查找

吉林大学计算机学院
谷方明
fmgu2002@sina.com

学习目标

- 掌握线性表上的顺序查找和自组织表；
- 掌握有序线性表上的折半查找，理解有序线性表上的一致折半查找和**Fibonacci**查找；
- 掌握基于分布信息的插值查找；
- 了解分块查找；

查找问题

- 计算机出现以前，对数表、三角函数表等已广泛发布，使得数学计算可以通过查表来完成。
- 计算机一出现，人们便发现：用计算机程序查找比人工查表更合算。但关于查找的研究比较少。
- 随着随机存储器的迅速发展，存储容量越来越大，查找逐渐成为令人感兴趣的问题；（**1950s**）
- 现在，查找是许多计算机程序中最耗费时间的部分,查找算法的优劣与查找操作的速度密切相关,从而极大的影响包含查找算法之程序的效率；

术语

- 查找表：是由同一类型的数据元素构成的集合。
- 关键词： 可标识数据元素的域
- 查找：在查找表中找出满足条件的数据元素
- 查找结果：查找成功、查找失败
  - ✓ 查找与插入：查找失败后，插入关键词新记录
- 平均查找长度：查找一个结点所作的平均比较次数（衡量一个查找算法优劣的主要标准）

查找算法特性

1. 内外有别：分为内查找和外查找；
2. 静态动态：静态查找查找时，表的内容不变；动态查找查找时，频繁地把新记录插入到表中，或者从表中删除记录，即表之内容不断地变化；
3. 原词变词：原词系指用原来的关键词，变词指使用经过变换过的关键词；
4. 数字文字：指比较的时候是否用数字的性质

线性表——顺序查找

- 从表头依次向后查找，直到匹配关键词或失败

```
int sfind( int A[ ], int N, int K)   //教材上算法S
{
    int i;
   for(i=1;i<=N;i++)  if(K==A[i]) return i;
   return -1;
}
```

改进1——引入虚拟记录（监视哨）

```
int qfind (int A[],int N,int K) //教材上算法Q
{
     A[N+1]=K;
     int i=1;
     while( K != A[i] ) i++;
     return i<=N ? i : -1;
}
```

- 加速原理：减少了**1**个比较条件，省了**20%**

改进1’（不推荐）

- 推进两步

```
int qfind(int A[],int N,int K)  //教材上算法Q’
{
       A[N+1]=K;
       int i=-1;
   q2:i += 2;
       if(K==A[i])  goto q5;
       if(K!=A[i+1]) goto q2; else i++;
   q5:return i<=N ? i : -1;
}
```

- 加速原理：减少了循环变量的加法操作，又省了10%

时间复杂性分析

- 成功查找的平均查找长度：

$$E(n)=\sum_{i=1\cdots n}P_i\times C_i$$

$$=\frac{1}{n}\sum_{i=1\cdots n}C_i=\frac{1}{n}\sum_{i=1\cdots n}i=\frac{1}{n}\frac{n(n+1)}{2}$$

$$=\frac{n+1}{2}\approx 0.5n$$

- 查找失败的查找长度：$n+1$
- 顺序查找的期望时间复杂性：$O(n)$

- 若表中 $P_1 \geq P_2 \geq \ldots \geq P_n$时，$E(n)$取最小值；若表中元素之概率是递增的，则$E(n)$取最大值。由此可见, 表中元素的不同排列 (按元素发生的概率P_i) 将影响顺序查找算法的时间复杂性。
- 启发：将经常出现的元素（概率大）自动向表前端移动(将不经常出现的元素自动向表后端移动)

自组织表(Self-Organizing List)

- **两种基本操作**
  - access(x)：访问元素x，耗时正比x在表中的位置
  - transpose(x)：交换x，**链表**耗时为O(1)。
- **交换策略**
  - MOVE-TO-FRONT (MTF)
  - MOVE-AHEAD-ONE
- **分析**
  - 最坏：每次都访问最后元素, s*n. （上帝视角构造）
  - 平均：$\sum p_i * i$ （拓展：竞争分析）
- **MTF应用：流行词 搜索（序列局部反应较好）**

有序表——顺序查找

- 利用序关系可以减少查找失败时间

```
int tfind(int A[],int N,int K) //教材上算法T
{
    A[N+1]=∞;
    int i=1;
    while( K > A[i] ) i++;
    return A[i]==K ? i : -1;
}
```

充分利用序关系

- 有序表中，K 和 K_i 比较后，有三种情况:

  $K < K_i$　　　$K = K_i$　　　$K > K_i$

① $K < K_i$，[不必再考虑子表 $R_i , R_{i+1} , \dots , R_N$]

② $K = K_i$，[查找成功结束]

③ $K > K_i$，[不必再考虑子表 $R_1 , R_2 , \dots , R_i$]

- 确定 *i* 的规则，设计算法
  - ✓ 最简单的二分

有序表——对半查找(二分查找)

- 算法思想： *K*与待查表中间记录比较

```
int bfind(int a[ ],int N,int K)   //教材上算法B
{
    int s = 1, e = N, i;
    while( s <= e ){
        i = ( s + e ) / 2;
        if ( a[i]==K ) return i;
        else if ( a[i]<K ) e = i-1;
        else s = i+1;
    }
    return -1;
}
```

例：查找 K=96 时二分查找过程(4次比较成功)

1	2	3	4	5	6	7	8	9	10
12	23	26	37	54	60	68	75	82	96
↑s=1									↑e=10
12	23	26	37	54	60	68	75	82	96
↑s=1				↑i=5					↑e=10
12	23	26	37	54	60	68	75	82	96
					↑s=6		↑i=8		↑e=10
12	23	26	37	54	60	68	75	82	96
								↑i=9 ↑s=9	↑e=10
12	23	26	37	54	60	68	75	82	96
									↑i=10 s=e=10↑

例查找 *K*=58 时的对半查找过程(3次失败)

1	2	3	4	5	6	7	8	9	10
12	23	26	37	54	60	68	75	82	96

12 23 26 37 54 60 68 75 82 96

s=1 i=5 e=10

12 23 26 37 54 60 68 75 82 96

s=6 i=8 e=10

12 23 26 37 54 60 68 75 82 96

s=6 i=6 e=7

12 23 26 37 54 60 68 75 82 96

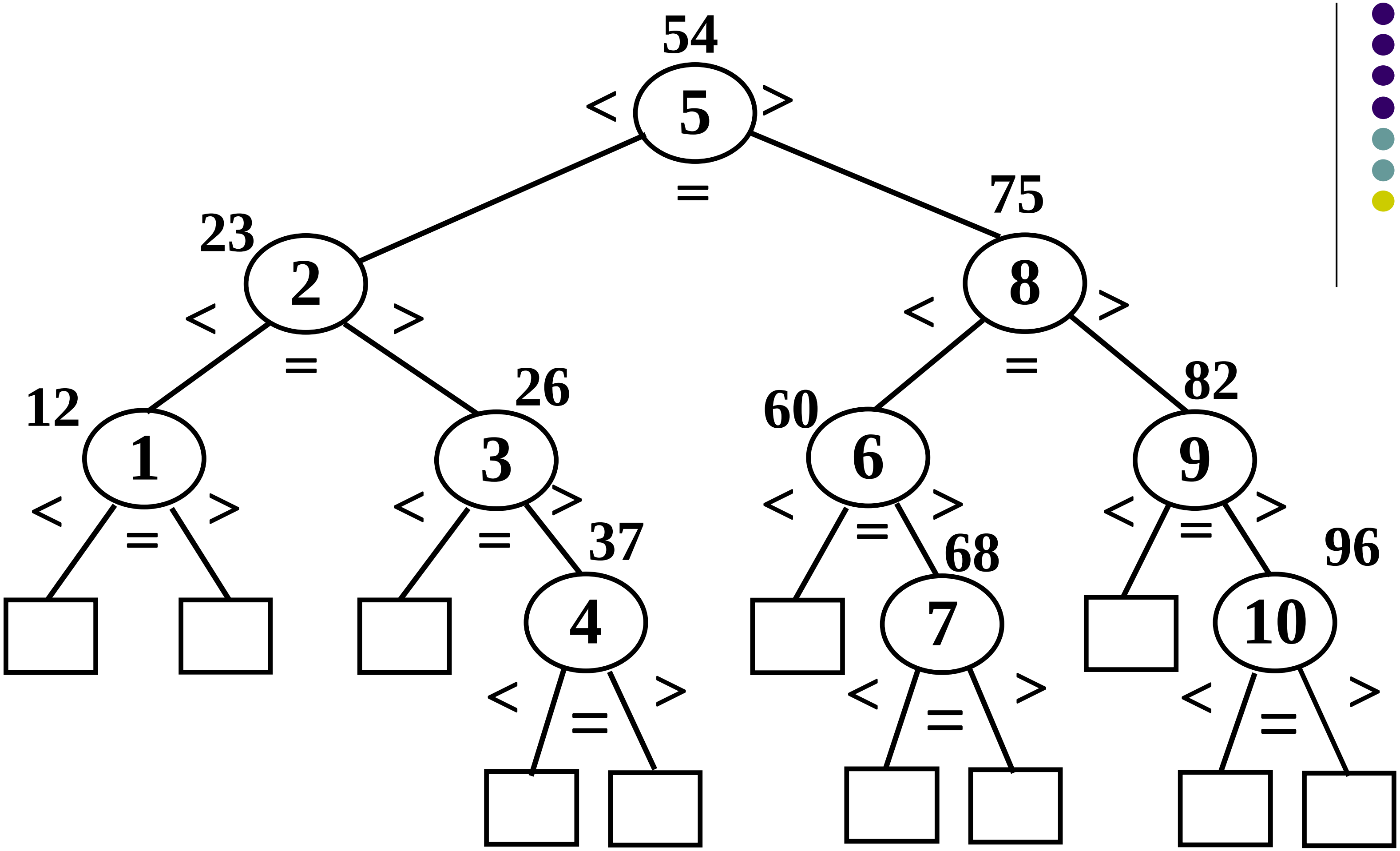
e=5 s=6

对半查找算法分析

- 二分查找判定树，T (s , e)的递归定义如下：

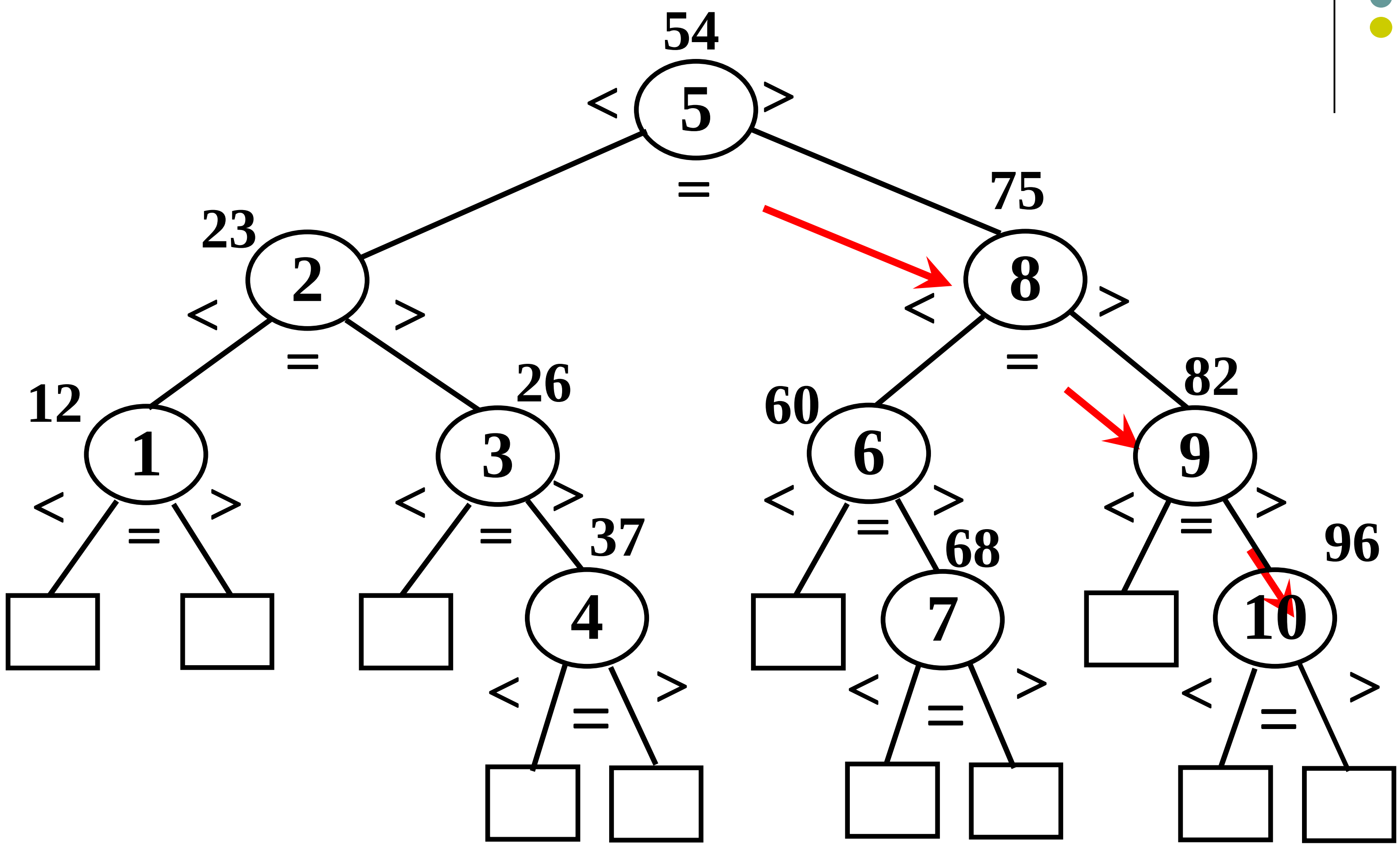
① 当 $e–s+1 \le 0$（$s = e+1$）时， T (s , e) 为空树；

② 当 $e–s+1 > 0$（$s \le e$）时，二分查找判定树的根结点是有序表中序号为$\lfloor(s+e)/2\rfloor$的记录;根结点的左子树是与有序表 R_s , R_{s+1}, … , $R_{\lfloor(s+e)/2\rfloor-1}$ 对应的二分查找判定树，根结点的右子树是与有序表 $R_{\lfloor(s+e)/2\rfloor+1}$, $R_{\lfloor(s+e)/2\rfloor+2}$, … , R_e 对应的二分查找判定树.

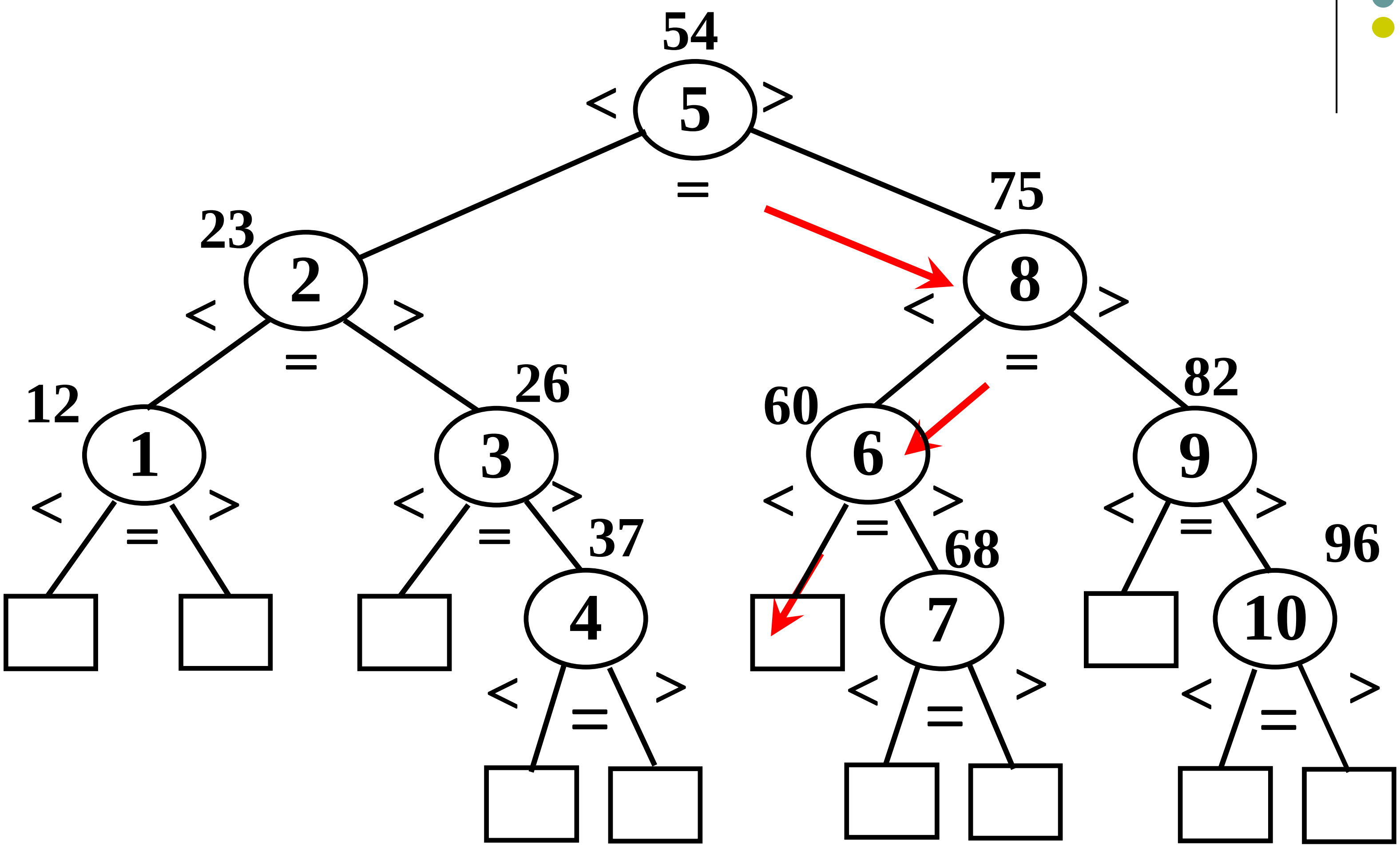


序列“12, 23, 26, 37, 54, 60, 68, 75, 82, 96”对半查找对应的二叉判定树T(1, 10) . 树中每个圆圈结点表示关键词比较 $K : K_i$， 每条边表示比较结果 。

搜索 K=96 成功的情况：



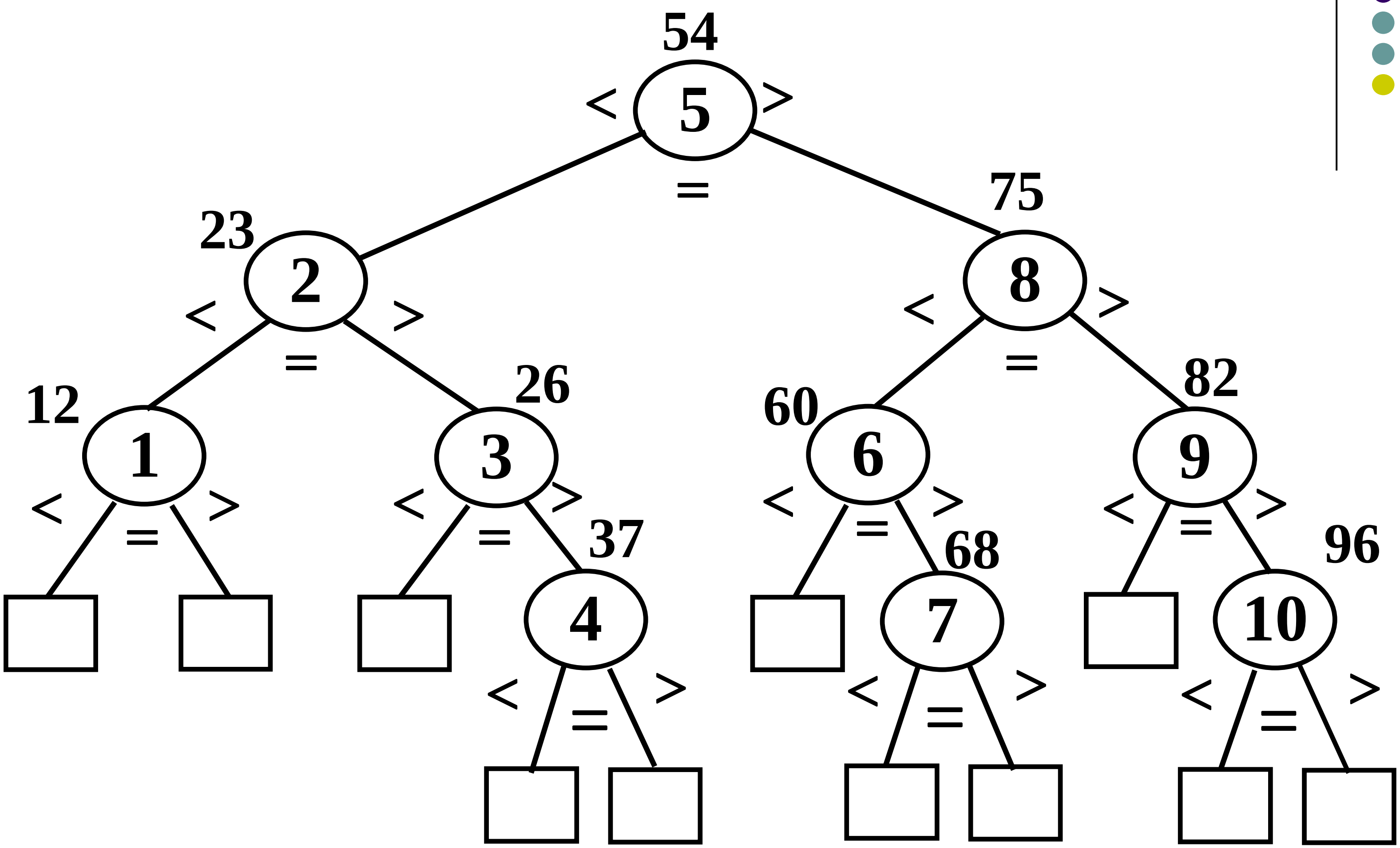
搜索 K=58 失败的情况



分析

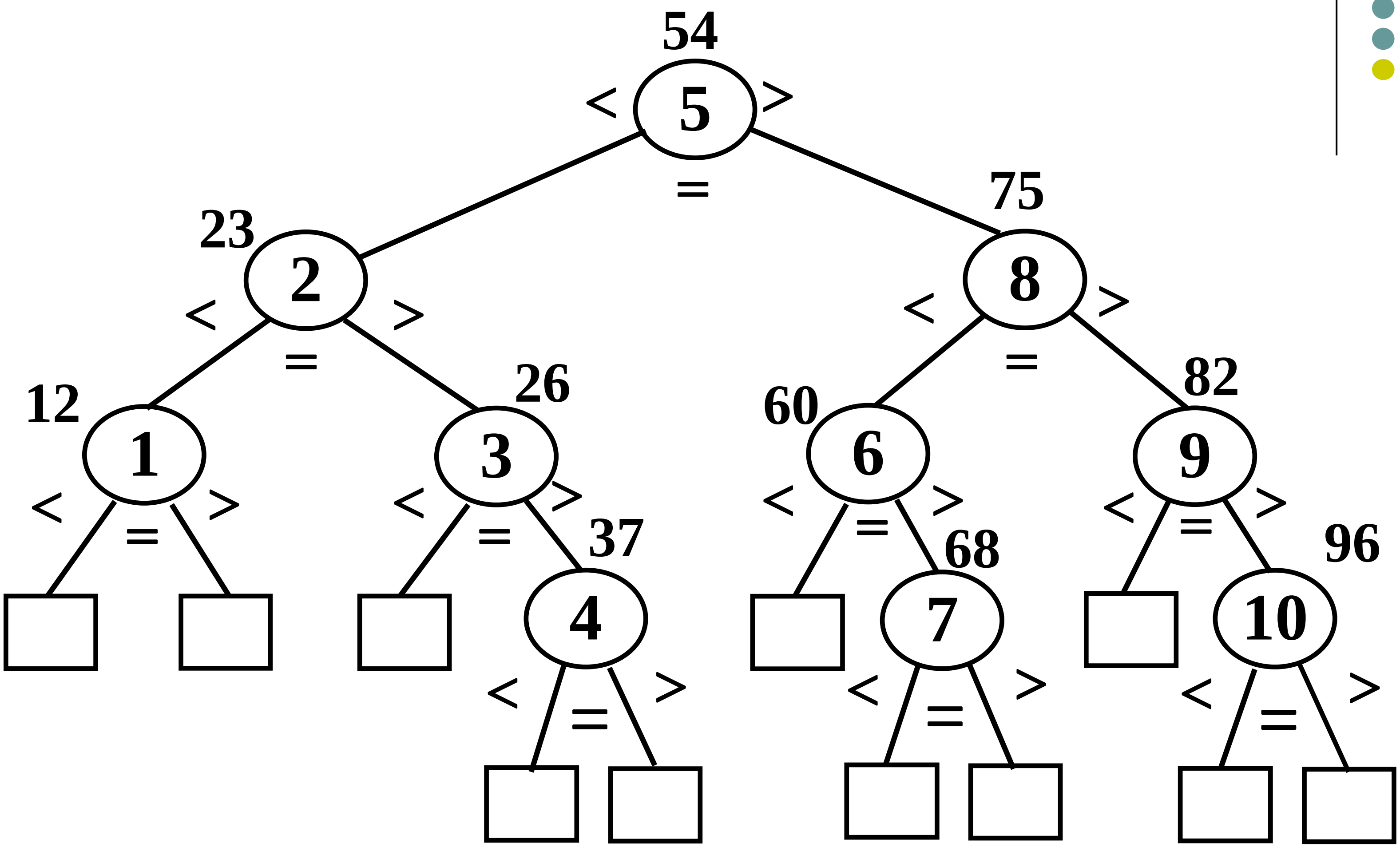
- 算法B 的第一个比较是 $K:K_5$ ，通过图中的根结点来表示。若$K<K_5$，则算法沿着左子树进行；若$K>K_5$，则算法沿着右子树进行；
- 每次成功查找对应判定树的一个内结点，关键词的比较次数：树根到该结点的路径长度加1；
- 每次不成功查找对应判定树的一个外结点，关键词的比较次数：根节点到该外结点的路径长度。
- 外结点编号为 $\boxed{0}$ 到 $\boxed{N}$ 。结点 $\boxed{i}$ ，当且仅当 $K_i < K < K_{i+1}$ ；

T(1, 10)查找成功的平均查找长度



$$ASL_{SUCC} = (1\times1+2\times2+3\times4+4\times3)/10 = 29/10 = 2.9$$

T(1, 10)查找失败的平均查找长度



$$ASL_{UNSUCC} = En/(n+1)=(3\times5+4\times6)/11 = 39/11$$

折半查找——效率分析

- 引理8.1

设算法B对 N 个记录的成功与不成功查找都是等概率的，则对于成功查找关键词的平均比较次数为 $S_N=1+I_N / N$，对于不成功查找关键词的平均比较次数为$U_N=E_N / (N+1)$，其中I_N，E_N分别为$T(1, N)$的内、外路径长. 则有 $S_N = (1+1/N)\, U_N - 1$.

证明：外路径长比内路径长大$2N$，故成立

- 引理8.2

二分查找判定树 $T(s, e)$ 的高度h是$\lceil \log(e-s+2) \rceil$.

令$N= e-s+1$，表记录数，$h = \lceil \log(N+1) \rceil = \lfloor \log N \rfloor +1$.

引理8.2证明：数学归纳法

- 基础：**e-s=0时,成立**
- 归纳：**假设e-s=m时成立，则当$e-s=$ m+1时，**
  - ✓ $h(T(s,e)) = \max\{h(T(s, \lfloor (e+s)/2 \rfloor - 1)),\ h(T(\lfloor (e+s)/2 \rfloor + 1, e))\} + 1.$
  - ✓ 令$h(T(s,e)) = h(T(\lfloor (e+s)/2 \rfloor + 1, e))+1$，因 $(e+s)/2 \geq \lfloor (e+s)/2 \rfloor \Rightarrow e - (\lfloor (e+s)/2 \rfloor + 1) \geq \lfloor (e+s)/2 \rfloor - 1 - s.$
  - ✓ 根据归纳假设，$h(T(s,e)) = \lceil \log_2(e - \lfloor (s+e)/2 \rfloor + 1) \rceil + 1 = \lceil \log_2(2e - 2\lfloor (s+e)/2 \rfloor + 2) \rceil$
  - ➢ 当s+e为偶数时，$h(T(s,e)) = \lceil \mathbf{log}(e-s+2) \rceil$.
  - ➢ 当s+e为奇数时，$h(T(s,e)) = \lceil \mathbf{log}(e-s+3) \rceil$. **此时, log($e$–$s$+2)不能是整数，故$\lceil \log(e-s+3) \rceil = \lceil \log(e-s+2) \rceil$.**

引理8.3

- **设T(1 , N) 是 N 个内结点的二分查找判定树，若不考虑外结点T(1, N)的高度为k，则T(1, N)之外结点均属于k 或 k+1层.**
- **证明：**对N用数学归纳法.

N = 1时结论成立. 假定结点数小于N (N >1)时成立，

- 设T(1, N) 的左子树形为T_1，右子树形为T_2，且T_1有m个内结点，由于T_2的内结点数等于m或m +1，不妨假定T_2有m +1个内结点.
- 假定T_1和T_2的高度分别为r和q(不考虑外结点),由归纳假设知:T_1的外结点或都在r +1层或在r层和r +1层，T_2的外结点或都在q +1层或在q和q +1层

- 上述情况有四种组合，分别讨论:

- ✓ (1) T_1 的外结点都在$r+1$层，T_2 的外结点都在$q+1$层。这不可能，两棵满二叉树的内结点数不能相差1
- ✓ (2) T_1 的外结点都在$r+1$层，T_2 的外结点在q和$q+1$层

T_1 的内结点数$m = 2^{r+1} - 1$，T_2 的内结点数$m+1$大于$2^q - 1$，小于$2^{q+1} - 1$，因此$2^q - 1 < 2^{r+1} - 1 + 1 < 2^{q+1} - 1$，显然有$r+1 = q$

- ✓ (3) T_1 的外结点在r层和$r+1$层，T_2 的外结点都在$q+1$层.
- ✓ (4) T_1 的外结点在r层和$r+1$层，T_2 的外结点在q和$q+1$层.

$2^r - 1 < m < 2^{r+1} - 1$ 和$2^q - 1 < m + 1 < 2^{q+1} - 1$，即

$2^r - 1 < m < 2^{q+1} - 2$，$2^q - 2 < m < 2^{r+1} - 1$.

m为自然数，故有 $2^r - 1 + 1 < 2^{q+1} - 2$，$2^q - 2 + 1 < 2^{r+1} - 1$，

由此可$q - 1 < r < q + 1$，$r = q$.

定理 8.1

- 在最坏情况下，算法B的关键词比较次数为 $\lceil \log(N+1) \rceil$；期望复杂性等于O(log$N$) .
  - ✓ E(n) = S_N * p + U_N * q = O (log N)
  - ✓ $U_n = (N+1)(\lfloor \log N \rfloor + 2) - 2^{\lfloor \log N \rfloor + 1}$

一致对半查找

- 对半查找改进策略：使用三个指针 (*s*、*i* 和 *e*) 减少所用指针的数量，减少维护计算。

- 一种具体思路是：用当前的位置 i 及其变化率 δ;
- **N = 10 ?**
- **N = 11 ?**
- 覆盖全部：多覆盖正确，少覆盖不正确
- $x = \lfloor x/2 \rfloor + \lceil x/2 \rceil$，因此 $i \leftarrow \lceil N/2 \rceil$，$m \leftarrow \lfloor N/2 \rfloor$

- 在每次 $K > K_i$ 或 $K < K_i$ 之后:

  置 $i \leftarrow i \pm \delta$ 和 $\delta \leftarrow \delta/2$（近似地）.

- N = 10 ?
- N = 11 ?

- $i \leftarrow i \pm \lceil m/2 \rceil$, $m \leftarrow \lfloor m/2 \rfloor$

算法U（N，R，K.i）

/*若 N为偶数，算法 U 涉及虚拟关键词$K_0 = -\infty$ */

U1. [初始化] 置$i \leftarrow \lceil N/2 \rceil$，$m \leftarrow \lfloor N/2 \rfloor$.

U2. [比较]若 K < Ki , 自然到步骤 U3；若 $K > K_i$, 转到U4; 若 $K = K_i$, 则算法成功结束.

U3. [减小i]若m =0, 则算法以失败告终; 否则置$i \leftarrow i - \lceil m/2 \rceil$; 然后置$m \leftarrow \lfloor m/2 \rfloor$ 并返回U2.

U4. [增大i]若m = 0, 则算法以失败告终;否则置$i \leftarrow i + \lceil m/2 \rceil$; 然后置$m \leftarrow \lfloor m/2 \rfloor$并返回U2. ▌

$\delta = \lceil m/2 \rceil, m = 5, 2, 1, 0$

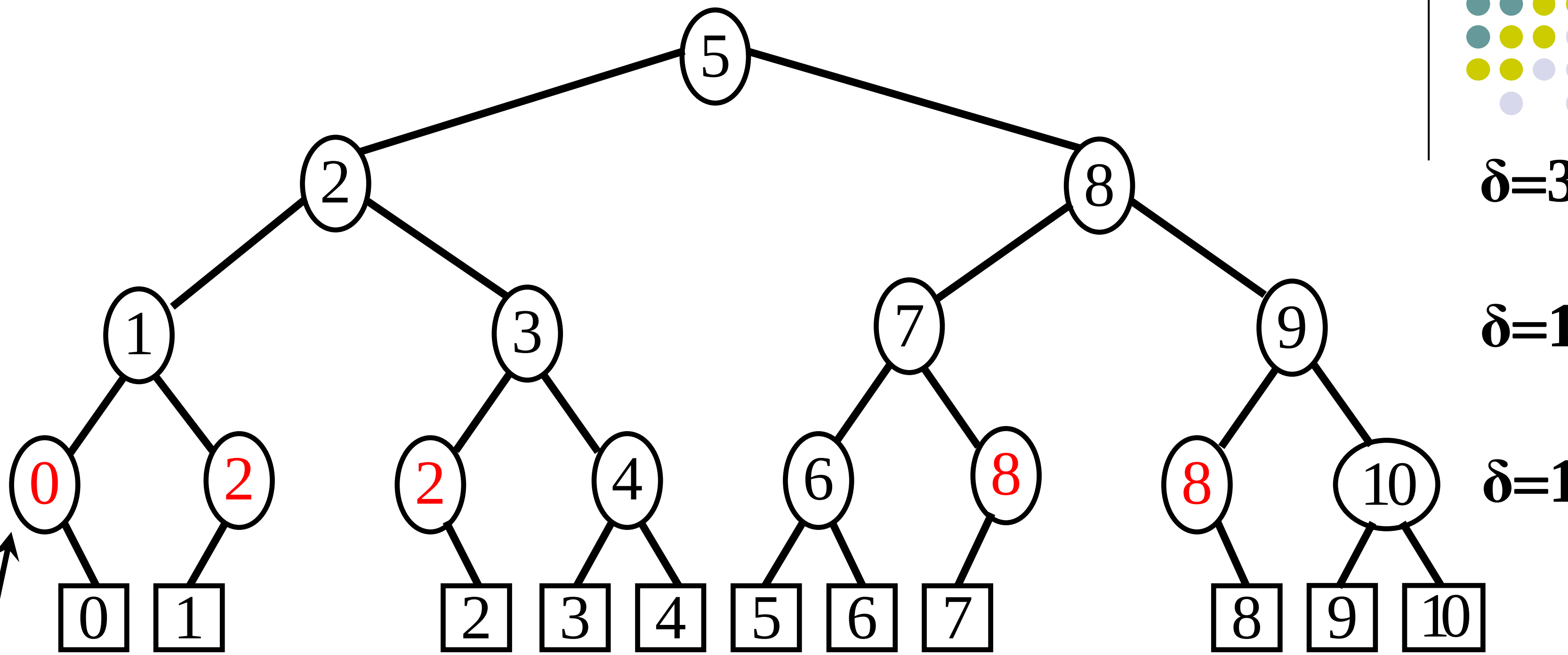


图8.4 当 N=10 时一株“一致”的二叉判定树

$K_0 = -\infty$

一致对半查找判定树的性质

- 外结点都集中在最下一层；可能有度为**1**的内结点
- 由于重复覆盖，内结点可能有重复；查找成功时，比较次数相同；查找失败，可能作一次冗余比较.
- **U** 被称为一致的原因是**:** 在 **l** 层上的一个结点的编号与其父亲结点的编号之差的绝对值，对 **l** 层上的所有结点均为同一个常数 δ ．

m 序列

- $\lfloor N/2 \rfloor$, $\lfloor \lfloor N/2 \rfloor /2 \rfloor$, $\lfloor \lfloor \lfloor N/2 \rfloor /2 \rfloor /2 \rfloor$,…, 0
- 证明：$\lfloor \lfloor N/2^k \rfloor /2 \rfloor = \lfloor N/2^{k+1} \rfloor$

  若$2^k \mid N$，显然成立；否则，有

  $N/2^k = Z$ .. R，Z是非负整数，$R<2^k$

  $\lfloor \lfloor N/2^k \rfloor /2 \rfloor = \lfloor Z/2 \rfloor$

  $\lfloor N/2^{k+1} \rfloor = \lfloor Z/2 + R/2^{k+1} \rfloor$， $R/2^{k+1} < ½$

  Z为偶数，相等； Z为奇数，也相等。

$$\lfloor n/2 \rfloor, \lfloor n/2^2 \rfloor, \lfloor n/2^3 \rfloor, \cdots, 0$$

中点 i 序列

- $\lceil N/2 \rceil$， $\lceil N/2 \rceil$+/-$\lceil \lfloor N/2 \rfloor /2 \rceil$,

$\lceil N/2 \rceil$+/-$\lceil \lfloor N/2 \rfloor /2 \rceil$+/-$\lceil \lfloor N/2^2 \rfloor /2 \rceil$，……

性质1：x为整数时，$\lceil x/2 \rceil = \lfloor (x+1)/2 \rfloor$

当x为奇数时，相等；当x 为偶数时，相等

性质2：$\lfloor x \rfloor + 1 = \lfloor x+1 \rfloor$

证明：$\lceil \lfloor N/2^{k-1} \rfloor /2 \rceil = \lfloor (\lfloor N/2^{k-1} \rfloor +1)/ 2 \rfloor$

$= \lfloor (N+2^{k-1}) / 2^k \rfloor$

$$\left\lfloor \frac{n+2^0}{2^1} \right\rfloor, \left\lfloor \frac{n+2^0}{2^1} \right\rfloor + / - \left\lfloor \frac{n+2^1}{2^2} \right\rfloor,$$

$$\left\lfloor (n+2^0)/2^1 \right\rfloor + / - \left\lfloor (n+2^1)/2^2 \right\rfloor + / - \left\lfloor (n+2^2)/2^3 \right\rfloor, \cdots$$

- 于是算法 U 又可以改进：在运行期间，不去计算m及i值，而是使用一张辅助表

$$DELTA[j] = \left\lfloor \frac{(N + 2^{j-1})}{2^j} \right\rfloor$$

$$for \quad 1 \le j \le \lfloor \log_2 N \rfloor + 2$$

算法C（N，R，$K.i$）

C1. [初始化] 置 $i \leftarrow$ DELTA[1]. $j \leftarrow 2$.

C2. [比较]若$K = Ki$， 则算法成功结束.

若$K < Ki$，则转 C3；

若$K > Ki$，则转 C4；

C3. [减小 i] 若 DELTA[j] = 0，则算法以失败告终；否则置 $i \leftarrow i -$ DELTA[j]. $j \leftarrow j + 1$，并转 C2 .

C4. [增大 i] 若 DELTA[j] = 0，则此算法以失败告终；否则，置 $i \leftarrow i +$ DELTA[j]. $j \leftarrow j + 1$，并转C2 . $\blacksquare$

算法C参考实现

```
int cfind(int A[],int n,int key) {   //n>0
        int i=delta[1], j;
        if( key==A[i]) return i;
        for(j=2;delta[j]>0;j++){
            if( key < A[i] ) i-=delta[j];else i+=delta[j];
             if( key==A[i]) return i;
        }
        return -1;
}
```

时间复杂度分析

- 成功的查找：算法C对应的二叉判定树与算法B对应的二叉判定树有相同的内路径长，所以平均比较次数与算法B一样。(虽然有重复结点，但成功时，只会触发一个，并在第一次触发时查找结束)
- 不成功的查找：算法C总是恰好进行$\lfloor \log_2 N \rfloor+1$次比较，比算法B的$\lceil \log_2(N+1) \rceil$比较次数多。
- 算法 C 中的算术运算仅包含加减法，且在算法运行期间未计算诸 m 之值，而是用一张辅助表来代替，从而明显提高了速度。算法C的时间花费不足算法B的二分之一。

斐波那契查找

- 对半查找改进策略2： 划分策略
- 斐波那契（Fibonacci）查找

  对半查找的替代，以Fibonacci序列的分划代替了对半查找的均匀分划。

- **Fibonacci 序列：**

0，1，1，2，3，5，8，13，21，34，…

$F_0=0$，$F_1=1$，

$F_j=F_{j-1}+F_{j-2}$，$j \geq 2$

$\lim F_k / F_{k+1} = 0.618$

黄金分割

假设有一个长度为F_{k+1}–1的文件，其记录下标[1，F_{k+1}–1]。记录 F_k 将文件分为三个部分：

① 左子文件[1，F_k–1]；

② F_k；

③ 右子文件[F_k+1，F_{k+1}–1]；

其中，左、右子文件的大小分别为F_k–1，F_{k-1}–1，故左、右子文件还可继续进行上面的划分过程。

$\lim (F_k-1) / (F_{k+1}-1) = \lim F_k / F_{k+1} = 0.618$

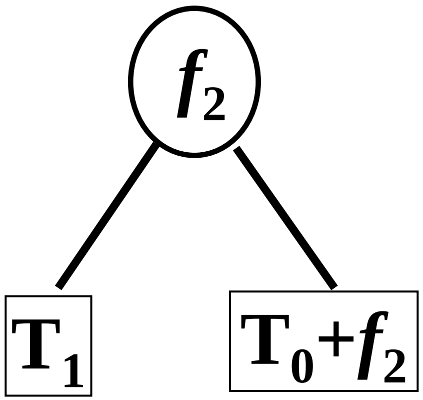
k 阶斐波那契树T_k的递归定义

(1) 当 $k = 0, 1$时，T_k为空树；

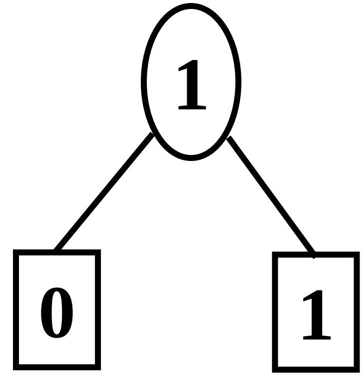
(2) 当 $k > 1$ 时，二叉判定树根是有序表中序号为f_k的记录，根结点的左子树是与有序表R_1，R_2，…，R_{f_k-1}对应的T_{k-1}是k−1阶斐波那契树，根为f_{k-1}；f_{k-1}

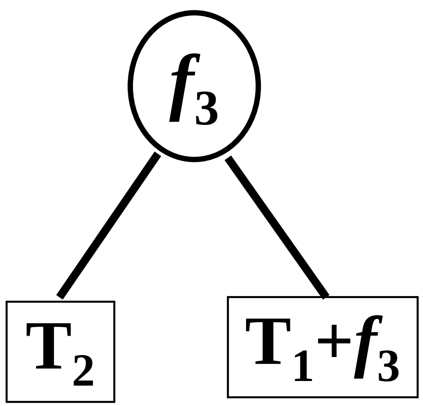
根结点的右子树是与有序表R_{f_k+1}，R_{f_k+2}，…，$R_{f_{k+1}-1}$对应的$k-2$阶且所有结点之编号都增加f_k的斐波那契树$T_{k-2}+f_k$，其根为f_k+f_{k-2}。

$f_{k-2}+f_k$

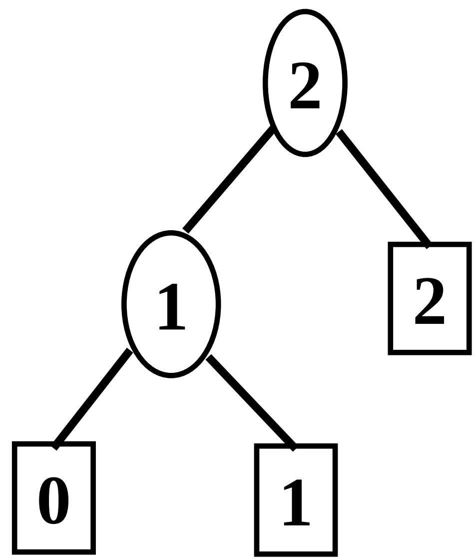


二阶斐波那契树T_2

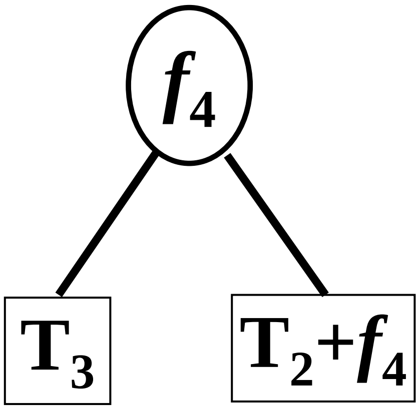


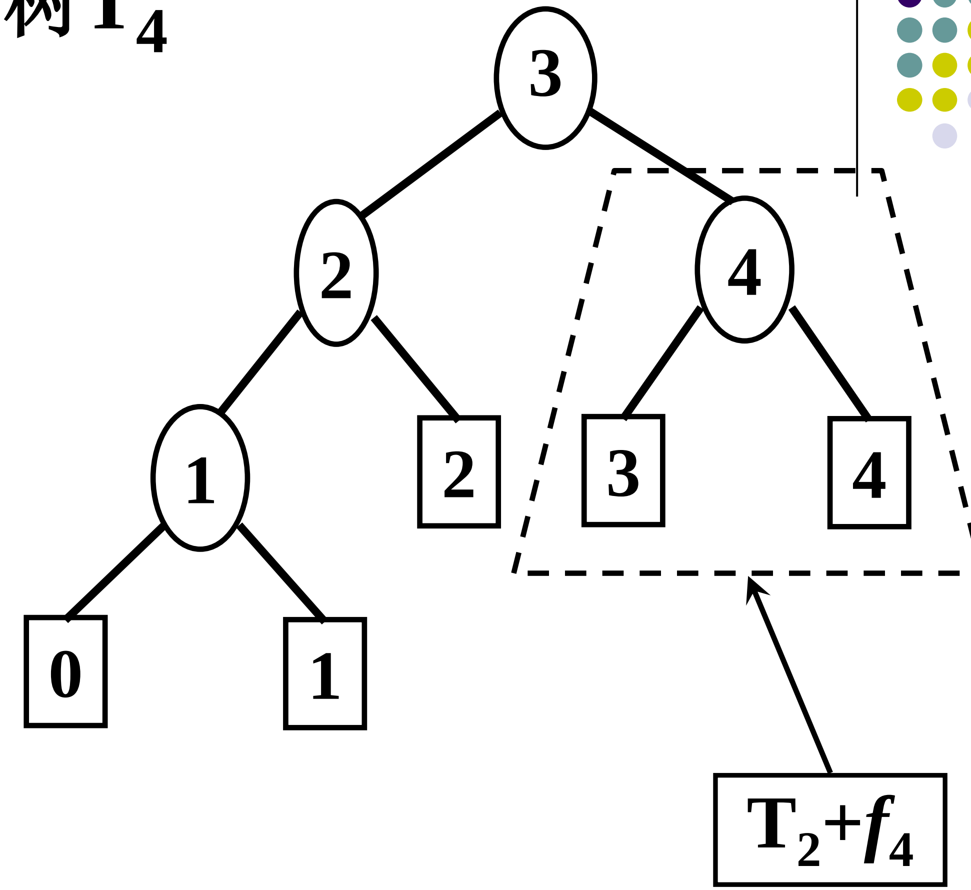


三阶斐波那契树T_3



四阶斐波那契树T_4

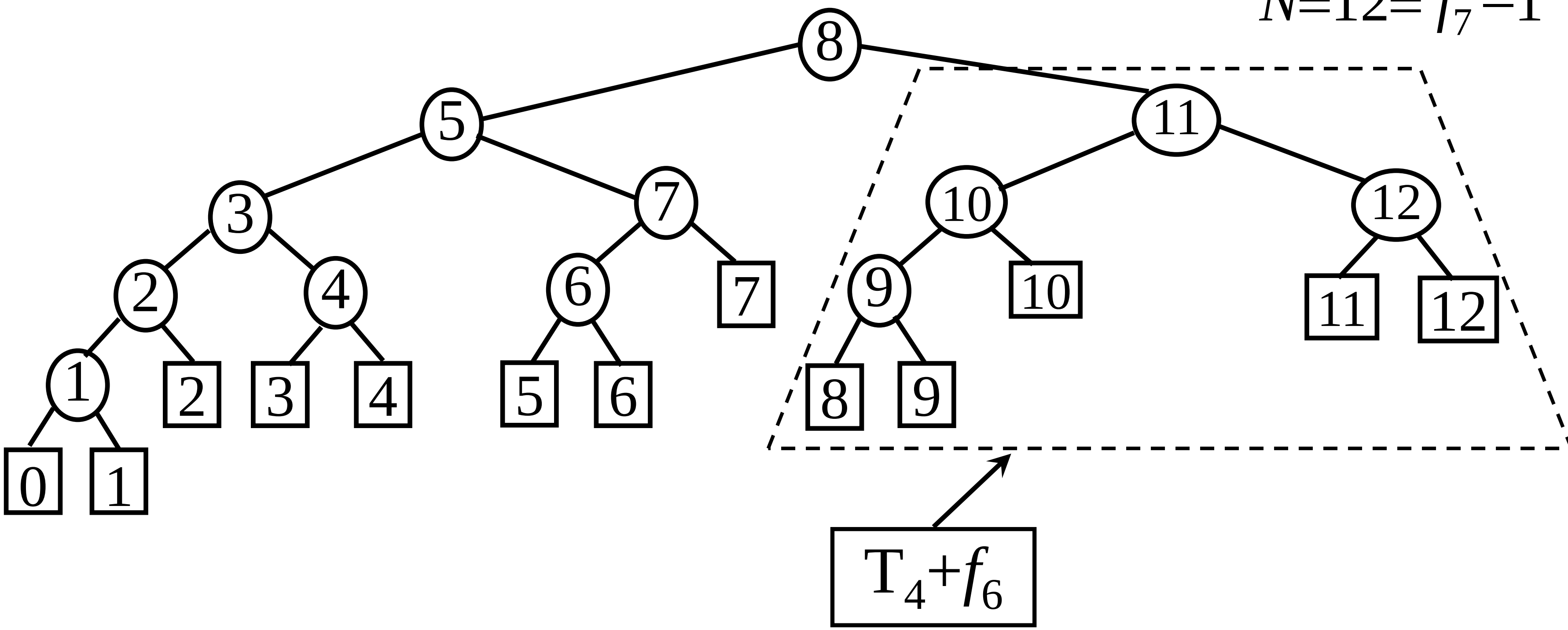




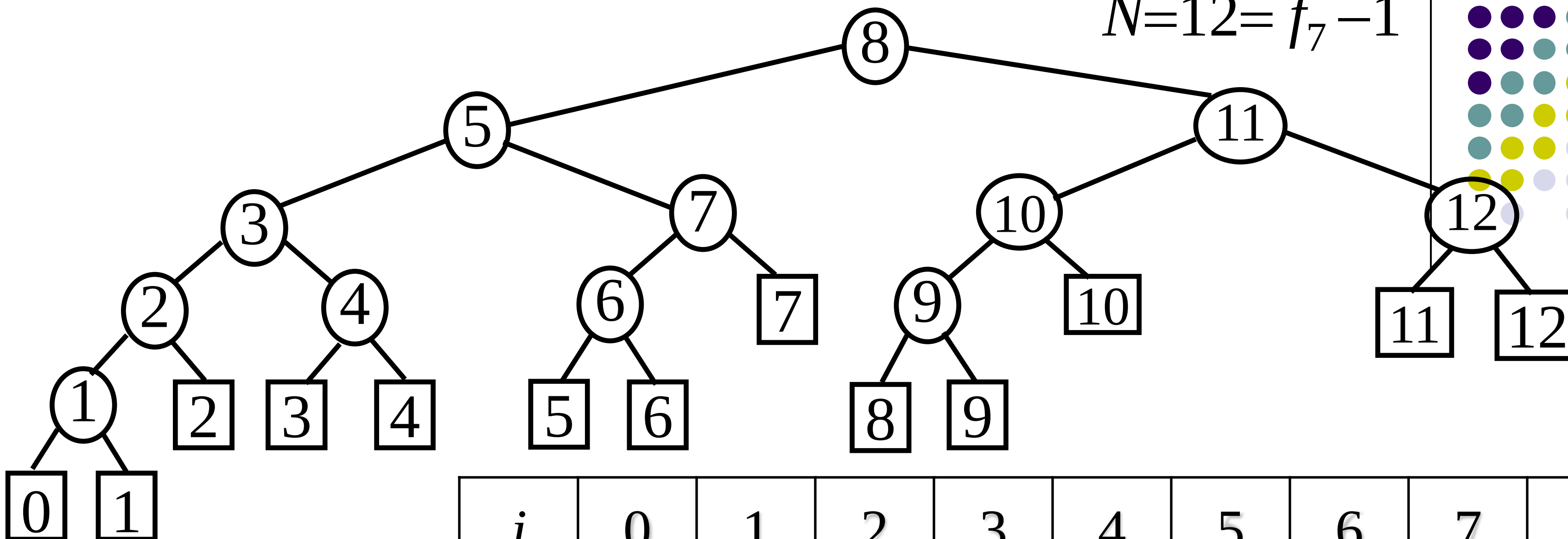
k 阶斐波那契树有 $f_{k+1}-1$ 个内结点和 f_{k+1}个外结点.

6 阶斐波那契树

$N=12=f_7-1$



$N=12=f_7-1$



j	0	1	2	3	4	5	6	7	8
f_j	0	1	1	2	3	5	8	13	21

- 只有度为**0**或度为**2**的结点；外结点数为**N+1**
- 内结点的两个子结点的编号与其父结点的编号之差的绝对值相同, 且这个绝对值是一个斐波那契数.
- **|**内结点与其父结点的编号之差**|** 是 $\mathbf{f_j}$ 时，若为左儿子，下层差为$\mathbf{f_{j-1}}$；若为右儿子，下层差为$\mathbf{f_{j-2}}$.

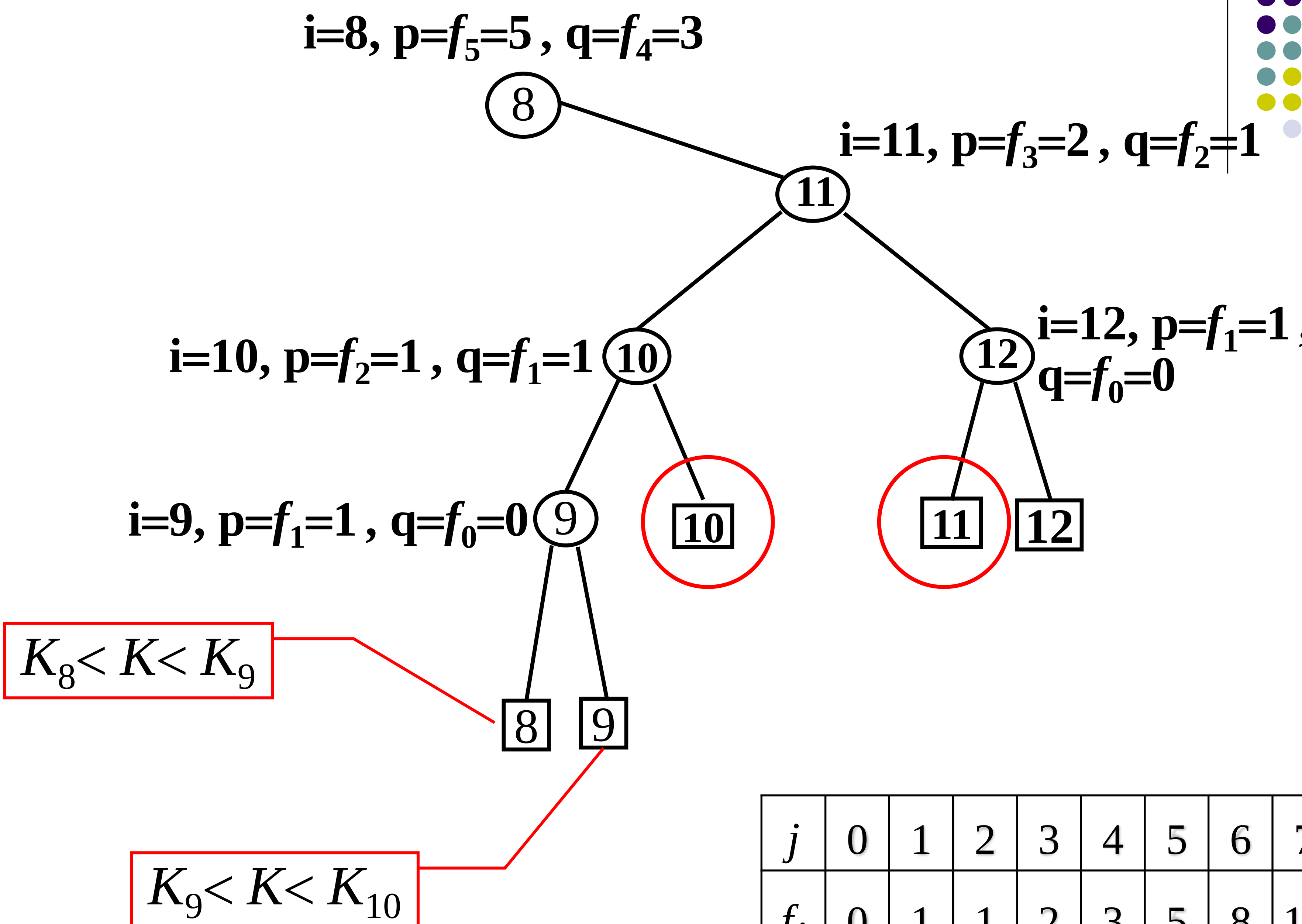
算法F(N，R，K . i)

F1. [初始化] 置 $i \leftarrow f_k$（ f_k为树根）, $p \leftarrow f_{k-1}$, $q \leftarrow f_{k-2}$.

F2. [比 较] 若$K < K_i$则转步骤F3；若$K > K_i$则转步骤F4；若$K = K_i$，则算法成功结束 .

F3. [i 减值] 若$q = 0$（已到树叶）, 则算法以失败告终, 否则置 $i \leftarrow i - q$，$t \leftarrow p$，$p \leftarrow q$，$q \leftarrow t - q$，并返回F2 .

F4. [i 增值] 若 $p = 1$（已到树叶），则算法以失败告终，否则置 $i \leftarrow i + q$，$p \leftarrow p - q$，$q \leftarrow q - p$，并返回F2. ▌



j	0	1	2	3	4	5	6	7
f_j	0	1	1	2	3	5	8	13

时间复杂度分析

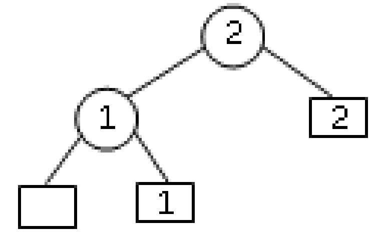


图 8.4-4　3 阶 Fibonacci 树形

- **引理8.4** 设$m \ge 3$，T_m是 m 阶斐波那契树形，则T_m的左子树形的高度等于右子树形的高度加1，且T_m 的高度为$m-1$.
- 证明：数学归纳法。
- 当$m = 3$时，引理成立。
- 假设小于m时引理成立, 故T_{m-1} 的高度为$m-2$，T_{m-2} 的高度为$m-3$。T_m 的左子树形是T_{m-1}，右子树形是$T_{m-2} + f_m$（T_{m-2} 的所有内结点之编号加上f_m 后得到的树），且$T_{m-2}+f_m$的高度等于T_{m-2} 的高度，由此可得T_m 的左子树形之高度等于其右子树形的高度加1，T_m 的高度等于T_{m-1} 的高度加1 ▌

- **引理8.5** 设 $n = F_{m+1}-1$，则 m 阶斐波那契树的高度约等于 $1.44 \log_2(n+1)$.

证明 由Fibonacci数的封闭型表达式，

$$N = f_{m+1} - 1 = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{m+1} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{m+1} - 1$$

当m比较大时，$N+1= \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{m+1} + O(1)$

T_m之高度$= m-1 = 1.44\times\log_2(N+1)$

- **定理8.2 令 $n = F_{m+1}-1$，则算法斐波那契在最坏情况下的时间复杂性为$O(\log_2 n)$，且期望复杂性亦为$O(\log_2 n)$.**

- 算法F的平均运行时间近似为：

$(7.05 \log N + 1.08)\, u$　　对于成功查找

$(7.05 \log N + 5.23)\, u$　　对于不成功查找

- 算法 C 的平均运行时间大约是算法 F 的 1.2 倍，算法 B 的平均运行时间大约是算法 F 的 2.5 倍。尽管在最坏的情况下，算法 F 的运行时间约为 8.6 log N，比算法 C 稍稍慢一点。

推广

- **Shar 变换**
  - ✓ $M = f_{k+1} - 1 - n$
  - ✓ $i = f_k$
  - ✓ K <= A[i]： 正常运行
  - ✓ K > A[i]： i = i – M, 向右多做一步

算法F参考实现

```
int ffind(int A[],int n,int key){
        int i=fk,p=fk_1,q=fk_2,t;
        if(key > A[i]){
                i=i-M;
                if(p==1) return -1;
                i=i+q; p=p-q,q=q-p;
        }
```

```
while(1){
        if(key==A[i]) return i;
        if(key<A[i]){
                if(q==0) break;
                i=i-q; t=p,p=q,q=t-p;
        }else{
                if(p==1) break;
                i=i+q; p=p-q,q=q-p;
        }
 }
 return -1;
}
```

均匀分布有序表——插值查找 (Interpolation Search)

- 前述讨论，均**假定对线性表中元素的分布一无所知**，故几种查找算法都是严格基于**比较**的.
- **很多查找问题涉及的表都满足某些统计特点.**
  - ✓ 例如在英汉词典中查找单词“apple”，不应该使用对半查找，先查找词典中间的页，然后查找词典的1/4或3/4处的页，……；应该**期望在前部查找**
- **在期望的地址附近开始查找，称为<u>插值查找</u>.**

插值公式

- 假定表中记录的关键词是数字类型，且 $K_1<K_2<\ldots<K_n$在(K_0, K_{n+1})区间上均匀分布。给定变元K，且$K_0<K<K_{n+1}$，则可用线性插值来决定 K 的期望地址 $n(K-K_0)/(K_{n+1}-K_0)$.
- 若$K_s< K < K_e$，则将应在

$$s + (e - s - 1)(K - K_s)/(K_e - K_s)$$处测试

算法Ip (N，R，K . i)

Ip1. [初始化] 置$s \leftarrow 0$. $e \leftarrow N+1$.

Ip2. [计算i的值] 如果$e - s \le 1$，则算法以失败告终；否则，置$i \leftarrow \left\lceil s + [(K - K_s)/(K_e - K_s)](e - s - 1) \right\rceil$.

Ip3. [比较] 如果$K < K_i$，则转到Ip4；如果$K > K_i$，则转到Ip5；如果$K = K_i$，则算法成功结束。

Ip4. [调整e] 置$e \leftarrow i$，并返回Ip2 .

Ip5. [调整s] 置$s \leftarrow i$，并返回Ip2 . $\blacksquare$

参考实现（类似二分查找）

- mid = $low + (high - low)(K - K_{low})/(K_{high} - K_{low})$

```
int Interpolation_Search(int arr[], int n, int key){
  int low = 1,high = n,mid;
  while (low <= high){
     mid = low;
     if(arr[low] != arr[high])
          mid+=(high-low)*(key - arr[low]) / (arr[high] - arr[low]);
     if (key < arr[mid]) high = mid - 1;
     else if(key > arr[mid]) low = mid + 1;
     else return mid;
  }
  return -1;
```

插值查找效率分析

- 数据分布均匀时
  - 例：查找表a中数据：1,2,…,100
  - 期望时间复杂度O(loglogn)
  - 渐进优于折半查找，相当于折半查找的改进
- 数据分布不均匀时，可能会退化
  - 例：查找表a中数据：1,2,3,4,10000
  - 退化到O(n)，不如折半查找
- 综合

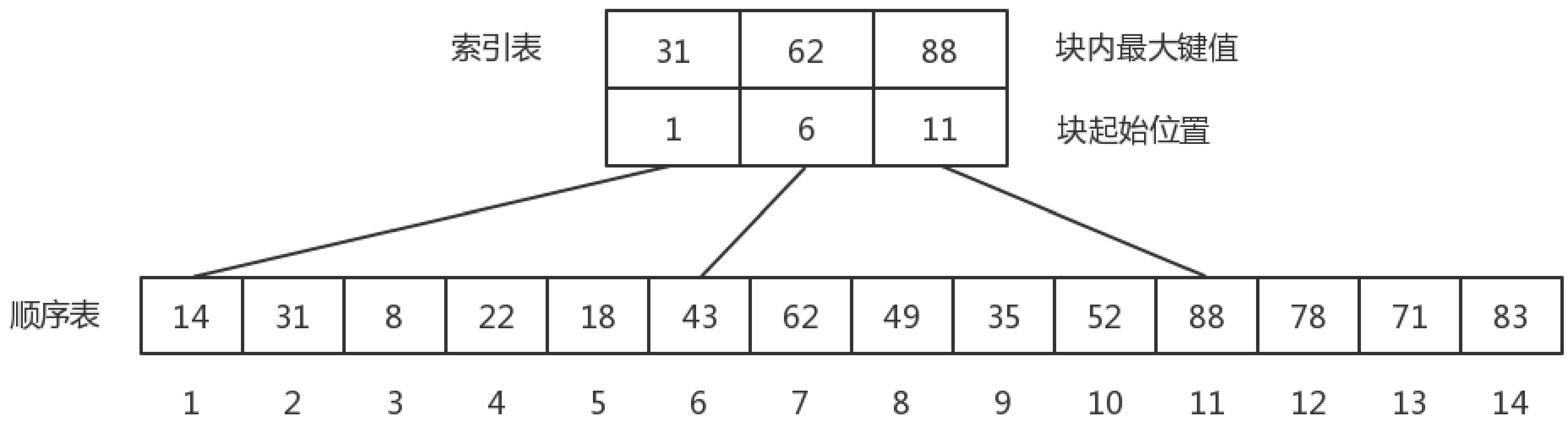
	最坏	最好	平均
查找成功	N	1	loglogn
查找失败	N	1	loglogn

分块查找(Blocking Search)

- **分块查找又称索引顺序查找。**
- **基本思想：**将n个数据元素"按块有序"划分为m块（m ≤ n）。每一块中的结点不必有序，但块与块之间必须"按块有序"；
- **操作步骤**
  - **取各块最大值，建索引表；**
  - **查找分两步：**先对**索引表**进行二分查找或顺序查找，确定块；然后，对**块内**进行顺序查找。

存储结构

- 主表：顺序表
- 索引表：结点结构(index,start)



插入操作

- 查找表动态形成
- 结点中引入length

分块查找效率分析

- 索引表二分 + 块内顺序
  - ✓ 时间复杂度：logm + n/m
- 索引表顺序 + 块内顺序
  - ✓ 时间复杂度：m + n/m
  - ✓ m = sqrt(n)取最小值

- 极端情形：块内有序；索引表、块内均二分
  - ✓ 时间复杂度：logm + log(n/m) = logn
- 分块查找是顺序查找的一种改进。性能介于顺序查找和二分查找之间。

小结

- **线性表**
  - ✓ 顺序查找
  - ✓ 自组织表
  - ✓ 分块查找（块间有序）
- **有序表**
  - ✓ 对半查找
  - ✓ 一致对半查找
  - ✓ Fibonacci查找
- **分布信息**
  - ✓ 插值查找

第8章 任务

- 慕课
  - ✓ 在线学习/预习 第 8 章 视频

- 作业
  - ✓ P340：8-4，8-7， 8-9，8-10，
    8-13，8-22，8-23
  - ✓ 在线提交