

向量

有序向量：二分查找（版本A）

02-D2

自从爷爷去后，这山被二郎菩萨点上火，烧杀了大半...及至火灭烟消，
出来时，又没花果养赡，难以存活，别处又去了一半。我们这一半，捱苦
的住在山中，这两年，又被些打猎的抢了一半去也

邓俊辉

deng@tsinghua.edu.cn

统一接口

```
template <typename T> //查找算法统一接口, 0 <= lo < hi <= _size
```

```
Rank Vector<T>::search( T const & e, Rank lo, Rank hi ) const {
```

```
    return ( rand() % 2 ) ? //等概率地随机选用
```

```
        binSearch( _elem, e, lo, hi ) //二分查找算法, 或
```

```
        : fibSearch( _elem, e, lo, hi ); //Fibonacci查找算法
```

```
}
```



有序向量中，每个元素都是轴点

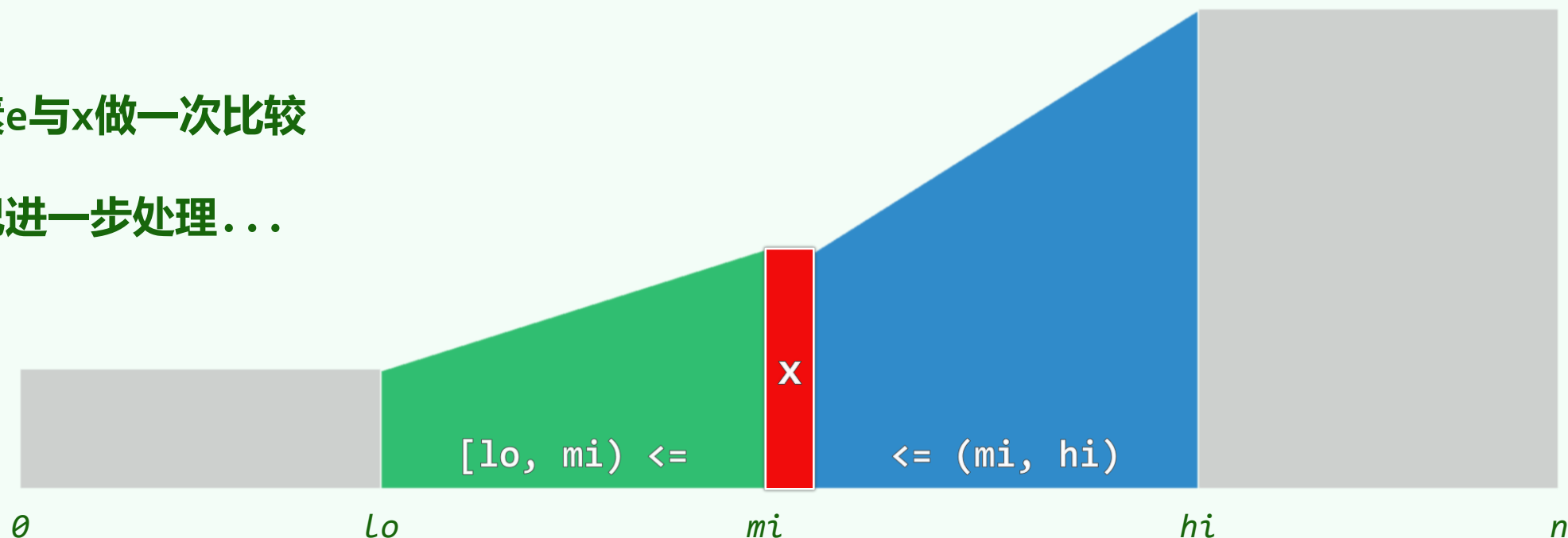
❖ 以任一元素 $x = [mi]$ 为界，都可将待查找区间 $[lo, hi)$ 分为三部分

$$[lo, mi) \leq [mi] \leq (mi, hi)$$

因此...

❖ 只需将目标元素 e 与 x 做一次比较

即可分三种情况进一步处理...



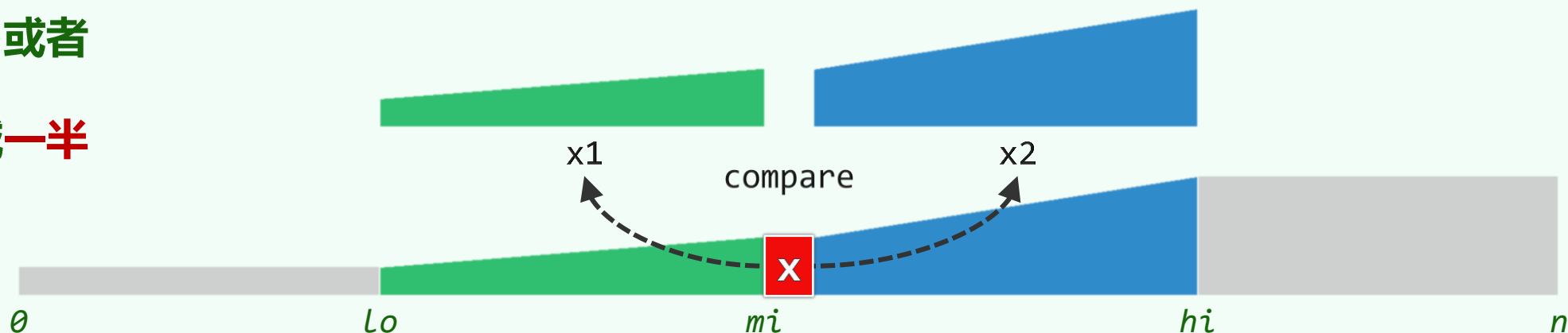
减而治之

- ❖ $e < x$: 则 e 若存在必属于左侧子区间, 故可 (减除 $[mi, hi)$ 并) 递归深入 $[lo, mi)$
- $x < e$: 则 e 若存在必属于右侧子区间, 亦可 (减除 $[lo, mi]$ 并) 递归深入 (mi, hi)
- $e = x$: 已在此处命中, 可随即返回 //若有多个, 返回何者?

- ❖ 若轴点 mi 取作中点, 则每经过至多两次比较

- ❖ 或者能够命中, 或者

将问题规模缩减一半



实现

```
template <typename T> //在有序向量[lo, hi)区间内查找元素e
static Rank binSearch( T * S, T const & e, Rank lo, Rank hi ) {
    while ( lo < hi ) { //每步迭代可能要做两次比较判断, 有三个分支
        Rank mi = ( lo + hi ) >> 1; //以中点为轴点 (区间宽度折半, 其数值表示右移一位)
        if ( e < S[mi] ) hi = mi; //深入前半段[lo, mi)
        else if ( S[mi] < e ) lo = mi + 1; //深入后半段(mi, hi)
        else return mi; //命中
    }
    return -1; //失败
}
```

The diagram illustrates the binary search process on a sorted array. The array is represented as a horizontal bar with indices 0, lo, mi, hi, and n. The segment from lo to mi is green, and the segment from mi to hi is blue. A red box with a white 'X' is at index mi. Dashed arrows labeled x1 and x2 point from the red box to the green and blue segments respectively, indicating the recursive search steps. The word 'compare' is written above the red box.

实例 + 复杂度

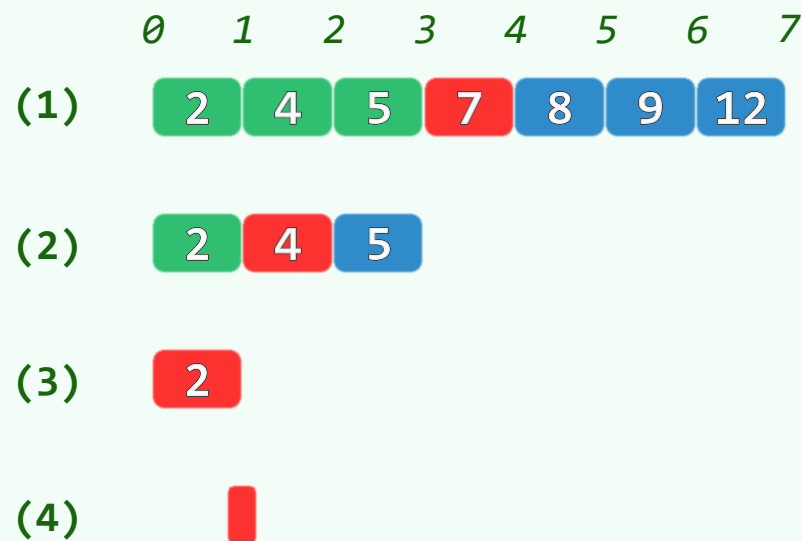
S.search(8, 0, 7):

经 $2 + 1 + 2 = 5$ 次比较, 在S[4]命中



S.search(3, 0, 7):

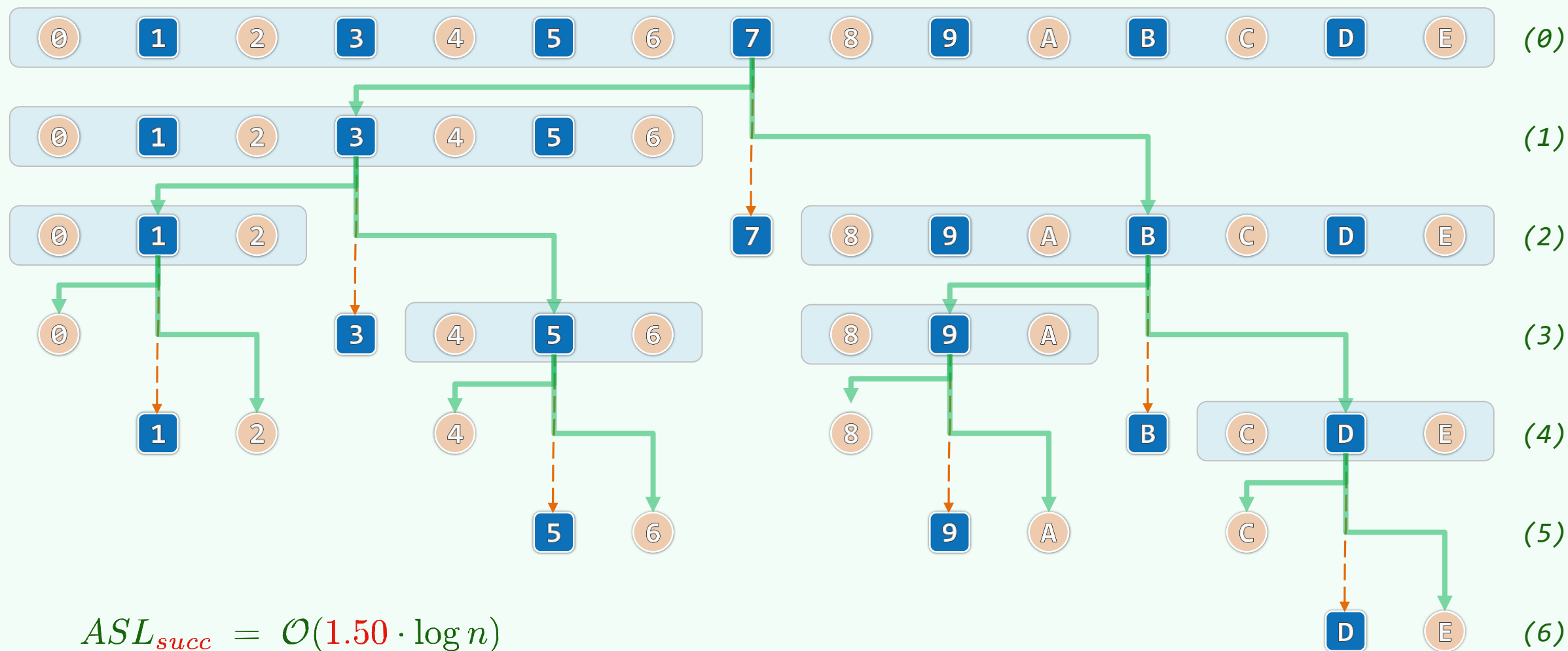
经 $1 + 1 + 2 = 4$ 次比较, 在S[1]失败



❖ 线性“递归”： $T(n) = T(n/2) + \mathcal{O}(1) = \mathcal{O}(\log n)$ ，大大优于顺序查找

“递归”跟踪：轴点总能取到中点，递归深度 $\mathcal{O}(\log n)$ ；各递归实例仅耗时 $\mathcal{O}(1)$

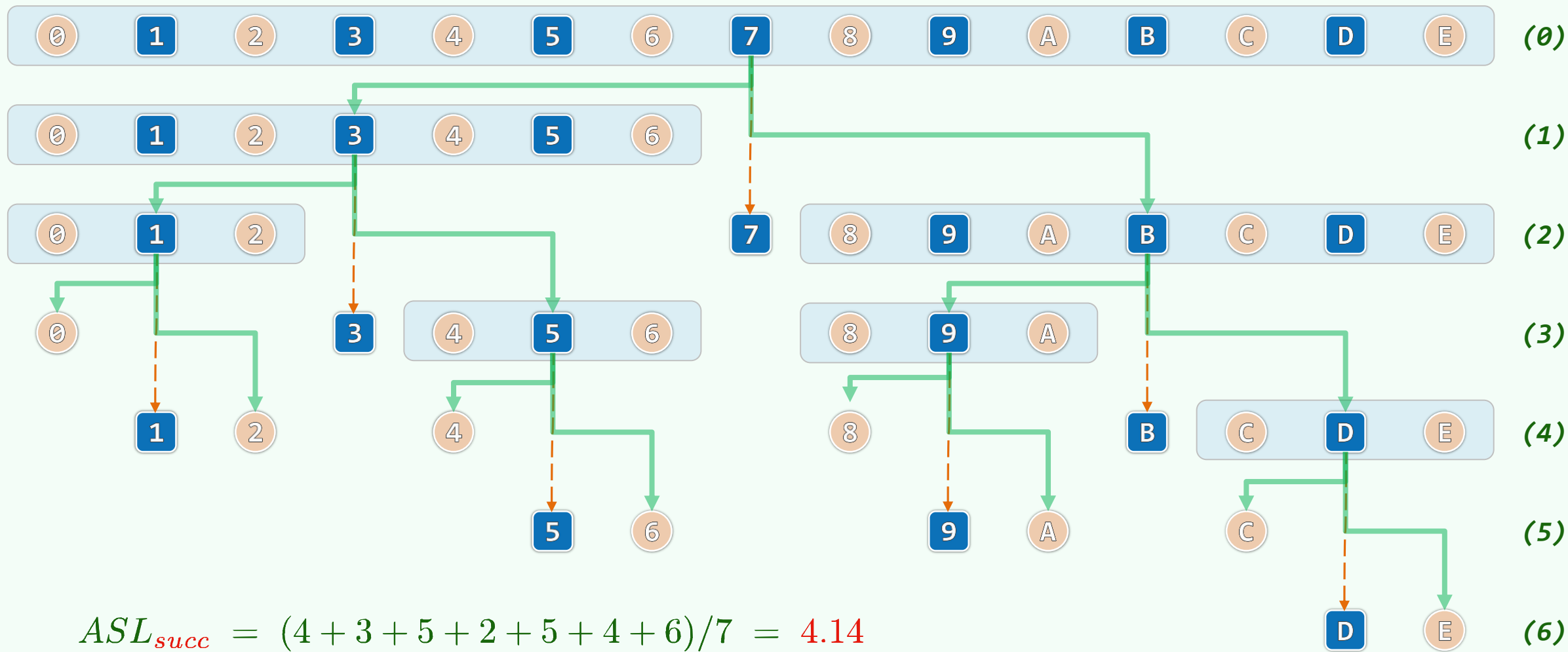
关键码的比较次数 ~ 平均查找长度 (Average Search Length)



$$ASL_{succ} = \mathcal{O}(1.50 \cdot \log n)$$

$$ASL_{fail} = \mathcal{O}(1.50 \cdot \log n)$$

实例：n = 7，按等概率估算



$$ASL_{succ} = (4 + 3 + 5 + 2 + 5 + 4 + 6)/7 = 4.14$$

$$ASL_{fail} = (3 + 4 + 4 + 5 + 4 + 5 + 5 + 6)/8 = 4.50$$