第9章 查找

- 查找是对已存入计算机中的数据所进行的一种运算，采用何种查找方法，首先取决于使用哪种数据结构来表示“表”，即表中数据元素是按何种方式组织的。
- 为了提高查找速度，常常用某些特殊的数据结构来组织表，或对表事先进行诸如排序这样的运算。
- 因此在研究各种查找方法时，必须弄清这些方法所需要的数据结构是什么？对表中关键字的次序有何要求，例如，是对无序表查找还是对有序表查找？
- 查找表的定义：是由同一类型的数据元素(或记录)构成的集合。由于“集合”中的数据元素之间存在着松散的关系，因此查找表是一种应用灵便的结构。

- 若对查找表只作：(1)查询某个“特定的”数据元素是否在查找表中；（2)检索某个“特定的”数据元素的各种属性，则称此类查找表为静态查找表(Static Search Table)。
- 若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，则称此类表为动态查找表(Dynamic Search Table)。

- 由于查找运算的主要运算是关键字的比较，所以通常把查找过程中和给定值进行比较的关键字个数的平均比较次数（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ASL（Average Search Length）定义为：

$$ASL = \sum_{i=1}^{n} p_i c_i$$

  - 其中，n是查找表中元素的个数。p_i是查找第i（1≤i≤n）个元素的概率，一般地，除特别指出外，均认为每个元素的查找概率相等，即$p_i=1/n$，c_i是找到第i个元素所需进行的比较次数。

平均查找长度分为：

- 查找成功情况下的平均查找长度ASL_{succ}，指在表中找到指定关键字的元素平均所需关键字比较的次数。
- 查找不成功情况下的平均查找长度ASL_{unsucc}，指在表中找不到指定关键字的元素平均所需关键字比较的次数。

本章及以后各章节的讨论中，涉及的关键字类型和数据元素类型统一说明如下：

```
//----- 典型的关键字类型说明 -----
typedef  float   KeyType ;        /*  实型  */
typedef  int   KeyType ;          /*  整型  */
typedef  char  KeyType ;          /*  字符串型  */

//----- 数据元素类型的定义 -----
typedef  struct  RecType {
    KeyType  key ;                /* 关键字码  */
    otherinfo                     /* 其他域  */
} RecType ;
```

本章及以后各章节的讨论中，涉及的关键字类型和数据元素类型统一说明如下：

```
//----- 对两个关键字的比较约定为如下带参数的宏定义 ----
/*  对数值型关键字  */
#define  EQ(a, b)   ((a)==(b))
#define  LT(a, b)   ((a)<(b))
#define  LQ(a, b)   ((a)<=(b))

/*  对字符串型关键字  */
#define  EQ(a, b)   (!strcmp((a), (b)) )
#define  LT(a, b)   (strcmp((a), (b))<0 )
#define  LQ(a, b)   (strcmp((a), (b))<=0 )
```

静态查找表采用顺序存储结构，顺序表是一种典型的顺序存储结构，本节静态查找表采用顺序表组织方式，并假设顺序表中数据元素类型的定义如下：

```
// ------ 静态查找表的顺序存储结构 ------
typedef struct {
    // 数据元素存储空间基址，建表时按实际长度分配，0号单元留空
    ElemType  *elem;
    int length;              // 表长度
} SSTable;
```

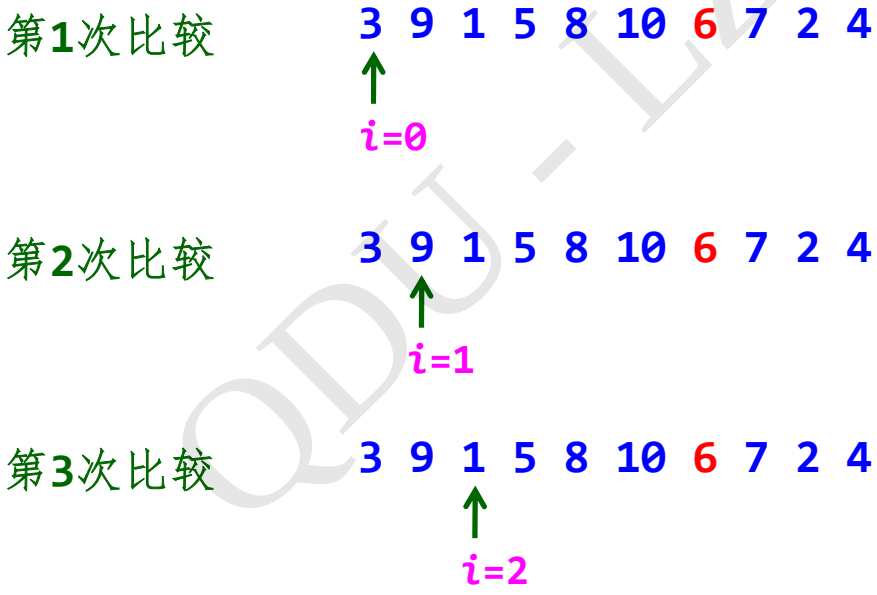
9.1 静态查找表

9.1.1 顺序表的查找

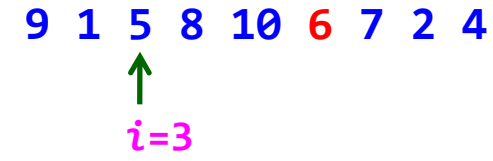
- 顺序查找又称为线性查找，是一种最简单的查找方法。
- **基本思路：**从表的一端开始顺序扫描顺序表，依次将扫描到的元素关键字和给定值k相比较，若当前扫描到的元素关键字与k相等，则查找成功；若扫描结束后，仍未找到关键字等于k的元素，则查找失败。

【示例-1】 在关键字序列为（3,9,1,5,8,10,6,7,2,4）的顺序表采用顺序查找方法查找关键字为6的元素。

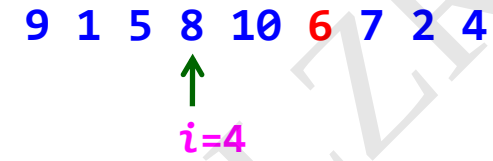
解：顺序查找过程如下：



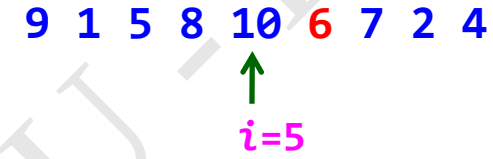
第4次比较



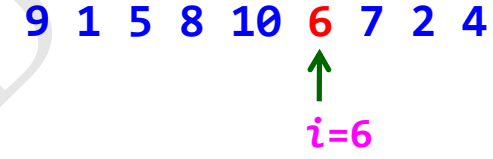
第5次比较



第6次比较



第7次比较



顺序查找算法如下（在顺序表ST中查找关键字为k的元素，成功时返回找到的元素的逻辑序号，失败时返回0）：

```
int SeqSearch(SSTable ST, KeyType k)  //顺序查找算法
{
   int i=0;
   while (i<ST.length && ST.elem[i]!=k)   //从表头往后找
        i++;

   if (i==ST.length)                      //未找到返回0
        return 0;
   else
        return i+1;                  //找到后返回其逻辑序号i+1
}
```

算法分析：对于含有n(即length)个元素的顺序表，元素的查找在等概率的前提下，查找成功的概率$p_i=1/n$。

第1个元素（序号为0的元素）查找成功需比较1次

第2个元素（序号为1的元素）查找成功需比较2次

……

第n个元素（序号为$n-1$的元素）查找成功需比较n次

即成功找到第i（$1\le i\le n$）个元素所需关键字比较次数$C_i=i$，所以查找成功时的平均查找长度为：

$$ASL_{succ}=\sum_{i=1}^{n}p_iC_i=\sum_{i=1}^{n}\frac{1}{n}\times i=\frac{1}{n}(1+2+\cdots+n)=\frac{n+1}{2}=O(n)$$

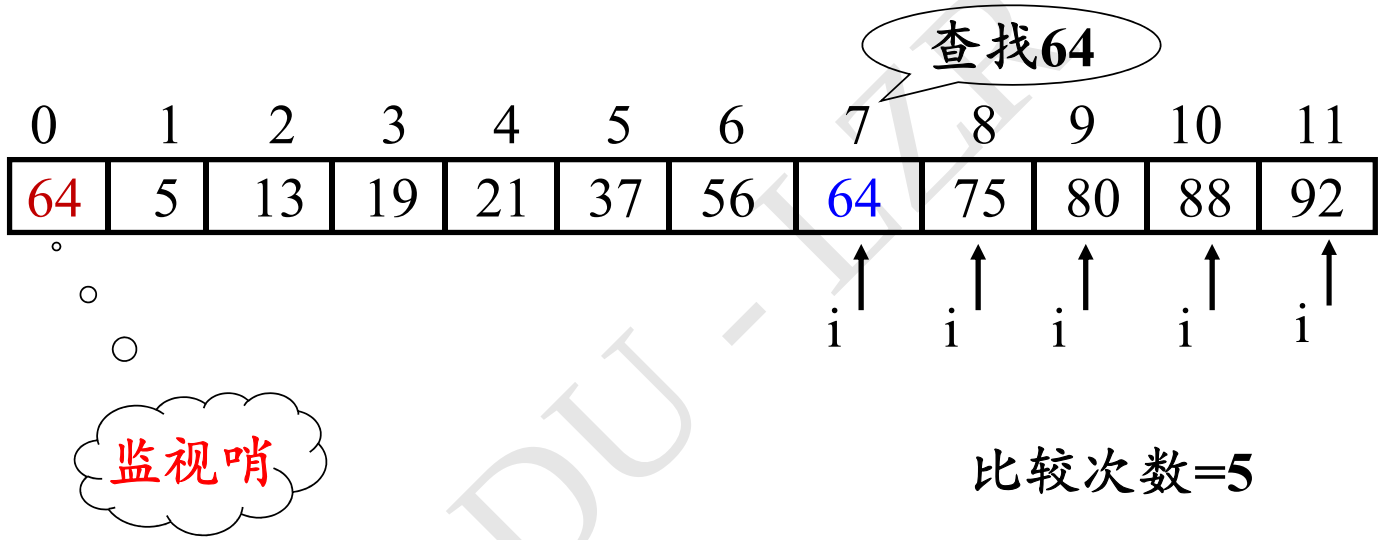
任何一次不成功的查找，都需要和顺序表中n个元素的关键字都比较一次，所以：

$$ASL_{unsucc}=n$$

顺序查找算法的改进（在顺序表ST中查找关键字为*k*的元素，成功时返回找到的元素的逻辑序号，失败时返回0）：

```
int Search_Seq(SSTable ST, KeyType key)
{   // 在顺序表ST中顺序查找其关键字等于key的数据元素。若找到，则返回
    // 该元素在表中的位置，否则返回0。算法9.1
    int i;
    ST.elem[0].key = key;     // 哨兵
    for(i = ST.length; !EQ(ST.elem[i].key, key); --i)
            ;                 // 从后往前找
    return i;                 // 找不到时，i为0
}
```

Keys = (5　13　19　21　37　56　64　75　80　88　92)



算法9-1　顺序查找示例

- 这个算法的思想和算法SeqSearch(ST，key)一致。只是在Search_Seq(ST，key)中，查找之前先对ST.elem[0]单元赋值关键字key，目的在于免去查找过程中每一步都要检测整个表是否查找完毕。在此，ST.elem[0]起到了监视哨的作用。这仅是一个程序设计技巧上的改进，然而实践证明，这个改进能使顺序查找在n≥1000时，进行一次查找所需的平均时间几乎减少一半！
- 监视哨也可设在高下标处。
- 算法性能分析和算法SeqSearch(ST，key)一致。

END