

## 02-D2-7 查找长度 & Fibonacci查找

#数据结构邓神

### 如何更为精细的评估查找算法的性能

考察关键码的比较次数，即查找长度(Search Length)

通常，需要针对成功和失败查找，从最好最坏平均等角度评估

比如成功和失败平均查找长度大致为 $O(1.50 \log n)$

还有改进的余地！

### Fibonacci 查找

前面查找的算法的复杂度为  $O(1.5 \log n)$

这个1.5是可以减少的！！

就要使用 Fibonacci查找

改进余地在于：转向左右分支的关键码比较次数不想等，但是递归深度却相等

像左侧需要一次比较但是像右侧却需要两次比较？

算法效率可能还没有达到最优

我们发现像左侧的成本要低于像右侧的成本

那么我们能不能让这颗树能够让左侧深度更深，让右侧深度更浅

简单来说：我们希望做更多次成本低的转向，成本高的转向少做

### 思路及原理

❖ 二分查找版本A的效率仍有改进余地，因为不难发现

转向左、右分支前的关键码比较次数不等，而递归深度却相同

❖ 若能通过递归深度的不均衡，对转向成本的不均衡进行补偿

平均查找长度应能进一步缩短...



## Fibonacci 查找

❖ 比如，若设  $n = \text{fib}(k) - 1$ ，则可取  $mi = \text{fib}(k - 1) - 1$   
于是，前、后子向量的长度分别为  $\text{fib}(k - 1) - 1$ 、 $\text{fib}(k - 2) - 1$

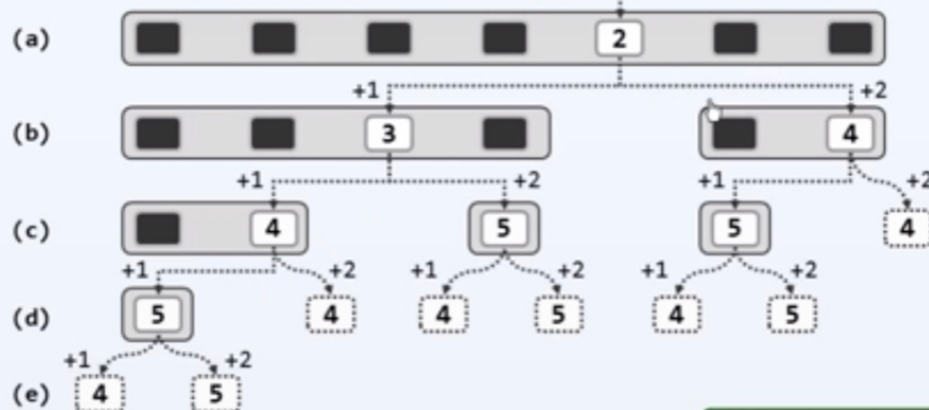
如果长度  $N$  刚好为一个斐波那契数列减去1，那么我们可以去中间的点的值为这个斐波那契数列的前一个减去1，这样就把数列分成三段

Fibonacci 查找的ASL(在常系数的意义上)优于二分查找

❖ 仍以  $n = \text{fib}(6) - 1 = 7$  为例，在等概率情况下

平均成功查找长度 =  $(5 + 4 + 3 + 5 + 2 + 5 + 4) / 7 = 28/7 = 4.00$

平均失败查找长度 =  $(4 + 5 + 4 + 4 + 5 + 4 + 5 + 4) / 8 = 35 / 8 = 4.38$



## 实现

```
template <typename T>
static Rank fibSearch(T* A, T const& e, Rank lo, Rank hi){
    Fib fib(hi - lo); // 用 O(logn) = O(log(hi - lo))时间来创建Fibonacci数列
    while(lo < hi){
        while(hi - lo < fib.get()){
            fib.prev();
        }
        Rank mi = lo + fib.get() - 1;
        if (e < A[mi]){
            hi = mi;
        }
    }
}
```

```
    else if (A[mi] < e){  
        lo = mi + 1;  
    }  
    else {  
        return mi;  
    }  
}  
return -1;  
}
```

具体的差别在于中点选择!

最关键就是确定mi

每次都取Fibonacci数列点下一项

我们通过 Fib 类取出他的下一项或者前面一项

整个Fibonacci关键点就是在于mi如何确定