数据结构

深圳技术大学
大数据与互联网学院

第九章 查找

9.1 静态查找表
9.2 动态查找表
9.3 哈希表

9.1 静态查找表

一. 查找表

- 数据的组织和查找是大多数应用程序的核心，而查找是所有数据处理中最基本、最常用的操作。特别当查找的对象是一个庞大数量的数据集合中的元素时，查找的方法和效率就显得格外重要。
- 静态查找表
  - 仅作查询和检索操作的查找表。
- 动态查找表
  - 在查找过程中同时插入查找表中不存在的数据元素，或者从查找表中删除已存在的某个数据元素

9.1 静态查找表

二. 查找表术语

- 关键字，是数据元素（或记录）中某个数据项的值，用以标识（识别）一个数据元素（或记录）
  - 主关键字：可以识别唯一的一个记录的关键字
  - 次关键字：能识别若干记录的关键字
- 查找，根据给定的某个值，在查找表中确定一个其关键字等于给定值的数据元素（或记录）
  - 查找成功：在查找表中查找到指定的记录
  - 查找不成功：在查找表中没有找到指定记录

9.1 静态查找表

二. 查找表术语

- 衡量查找算法的标准
  - 时间复杂度
  - 空间复杂度
  - 平均查找长度ASL

9.1 静态查找表

二. 查找表术语

- 平均查找长度，是为确定记录在表中的位置所进行的和关键字比较的次数的平均值

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

n为查找表的长度，即表中所含元素的个数
P_i为查找第i个元素的概率（$\sum P_i=1$）
C_i是查找第i个元素时同给定值K比较的次数

9.1 静态查找表

三. 顺序查找

- 顺序查找算法是顺序表的查找方法，以顺序表或线性链表表示静态查找表

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

3. 若某个记录比较相等，则查找成功

4. 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

三. 顺序查找

- 顺序查找算法实现

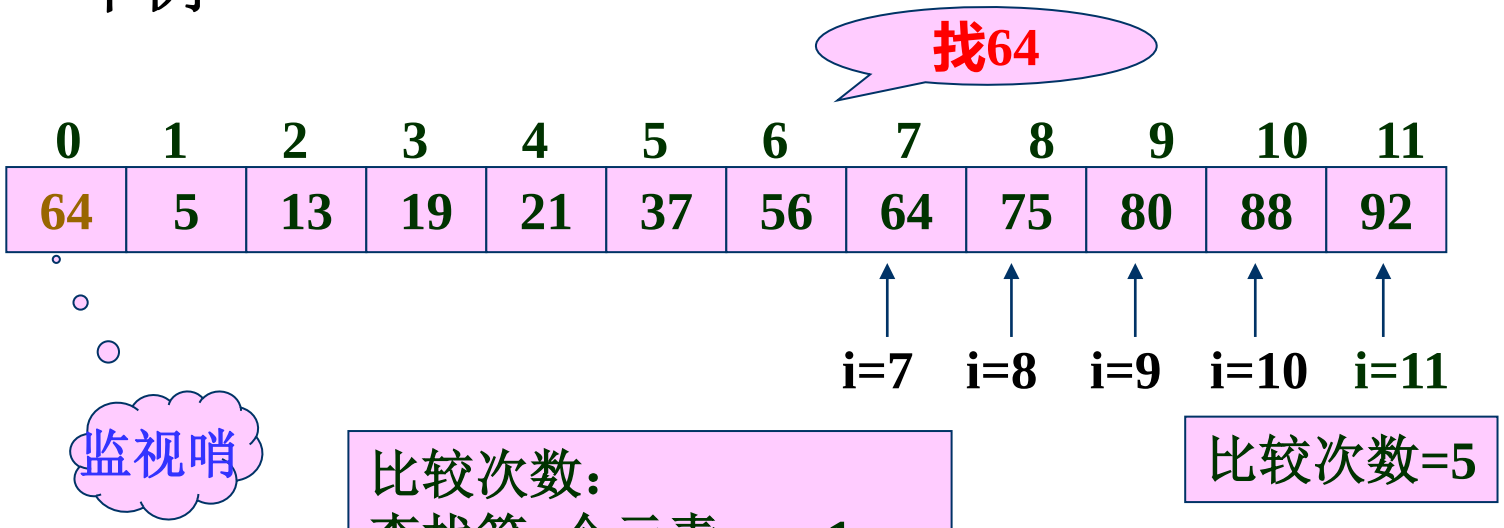
```
int Search_Seq(SSTable ST, KeyType key) {
   // 若查找成功，返回位置
   ST.elem[0].key = key;                    // “哨兵”
   for (i=ST.length; ST.elem[i].key!=key; --i); // 从后往前找
   return i;                                // 找不到时,i=0
} // Search_Seq
```

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

三. 顺序查找

- 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+1-i
查找失败： n+1

9.1 静态查找表

三. 顺序查找

- 算法性能分析
  - 对顺序表而言，$C_i=n-i+1$
  - 在等概率查找的情况下，$P_i=1/n$
  - $ASL=n*P_1+(n-1)P_2+\ldots+2P_{n-1}+P_n=(n+1)/2$

9.1 静态查找表

三. 顺序查找

- 顺序查找的优化
  - 如果被查找的记录概率不等时，设

    $$P_n \geqslant P_{n-1} \geqslant \cdots \geqslant P_2 \geqslant P_1$$

    记录按照概率来排列，这样使得ASL可以达到极小值

  - 若查找概率无法事先测定，增加一个访问频度域存储访问频率，然后记录按照访问频率来排序
  - 在每次查找之后，将刚刚查找到的记录直接移至表尾的位置上。

9.1 静态查找表

三. 顺序查找

- 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的查找次数

9.1 静态查找表

四. 折半查找

- 折半查找算法是有序表的查找方法
  - 在折半查找算法中，静态查找表按关键字大小的次序，有序地存放在顺序表中
- 折半查找的原理是：
  - 先确定待查记录所在的范围(前部分或后部分)
  - 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

四. 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. 让k与mid指向的记录比较
   若k=ST[mid].key，查找成功
   若k<ST[mid].key，则high=mid-1　　[上半区间]
   若k>ST[mid].key，则low=mid+1　　[下半区间]
5. 重复3,4操作，直至low>high时，查找失败。

9.1 静态查找表

四. 折半查找

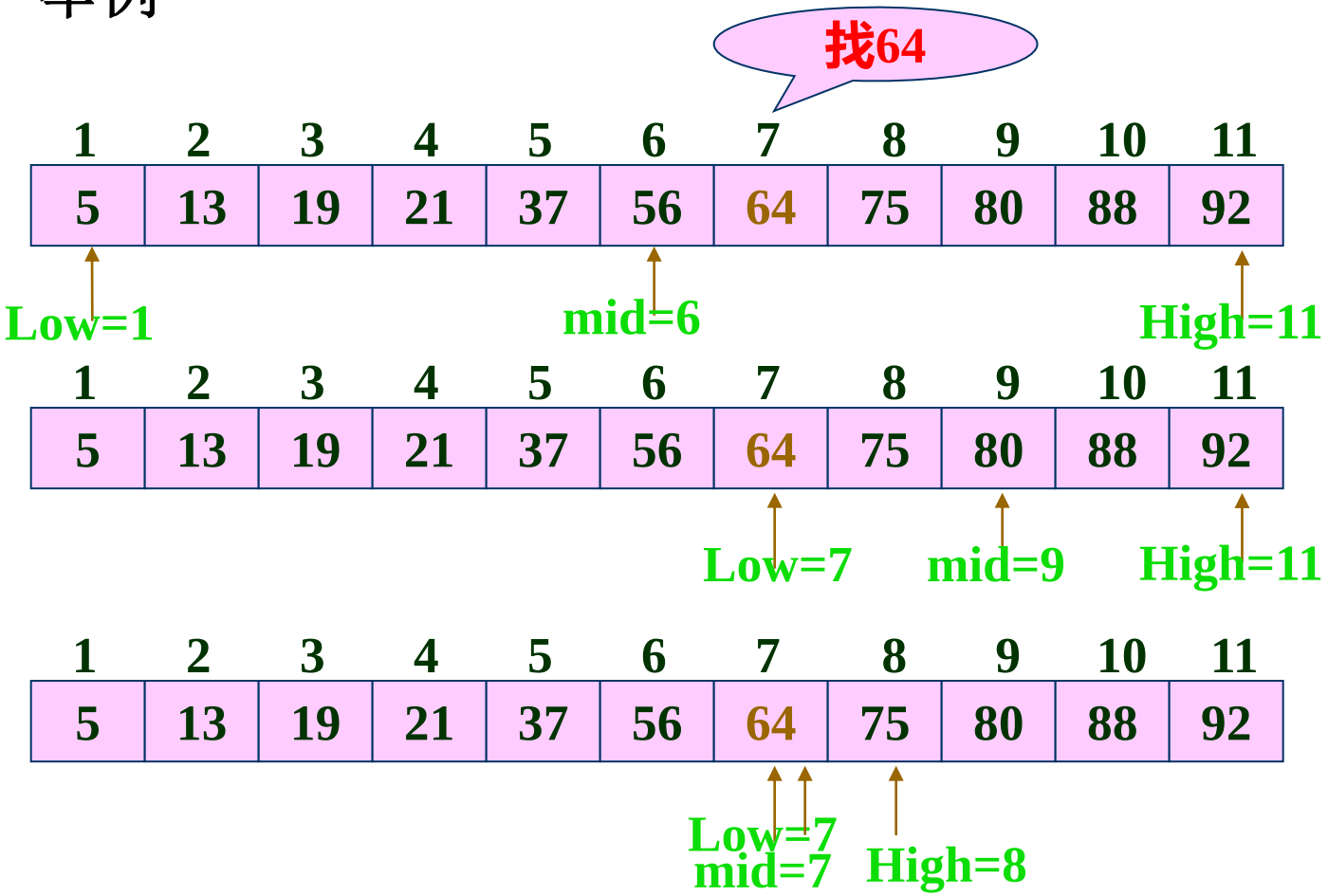
- 算法实现

```
int Search_Bin ( SSTable ST, KeyType key ) {
// 若找到，则函数值为该元素在表中的位置，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

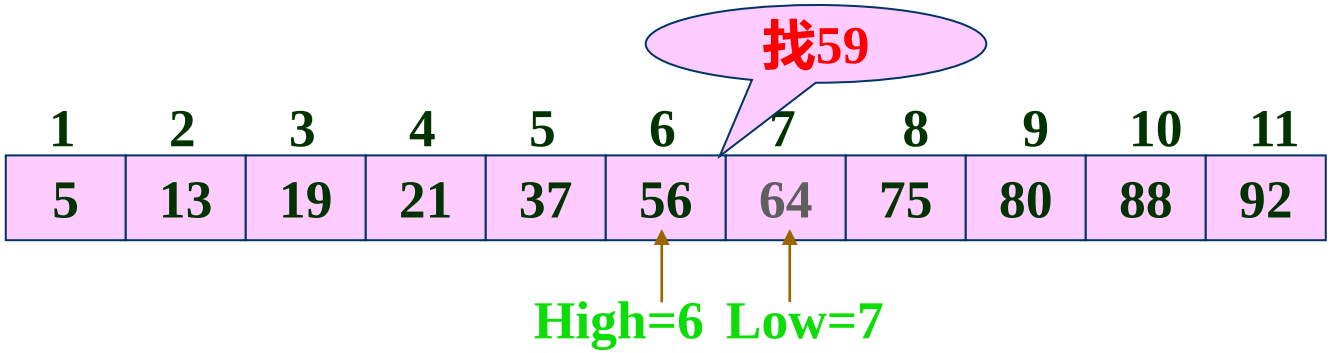
- 举例



9.1 静态查找表

四. 折半查找

- 举例



- 当下界low大于上界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

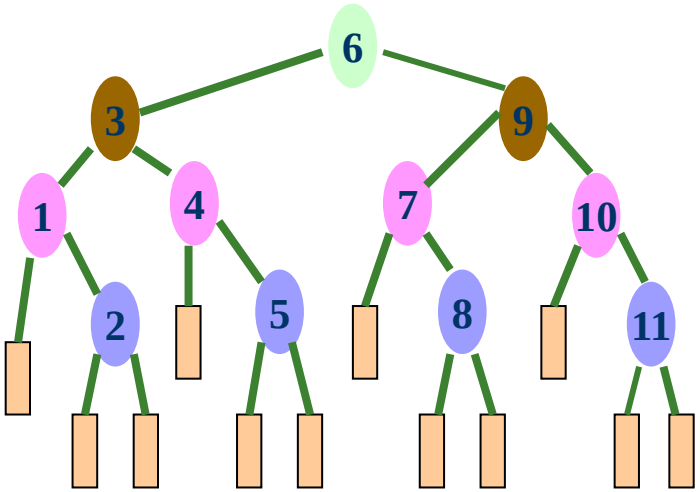
四. 折半查找

- 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：
  - 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作为左子树的结点；
  - 排在中间位置后面的作为右子树的结点；
- 这样得到的二叉树称为判定树(Decision Tree)。

9.1 静态查找表

四. 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$\lfloor \log_2 n \rfloor +1$



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

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

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

9.1 静态查找表

四. 折半查找

- 算法特点
  - 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)
  - 折半查找只能适用于有序表，并且以顺序存储结构存储

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的查找过程和查找次数

9.1 静态查找表

五. 索引顺序表查找

- 又称分块查找，是一种索引顺序表(分块有序表)查找方法，是折半查找和顺序查找的简单结合
- 索引顺序表(分块有序表)将整个表分成几块，块内无序，块间有序
- 所谓块间有序是指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

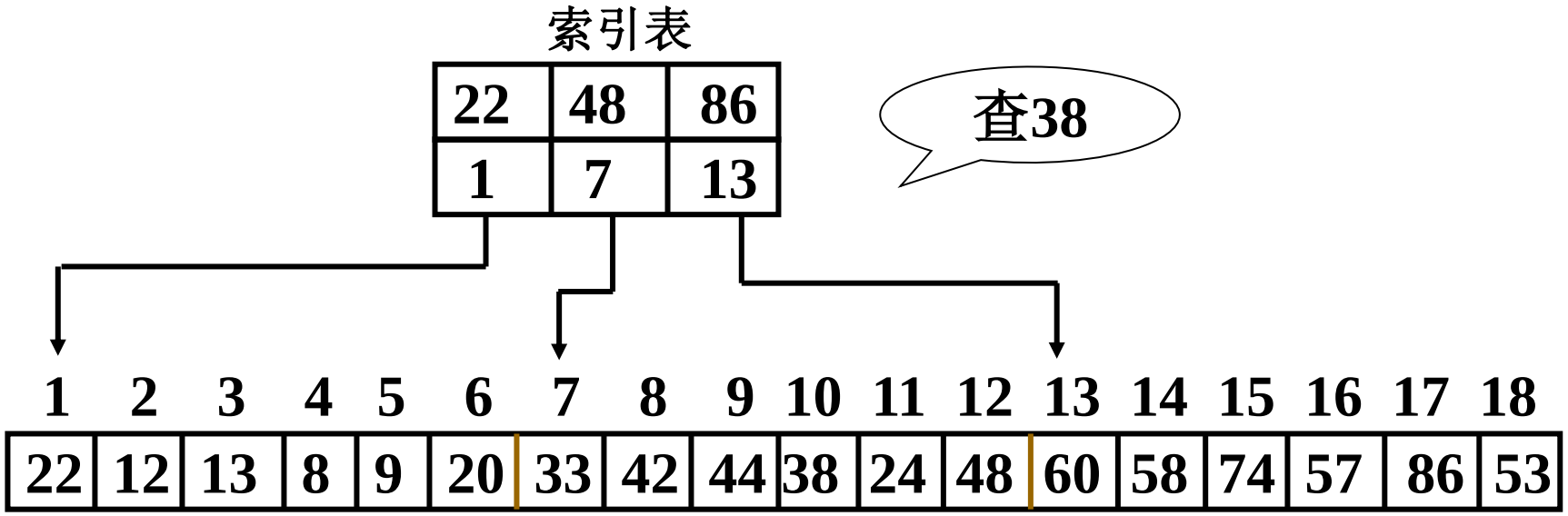
五. 分块查找

- 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：索引表是按关键字有序的，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

9.1 静态查找表

五. 分块查找

- 算法思想：
  - 用折半或顺序查找方法，在索引表中找到块
  - 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

五. 分块查找

- 程序实现

//索引表结点的定义

```
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

五. 分块查找

- 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) {  cout << "\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; cout << "\nNot found";
    }
    return(j);
}
```

9.1 静态查找表

五. 分块查找

- 算法性能
  - 若将长度为n的表分成b块，每块含s个记录，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

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

    $ASL_{块内} = s/2$
  - $ASL \approx \log_2(n/s+1) + s/2$

9.1 静态查找表

五. 分块查找

- 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

一. 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，块内用带哨兵顺序查找，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

五. 分块查找

- 三种查找方法的比较

	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	顺序存储结构 线性链表	顺序存储结构	顺序存储结构 线性链表