

Lecture 25: 顺序查找和折半查找

25.1 顺序查找

概念

- 效率：三种查找方式，时间复杂度都是 $O(n)$

25.1.1 顺序查找：原始

概念

- 思想：
 - 凑头到尾每一个都找一次；
- 概念：
 - 顺序查找又称之为线性查找，通常用于线性表；
- 效率：
 - $O(n)$

代码实现

```
typedef struct{           //查找表的数据结构(顺序表)
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}
//顺序查找
int Search_Seq(SSTable ST, ElemType key){
    int i;
    for(i=0; i<ST.TableLen && ST.elem[i]!=key; ++i);
    return i==ST.TableLen?-1:i;
}
```

代码实现：优化

- 优点：实现的逻辑更简单，只需要判断当前查找的元素和当前数组中的元素是否相等即可（因为把当前的目标元素还作为“哨兵”存储在了数组的最头部，然后当前查找是从尾部往头部进行的）
- 效率：

$$ASL_{成功} = \frac{1 + 2 + 3 + \dots + n}{n} = \frac{n + 1}{2}$$

- 代码实现：

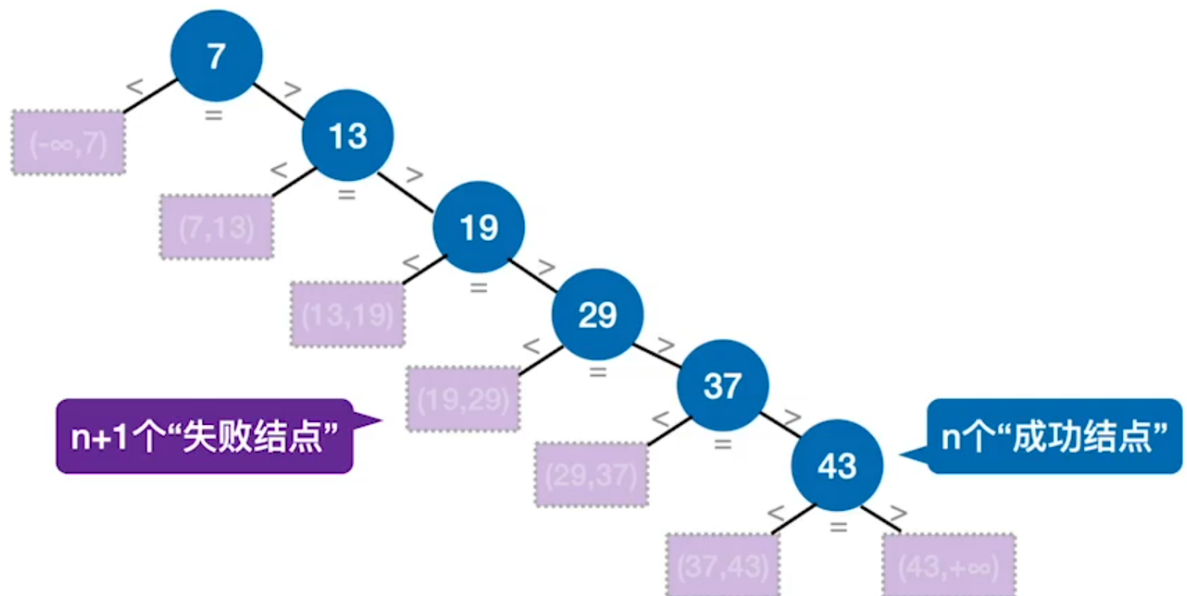
```
//顺序查找的优化
int Search_Seq(SSTable ST, ElemType key){
    ST.elem[0]=key; //哨兵
    int i;
```

```
for(i=ST.TableLen ;ST.elem[i]!=key;--i); //从后往前找
return i; //查找成功返回元素下标，查找失败，则返回0
}
```

25.1.2 顺序查找：基于有序表

概念

- 概念：
 - 元素在顺序表中是顺序排列的，当需要查找时，只需要从 7 开始逐次往下进行判断大小，因此构成了一个判断树；
- 性质：
 - 一个成功结点的查找长度=自身所在层数；
 - 一个失败结点的查找长度=其父节点所在层数；
- 判定树：
 - 方形：失败节点（ $n+1$ 个“失败结点”）；
 - 圆形：成功结点（ n 个“成功结点”）；



25.1.3 顺序查找：基于概率

概念

- 思想：
 - 被查概率大的放在靠前位置；
- 特征：
 - 可以提升查找成功情况下的查找效率，但是对于查找失败的情况并没有什么帮助；
- 举例：
 - 背景：
 - 7:15%
 - 13:5%
 - 19:10%

- 29:40%
- 37:28%
- 43:2%
- $ASL_{成功} = 1 * 0.4 + 2 * 0.28 + 3 * 0.15 + 4 * 0.1 + 5 * 0.05 + 6 * 0.02 = 2.18$

25.2 折半查找

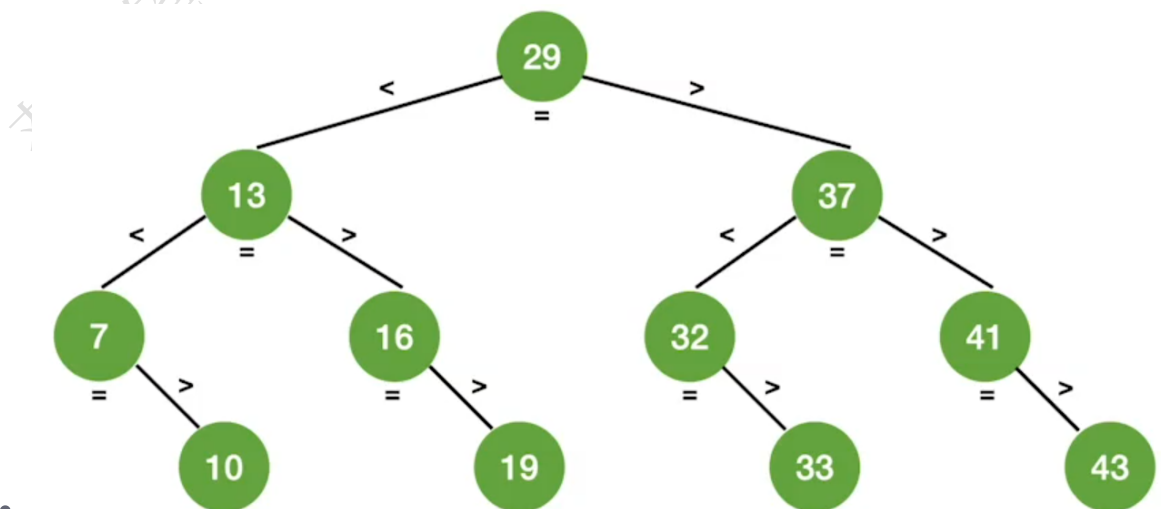
25.2.1 基本概念

概念：折半查找

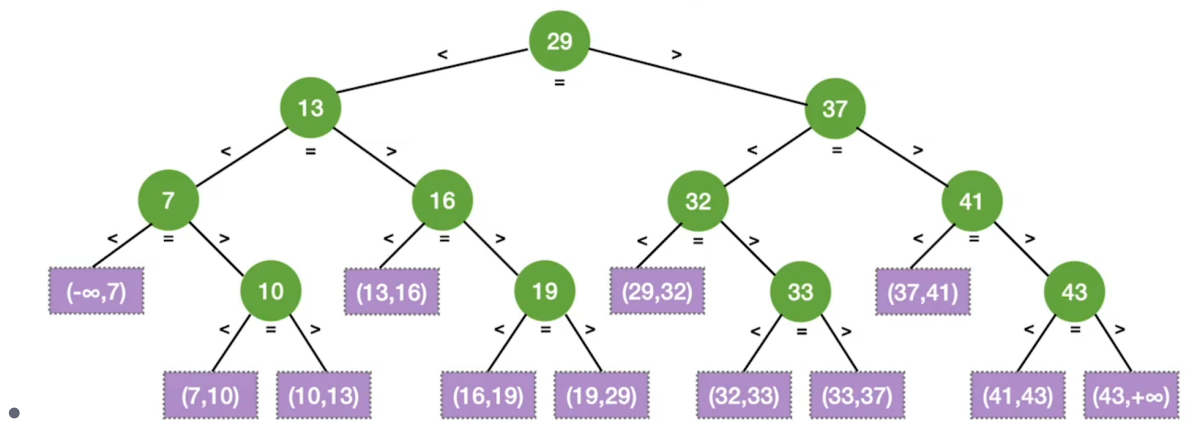
- 思想：
 - 折半查找，又称“二分查找”，仅适用于**有序的顺序表**；
 - 先检查最中间的元素，然后根据和中间元素的大小，分别往左或者往右递归；
 - 递归完一次后：
 - 若元素值小于 mid：让 $high = mid - 1$ ，表示将当前的 high 移到当前 Mid 的左边一个位置；
 - 若元素值大于 mid：让 $low = mid + 1$ ，表示将当前的 low 移到当前 Mid 的右边一个位置；
 - 当 $low > high$ 元素时，则表示查找失败；
- 注意：
 - 因为顺序表拥有随机访问的特性，链表没有；
 - 所以折半查找只能通过顺序表实现；
- 图示：数据结构



- 图示：查找树
 - 查找成功：



- 查找失败：



效率

- 查找成功:

$$ASL_{\text{成功}} = (1 * 1 + 2 * 2 + 3 * 4 + 4 * 4) / 11 = 3$$

- 查找失败:

$$ASL_{\text{失败}} = (3 * 4 + 4 * 8) / 12 = 11/3$$

25.2.2 算法实现

折半查找的实现

```
typedef struct{           //查找表的数据结构(顺序表)
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;

//折半查找
//条件: 增序排列(降序类似)
int Binary_Search(SSTable L, ElemType key){
    int low=0, high=L.TableLen-1, mid;
    while(low<=high){
        mid=(low+high)/2;    //取中间位置
        if(L.elem[mid]==key)
            return mid;      //查找成功则返回所在位置
        else if(L.elem[mid]>key)
            high= mid-1;     //从前半部分继续查找
        else
            low= mid+1;      //从后半部分继续查找
    }
    return -1;              //查找失败, 返回-1
}
```

25.2.3 查找判定树

判定树

- 结论:

- 如果当前 low 和 high 之间有奇数个元素, 则 mid 分隔后, 左右两部分元素个数相等;
- 如果当前 low 和 high 之间有偶数个元素, 则 mid 分隔后, 左半部分比右半部分少一个元素;

- 性质:

- 1. 折半查找的判定树一定是平衡二叉树, 且满足二叉排序树的定义;
- 2. 元素个数为 n 时树高:

- $$h = \lceil \log_2(n + 1) \rceil$$

- 3. 失败节点 = $n+1$;
- 4. 查找成功的 $ASL \leq h$;
- 5. 查找失败的 $ASL \leq h$;

- 效率:

- $$O(\log_2 n)$$

- 公式:

- 折半查找的判定树中, 若右半部分比左半部分多一个元素, 且 $mid = \lfloor (low + high)/2 \rfloor$, 则对于任何一个结点, 必有:

- $$\text{右子树结点数} - \text{左子树结点数} = 0 \text{ 或 } 1$$

- 折半查找的判定树中, 若左半部分比右半部分多一个元素, 且 $mid = \lfloor (low + high)/2 \rfloor$, 则对于任何一个结点, 必有:

- $$\text{左子树结点数} - \text{右子树结点数} = 0 \text{ 或 } 1$$

- 注意:

- 1. 折半查找大部分情况下效率比顺序查找高, 但不能说其效率一定比顺序查找高;
- 2. 如果当前 low 和 high 之间有奇数个元素, 则 mid 分隔后, 左右两部分元素个数相等;

- 执行过程:

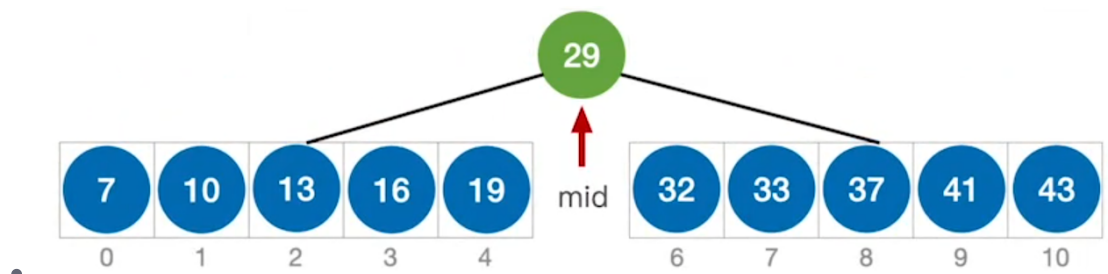
- 奇数:

- 第一次:

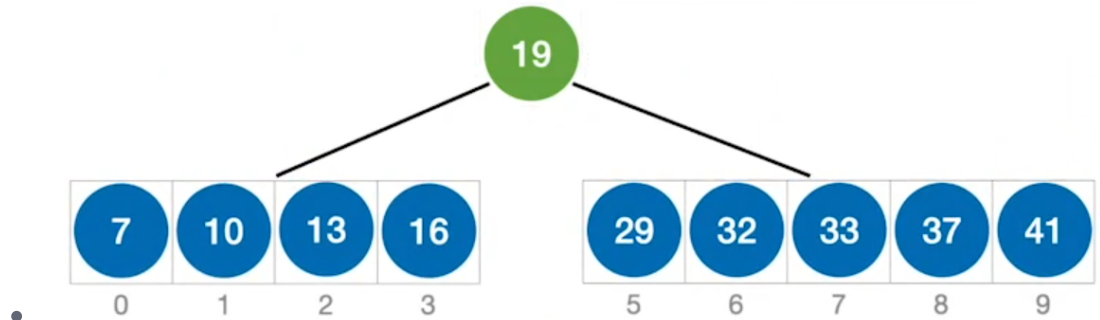
- 如果当前 low 和 high 之间有奇数个元素, 则 mid 分隔后, 左右两部分元素个数相等;



- 第二次: 划分



- 偶数：
 - 划分后：

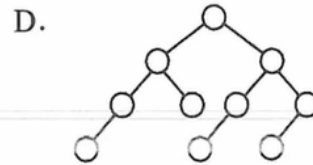
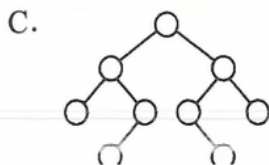
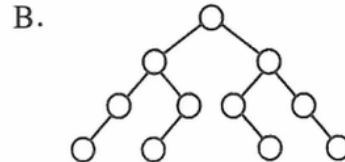
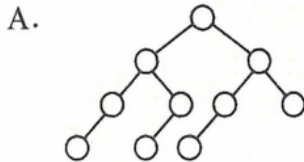


25.2.4 补充：折半查找判定树正确性判断

补充：折半查找判定树正确性判断

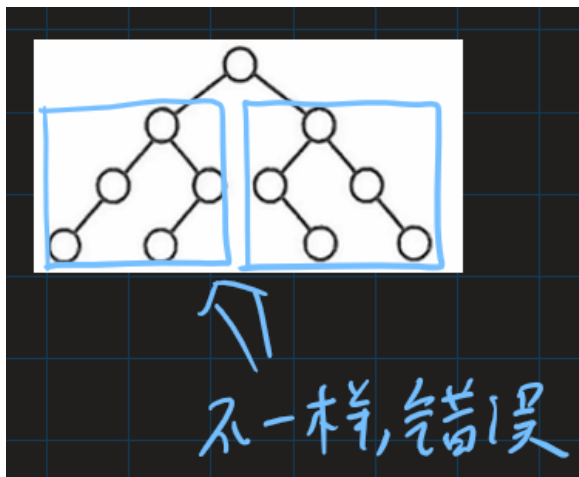
- 题目：

【2017 统考真题】下列二叉树中，可能成为折半查找判定树（不含外部结点）的是（ ）。



- 分析：

- 1. 先找左右子树中元素数量一致的树，若左右子树的数量一致、但左右子树的结构不是完全一样，则错误；



- 任意两组相同数目的序列，其构成的折半查找判定树、其结构是一样的；
- 因为折半查找判定树的构建过程与序列中的元素值无关，**只和当前序列中元素的个数有关**；
 - 树(或者子树)的root = 新的mid = $(low_index + high_index)/2$

- 2. 左右子树中元素数量不一致时，依据以下规则、判定是否当前树是何种取整方式

• 若根节点的左子树个数 > 右子树个数，则当前是 向上取整

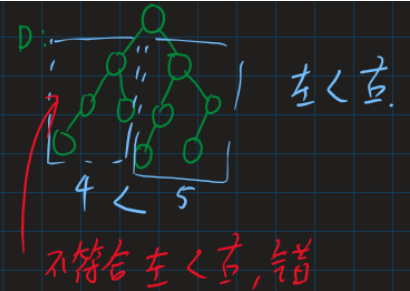
12345678 -> $mid = (1+8)/2 = 4.5 = 5$

1234 5 678

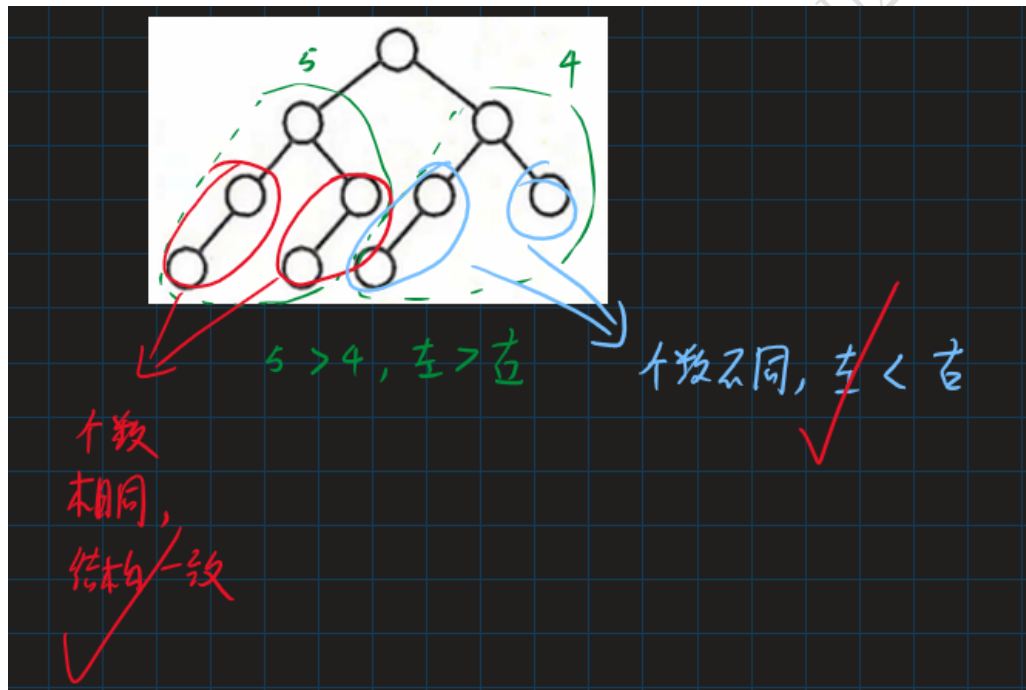
• 若根节点的左子树个数 < 右子树个数，则当前是 向下取整

12345678 -> $mid = 4.5 = 4$

123 4 5678



- 3. 根据计算得到的取整方式，判断其子树是否符合当前的取整方式。若不符合则不是合理判定树，符合则表示是合理的判定树；



25.3 分块查找

25.3.1 基础概念

概念

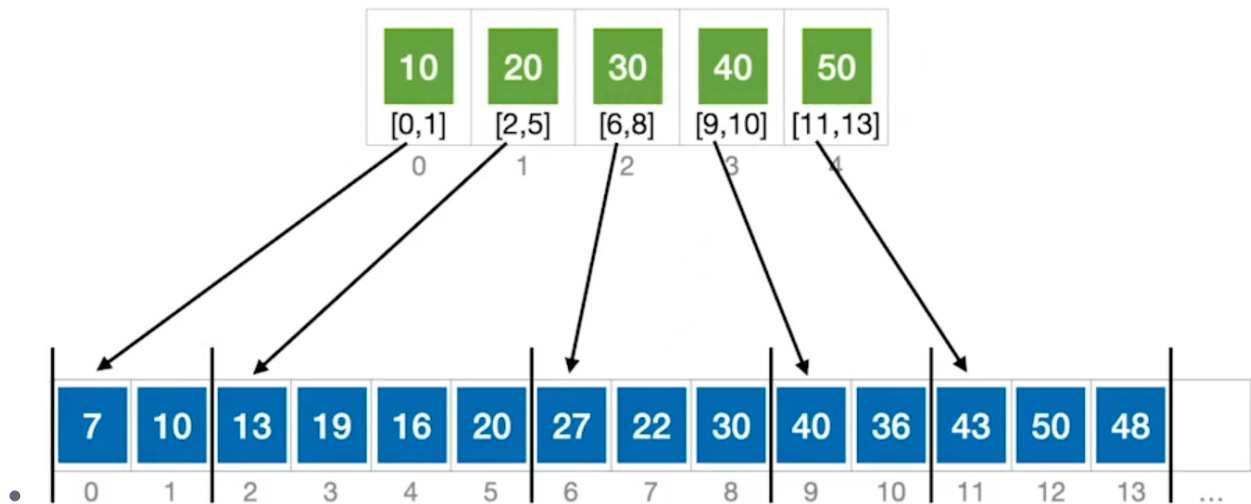
- 概念：
 - 分块查找，又称索引顺序查找；
 - 1. 在索引表中确定待查记录所属的分块 (可顺序、可折半)；
 - 2. 在**块内顺序查找**；
- 思想：
 - 索引表中保存每个分块的最大关键字和分块的存储区间；
 - 块内无序、块间有序；
- 效率：

- 如图示：共有 14 个元素，各自被查概率为 $1/14$
- 若索引表采用顺序查找，则 7:2 次、10:3 次、13:3 次...
- 若索引表采用折半查找，则 30:4 次、27:2 次？

• 图示：数据结构



• 图示：建立索引



25.3.2 代码实现

```
//索引表
typedef struct{
    ElemType maxValue;
    int low,high;
}Index;

//顺序表存储实际元素
ElemType List[100];
```

25.3.3 特殊情况分析

分析

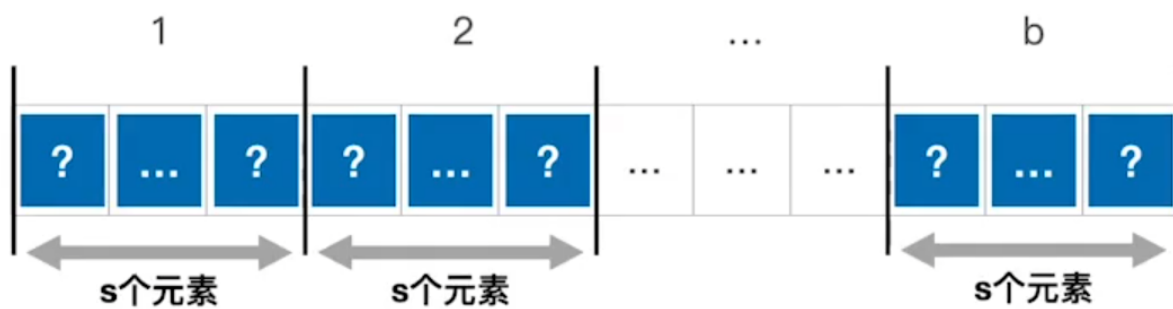
- 背景：
 - 假设，长度为 n 的查找表被均匀地分为 b 块，每块 s 个元素
- 分析：
 - 设索引查找和块内查找的平均查找长度分别为 L_I 、 L_S ，则分块查找的平均查找长度为：

$$ASL = L_I + L_S$$

- 用顺序查找查索引表，则： $L_I = \frac{(1+2+\dots+b)}{b} = \frac{b+1}{2}$, $L_S = \frac{(1+2+\dots+s)}{s} = \frac{s+1}{2}$
- 则： $ASL = \frac{b+1}{2} + \frac{s+1}{2} = \frac{s^2+2s+n}{2s}$,
- 结论：最好效率情况

• 当 $s = \sqrt{n}$ 时, $ASL_{\text{最小}} = \sqrt{n} + 1$

• 图示:



获取笔记持续更新可加微信: WandD2277