福州大学至诚学院
FUZHOU UNIVERSITY ZHICHENG COLLEGE

高级语言程序设计（C语言与数据结构）

杨雄

83789047@qq.com

9.查找

01 静态查找表

学习目的

- **掌握静态查找表的概念**
- **顺序查找、折半查找**

Part.1

9.1 静态查找表

查 找 概 论

- **查找表** (search table):

  同一类型数据元素构成的集合。

- **关键字(KEY):**
  - 数据元素(或记录)的某个数据项的值。
  - 可以唯一标识一个记录的关键字称为主关键字；

    否则称为次关键字。

- **查找(Searching)**
  - 根据给定的值,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 概 论

- 查找操作:

  (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;

  (3)在查找表中插入一个数据元素;

  (4)从查找表中删除某个数据元素;

- 静态查找表:对查找表只作(1)(2)操作;
- 动态查找表:可以对查找表作(1)-(4)操作

关键字类型

典型的关键字类型说明：

```
typedef float KeyType;
typedef int   KeyType;
typedef char KeyType;
```

数据元素类型定义为：

```
typedef struct {
    KeyType key;
    ……
}SElemType;
```

9.1 静态查找表

- 静态查找表可以用顺序表，也可以用线性链表来表示。
  - 顺序表的查找

```
//静态查找表的顺序存储结构
typedef struct{
    SElemType *elem;
    int length;
}SSTable;
```

- 顺序查找：

从表中第一个（或最后一个）记录开始，逐个进行记录的关键字和给定值比较，若某个记录的关键字和给定值相等，则查找成功，找到所查的记录；

如果直到最后一个（或第一个）记录，其关键字和给定值比较都不等时，则表中没有所查的记录，则查找不成功。

- 顺序查找：

从表中第一个（或最后一个）记录开始，逐个进行记录的关键字和给定值比较，若某个记录的关键字和给定值相等，则查找成功，找到所查的记录；

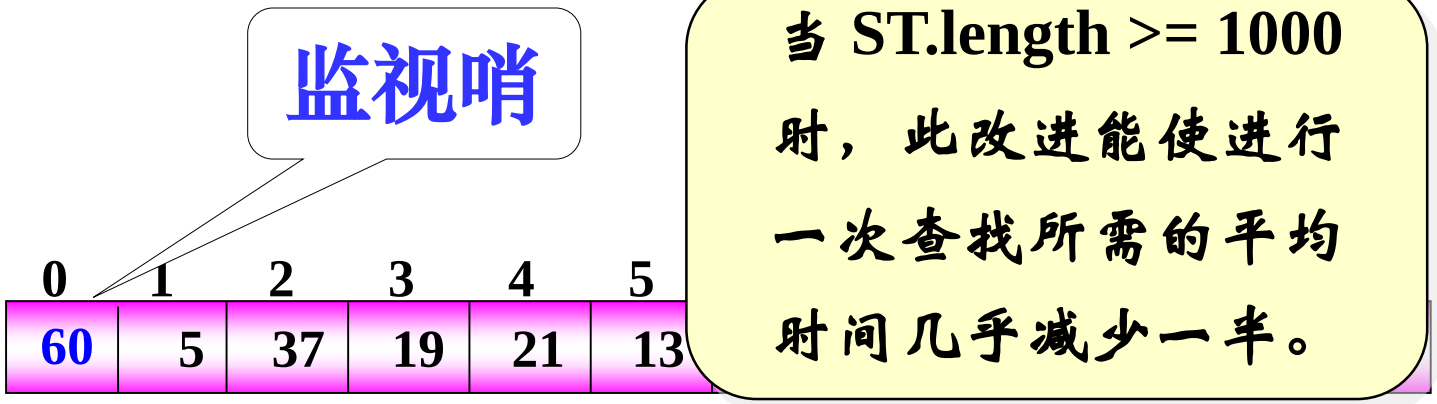
如果直到最后一个（或第一个）记录，其关键字和给定值比较都不等时，则表中没有所查的记录，则查找不成功。

- 顺序查找的算法实现：

```
int Search_Seq(SSTable ST,  KeyType key)
{
  for (i = ST.length ;  ST.elem[i].key != key ;  - - i )
       if (i <= 0) break ;
  if (i > 0) return i ;
  else return 0 ;
}
```

- 顺序查找的算法改进实现：

```
int Search_Seq(SSTable ST,  KeyType key)
{
  ST.elem[0].key = key ;
  for (i = ST.length ;  ST.elem[i].key != key ;  - - i )
    return i ;
}
```



当 ST.length >= 1000 时，此改进能使进行一次查找所需的平均时间几乎减少一半。

顺序查找
- 优点：算法简单，适应面广。
- 缺点：平均查找长度大，不适用于表长大的查找表。

查找方法评价：

时间复杂度；空间复杂度；算法本身复杂程度。

因为查找算法的基本操作为：将记录的关键字同给定值进行比较，所以，通常以关键字同给定值进行过比较的记录的个数的平均值作为衡量查找算法好坏的依据。

平均查找长度 ASL (Average Search Length)：为确定记录在表中的位置，需要与给定值进行比较的关键字的个数的期望值叫做查找算法的平均查找长度。

0	1	2	3	4	5	6	7	8	9	10	11
	5	37	19	21	13	56	64	92	88	80	75
	11	10	9	8	7	6	5	4	3	2	1

对含有 n 个记录的表，查找**成功**时：

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

C_i：找到第 i 个记录需比较的次数。

P_i：第 i 个记录被查找的概率。

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

顺序查找的平均查找长度：

$$ASL=nP_1+(n-1)P_2+\ldots+2P_{n-1}+P_n$$

假设每个记录的查找概率相等：$P_i=1/n$

则：$$ASL_{SS}=\sum_{i=1}^{n}P_iC_i=\frac{1}{n}\sum_{i=1}^{n}(n-i+1)=\frac{n+1}{2}$$

0	1	2	3	4	5	6	7	8	9	10	11
	5	37	19	21	13	56	64	92	88	80	75
12	11	10	9	8	7	6	5	4	3	2	1

查找不成功时，关键字的比较次数总是 $n+1$ 次。

假设：1、查找成功与不成功的概率相同。

2、每个记录被查找的概率相同。

则：平均查找长度（成功与不成功的平均查找长度之和）为

$$ASL_{SS}=\frac{1}{2n}\sum_{i=1}^{n}(n-i+1)+\frac{1}{2}(n+1)=\frac{3(n+1)}{4}$$

- 有序表的查找

  - **折半查找：**

确定待查记录的区间，逐步缩小范围直到找到或找不到该记录为止。

前提：

1）线性表中的记录必须是关键字有序；

2）线性表必须采用顺序存储。

折半查找

基本思路：

1）在有序表中，取中间记录作为比较对象，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，则在中间记录的左半区继续查找；

3）若给定值大于中间记录的关键字，则在中间记录的右半区继续查找；

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。

■ 折半查找

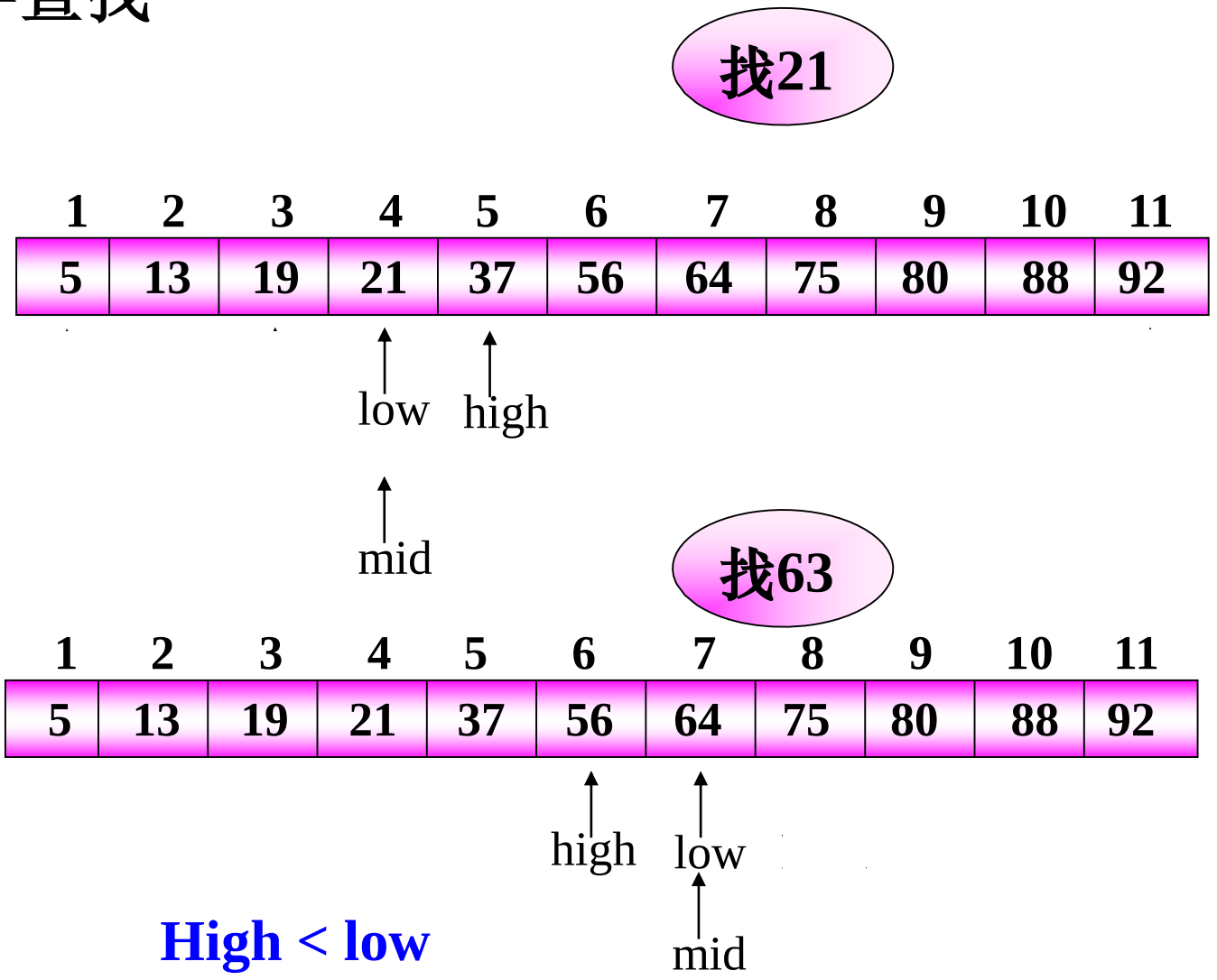
有序表折半查找关键字key的数据元素过程：

(1) key = ST.elem[mid]. key查找成功;

(2) 当key < ST.elem[mid].key时,下一个待查区间为[low,mid-1]

(3) 当key > ST.elem[mid].key时,下一个待查区间为[mid+1,high]

- 折半查找


找21
1 2 3 4 5 6 7 8 9 10 11
5 13 19 21 37 56 64 75 80 88 92
low
high
mid
找63
1 2 3 4 5 6 7 8 9 10 11
5 13 19 21 37 56 64 75 80 88 92
high
low
mid
High < low

- **折半查找算法描述：**

```
int Search_Bin ( SSTable ST,  KeyType key )
{ // 在有序表 ST 中折半查找其关键字等于 key 的数据元素。
  // 若找到，则函数值为该元素在表中的位置，否则为0。
   low = 1 ;       high = ST.length ;        // 置区间初值
   while (low <= high)
   {   mid = (low + high) / 2 ;
      if (ST.elem[mid].key = key)   return mid ;    // 找到待查元素
      else  if (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
```

■ 折半查找

i	1	2	3	4	5	6	7	8	9	10	11
	5	13	19	21	37	56	64	75	80	88	92
C_i	3	4	2	3	4	1	3	4	2	3	4

性能分析：

查找成功：

比较次数 = 路径上的结点数

比较次数 = 结点 4 的层数

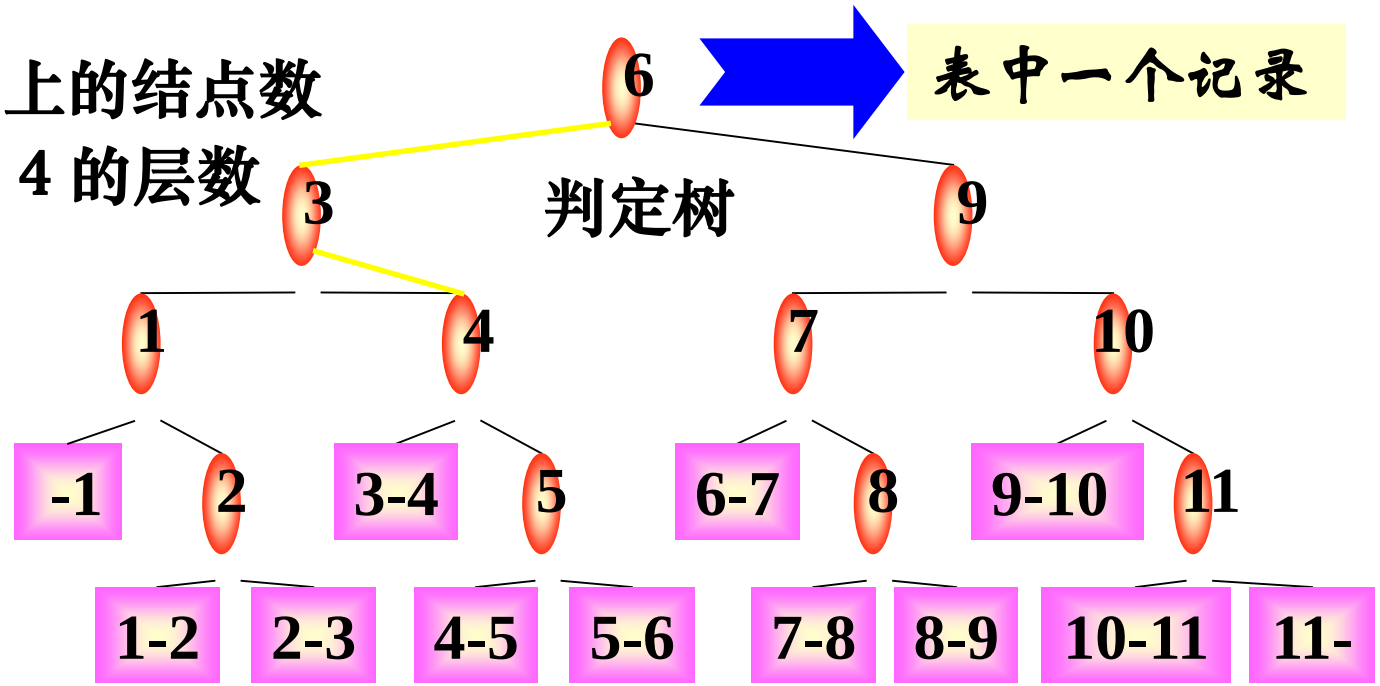
比较次数

⋀

树的深度

‖

$\lfloor \log_2 n \rfloor + 1$



查找不成功：

比较次数 = 路径上的内部结点数　　比较次数 ≤ $\lfloor \log_2 n \rfloor + 1$

■ **折半查找性能分析：**

平均查找长度ASL（成功时）：

设表长 $n = 2^h - 1$，则 $h = \log_2(n+1)$（此时，判定树为深度$= h$的满二叉树），且表中每个记录的查找概率相等：$P_i = 1/n$。

第 j 层的每个结点要比较的次数

$$\text{则：}\ ASL_{bs} = \sum_{i=1}^{n} p_i c_i = \frac{1}{n}\sum_{i=1}^{n} c_i = \frac{1}{n}\sum_{j=1}^{h} j \cdot 2^{j-1}$$

2^{j-1} → 第 j 层结点数

j → 第 j 层要比较的总次数

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

$$ASL_{SS} = \frac{n+1}{2} \approx \log_2(n+1) - 1 \quad (n > 50)$$

折半查找优点：效率比顺序查找高。

折半查找缺点：只适用于有序表，且限于顺序存储结构

- **索引顺序表的查找（分块查找）**

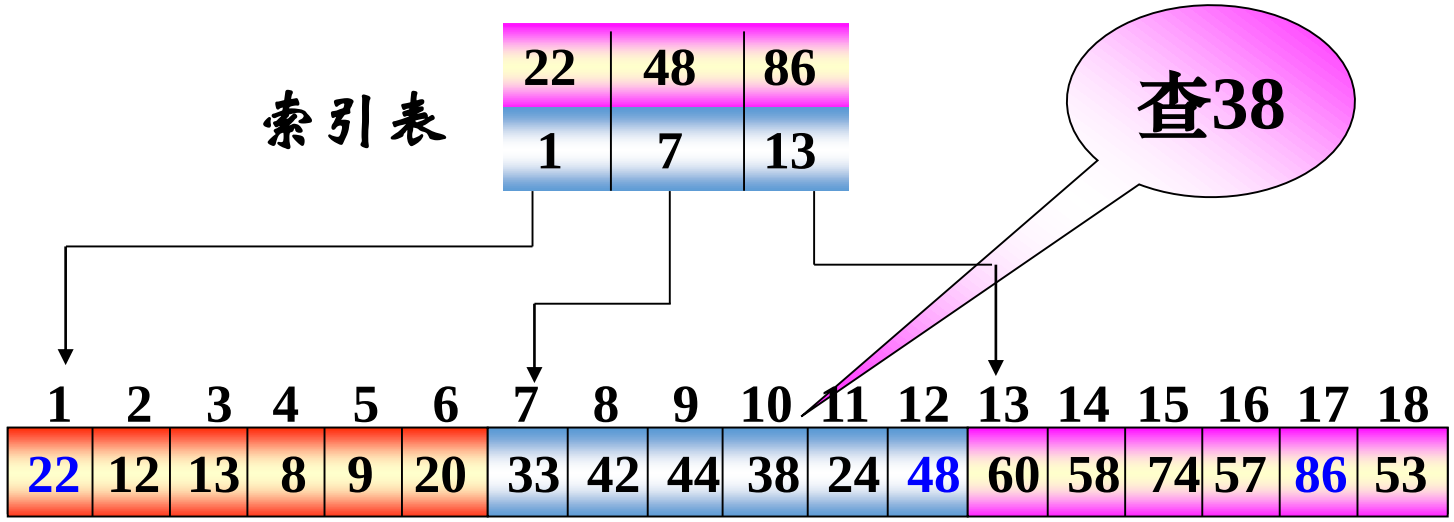
若 $i<j$，则第 j 块中所有记录的关键字均大于第 i 块中的最大关键字。

条件： 1、**块内无序**：即每一块内的记录不要求有序；

2、**块间有序**

3、建立“索引表”

查找过程： 先确定待查记录所在块（顺序或折半查找），再在块内查找（顺序查找）。



查找方法比较

	顺序查找	折半查找	分块查找
ASL	最大	最小	中间
表结构	有序表、无序表	有序表	分块有序
存储结构	顺序表、线性链表	顺序表	顺序表、线性链表

Part.2

总结

总 结

- **查找表**:同一类型数据元素构成的集合。

- **查找表**
  - **静态查找表**：查找、读表元素；
  - **动态查找表**：查找、读表元素、**插入、删除**

- **关键字**:数据元素(或记录)的某个数据项的值。
  - 主关键字：可以唯一标识一个记录的关键字。

- **查找结果**
  - 成功
  - 失败

平均查找长度：为确定记录在表中的位置，需要与给定值进行比较的关键字的个数的期望值。 $ASL = \sum_{i=1}^{n} P_i C_i$

总 结

- 顺序查找的算法改进实现：

```
int Search_Seq(SSTable ST,  KeyType key)
{
   ST.elem[0].key = key ;
   for (i = ST.length ;  ST.elem[i].key != key ;  - - i )
     return i ;
}
```

总 结

顺序查找成功的平均查找长度：

$$ASL_{SS}=\sum_{i=1}^{n}P_iC_i=\frac{1}{n}\sum_{i=1}^{n}(n-i+1)=\frac{n+1}{2}$$

顺序查找的时间复杂度： **O(n)**

顺序查找

- **优点：** 算法简单，适应面广。
- **缺点：** 平均查找长度大，不适用于表长大的查找表。

总 结

- 折半查找算法描述：

```
int Search_Bin ( SSTable ST,  KeyType key )
 {
   low = 1 ;       high = ST.length ;        // 置区间初值
   while (low <= high)
   {

      if (ST.elem[mid].key == key)
         return mid ;    // 找到待查元素
      else  if (key < ST.elem[mid].key)

     else

   }
   return 0 ;          // 顺序表中不存在待查元素
 } // Search_Bin
```

总 结

例题：

若查找每个记录的概率均等，则在具有n个记录的连续顺序文件中采用顺序查找法查找一个记录，其平均查找长度ASL为(　)。

A.　(n-1)/2　　B. n/2　　C. (n+1)/2　　D. n

总 结

例题：

对长度为10的顺序表进行查找，若查找前5个元素的概率相同，均为1/8，查找后5个元素的概率相同，均为3/40，则查找任一元素的平均查找长度为(　)。

A．5.5　　B．5

C．39/8　　D．19/4

$$ASL=(1+2+3+4+5)\times 1/8+(6+7+8+9+10)\times \frac{3}{40}=\frac{39}{8}。$$

总 结

例题：

折半查找有序表（4，6，10，12，20，30，50，70，88，100）。若查找表中 元素58，则它将依次与表中（)比较大小，查找结果是失败。

A．20，70，30，50

B．30，88，70，50

C．20，50

D．30，88，50

总 结

例题：

对 22 个记录的有序表作折半查找，当查找失败时，最多需要比较（ ） 次关键字

A．3　　B．4　　C．5　　D． 6

总 结

例题：

对线性表进行二分查找时，要求线性表必须(　)。

A．以顺序方式存储

B．以链接方式存储

C．顺序存储，且结点按关键字有序排序

D．链式存储，且结点按关键字有序排序