

答案：D

真题演练

下列关于m阶B树的叙述中，错误的是（　）。

A:根结点至多有m棵子树

B:所有叶子都在同一层次上

C:每个非根内部结点至少有⌈m/2⌉棵子树

D:结点内部的关键字可以是无序的

真题演练

下列关于m阶B树的叙述中，错误的是（　）。

A:根结点至多有m棵子树

B:所有叶子都在同一层次上

C:每个非根内部结点至少有⌈m/2⌉棵子树

D:结点内部的关键字可以是无序的

答案：A

真题演练

下列叙述中，不符合m阶B树定义的是（　）。

A:根结点最多有m棵子树

B:所有叶结点都在同一层上

C:各结点内关键字均升序或降序排列

D:叶结点之间通过指针链接

真题演练

下列叙述中，不符合m阶B树定义的是（　）。

A:根结点最多有m棵子树

B:所有叶结点都在同一层上

C:各结点内关键字均升序或降序排列

D:叶结点之间通过指针链接

答案：D

真题演练

下列关于m阶B树的叙述中，错误的是（　）。

A:每个结点至多有m个关键字

B:每个结点至多有m棵子树

C:插入关键字时，通过结点分裂使树高增加

D:删除关键字时通过结点合并使树高降低

真题演练

下列关于m阶B树的叙述中，错误的是（　）。

A:每个结点至多有m个关键字

B:每个结点至多有m棵子树

C:插入关键字时，通过结点分裂使树高增加

D:删除关键字时通过结点合并使树高降低

答案：A

8.3.3三、B+树

B^+树是一种常用于文件组织的B树的变形树。一棵 m 阶的 B^+树和 m 阶的B树的差异在于：

（1）有 k 个孩子的结点必含有 k 个关键字。

（2）所有的叶结点中包含了关键字的信息及指向相应结点的指针，且叶子结点本身依照关键字的大小自小到大顺序链接。

（3）所有非终端结点可看成是索引部分，结点中仅含有其子树（根结点）中的最大关键字（或最小）关键字。

例如，图8.18所示是一棵3阶的 B^+树。通常在 B^+树上有两个头指针root和sqt，前者指向根结点，后者指向关键字最小的叶子结点。因此，可以对 B^+树进行两种查找运算：一种是从最小关键字起进行顺序查找，另一种是从根结点开始进行随机查找。

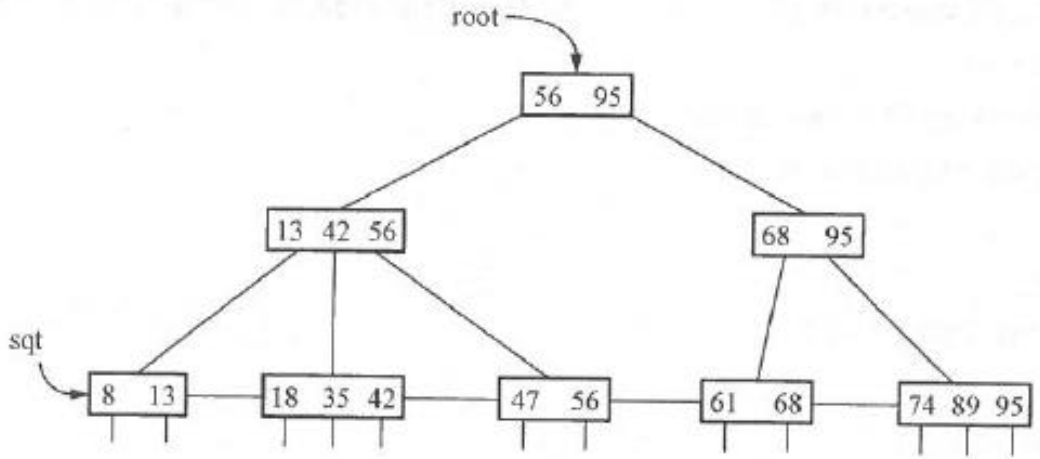


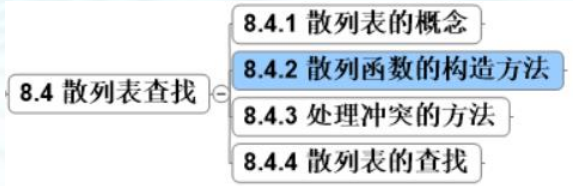
图 8.18　一棵3阶的 B^+树

散列（Hash）同顺序、链式和索引存储结构一样，是存储线性表的又一种方法。散列存储的基本思想是：以线性表中的每个元素的关键字key为自变量，通过一种函数H (key) 计算出函数值，把这个函数值解释为一块连续存储空间的单元地址（即下标），将该元素存储到这个单元中。散列存储中使用的函数H (key)称为散列函数或哈希函数，它实现关键字到存储地址的映射（或称转换）。H (key)的值称为散列地址或哈希地址，使用的数组空间是线性表进行散列存储的地址空间，所以被称之为散列表或哈希表。当在散列表上进行查找时，首先根据给定的关键字key，用与散列存储时使用的同一散列函数H(key) 计算出散列地址，然后按此地址从散列表中取对应的元素。

例如，有个线性表A= (31, 62, 74, 36, 49, 77)，其中每个整数可以是元素本身，也可以仅是元素的关键字。为了散列存储该线性表，假设选取的散列函数为

H (key) =key % m　　取m=11，表长也为11

散列地址（下标）	0	1	2	3	4	5	6	7	8	9	10
	77			36		49		62	74	31	



1. 直接地址法

直接地址法是以关键字key本身或关键字加上某个常量C作为散列地址的方法。对应的散列函数H (key)为

H (key) =key+C

在使用时，为了使散列地址与存储空间吻合，可以调整C。这种方法计算简单，并且没有冲突。它适合于关键字的分布基本连续的情况，若关键字分布不连续，空号较多，将会造成较大的空间浪费。

8.4.2 散列函数的构造方法

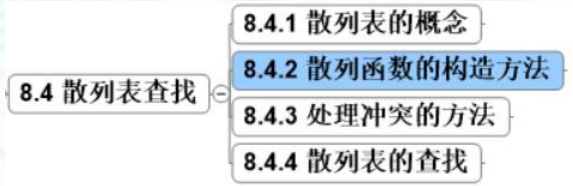
2. 数字分析法

数字分析法是假设有一组关键字，每个关键字由 n 位数字组成，如 k_1，$k_2 \cdots k_n$。数字分析法是从中提取数字分布比较均匀的若干位作为散列地址。

例如，有一组有 6 位数字组成的关键字，如下表左边一列所示。

关键字	散列地址（0..99）	关键字	散列地址（0..99）
912356	13	892556	95
952456	54	⋮	⋮
964852	68	872265	72
982166	81		

分析这一组关键字会发现，第1、3、5和6位数字分布不均匀，第1位数字全是9或8，第3位基本上都是2，第5、6两位上也都是5和6，故这4位不可取。而第2、4两位数字分布比较均匀，因此可取关键字中第2、4两位的组合作为散列地址。



3. 除余数法

除余数法是选择一个适当的p（p≤散列表长m）去除关键字k, 所得余数作为散列地址的方法。对应的散列函数为H（k）即：分析这一组关键字会发现，第1、3、5和6位数字分布不均匀，第1位数字全是9或8，第3位基本上都是2，第5、6两位上也都是5和6，故这4位不可取。而第2、4两位数字分布比较均匀，因此可取关键字中第2、4两位的组合作为散列地址。

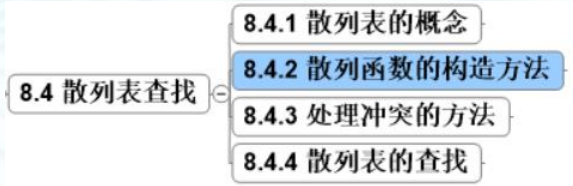
H（k）=k%p

其中，p最好选取小于或等于表长m的最大素数。如表长为20，那么p选19；若表长为25，则p可选23，... 。表长m与模p的关系可按下表对应：

m=8, 16, 32, 64, 128, 256, 512, 1024. ...

p=7, 13, 31, 61, 127, 251, 503, 1019, ...

这是一种最简单也最常用的一种散列函数构造方法，在第8.4.1中已经使用过。



4. 平方取中法

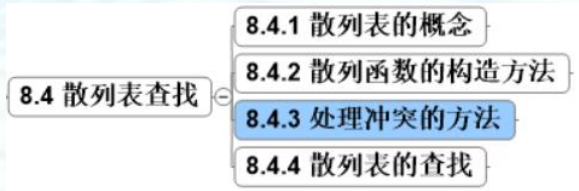
平方取中法是取关键字平方的中间几位作为散列地址的方法，因为一个乘积的中间几位和乘数的每一位都相关，故由此产生的散列地址较为均匀，具体取多少位视实际情况而定。例如有一组关键字集合（0100, 0110, 0111, 1001, 1010, 1110），平方之后得到新的数据集合（0010000, 0012100, 0012321, 1002001. 1020100, 123210）。那么，若表长为1000，则可取其中第3、4和5位作为对应的散列地址（100, 121, 123, 020, 201, 321）。

5. 折叠法

折叠法是首先把关键字分割成位数相同的几段（最后一段的位数可少一些），段的位数取决于散列地址的位数，由实际情况而定，然后将它们的叠加和（舍去最高进位）作为散列地址的方法。

折叠法又分移位叠加和边界叠加。移位叠加是将各段的最低位对齐，然后相加；边界叠加则是将两个相邻的段沿边界来回折叠，然后对齐相加。

例如，关键字k=98 123 658，散列地址为3位，则将关键字从左到右每三位一段进行划分，得到的三个段为981、236和58，叠加后值为1275，取低3位275作为关键字 98 123 658的元素的散列地址；如若用边界叠加，即为981、632和58叠加后其值为1671， 取低3位得671作为散列地址。

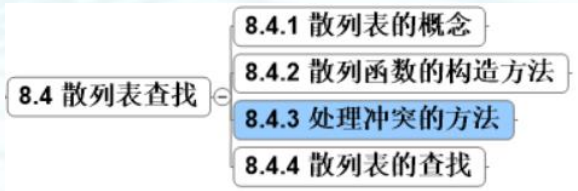


1. 开放定址法

开放定址法又分为**线性探插法、二次探查法和双重散列法**。开放定址法解决冲突的基本思想是使用某种方法在散列表中形成一个探查序列，沿着此序列逐个单元进打查找，直到找到一个空闲的单元时将新结点存入其中。假设散列表空间为T[0..m-1]，散列函数为H (key)，开放定址法的一般形式为：

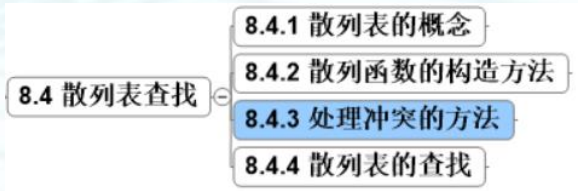
$$h_i = (H(key) + d_i)\%m \quad 0 \le i \le m-1$$

其中 d_i 为增量序列，m 为散列表长。$h_0 = H(key)$ 为初始探查地址（假设 $d_0 = 0$），后续的探查地址依次是 $h_1, h_2, \ldots, h_{m-1}$。



（1）线性探查法

其基本思想是：将散列表T[0..m-1]看成是一个循环向量，若初始探查的地址为d（即H（key）=d），则后续探查地址的序列为：d+1，d+2，…，m-1，0，1，…，d-1。也就是说，探查时从地址d开始，首先探查T[d]，然后依次探查T[d+1]，…，T[m-1]，此后又循环到T[0]，T[1]，…，T[d-1]。分两种情况分析：一种运算是插入，若当前探查单位为空，则将关键字key写入空单元，若不空，则继续后续地址探查，直到遇到空单元插入关键字，若探查到T[d-1]时仍未发现空单元，则插入失败（表满）；另一种预算是查找，若当前探查单元中的关键字值等于key，则表示查找成功，若不等，则继续地址探查，若遇到单元中的关键字值等于key时，查找成功，若再探查T[d-1]单元时，仍未发现关键字值等于key，则查找失败。



（1）线性探查法

其基本思想是：将散列表T[0..m-1]看成是一个循环向量，若初始探查的地址为d（即H（key）=d），则后续探查地址的序列为：d+1，d+2，…，m-1，0，1，…，d-1。也就是说，探查时从地址d开始，首先探查T[d]，然后依次探查T[d+1]，…，T[m-1]，此后又循环到T[0]，T[1]，…，T[d-1]。分两种情况分析：一种运算是插入，若当前探查单位为空，则将关键字key写入空单元，若不空，则继续后续地址探查，直到遇到空单元插入关键字，若探查到T[d-1]时仍未发现空单元，则插入失败（表满）；另一种预算是查找，若当前探查单元中的关键字值等于key，则表示查找成功，若不等，则继续地址探查，若遇到单元中的关键字值等于key时，查找成功，若再探查T[d-1]单元时，仍未发现关键字值等于key，则查找失败。

（2）二次探查法。二次探查法的探查序列是：

$$hi = (H(key) \pm i^2)\%m (0 \le i \le m-1)$$

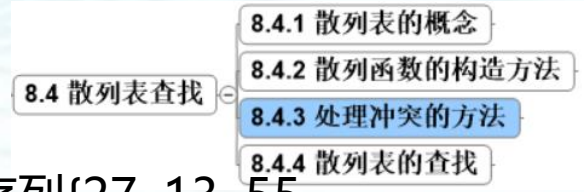
即探查序列为 $d = H(key), d+1^2, d-1^2, d+2^2, d-2^2, \ldots$ 等。也就是说，探查从地址 d 开始，先探查 T[d]，然后再依次探查 $T[d+1^2], T[d-1^2], T[d+2^2], T[d-2^2], \ldots$。

（3）双重散列法。双重散列法是几种方法中最好的方法，它的探查序列为

$$hi = (H(key) + i^* H_1(key))\%m (0 \le i \le m-1)$$

即探查序列为 $d = H(key), (d+1^* H_1(key))\%m, (d+1^* H_1(key))\%m, \ldots$ 等。

8.4.3 处理冲突的方法



【例8.6】设散列函数为h (key) =key % 11；散列地址表空间为0～10，对关键字序列{27, 13, 55, 32, 18, 49, 24, 38, 43}，利用线性探测法解决冲突，构造散列表。

解：首先根据散列函数计算散列地址

A (27)=5；　　h (13)=2；

h (55)=0；　　h (32) =10；

h (18)=7；　　h (49) =5；

h (24)=2；　　h (38)=5；

h (43)=10；　　(散列表各元素查找比较次数标注在结点的上方或下方)

下标（地址）	0	1	2	3	4	5	6	7	8	9	10
散列表 T[0..10]	55	43	13	24		27	49	18	38		32
探查次数	1	3	1	2		1	2	1	4		1

图 8.19 线性探测法构造散列表

ASL=(1X5+2X2+3X1+4X1) /9≈1.78

2. 拉链法（链地址法）

当存储结构是链表时，多采用拉链法。用拉链法处理冲突的办法是：把具有相同散列地址的关键字（同义词）值放在同一个单链表中，称为同义词链表。有m个散列地址就有m个链表，同时用指针数组T[0..m-1]存放各个链表的头指针，凡是散列地址为i的记录都以结点方式插入到以T[i]为指针的单链表中，T中个分量的初值应为空指针，例如，按例8.6所给的关键字序列，用拉链法构造散列表如图8.20所示。

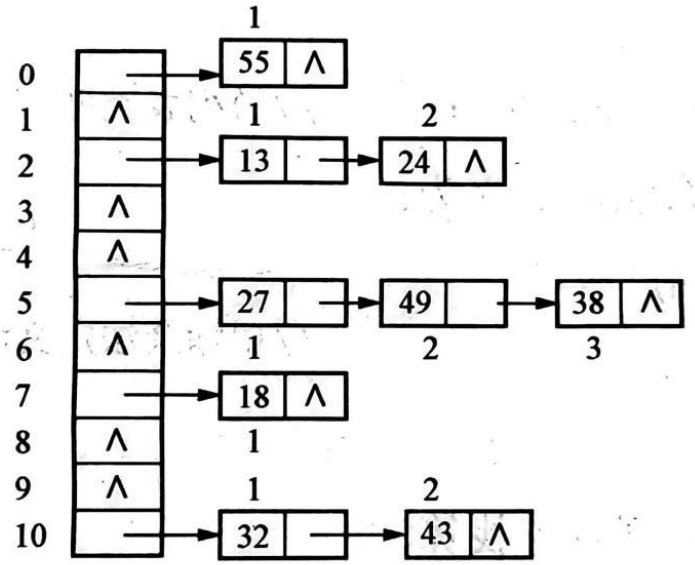


图 8.20　拉链法构造散列表

ASL=(1X5+2X3+3X1) /9≈1.55

8.4.4四、散列表的查找

【例 8.7】 设散列函数 $f(k)=k\%13$，散列表地址空间为 0～12，对给定的关键字序列（19，14，01，68，20，84，27，26，50，36）分别以拉链法和线性探查法解决冲突构造散列表，画出所构造的散列表，指出在这两个散列表中查找每一个关键字时进行比较的次数，并分析在等概率情况下查找成功和不成功时的平均查找长度以及当结点数为 $n=10$ 时的顺序查找和二分查找成功与不成功的情况。

分析：

（1）用线性探测法解决冲突，其散列表如图 8.21 所示。

下标（地址）	0	1	2	3	4	5	6	7	8	9	10	11	12
散列表T[0..12]	26	14	01	68	27		19	20	84		36	50	
探查次数	1	1	2	1	4		1	1	3		1	1	

图 8.21　线性探查法建立的散列表

因此，在该表上的平均查找长度为

$$ASL=(1+1+2+1+4+1+1+3+1+1)/10=1.6$$

8.4.4 散列表的查找

（2）用拉链法解决冲突，构造的散列表如图 8.22 所示。

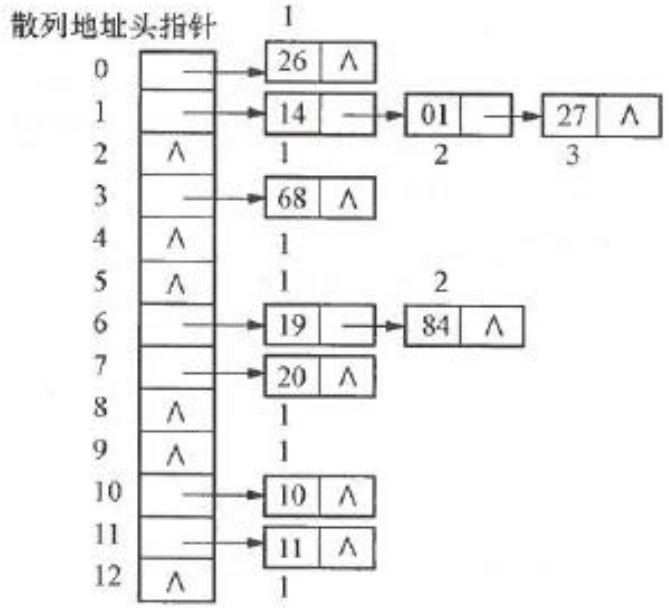


图 8.22　拉链法建立的散列表

在该散列表上的平均查找长度为

$$ASL=(1\times7+2\times2+3\times1)/10=1.4$$

而当 $n=10$ 时，顺序查找和二分查找的平均长度分别为

$$ASL_{seq}=(10+1)/2=5.5$$

$$ASL_{bin}=(1\times1+2\times2+3\times4+4\times3)/10\approx3$$

对于查找不成功的情况，顺序查找和二分查找所需要进行的关键字仅取决于表长，而散列表查找所需要进行的比较次数和待查结点有关。

8.4.4四、散列表的查找

而散列表查找所需要进行的比较次数和待查结点有关。因此，在等概率情况下，也可以将散列表在查找不成功时对关键字需要执行的平均比较次数，定义为查找不成功时的平均查找长度。在图 8.21 所示的线性探查法中，假设所查的关键字 K 不在散列表中，若 $h(K)=0$，则必须依次在表 T[0..5] 中的关键字 K 或空值进行比较之后才遇到 T[5] 为空，即比较次数为 6；若 $h(K)=1$，则需要比较 5 次才能确定查找不成功。类似地，对 $h(K)=2$，3，4，5 进行分析，其比较次数分别为：4，3，2，1。若 $h(K)=6$，7，8，9 时，则类似的需要比较次数分别为：4，3，2，1；而 $h(K)=10$，11，12 时，需要比较次数分别为 3，2，1 次才能确定查找不成功，所以查找不成功时的平均查找长度为

$$ASL=(6+5+4+3+2+1+4+3+2+1+3+2+1)/13\approx 2.85$$

请注意，在计算查找成功的平均查找长度时，除数是结点的个数，而在计算查找不成功的平均查找长度时，除数却是表长。因此，同样的一组关键字对应的散列表，因表长不同，其查找成功和查找不成功时的平均查找长度都是不同的。

另外，在拉链法建立的散列表中，若待查关键字 K 的散列地址为 $d=h(K)$，且第 d 个链表上具有 i 个结点，则当 K 不在链表上时，就需要做 K 次关键字比较（不包括空指针比较），因此查找不成功时的平均查找长度为

$$ASL=(1+3+0+1+0+0+2+1+0+0+1+1+0)/13\approx 0.7$$

散列表的平均查找长度要比顺序查找小得多，比二分查找也小。

真题演练

假设散列表长m=11，散列函数H(key)=key%11。表中已有4个结点：H(39)=6，H(41)=8，H(53)=9，H(76)=10，占了4个位置，其余位置为空。现采用线性探查法处理冲突，存储关键字85时需要探查的次数是（　）。

A:2

B:3

C:4

D:5

真题演练

假设散列表长m=11，散列函数H(key)=key%11。表中已有4个结点：H(39)=6，H(41)=8，H(53)=9，H(76)=10，占了4个位置，其余位置为空。现采用线性探查法处理冲突，存储关键字85时需要探查的次数是（　）。

A:2

B:3

C:4

D:5

答案：C

真题演练

在下列查找方法中，平均查找长度与结点数量无直接关系的是（　）。

A:顺序查找

B:分块查找

C:散列查找

D:基于B树的查找

真题演练

在下列查找方法中，平均查找长度与结点数量无直接关系的是（　）。

A:顺序查找

B:分块查找

C:散列查找

D:基于B树的查找

答案：C

真题演练

设散列表长m=16，散列函数H(key)=key%15。表中已保存4个关键字：addr(18)=3，addr(35)=5，addr(51)=6，addr(22)=7，其余地址均为开放地址。存储关键字36时存在冲突，采用线性探测法来处理。则查找关键字36时的探查次数是（　）。

A:1

B:2

C:3

D:4

真题演练

设散列表长m=16，散列函数H(key)=key%15。表中已保存4个关键字：addr(18)=3，addr(35)=5，addr(51)=6，addr(22)=7，其余地址均为开放地址。存储关键字36时存在冲突，采用线性探测法来处理。则查找关键字36时的探查次数是（　）。

A:1

B:2

C:3

D:4

答案：C

本章总结


第8章 查找
8.1 基本概念
8.2 顺序表的查找
8.2.1 顺序查找
8.2.2 二分查找
8.2.3 索引顺序查找
8.2.4 三种查找方法的比较
8.3 树表的查找
8.3.1 二叉排序树
8.3.2 B树
8.3.3 B+树
8.4 散列表查找
8.4.1 散列表的概念
8.4.2 散列函数的构造方法
8.4.3 处理冲突的方法
8.4.4 散列表的查找

祝大家顺利通过考试！