



第九章 查找



何谓查找表？

- 查找表是由**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。
- 比如：城市的电话号码簿就是一个查找表



对查找表经常进行的操作

- 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的各种属性；
- 在查找表中插入一个数据元素；
- 从查找表中删去某个数据元素。
- 静态查找表: 仅作查询和检索操作的查找表。
- 动态查找表: 有时在查询之后，还需要将查询结果为不在查找表中的数据元素插入到查找表中；或：从查找表中删除其查询结果为不在查找表中的数据元素。
- 查询和检索统称为查找



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）。
- 若此关键字可以识别唯一的一个记录，则称之为“主关键字”。
- 若此关键字能识别若干记录，则称之为“次关键字”。
- 查找表的查询和检索通常是根据关键字进行的



查找

- 根据给定的值，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
- 若查找表中存在这样一个记录（数据元素），则称**查找成功**。查找结果给出整个记录（数据元素）的信息，或指示该记录（数据元素）在查找表中的位置；否则称此**查找不成功**。查找结果给出“空记录”或“空指针”



如何进行查找?

- 查找的方法的选择和设计要看查找表的存储方式
- 由于查找表中的数据元素之间存在着“同属于一个集合的松散关系”，因此不便于查找。为了提高查找的效率，需要在查找表中的元素之间人为地附加某种确定的关系，换句话说，用另外一种结构来表示查找表。比如：把查找表组织成线性表、二叉树等等。



第9章的内容

■ 9.1 静态查找表

- 顺序查找表
- 有序查找表
- 静态查找树表
- 索引顺序表

■ 9.2 动态查找表

- 二叉排序树和平衡二叉树
- B-树和B+树
- 键树

■ 9.3 哈希表



第9章的内容

■ 9.1 静态查找表

- 顺序查找表
- 有序查找表
- 静态查找树表
- 索引顺序表

■ 9.2 动态查找表

- 二叉排序树和平衡二叉树
- B-树和B+树
- 键树

■ 9.3 哈希表



9.1 静态查找表--顺序查找

- 查找表的组织方式：以线性表表示静态查找表
- 查找方法：从表的一端顺序找到表的另一端
- 说明：存储结构无特殊要求（数组或链表都可以的），元素顺序无要求（有序、无序都可以的）

9.1 静态查找表--顺序表查找

- 若查找表组织成线性表，存储结构为数组，则称为顺序表

- 顺序表定义

```
typedef struct{  
    L.elem[0]  
    int key; .....; L.elem[1]  
    L.elem[2]  
}elemType;
```

<i>key</i>	数学
张卓	114
王力	98
李华	73

```
typedef struct  
{ elemType * elem; int length;  
}SqList;  
SqList L;
```



9.1 静态查找表--顺序查找

```
int search(SqList L, int x)
{ int i;
  for(i=0;i<L.length;i++)
    if(L.elem[i].key==x) return i+1;
  return 0;
}
```

说明:返回0说明没找到。

为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

	key	数学
L.elem[0]			
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73

为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找x?

	key	数学
L.elem[0]			
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113

为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =王力?

	key	数学
L.elem[0]			
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113

为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =王力?

	key	数学
L.elem[0]	王力		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113

← i

为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =王力?

	key	数学
L.elem[0]	王力		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113

← i

为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =王力?

	key	数学
L.elem[0]	王力		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113



为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =王力?

	key	数学
L.elem[0]	王力		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113



为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =王力?

	key	数学
L.elem[0]	王力		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113

找到!



为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =李刚?

	key	数学
L.elem[0]	王力		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113



为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =李刚?

	key	数学
L.elem[0]	李刚		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113

← i

为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =李刚?

	key	数学
L.elem[0]	李刚		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113

← i

为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =李刚?

	key	数学
L.elem[0]	李刚		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113



为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =李刚?

	key	数学
L.elem[0]	李刚		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113



为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

查找 x =李刚?

	key	数学
L.elem[0]	李刚		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113

← i

为提高查找速度，设置哨兵项

	key	数学
L.elem[0]	张卓	114
L.elem[1]	王力	98
L.elem[2]	李华	73

	key	数学
L.elem[0]	李刚		
L.elem[1]	张卓	114
L.elem[2]	王力	98
L.elem[3]	李华	73
L.elem[4]	李阳	93
L.elem[5]	黄日华	113

查找 x =李刚?
没找到!





9.1 静态查找表--顺序查找

```
int ssearch(SqList L,int x)
{ int k=L.length;
  L.elem[0].key=x;
  while(x!=L.elem[k].key)
    k--;
  return k;
}
//L.elem[0]哨兵项
```



9.1 静态查找表--顺序查找性能分析

- 查找成功的平均查找长度:

(查找成功的情况下, 平均找一个数据元素需要的比较次数称为查找成功的平均查找(检索)长度)

$$ASL = (1 + 2 + \dots + n) / n = (n + 1) / 2$$

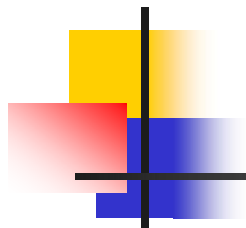
- 查找一次的平均检索长度 (成功, 失败)

$$ASL = (n + 1) / 2 + (1 + 2 + \dots + n) / (2 * n) = 3(n + 1) / 4$$



9.1 静态查找表--二分查找

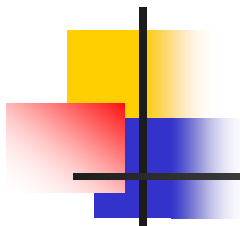
- 采用二分查找的要求：查找表组织成有序线性表（递增或递减），且采用顺序存储结构
- 特点：通过一次比较，将查找范围缩小一半



查找91

0	1	2	3	4	5	6	7	8	9	10
8	14	23	37	46	55	68	79	91	94	98

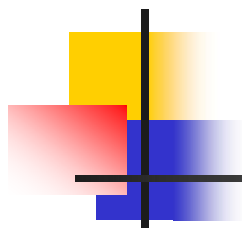
low mid high



查找91

0	1	2	3	4	5	6	7	8	9	10
8	14	23	37	46	55	68	79	91	94	98

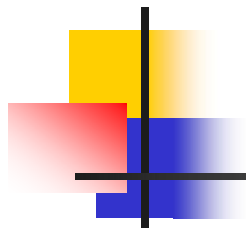
Diagram illustrating a binary search process on a sorted array. The array contains the values: 8, 14, 23, 37, 46, 55, 68, 79, 91, 94, 98. The search target is 91. The current search range is defined by **mid** (index 5, value 55) and **low** (index 6, value 68). The **high** pointer is at index 10 (value 98).



查找91

0	1	2	3	4	5	6	7	8	9	10
8	14	23	37	46	55	68	79	91	94	98

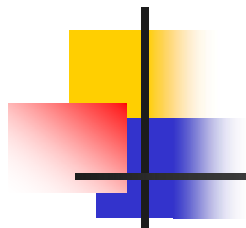
low mid high



查找26

0	1	2	3	4	5	6	7	8	9	10
8	14	23	37	46	55	68	79	91	94	98

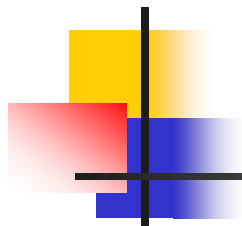
low mid high



查找26

0	1	2	3	4	5	6	7	8	9	10
8	14	23	37	46	55	68	79	91	94	98

low high mid



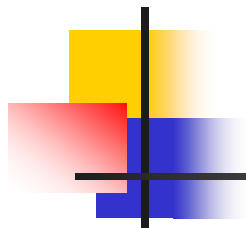
查找26

0	1	2	3	4	5	6	7	8	9	10
8	14	23	37	46	55	68	79	91	94	98

↑
low

↑
mid

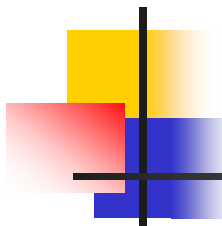
↑
high



查找26

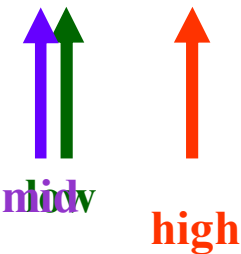
0	1	2	3	4	5	6	7	8	9	10
8	14	23	37	46	55	68	79	91	94	98

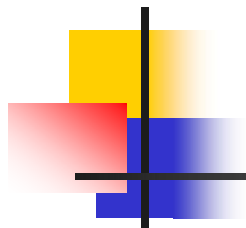
↑ mid ↑ low ↑ high



查找26


0	1	2	3	4	5	6	7	8	9	10
8	14	23	37	46	55	68	79	91	94	98







查找26

0	1	2	3	4	5	6	7	8	9	10
8	14	23	37	46	55	68	79	91	94	98


high


mid


low

High < low, 查找范围不能存在, 没找到!



9.1 静态查找表--二分查找

- 有序（递增或递减），顺序存储结构

- ```
int binaryS(SqList L, int x)
{
 int low=0, high=L.length-1, m;
 while(low<=high)
 {
 m=(low+high)/2;
 if(L.elem[m].key==x) return m+1;
 if(L.elem[m].key>x) high=m-1;
 else low=m+1;
 }
 return 0;
}
```



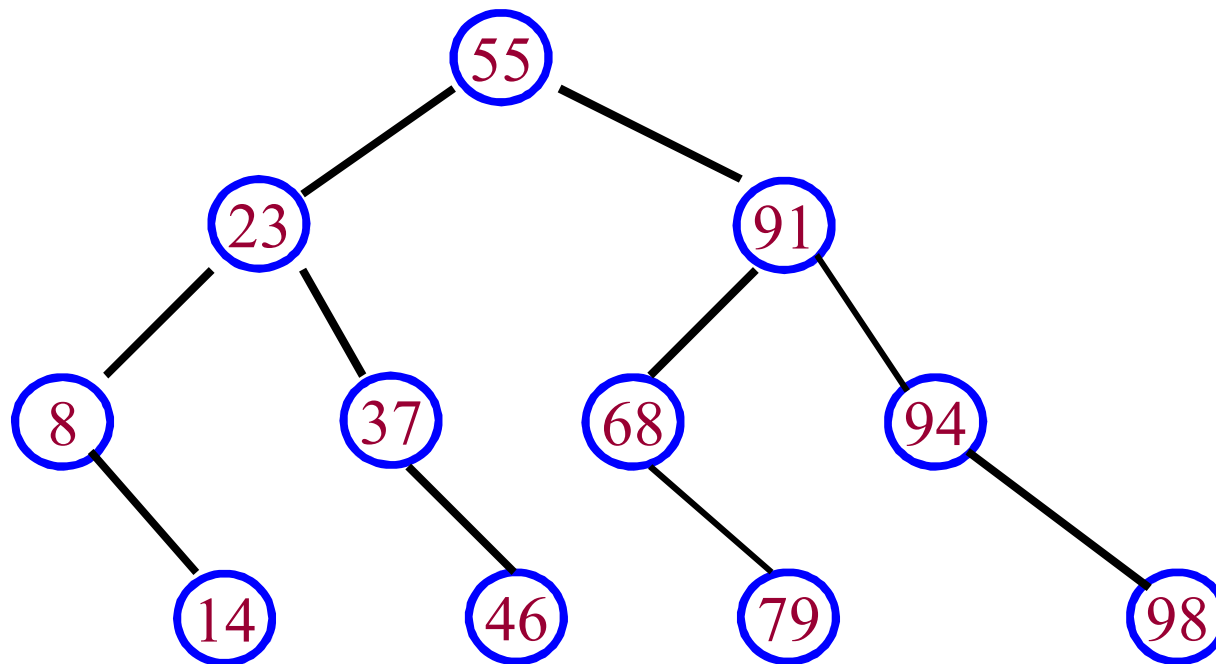
## 9.1 静态查找表--二分查找

- 二分查找的效率高，但是要将表按关键字排序。而排序本身是一种很费时的运算。即使采用高效率的排序方法也要花费  $O(n\log n)$  的时间。
- **二分查找只适用顺序存储结构**。为保持表的有序性，在顺序结构里插入和删除都必须移动大量的数据元素。因此，二分查找特别适用于那种一经建立就很少改动、而又经常需要查找的线性表。
- 对那些查找少而又经常需要改动的线性表，可采用链表作存储结构，进行顺序查找。**链表上无法实现二分查找。**



# 二分查找判定树

| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
|---|----|----|----|----|----|----|----|----|----|----|
| 8 | 14 | 23 | 37 | 46 | 55 | 68 | 79 | 91 | 94 | 98 |



二分查找可用二分查找判定树表示



## 9.1 静态查找表--二分查找

---

- 二分查找成功时的平均查找长度为：

$$ASL_{bn} \approx \log(n+1) - 1$$

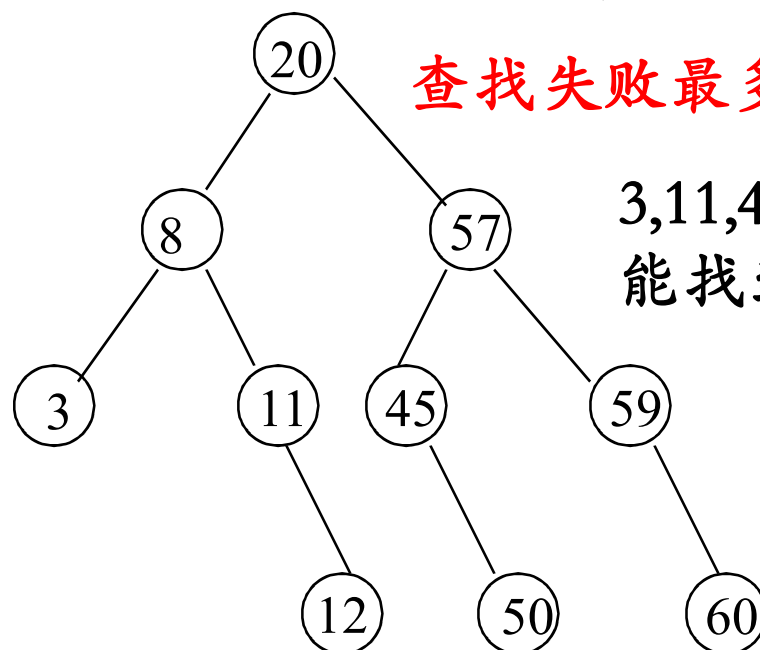
- 二分查找在查找失败时所需比较的关键字个数不超过判定树的深度，在最坏情况下查找成功的比较次数也不超过判定树的深度。即为：

$$\lfloor \log_2 n \rfloor + 1$$



3, 8, 11, 12, 20, 45, 50, 57, 59, 60

$$ASL = (1 + 4 + 12 + 12) / 10 = 2.5$$



查找失败最多比较4次

3, 11, 45, 59 是通过3次比较能找到的数据元素

二分查找判定树示例



## 9.1 静态查找表——索引顺序表的查找

- 线性表分成若干块，每一块中的键值存储顺序是任意的，块与块之间按键值排序，即后一块中的所有记录的关键字的值均大于前一块中最大键值。
- 建立一个索引表，该表中的每一项对应于线性表中的一块，每一项由键域（存放相应块的最大键值）和链域（存放指向本块地一个结点的指针）组成。

# 索引顺序表的查找

## 索引表

最大关键字  
起始地址

|    |    |    |
|----|----|----|
| 22 | 48 | 86 |
| 1  | 7  | 13 |

|    |    |    |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 22 | 12 | 13 | 8 | 9 | 20 | 33 | 42 | 44 | 38 | 24 | 48 | 60 | 58 | 74 | 49 | 86 | 53 |
|----|----|----|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|



## 索引顺序表的查找

- 首先对索引表采用**二分查找**或**顺序**查找方法查找，以确定待查记录所在的**块**。
- 在线性表查找 **$k$** ，若其在线性表中存在，且位于第 **$i$** 块，那么一定有：**第 **$i-1$** 块的最大键值 **$< k \leq$** 第 **$i$** 块的最大键值**。
- 然后在相应的块中进行**顺序查找**。



## 索引顺序表的查找

---

- 表长为 $n$ 的线性表，整个表分为 $b$ 块，前 $b-1$ 块中的记录数为 $s=[n/b]$ ，第 $b$ 块中的记录数小于或等于 $s$ 。
- $ASL=(b+1)/2+(s+1)/2$
- 当  $s = \sqrt{n}$  ，  $ASL$ 最小  $\sqrt{n} + 1$