第九章 查找

9.1 静态查找表

9.2 动态查找表

9.3 哈希表

9. 查找表

- 查找表(Search Table)：是由同一类型的数据元素(记录)构成的集合，以查找为核心运算的数据结构，每个元素通常由若干数据项构成。
- 查找表的两种基本形式
  - 静态查找表：查找表不变
  - 动态查找表：查找表可能变化

9. 查找表

- **静态查找表(Static Search Table)**：只作静态查找操作的查找表。主要操作有：
  - 查询某个“特定的”数据元素是否在查找表中
  - 检索某个“特定的”数据元素和各种属性。
- **动态查找表(Dynamic Search Table)**：动态表的特点是表结构本身是在查找过程中动态生成的。同时在查找过程中插入查找表中不存在的数据元素，或者从查找表中删除已经存在的某个数据元素。主要操作有：
  - 查找时插入数据元素
  - 查找时删除数据元素

9. 查找表

- 关键字(Key，码)：数据元素中某个(或几个)数据项的值，它可以标识一个数据元素。
  - 若关键字能唯一标识一个数据元素，则关键字称为主关键字；
  - 将能标识若干个数据元素的关键字称为次关键字。
- 查找/检索(Searching)：根据给定的K值，在查找表中确定一个关键字等于给定值的记录或数据元素。
  - 查找表中存在满足条件的记录：查找成功；输出结果：所查到的记录信息或记录在查找表中的位置。
  - 查找表中不存在满足条件的记录：查找失败；输出结果：空记录或空指针。

9. 查找表

- 查找的过程是依赖于“特定的”数据元素在查找表中的位置
- 查找的方法取决于查找表中数据元素的组织方式

查找表是记录的集合，而集合中的元素之间是一种完全松散的关系，因此，查找表是一种非常灵活的数据结构，可以用多种方式来存储表示。

9. 查找表

- 静态表：
  - 顺序表
  - 有序表（折半查找）
  - 索引顺序表
  - 插值查找、斐波那契查找
- 动态表：
  - 二叉排序树
  - 平衡二叉树
  - B树
  - 散列表

9. 查找表

- 查找方法评价指标
  - **平均查找长度ASL (Average Search Length)**：查找过程中和给定值比较的关键字个数的期望值。

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

其中， n为查找表中记录个数，C_i为查找第i个记录需要进行比较的次数

$$\sum_{i=1}^{n} P_i = 1$$

P_i：查找第i个记录的概率，一般认为查找每个记录的概率相等，即$P_i=1/n$；

  - 查找过程中主要操作是关键字的比较，ASL是衡量一个查找算法效率高低的标准。

9.1 静态查找表

- 顺序表的查找
  - 静态查找表的抽象数据类型定义如下：

ADT Static_SearchTable{

数据对象**D**：**D**是具有相同特性的数据元素的集合，各个数据元素有唯一标识的关键字。

数据关系**R**：数据元素同属于一个集合。

基本操作**P**：

Search(ST, key);

⋮

} ADT Static_SearchTable

9.1 静态查找表

- 顺序表和链表的查找：将给定的K值与查找表中记录的关键字逐个进行比较， 找到要查找的记录；
- 索引查找表的查找：首先根据索引确定待查找记录所在的块，然后再从块中找到要查找的记录。

9.1 静态查找表

顺序表的查找

顺序查找方法是以顺序表或链表表示静态查找表，从表的一端（第一个或最后一个记录）开始逐个将记录的关键字和给定K值进行比较，

- ✓ 若某个记录的关键字和给定K值相等，查找成功；
- ✓ 否则，若扫描完整个表，仍然没有找到相应的记录，则查找失败。

9.1 静态查找表

顺序表的查找

具体步骤：

1. 从表中最后一个记录开始；
2. 逐个进行记录的关键字和给定值的比较：
   - 若某个记录关键字比较相等，则查找成功；
   - 若直到第1个记录都比较不等，则查找不成功。

9.1 静态查找表

顺序表的查找

算法实现

- 无“哨兵”算法

```
// 顺序查找，ST为顺序表，key为要查找的关键字

int SeqSearch( SSTable ST, KeyType key) {
     for (int i = 0; i < ST.length; i++)
     { if ( ST.elem [i] == key)  return i; } //查找成功则返回i
     return -1; //查找不成功，返回-1
}
```

9.1 静态查找表

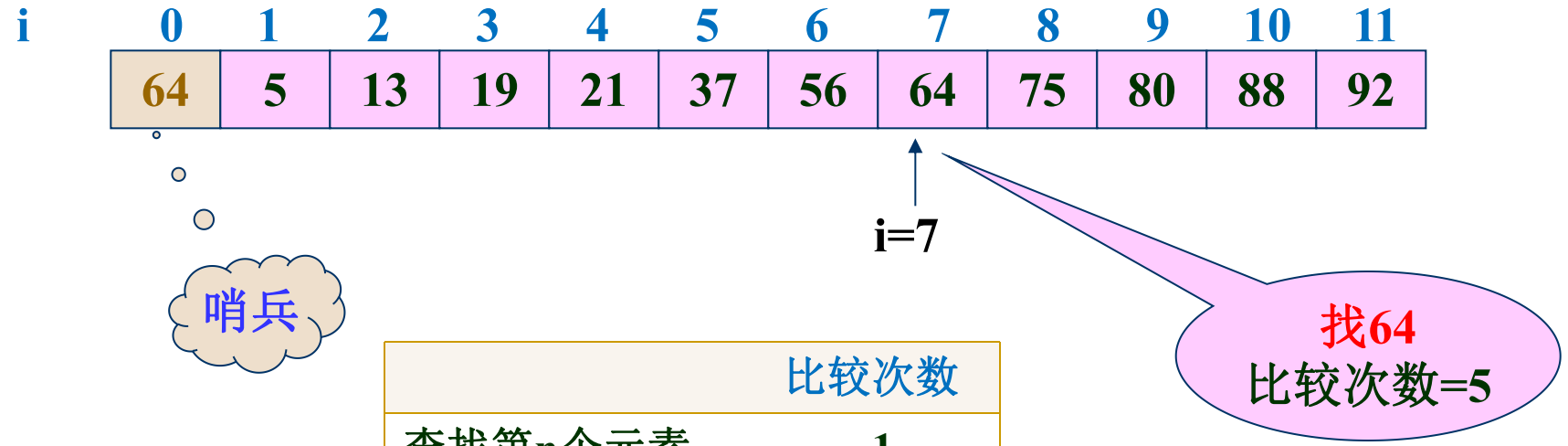
- 顺序表的查找
  - 带“哨兵”算法

```
int Search_Seq(SSTable ST, KeyType key) {
    //设置数组的下标为[0]的元素为“哨兵”
    ST.elem[0].key = key;
    // 从顺序表尾部开始从后往前找,若查找成功，返回位置i
    for ( i=ST.length; ST.elem[i].key!=key; --i) ;
    // 若查找不成功，i=0
    return i;
} // Search_Seq
```

※ 设置“哨兵”的目的是省略对下标越界的检查，提高算法执行速度

9.1 静态查找表

- 顺序表的查找
- 举例



	比较次数
查找第n个元素：	1
查找第n-1个元素：	2
查找第1个元素：	n
查找第i个元素：	n-i+1
查找失败：	n+1

9.1 静态查找表

- 顺序表的查找
- 算法性能分析

一般设查找每个记录成功的概率相等

- 查找成功时，$P_i=1/n$, 查找第i个元素成功的比较次数$C_i=n-i+1$，

$$ASL=n*P_1+(n-1)P_2+\ldots+2P_{n-1}+P_n$$

$$=\sum_{i=1}^{n}P_i\times C_i = \frac{1}{n}\sum_{i=1}^{n}(n-i+1) = \frac{n+1}{2}$$

- 查找不成功时：查找失败的比较次数为n+1，若成功与不成功的概率相等，对每个记录的查找概率为$P_i=1/(2n)$，则

$$\mathbf{ASL}=\sum_{i=1}^{n}P_i\times C_i = \frac{1}{2n}\sum_{i=1}^{n}(n-i+1)+\frac{n+1}{2}=3(n+1)/4$$

9.1 静态查找表

- 顺序表的查找
  - 优点

    简单，适应面广(对表的结构无任何要求)

  - 缺点

    平均查找长度较大，特别是当n很大时，查找效率很低

练习

一. 已知初始数列为33、66、22、88、11、27、44、55，采用带哨兵的顺序查找法，请写出数据22、11、99的比较次数。

9.1 静态查找表

- 折半查找
- 折半查找（又称为二分查找）算法是有序表的查找方法
  - **前提**：
    - 表中的记录必须是关键字有序（通常从小到大有序）；
    - 表必须采用顺序存储。
  - **策略**：

    逐步缩小(一半)范围直到找(不)到该记录为止

9.1 静态查找表

折半查找

步骤：

在有序表中，先确定待查记录所在的范围(前半部分或后半部分)，取中间记录作为比较对象，

（1）若给定值与中间记录的关键字相等，则查找成功；

（2）若给定值小于中间记录的关键字，则在中间记录的前半区继续查找；

（3）若给定值大于中间记录的关键字，则在中间记录的后半区继续查找。

不断重复上述过程，直到查找成功，或所有查找区域无记录，查找失败为止。

9.1 静态查找表

- 折半查找
- 算法流程：

1. n个对象从小到大存放在有序顺序表ST中，k为给定值
2. 设low、high指向待查元素所在区间的下界、上界，即low=1, high=n
3. 设mid指向待查区间的中点，即 mid=⌊(low+high)/2⌋
4. 比较中间位置记录的关键字与给定的k值
   ① k=ST[mid].key ： 查找成功
   ② k<ST[mid].key ：待查记录在[上半区间]，修改上界指针：High=Mid-1
   ③ k>ST[mid].key ：待查记录在[下半区间]，修改下界指针：Low=Mid+1
5. 重复3,4操作，直至越界(low>high)时，查找失败。

9.1 静态查找表

折半查找

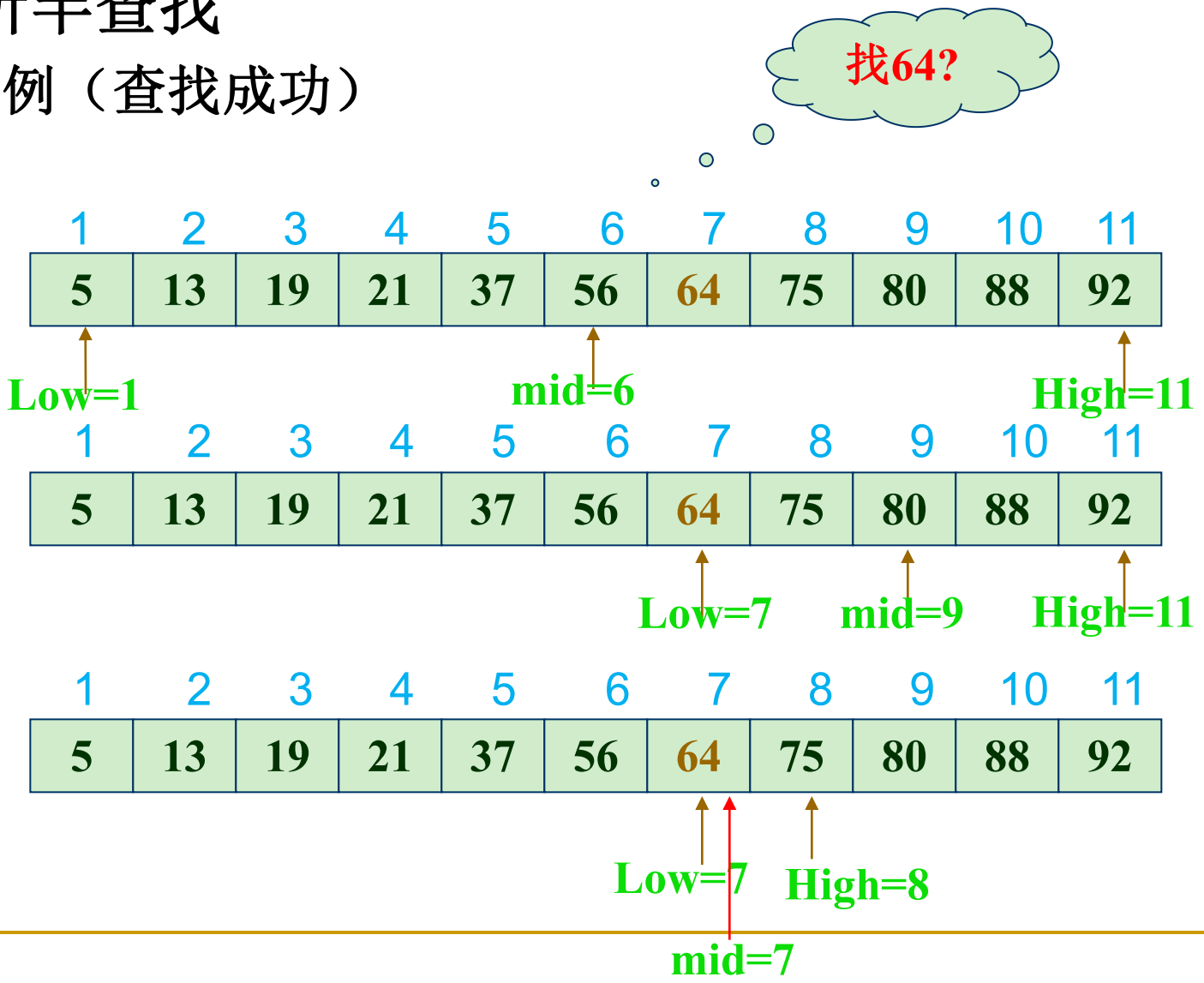
算法实现

```
int Search_Bin ( SSTable ST, KeyType key ) {
// 若找到，则返回该记录在表中的位置，否则为0。
   int low, high, mid;
   low = 1;  high = ST.length;     // 置区间初值
   while (low <= high) {
      mid = (low + high) / 2;
      if (EQ(key , ST.elem[mid].key)) return mid;  // 找到待查元素
      else if (LT(key, ST.elem[mid].key)) high = mid - 1;
                                   // 继续在前半区间进行查找
      else low = mid + 1;          // 继续在后半区间进行查找
   }
   return 0;                       // 顺序表中不存在待查元素
} // Search_Bin
```

9.1 静态查找表

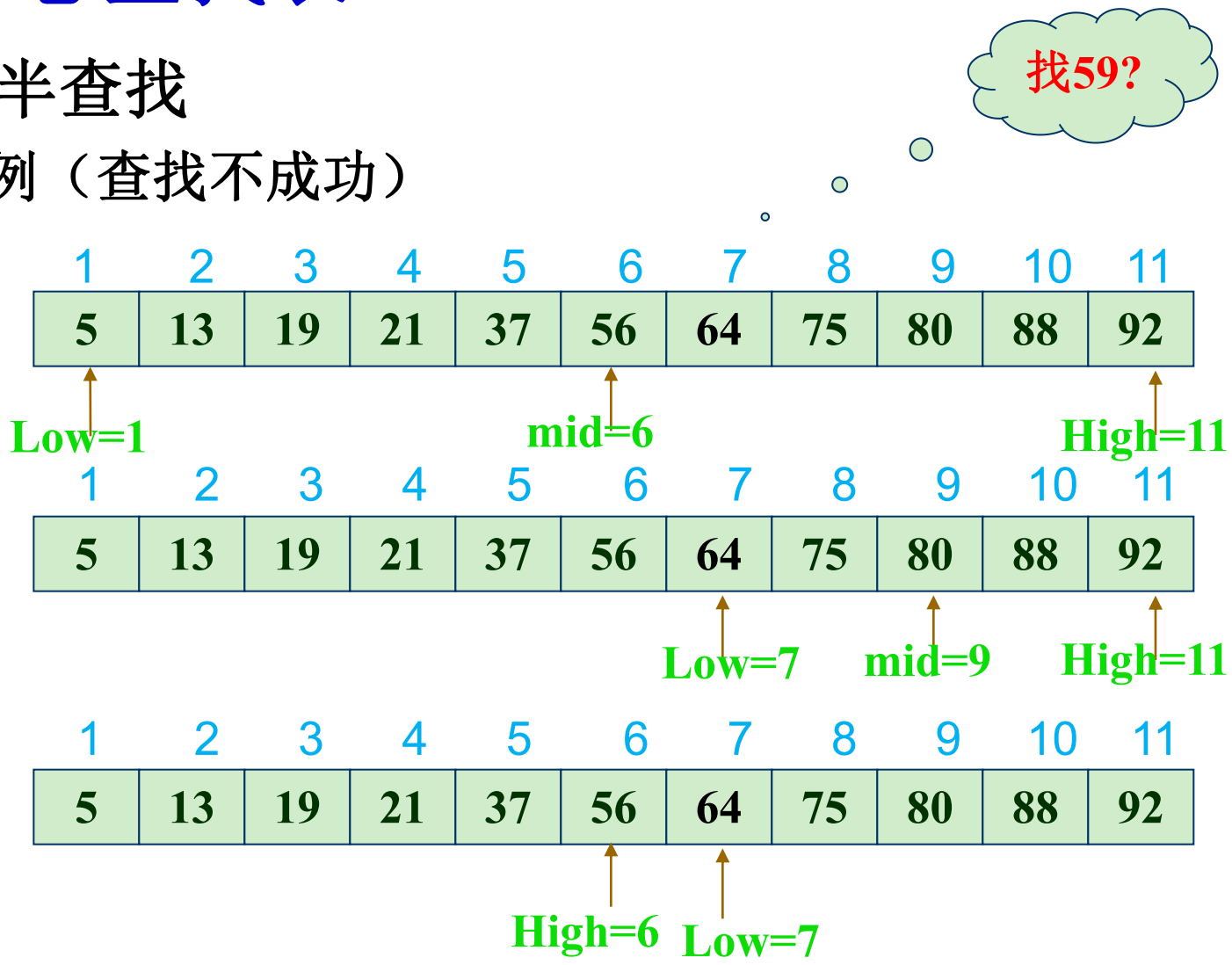
四. 折半查找

- 举例（查找成功）



9.1 静态查找表

- 折半查找
- 举例（查找不成功）



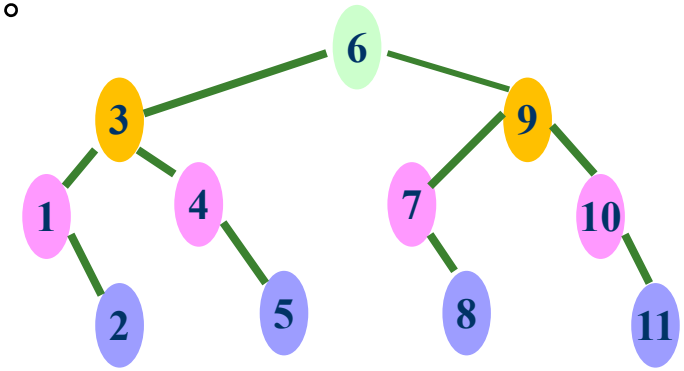
- 当下界low>high时，说明有序表中没有关键字等于K的元素，查找不成功

9.1 静态查找表

- 折半查找
- 查找时每经过一次比较，查找范围就缩小一半，该过程可用一棵二叉树表示，树中每个结点表示一个记录，结点的值是该记录在表中的位置：
  - 根结点就是第一次进行比较的中间位置的记录；
  - 排在中间位置前面的记录作为左子树的结点；
  - 排在中间位置后面的记录作为右子树的结点；
- 这样得到的二叉树称为判定树(Decision Tree)。

9.1 静态查找表

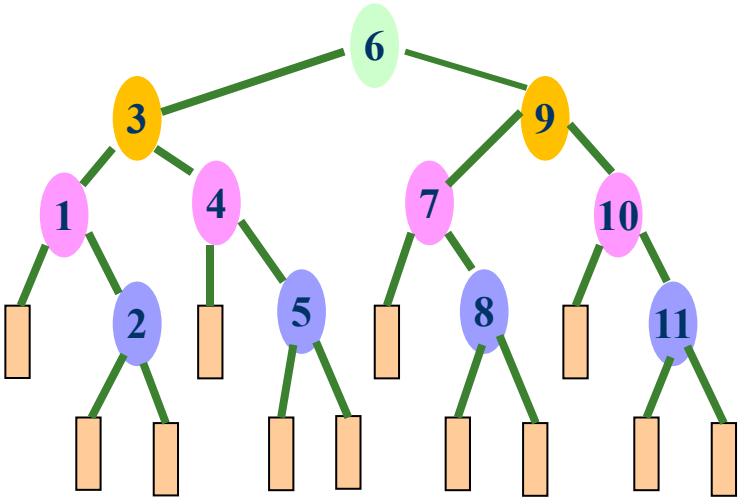
- 折半查找
- 判定树——描述查找过程的二叉树
  - 找到有序表中任一记录的过程就是走了一条从根结点到与该纪录相应的结点的路径，查找的比较次数就是该结点在判定树上的层次数。



位置 i	1	2	3	4	5	6	7	8	9	10	11
比较次数	3	4	2	3	4	1	3	4	2	3	4

9.1 静态查找表

- 折半查找
- 判定树——描述查找过程的二叉树
  - 有n个结点的判定树的深度为$\lfloor \log_2 n \rfloor + 1$，即折半查找法在查找过程中进行的比较次数最多不超过$\lfloor \log_2 n \rfloor + 1$



9.1 静态查找表

四. 折半查找

- 算法性能分析
  - 设有序表的长度n=2^h-1（即$h=\log_2(n+1)$），则描述折半查找的判定树是深度为h的满二叉树
  - 树中层次为1的结点有1个，层次为2的结点有2个，层次为h的结点有2^{h-1}个
  - 假设表中每个记录的查找概率相等，则查找成功时折半查找的平均查找长度：

$$ASL_{bs}=\frac{1}{n}\sum_{i=1}^{n}C_i=\frac{1}{n}\left[\sum_{j=1}^{h}j\times 2^{j-1}\right]=\frac{n+1}{n}\log_2(n+1)-1$$

$$ASL_{bs}\approx\log_2(n+1)-1 \qquad \text{当}n>50\text{时}$$

9.1 静态查找表

四. 折半查找

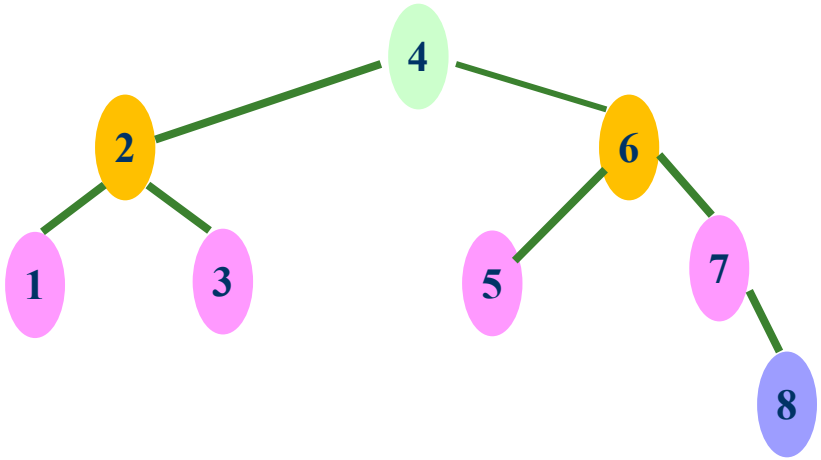
- 算法特点
  - 折半查找只适用于有序表，并且以顺序存储结构存储
  - 折半查找的效率比顺序查找高(特别是在静态查找表的长度很长时)

二分查找把静态有序查找分成了两棵子树，即查找结果只需要找其中的一半数据记录即可，然后继续折半查找。最坏情况下查找到关键字或查找失败的次数是$\lfloor \log_2 n \rfloor +1$，最好的情况是1次，因此二分查找的时间复杂度为$O(\log_2 n)$ 。

练习

一. 已知初始数列为11、22、27、33、44、55、66、88，对数列进行从小到大的排序，然后采用折半查找法，请写出数据11、66、99的查找过程和比较次数。

位置 i	1	2	3	4	5	6	7	8
比较次数	3	2	3	1	3	2	3	4



9.1 静态查找表

- 索引顺序表查找
- 索引的定义

  - 索引：就是把一个关键字与它对应的记录相关联的过程。

一个索引由若干个索引项构成，每个索引项至少应包含关键字和其对应的记录在存储器中的位置等信息。

  - 索引技术是大型数据库以及磁盘文件的一种重要技术。
  - 将索引项集合组织为线性表结构，称为索引表。

9.1 静态查找表

- 索引顺序表查找
- 索引存储结构

索引存储结构 = 数据主表 + 索引表

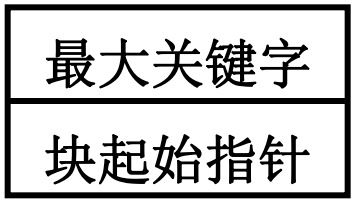
索引表中的每一个索引项的一般形式是：

（关键字值，地址）

9.1 静态查找表

- 索引顺序表查找
  - 主表：用数组存放待查记录，每个数据元素至少含有关键字域。
  - 索引表：索引表按关键字有序，表中每个结点含有最大关键字域和指向本块第一个结点的指针，结构如下：

```
typedef struct IndexType
{    keyType  maxkey ;    /*  块中最大的关键字  */
     int   startpos ;     /*  块的起始位置指针  */
} Index;
```

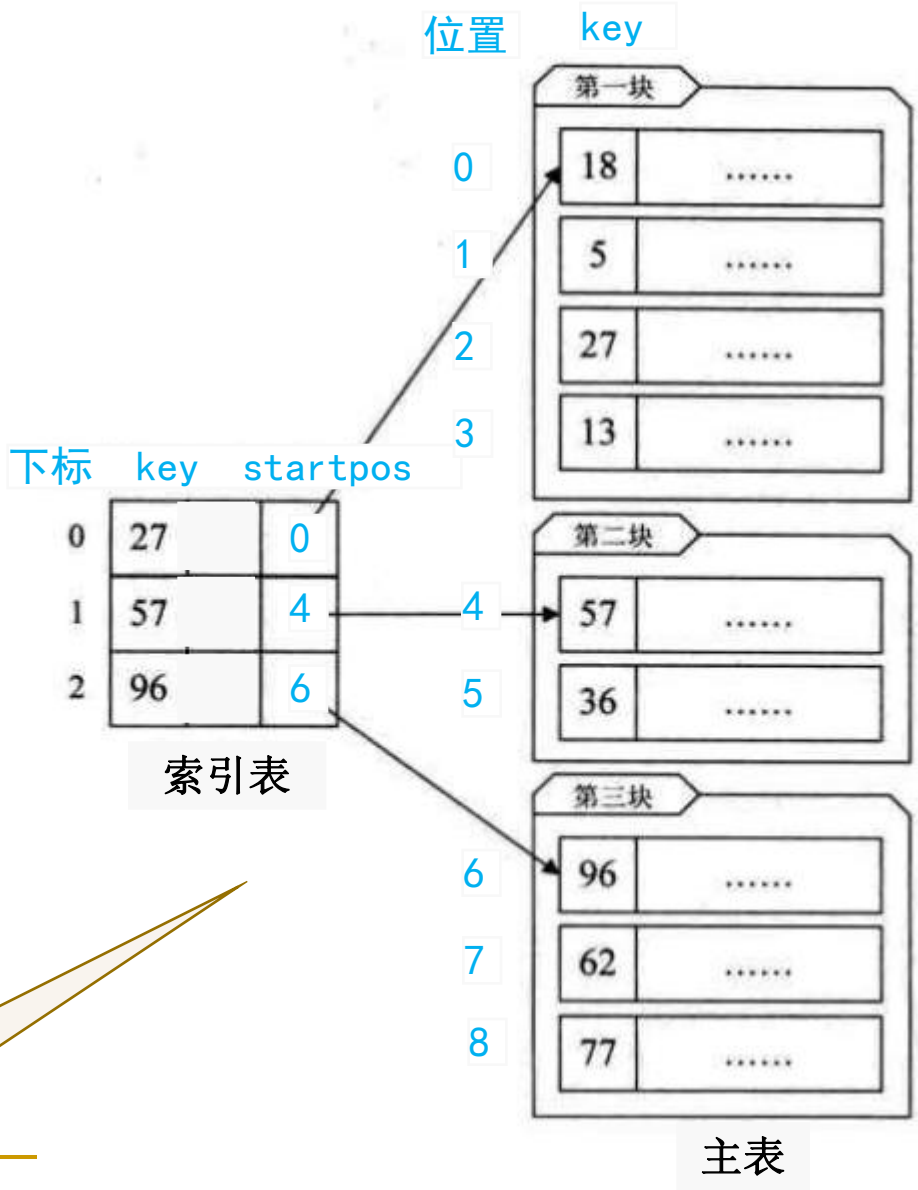


9.1 静态查找表

- 索引顺序表查找
  - 索引顺序表(分块有序表)将整个表分成几块，“分块有序，块内无序”。
  - 分块有序，是指第i+1块的所有记录关键字均大于(或小于)第i块记录关键字，即后一块表中所有记录的关键字均大于前一块表中的最大关键字。

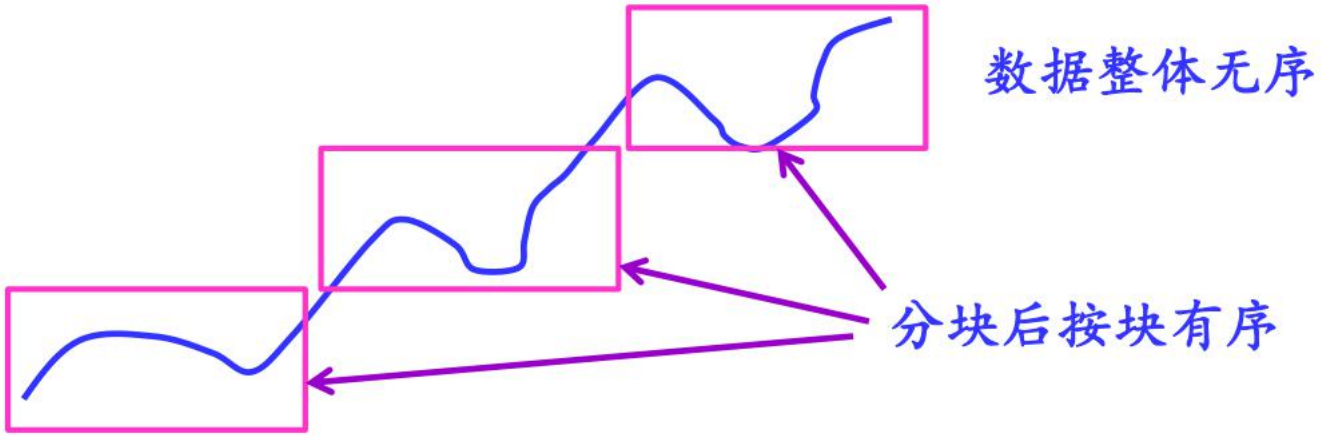
9.1 静态查找表

- 分块查找



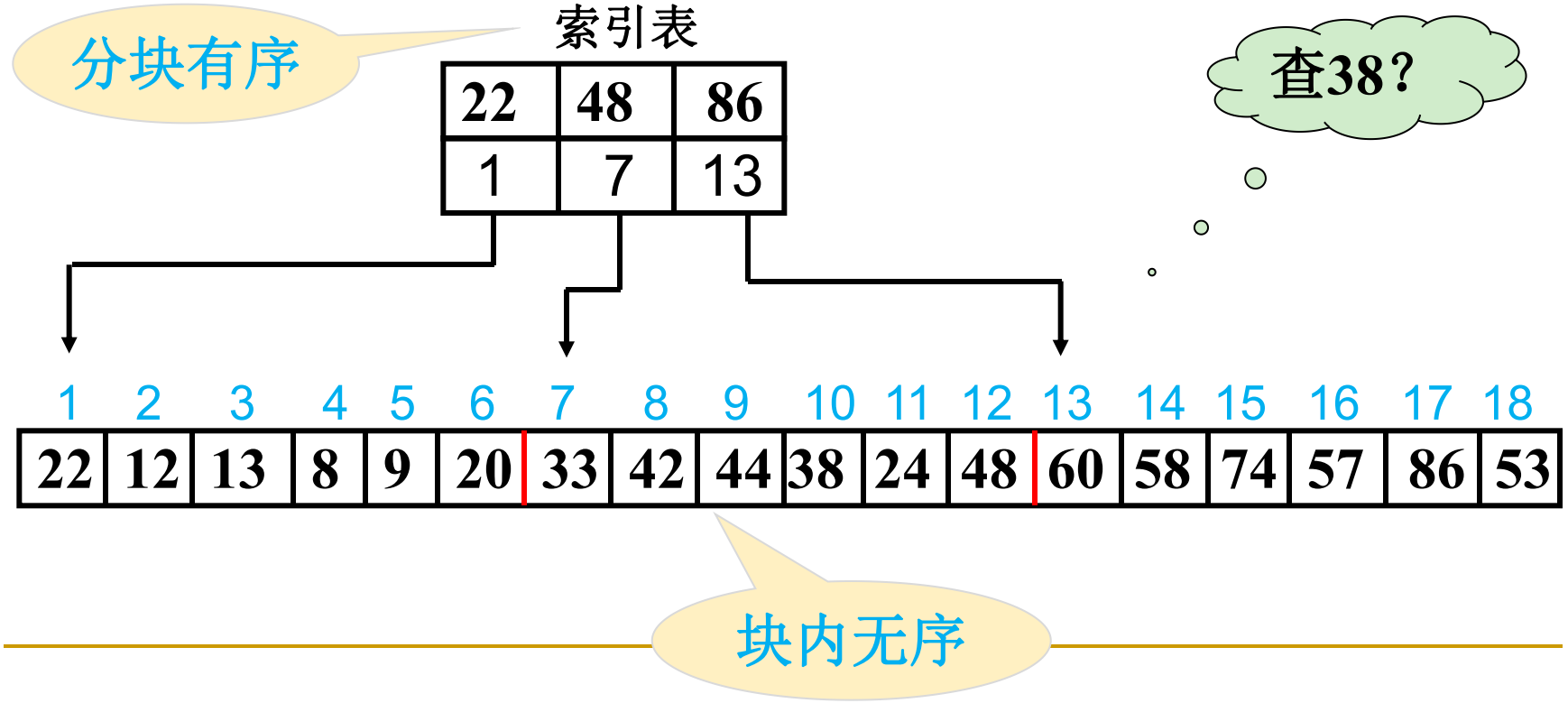
9.1 静态查找表

- 索引顺序表查找
  - **索引表（有序）**：可以顺序查找块，也可以二分查找块。
  - **数据块（无序）**：只能顺序查找块中元素。
  - **索引顺序表查找**：可以将折半查找和顺序查找方法结合，又称分块查找。



9.1 静态查找表

- 索引顺序表查找
- 算法思想：
  - 先用折半或顺序查找方法，在索引表中确定所在块
  - 再用顺序查找方法，在主表对应块中找到记录



9.1 静态查找表

- 索引顺序表查找
- 程序实现

```
int Block_search(RecType ST[] , Index ind[] , KeyType key , int n , int b)
//待查记录key，表长为n ，块数为b
    {   int i=0 , j , k ;
        while ((i<b)&&LT(ind[i].maxkey, key) )  i++  ;
            if (i>b) {  printf("\nNot found");   return(0);  }
        j=ind[i].startpos ;
        while ((j<n)&&LQ(ST[j].key, ind[i].maxkey) )
            {   if ( EQ(ST[j].key, key) )  break ;
                j++ ;
            }    // 在块内查找
        if (j>n||!EQ(ST[j].key, key) )
            {  j=0; printf("\nNot found");  }
        return(j);
    }
```

9.1 静态查找表

- 索引顺序表查找
- 算法性能
  - 若将长度为n的表分成b块，每块含s个记录，则 b=⌈ n/s⌉ 。设记录的查找概率相等，每块的查找概率为1/b，块中记录的查找概率为 1/s，并设表中每个记录查找概率相等
    - 用折半查找方法在索引表中查找索引块

      $ASL_{块间} \approx \log_2(b+1)$
    - 用顺序查找方法在主表对应块中查找记录

      $ASL_{块内} = s/2$
  - 分块查找的平均查找长度：

    $ASL \approx \log_2(n/s+1) + s/2$

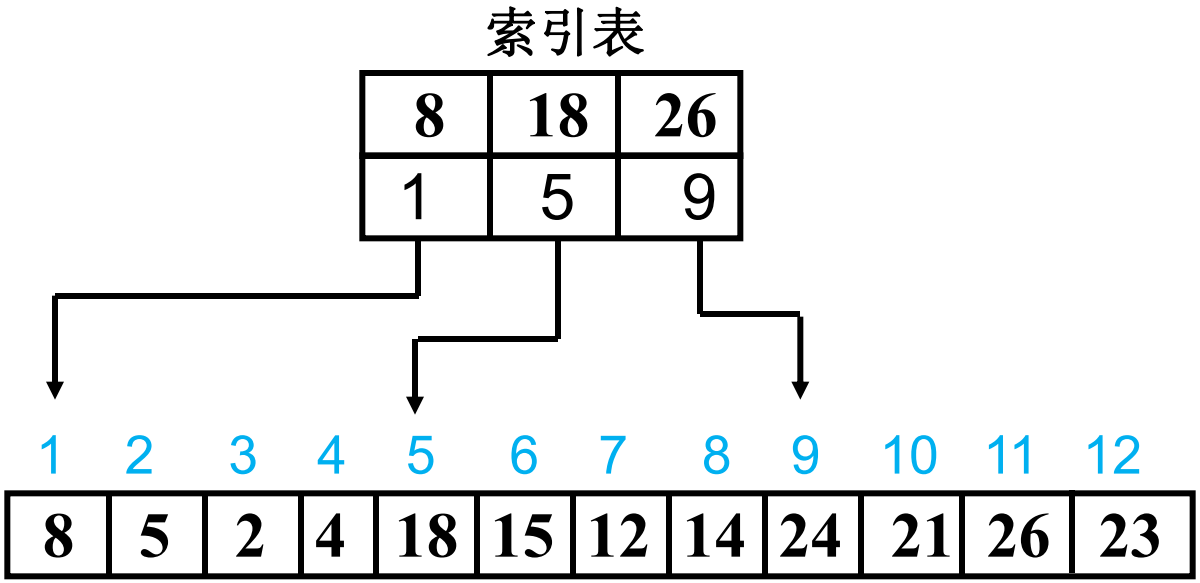
9.1 静态查找表

- 索引顺序表查找
- 分块索引查找的隐含条件
  - 如果在索引表中采用折半查找算法，则要求每个块的最大值在索引表中有序的，在实际应用中要求主表数据要简单有序，尽量不要完全无序，即每个块的数据都应该大于前一块的最大值
  - 如果主表数据是完全无序，则需要查找多个块

练习

1、已知初始数列为8, 5, 2, 4, 18, 15, 12, 14, 24, 21, 26 , 23，

把数列分成3块并建立索引表，使用顺序索引法，请写出数据21和17的查找过程次数。



练习

2、已知数列如下，建立的索引表如下，数列位置从1开始编号，采用顺序索引查找方法，索引表用折半查找，块内用顺序查找，写出数据23、66、98的查找过程和次数。

- 数列：

  19 28 23 11 8 33 48 51 35 37 62 66 86 99 72 81

- 索引表：

位置	数值
1	33
7	66
13	99

9.1 静态查找表

- 三种查找方法的比较

方法 / 特征	顺序查找	折半查找	分块查找
ASL	最大	最小	两者之间
表结构	有序表、无序表	有序表	分块有序表
存储结构	顺序存储 链表存储	顺序存储	顺序存储 链表存储