

向量

有序向量：二分查找（版本B）

02-D4

和微风匀到一起的光，象冰凉的刀刃儿似的，把宽静的大街切成两半，一半儿黑，一半儿亮。那黑的一半，使人感到阴森，亮的一半使人感到凄凉。

邓俊辉

deng@tsinghua.edu.cn

## 改进思路

❖ 二分查找中左、右分支转向代价不平衡的问题，也可**直接**解决，比如...

每次迭代仅做**1次**关键码比较；如此，所有分支只有**2个**方向，而不再是**3个**

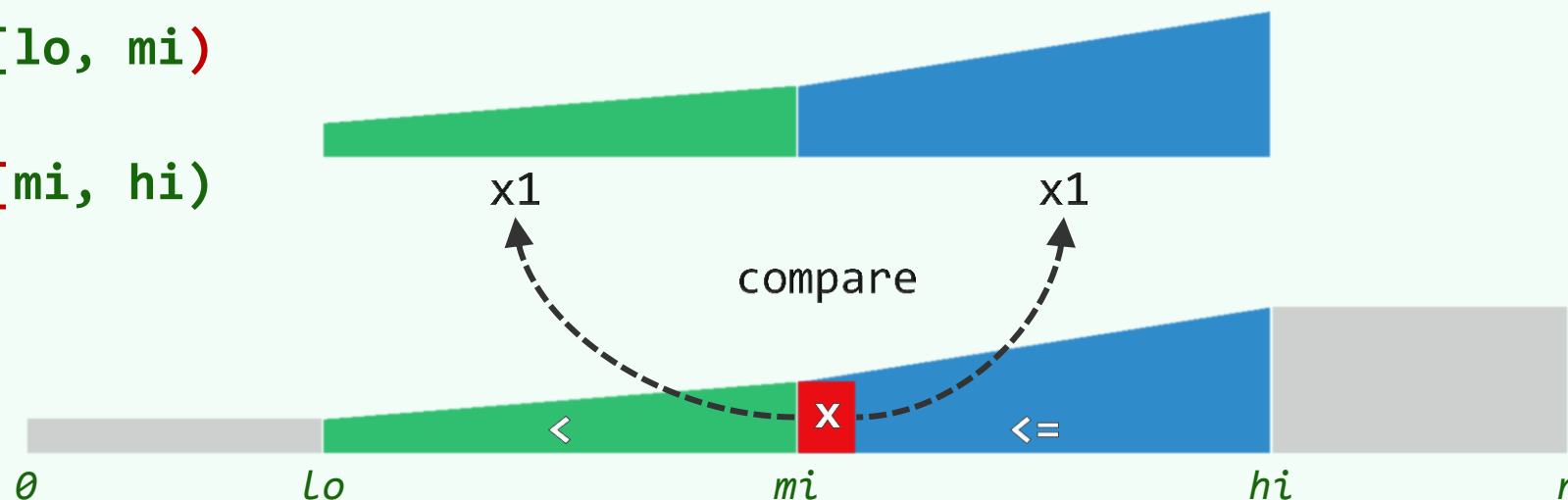
❖ 同样地，轴点 $mi$ 取作中点，则查找每深入一层，问题规模也缩减一半

-  $e < x$  : 则深入左侧的 $[lo, mi)$

-  $x \leq e$  : 则深入右侧的 $[mi, hi)$

❖ 直到 $hi - lo = 1$

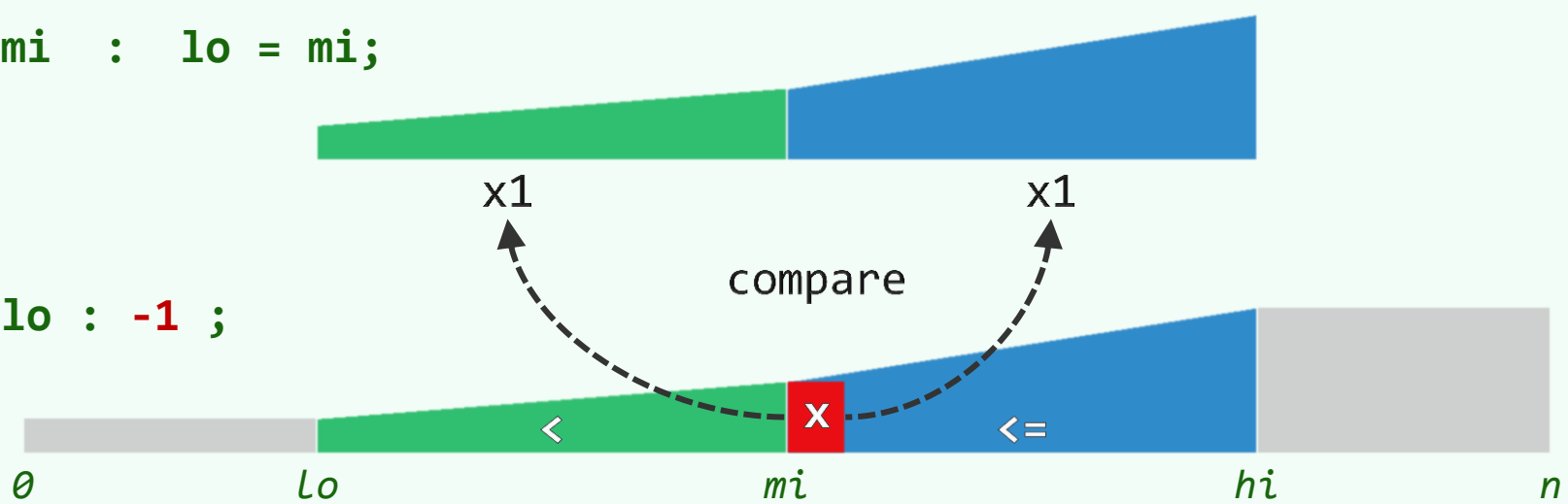
才判断是否命中



❖ 相对于版本A，最好（坏）情况下更坏（好），整体性能更趋均衡

# 实现

```
❖ template <typename T> static Rank binSearch( T * S, T const & e, Rank lo, Rank hi ) {  
  
    while ( 1 < hi - lo ) { //有效查找区间的宽度缩短至1时，算法才终止  
  
        Rank mi = (lo + hi) >> 1; //以中点为轴点，经比较后确定深入[lo, mi)或[mi, hi)  
  
        e < S[mi] ? hi = mi : lo = mi;  
  
    } //出口时hi = lo + 1  
  
    return e == S[lo] ? lo : -1;  
}
```



The diagram illustrates the binary search process on an array  $S$ . The array is represented as a horizontal bar with segments of different colors: gray for indices  $0$  to  $lo$ , green for  $lo$  to  $mi$ , blue for  $mi$  to  $hi$ , and gray for  $hi$  to  $n$ . A red box with a white 'X' is at index  $mi$ . A dashed arrow labeled 'compare' points from the element at index  $mi$  to the element at index  $x1$ . The green segment is labeled with '<' and the blue segment with '<='.

❖ 返回命中元素的秩，或者（失败时）非法的秩

## 返回更多信息

❖ 各种特殊情况，如何统一地处置？比如

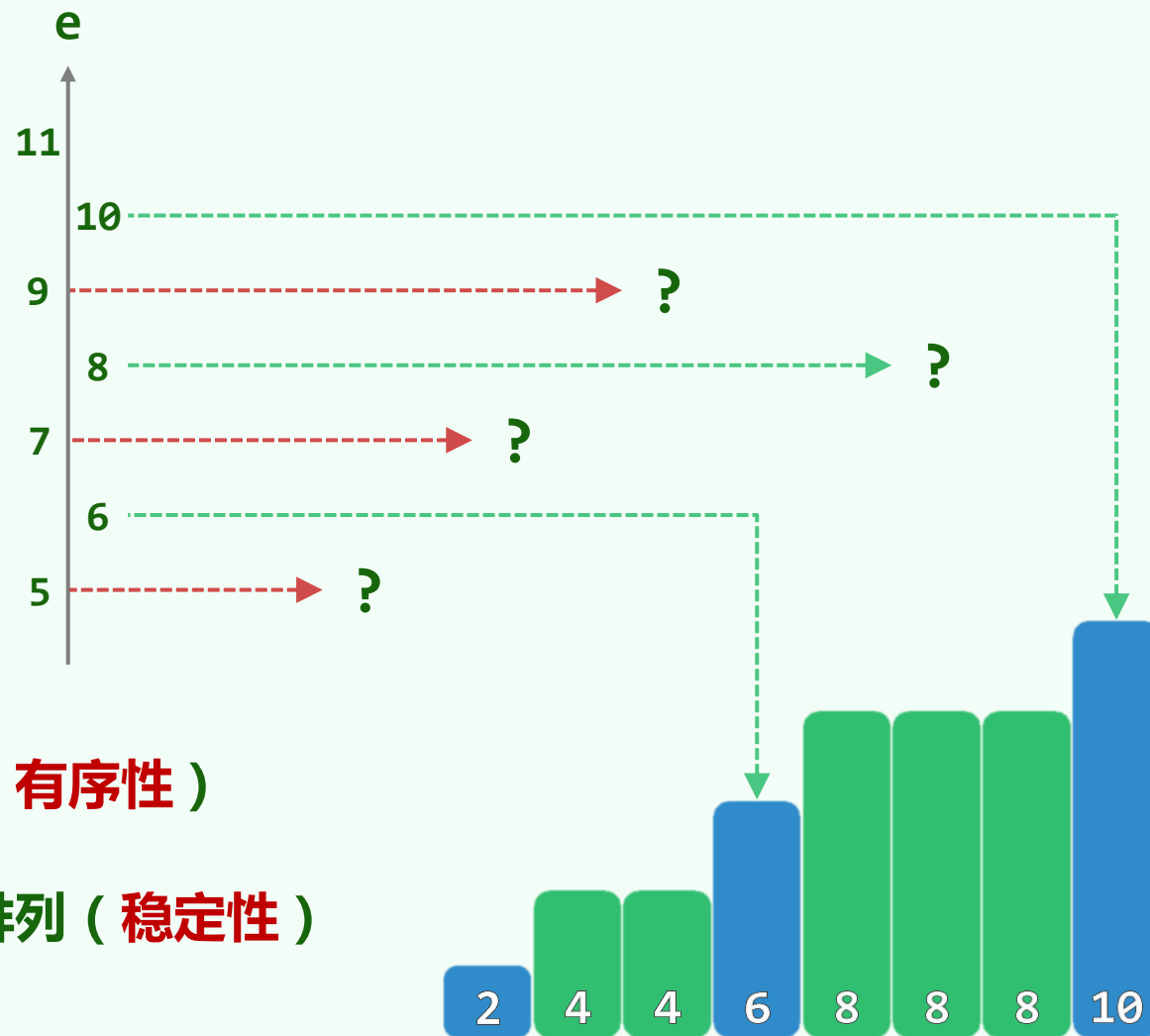
- 目标元素**不存在**；或反过来
- 目标元素同时存在**多个**

❖ 有序向量自身，如何便捷地维护？

比如：`V.insert( 1 + V.search(e), e )`

- 即便失败，也应给出新元素适当的插入位置（**有序性**）
- 若有重复元素，每一组也需按其插入的次序排列（**稳定性**）

❖ 为此，需要更为精细、明确、简捷地定义`search()`的返回值



# 返回值语义的扩充

❖ 约定：search()总是返回

$m$  = 不大于 $e$ 的最后一个元素

(其后继  $M$  = 大于 $e$ 的第一个元素)

❖ 改进版本B：

```
return e == S[lo] ? lo : -1 ;
```

```
return e < S[lo] ? lo-1 : lo ;
```

❖ 虽可行，但不免有些蹩脚

有没有...更为...简明、高明的...实现方式？

