逻辑教育
Logic education

Hello 数据结构与算法

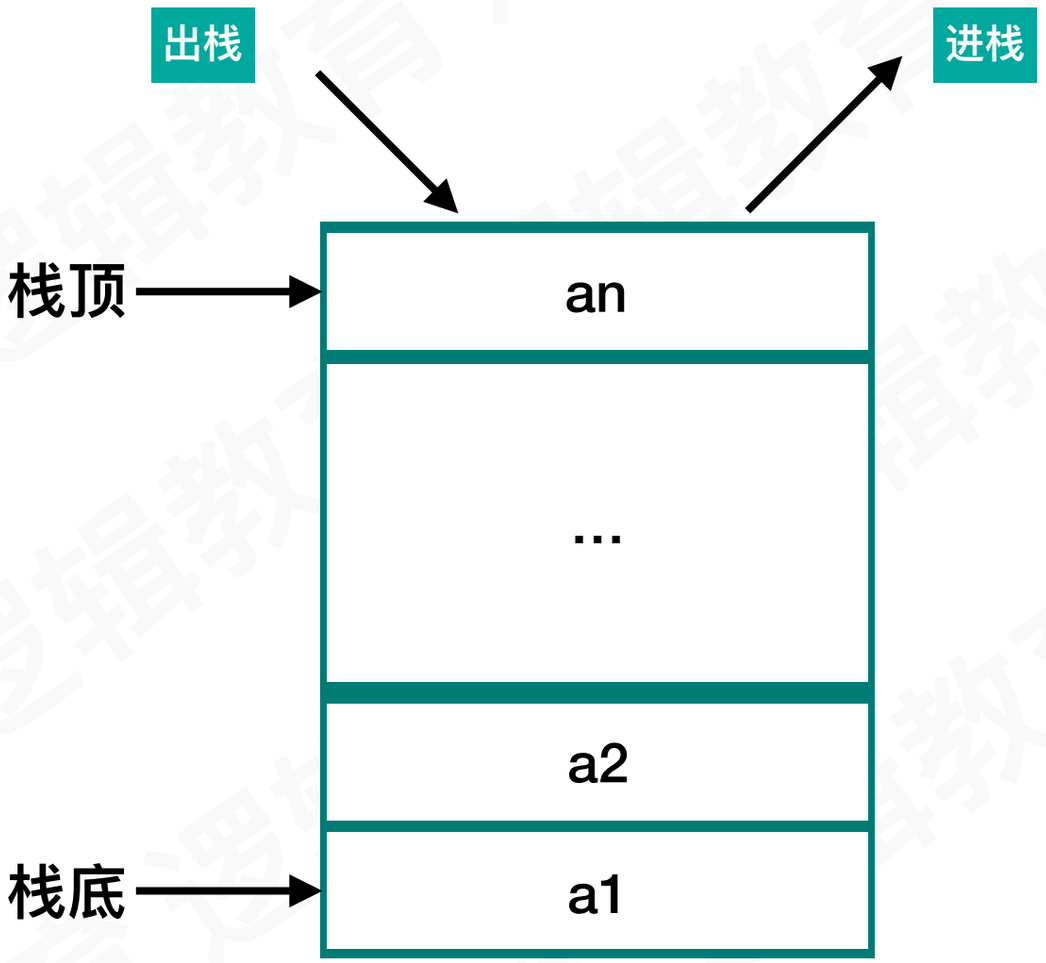
数据结构与算法—栈和队列 数据结构与算法主题[3]

@HelloCoder_CC

全力以赴．非同凡"想"

课程研发:CC老师
课程授课:CC老师

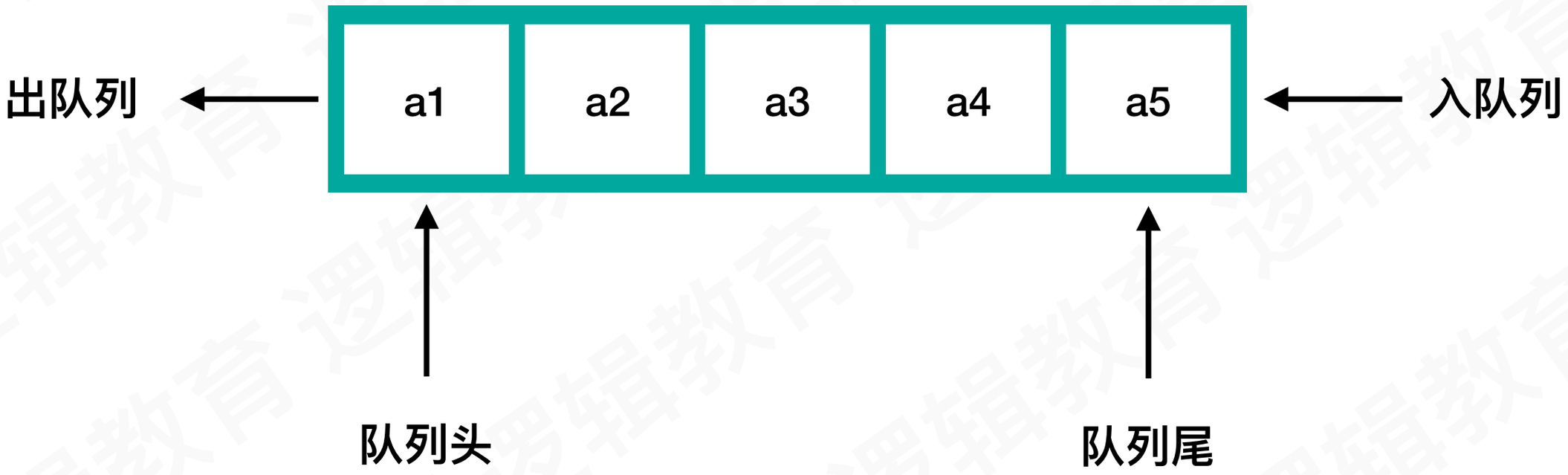
栈结构



栈的示意图

逻辑教育
Logic education

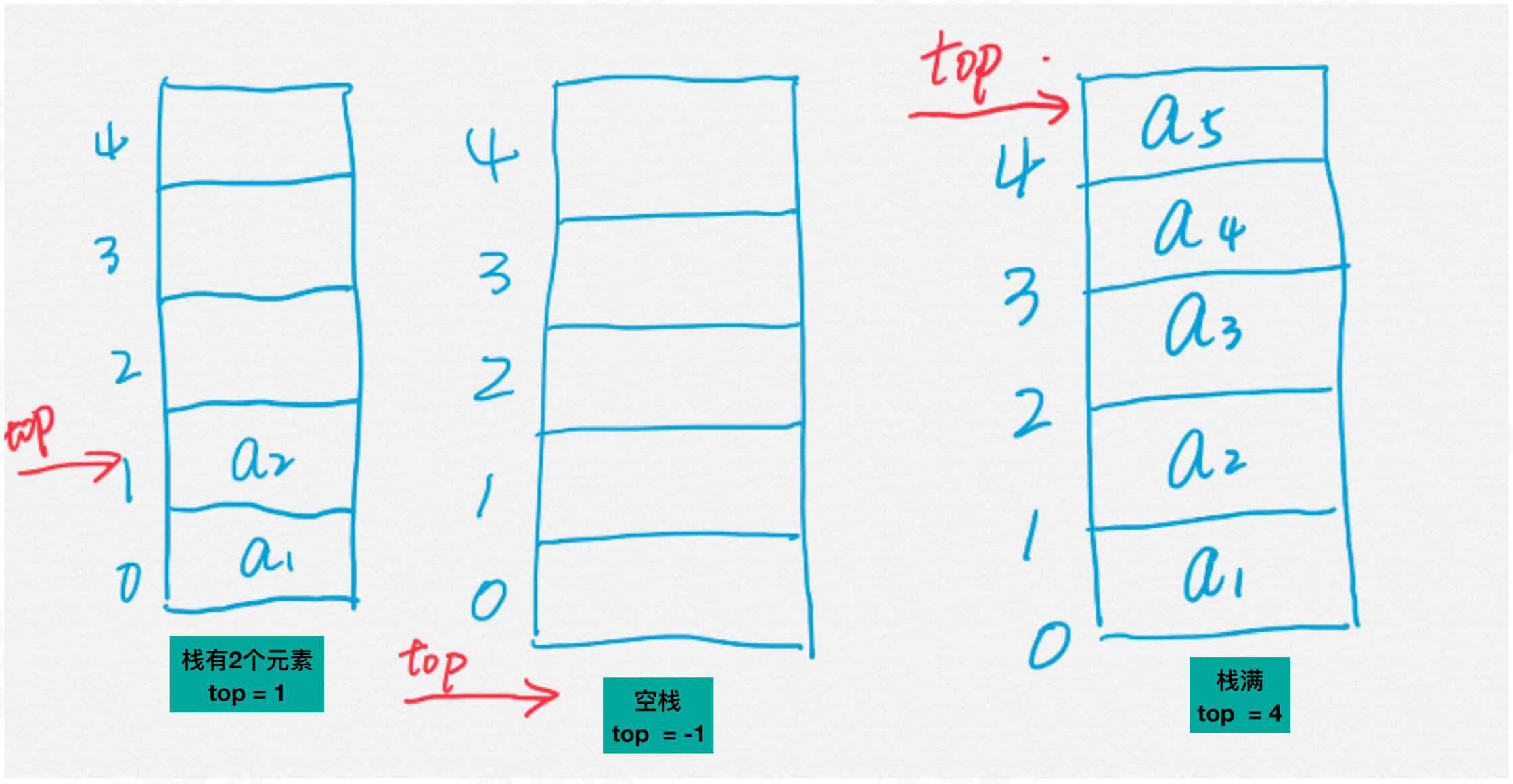
队列结构



队列的示意图

课程研发:CC老师
课程授课:CC老师

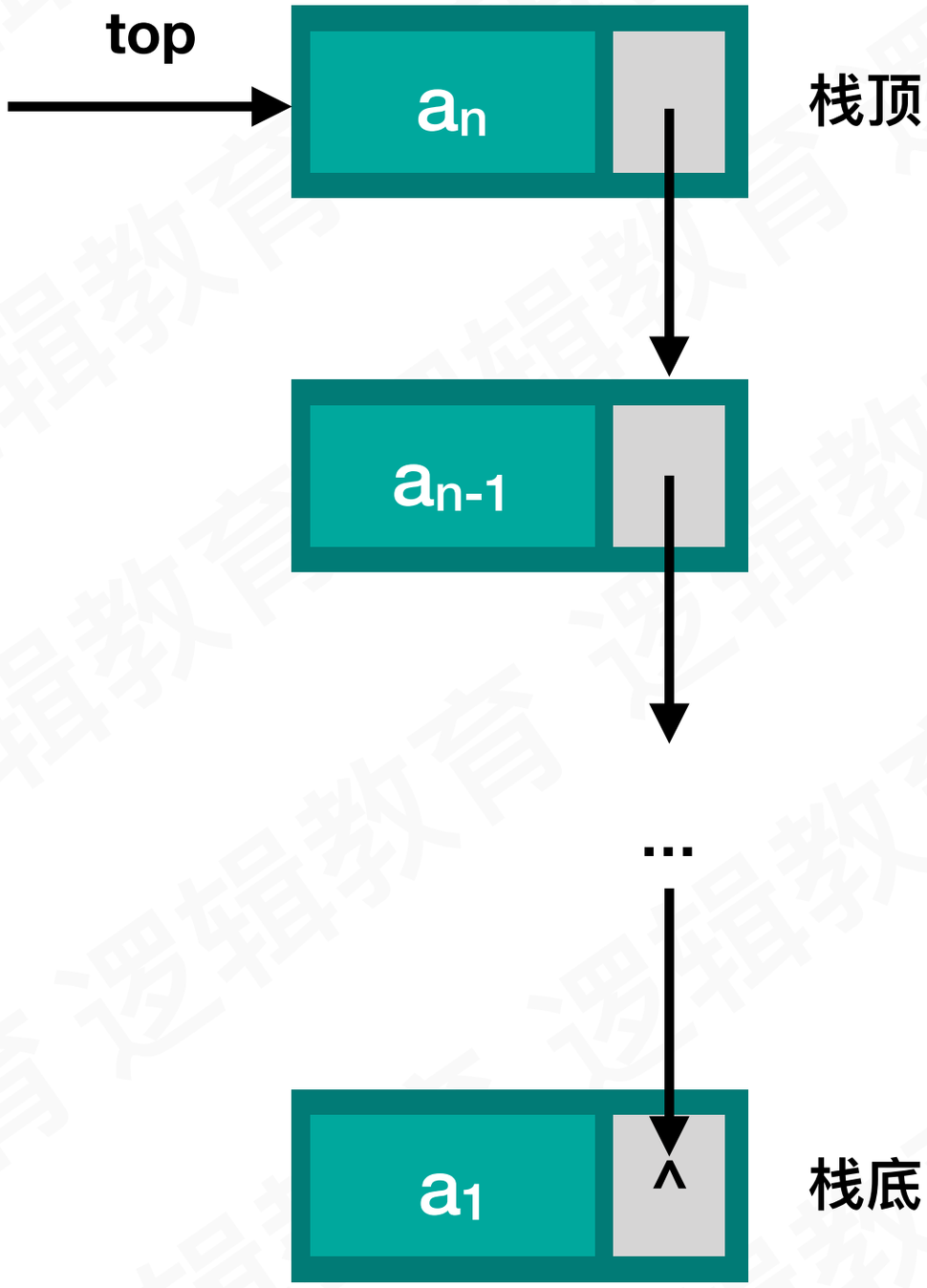
了解栈top信息



课程研发:CC老师
课程授课:CC老师

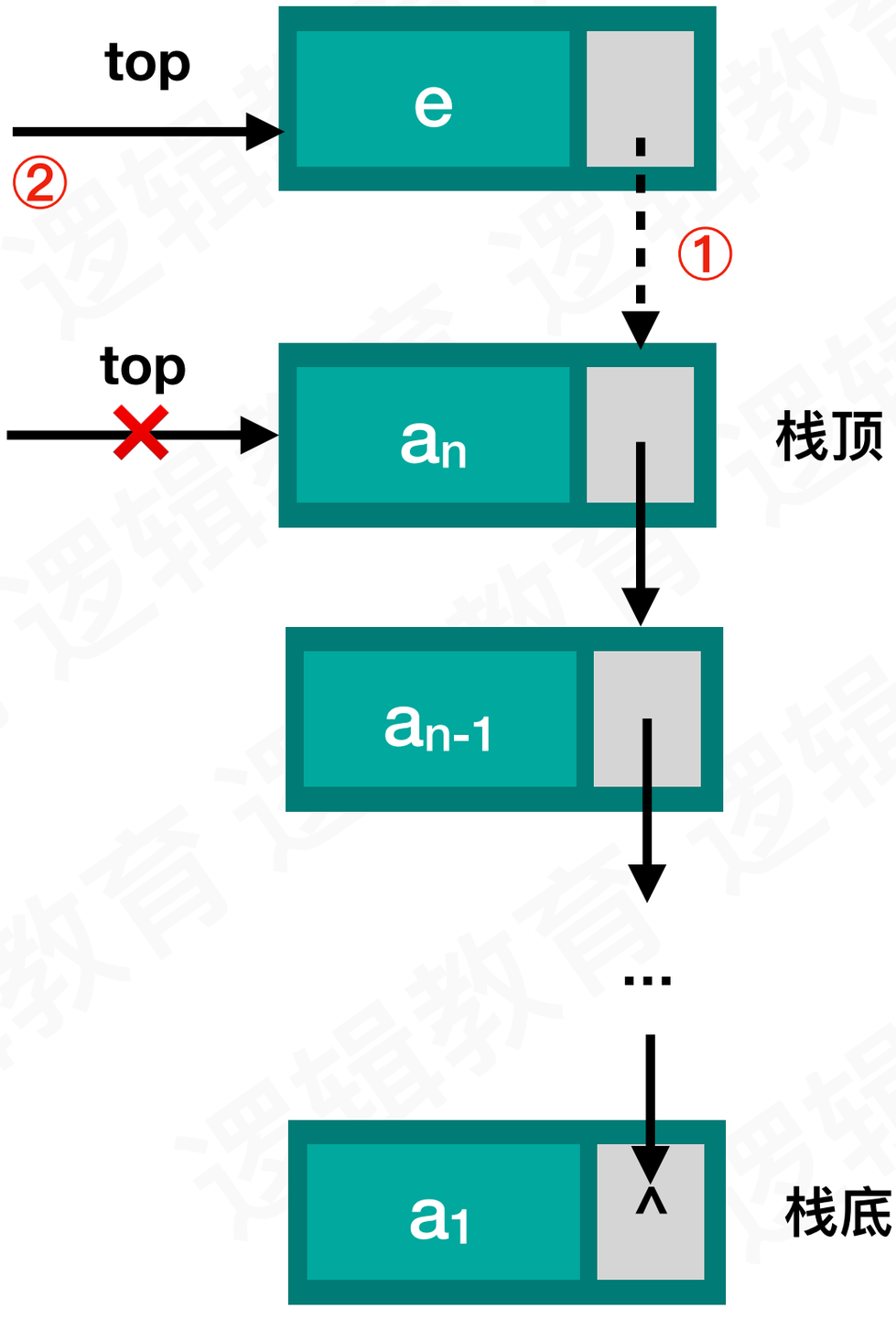
逻辑教育
Logic education

链式栈结构



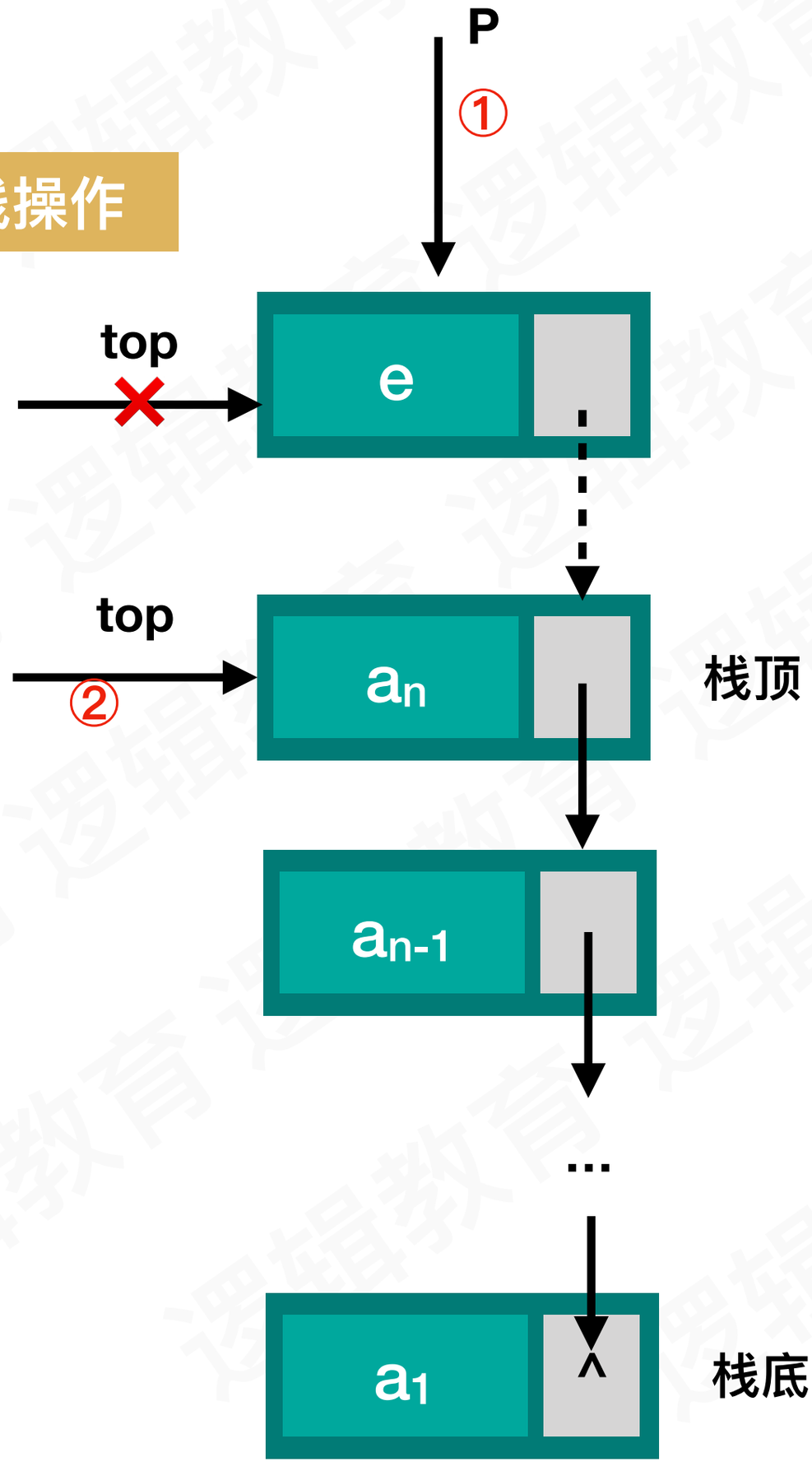
课程研发:CC老师
课程授课:CC老师

链式栈结构—进栈操作



课程研发:CC老师
课程授课:CC老师

链式栈结构—出栈操作



逻辑教育
Logic education

栈与递归

下面3种情况下,我们会使用到递归来解决问题

- 定义是递归的
- 数据结构是递归的
- 问题的解法是递归的

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

兔子繁衍问题:

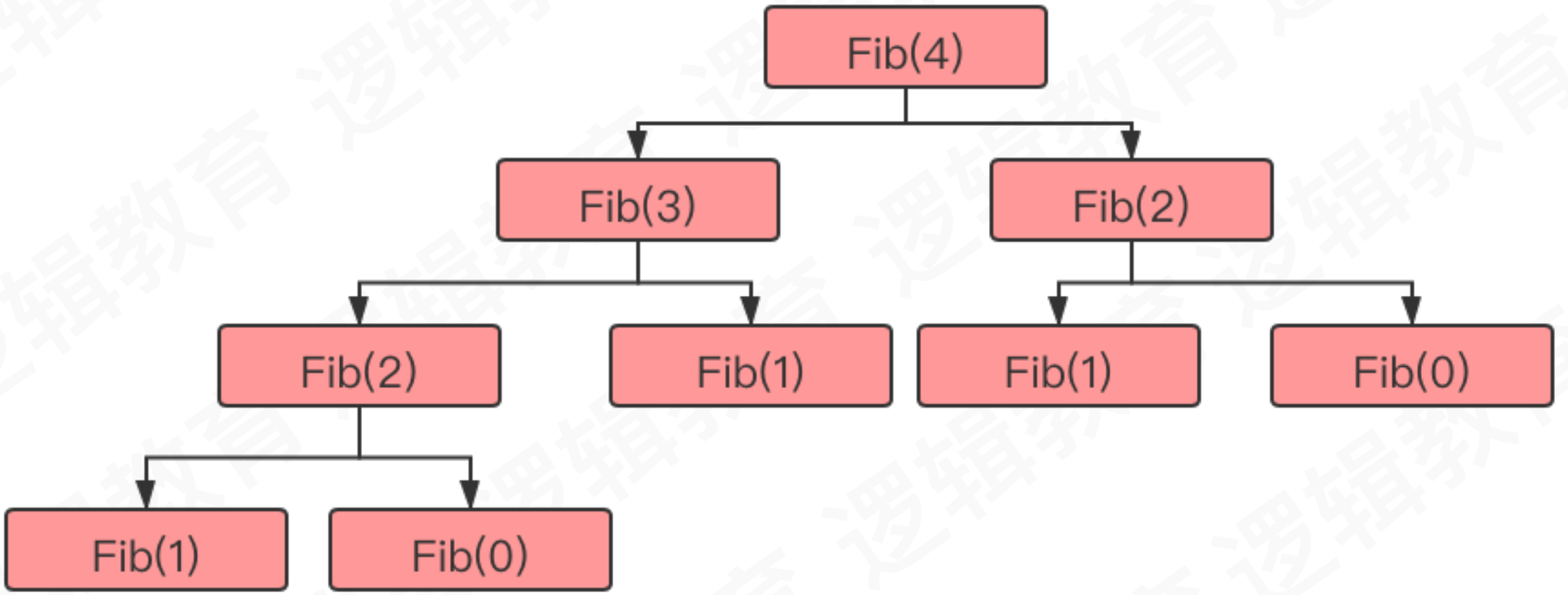
如果兔子2个月之后就会有繁衍能力,那么一对兔子每个月能生出一对兔子; 假设所有的兔子都不死,那么n个月后能生成多少只兔子?

经过的月数	1	2	3	4	5	6	7	8
兔子对数	1	1	2	3	5	8	13	21

课程研发:CC老师
课程授课:CC老师

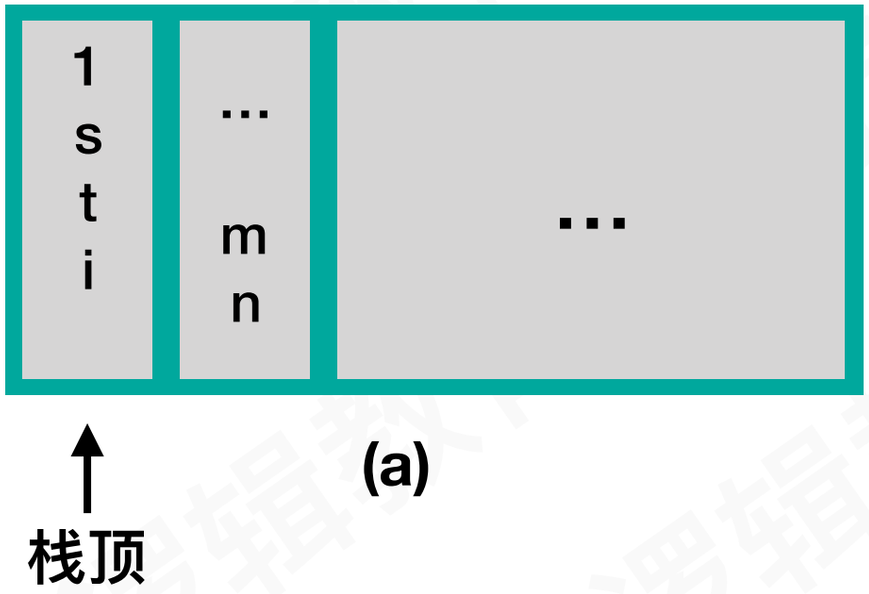
逻辑教育
Logic education

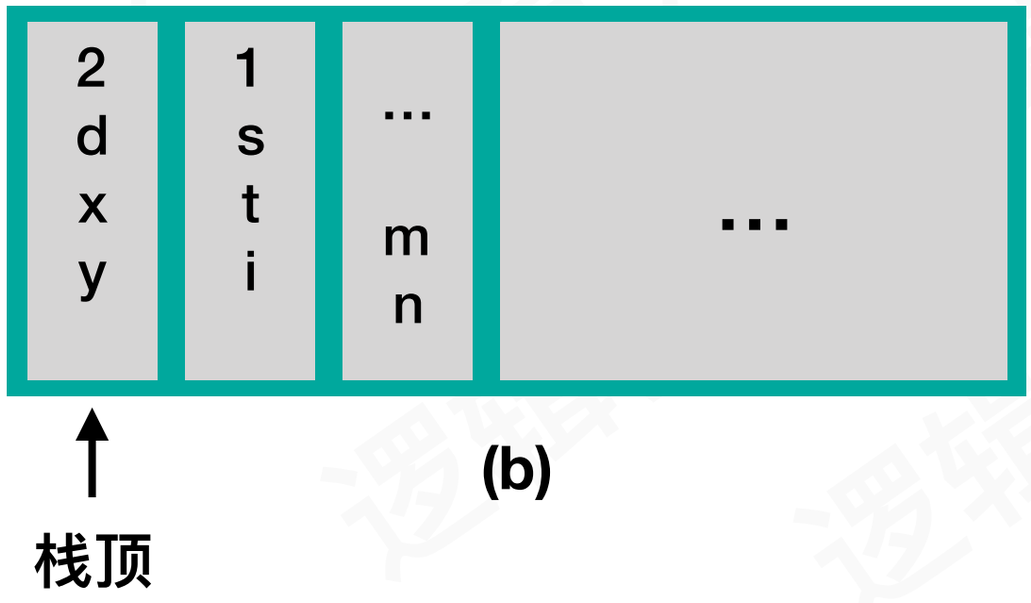
递归函数调用分析



课程研发:CC老师
课程授课:CC老师

递归过程与递归工作栈





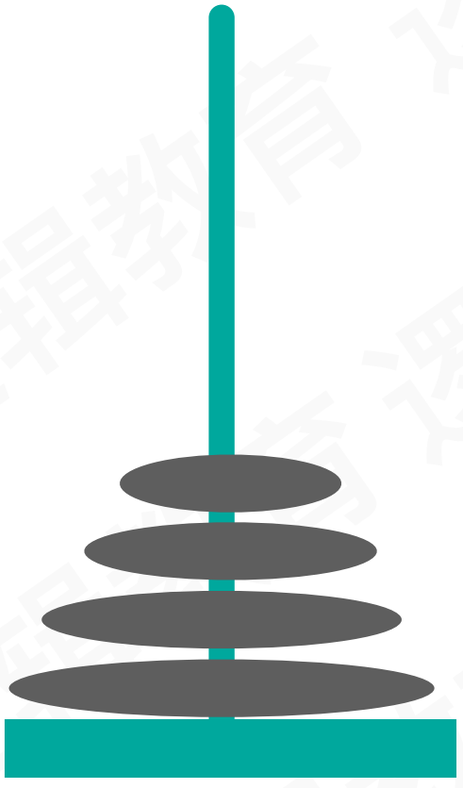

```
int second(int d){
        int x,y;
        //. . .
}

int first(int s ,int t){
        int i;
        //. . .
        second(i)
        //2. 入栈
        //. . .
}

void main( ){
        int m,n;
        first(m ,n);
        //1. 入栈
        //. . .
}
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

Hanoi 塔问题:

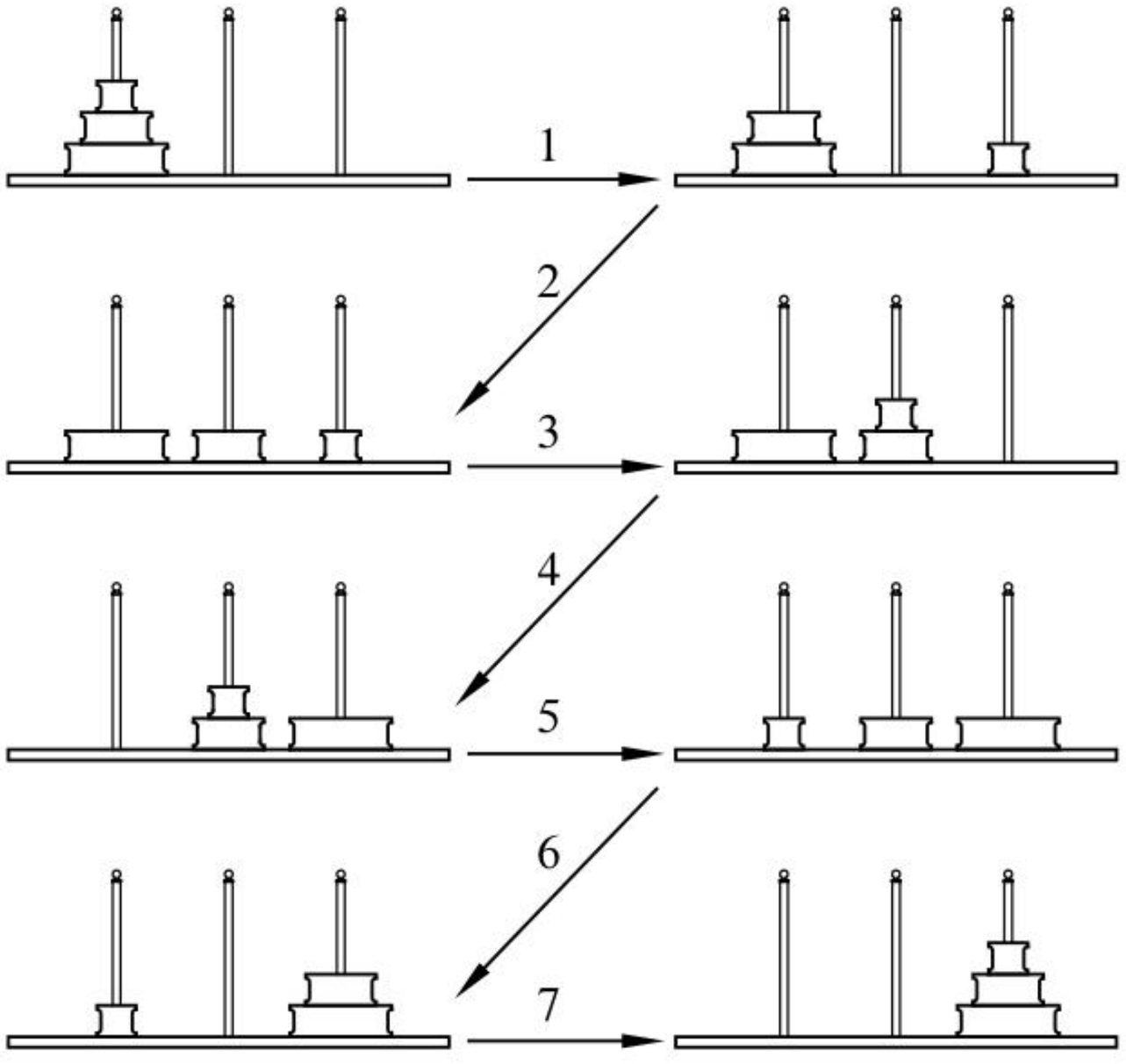
问题描述： 假如有3个分别命名为A,B,C的塔座,在塔座A上插有n个直接大小各不相同的,从小到大的编号为1,2,3...n的圆盘． 现在要求将塔座A上的n个圆盘移动到塔座C上． 并仍然按照同样的顺序叠排． 圆盘移动时必须按照以下的规则:1． 每次只能移动一个圆盘;2． 圆盘可以插在A,B,C的任一塔座上;3． 任何时刻都不能将一个较大的圆盘压在小的圆盘之上．



课程研发:CC老师
课程授课:CC老师

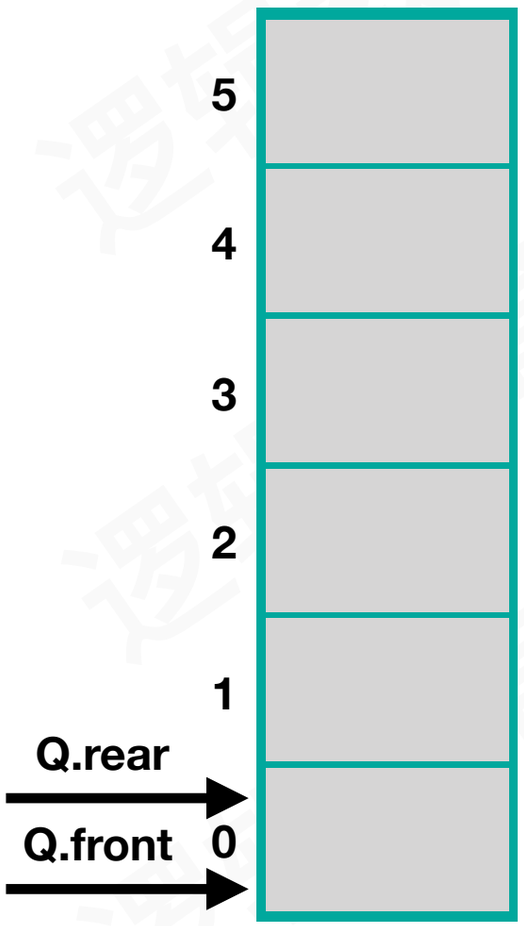
逻辑教育
Logic education

Hanoi 塔问题:

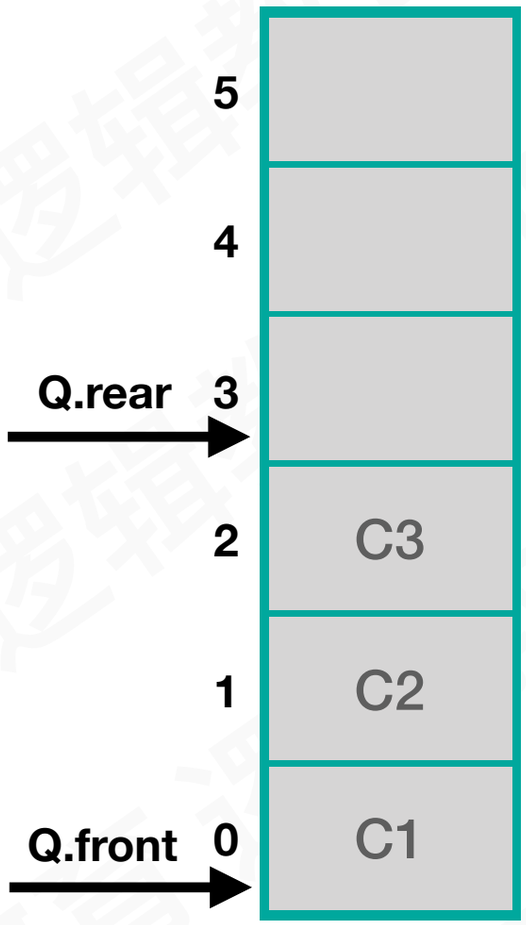


课程研发:CC老师
课程授课:CC老师

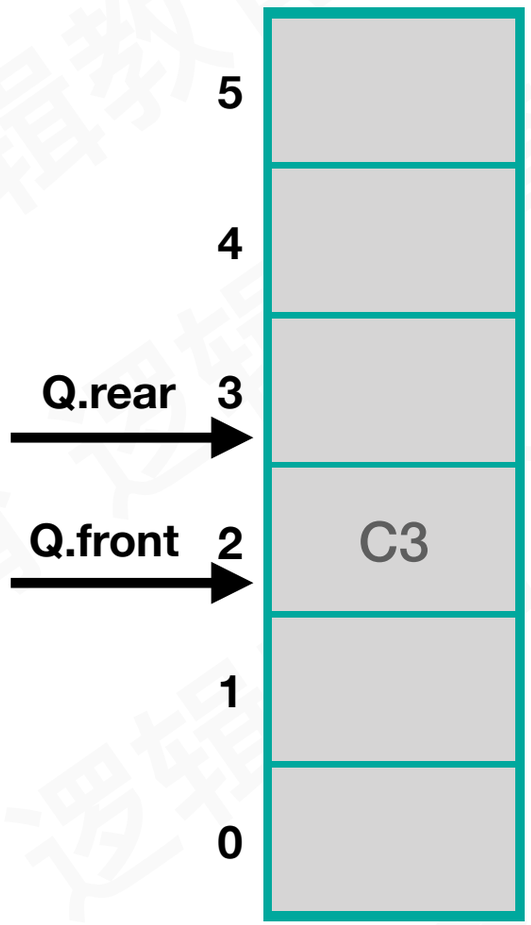
队列的表示与操作实现



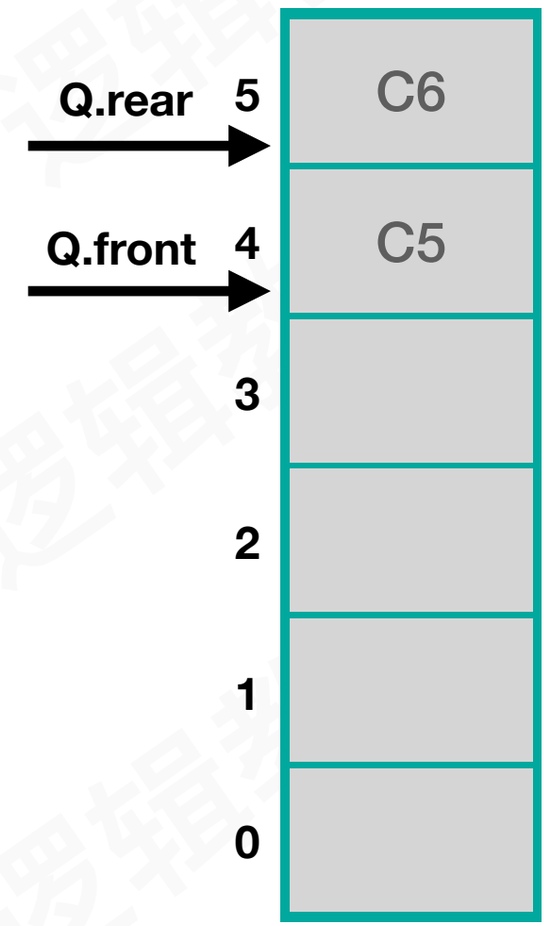
(a) 空队列



(b) c1 c2 c3相继入队



(c) c1 c2 相继出队

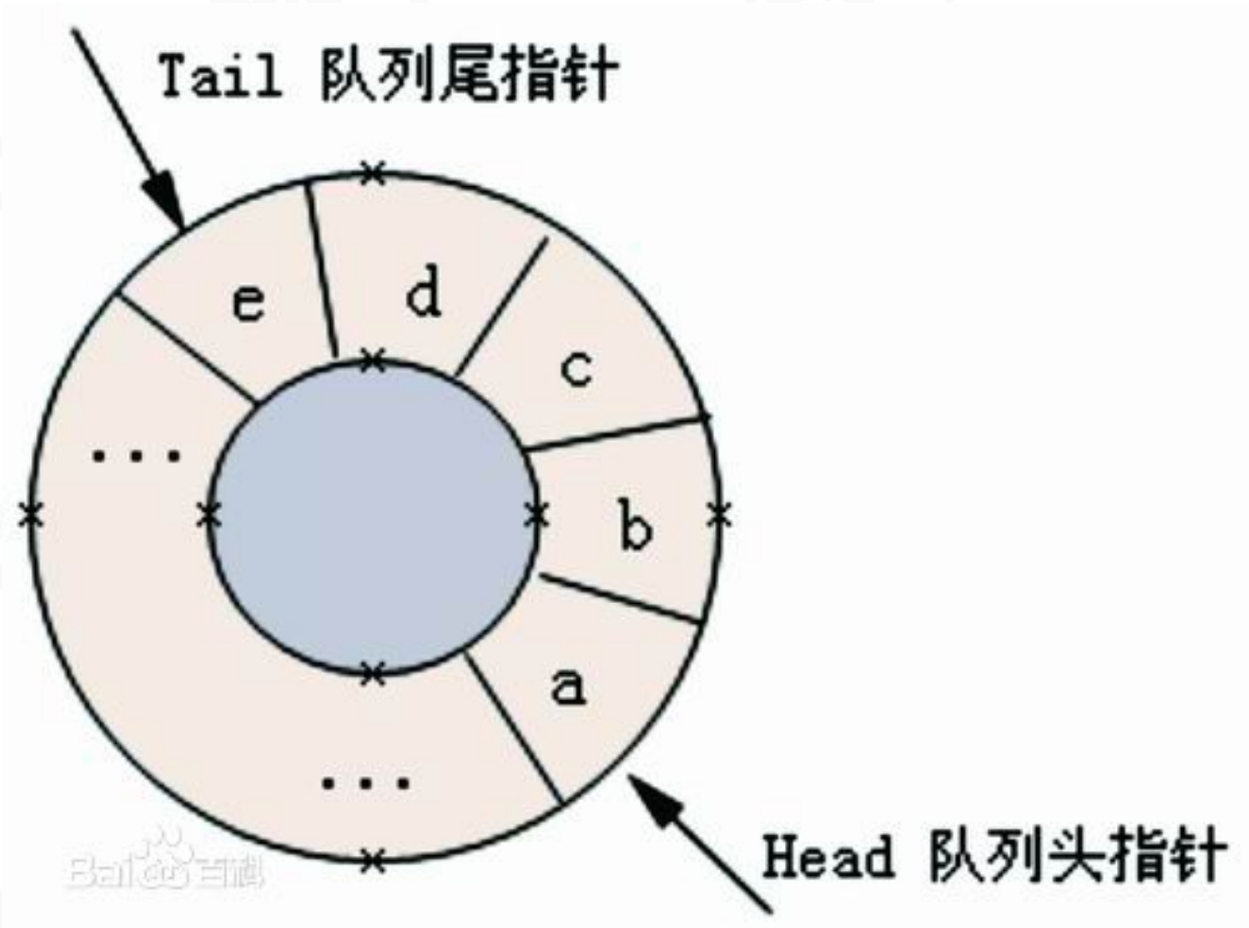


(d) C4 C5 C6相继入队;
C3 C4 相继出队

课程研发:CC老师
课程授课:CC老师

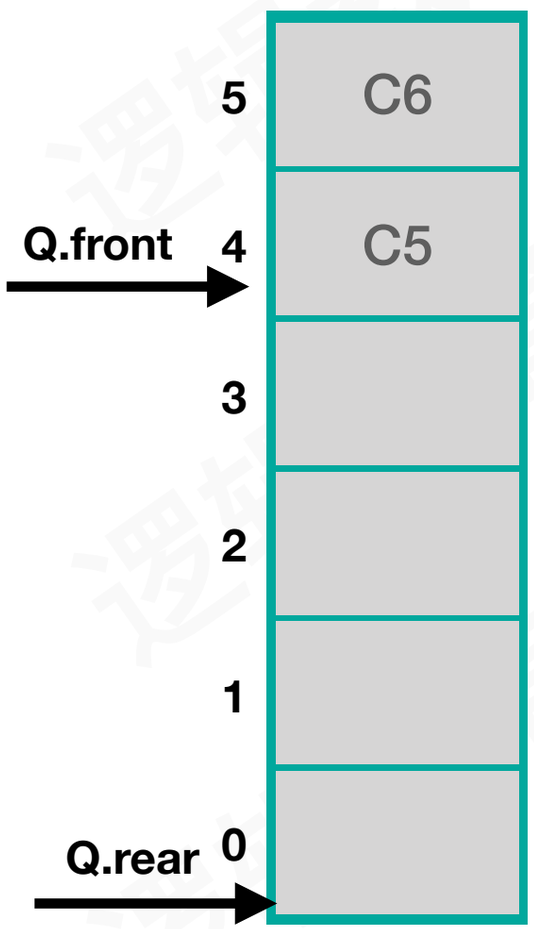
逻辑教育
Logic education

循环队列

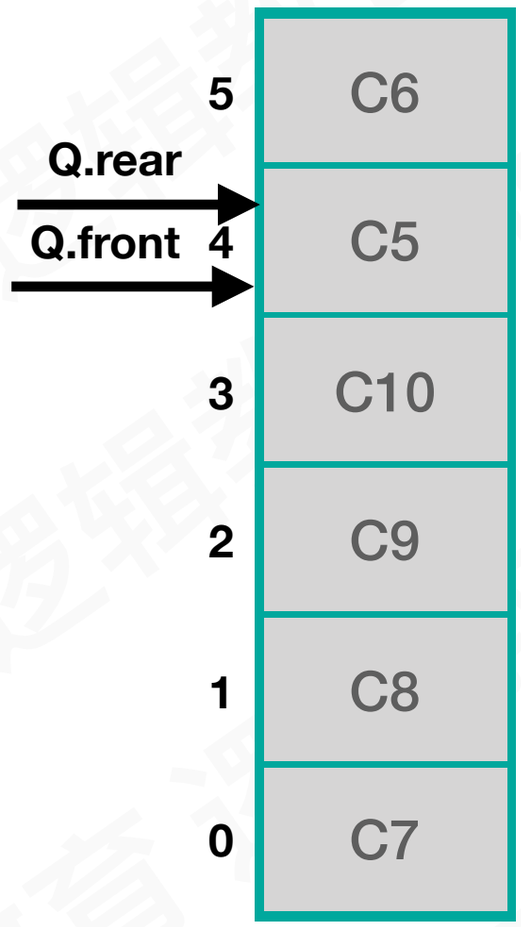


课程研发:CC老师
课程授课:CC老师

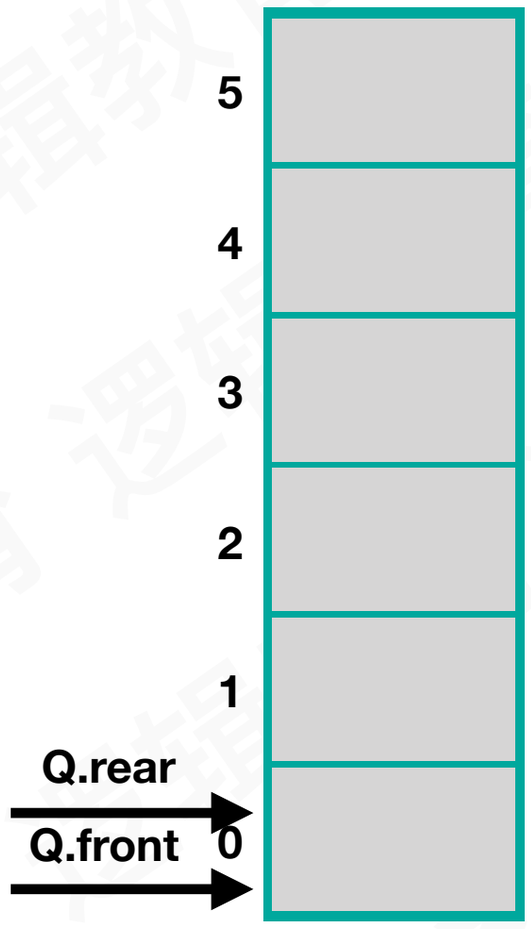
循环队列中头尾指针与元素之间关系



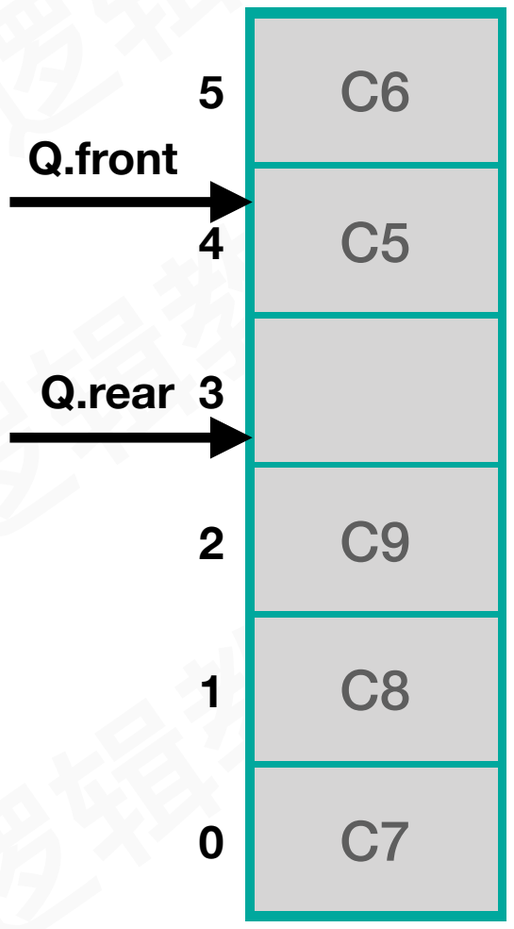
(a) 一般情况



(b) 队列空间被占满



(c) 空队列



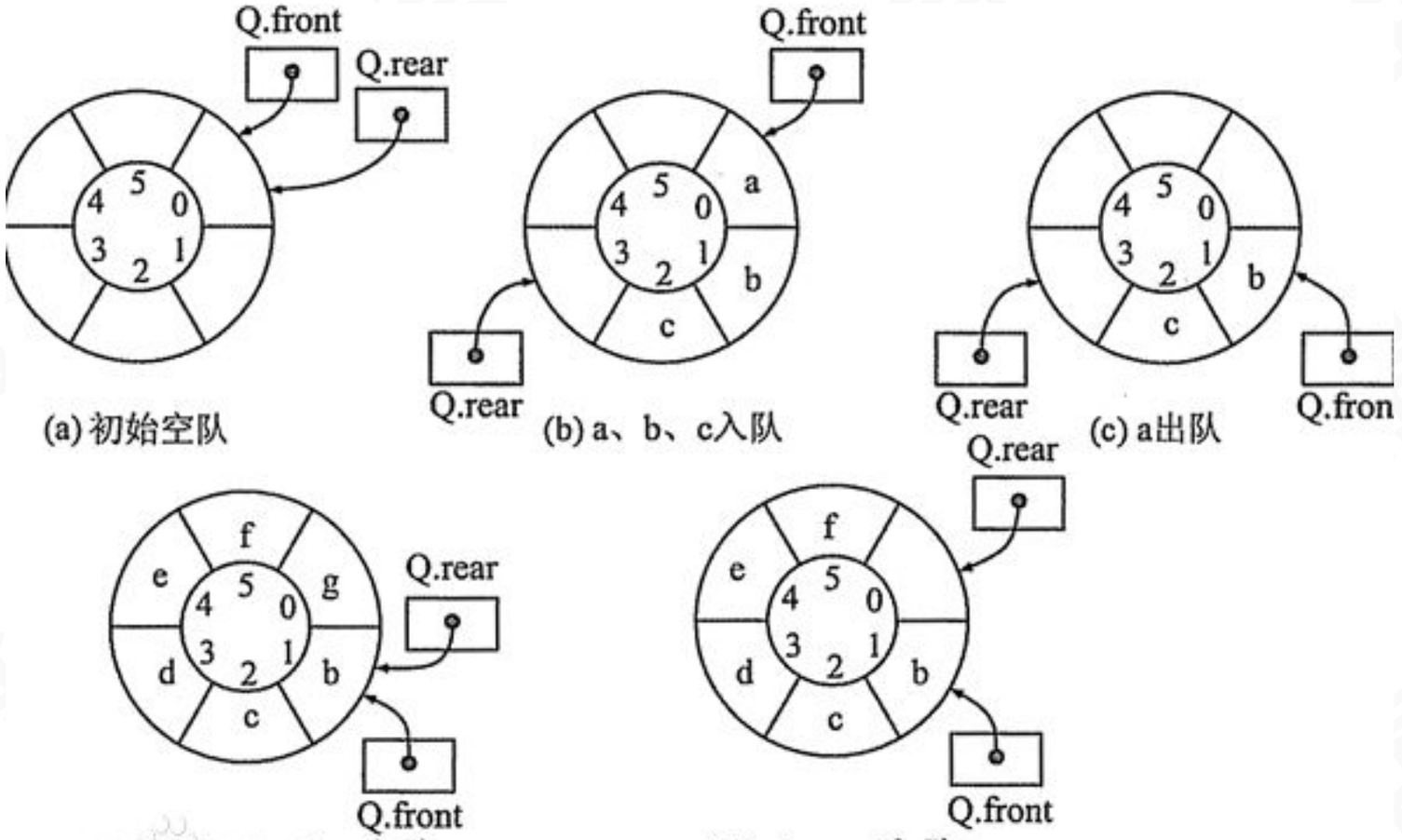
(d) 呈”满”状态循环队列

判断队空: Q.front == Q.rear;

判断队满: (Q.rear + 1) % MAXSIZE == Q.front

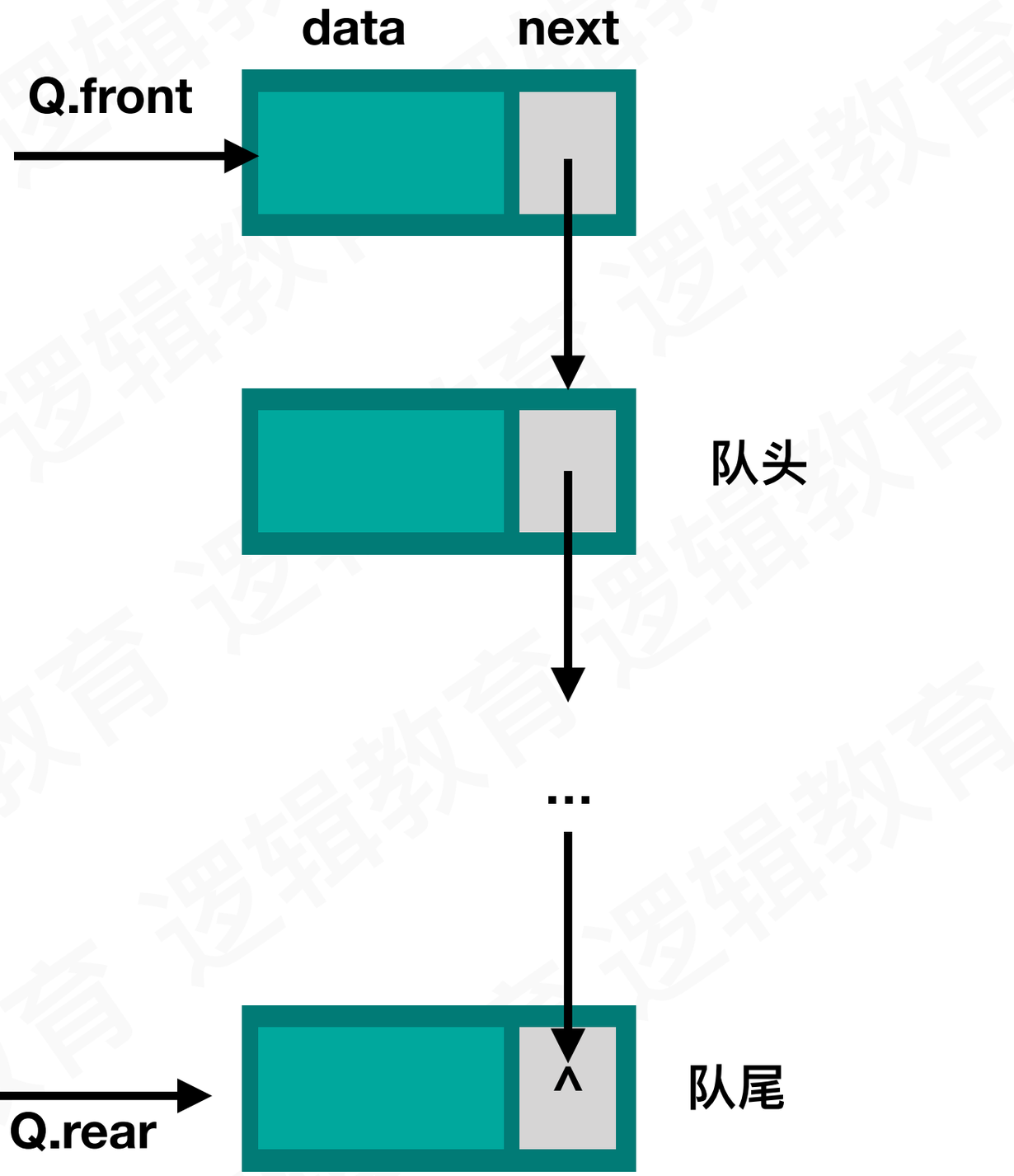
课程研发:CC老师
课程授课:CC老师

循环队列



逻辑教育
Logic education

链式队列表示与实现



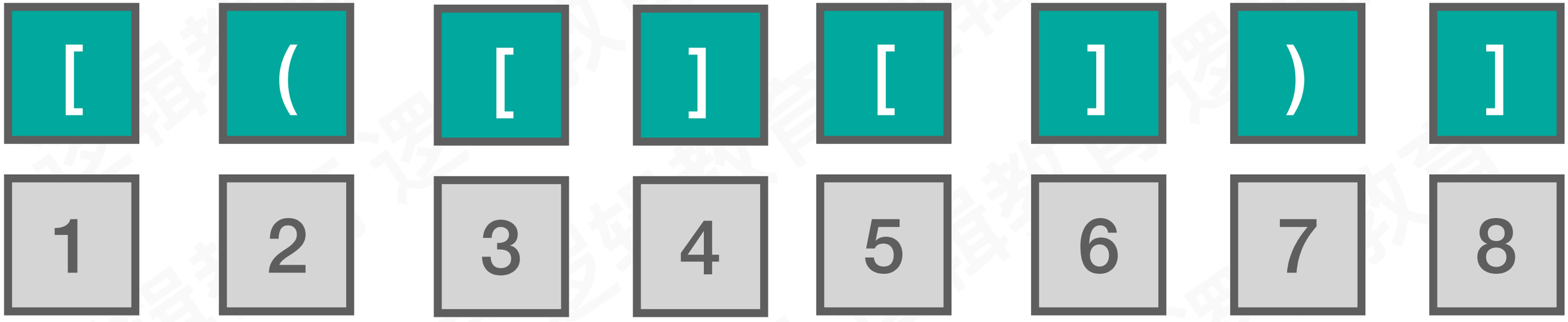
课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

算法题目(字节跳动算法面试题):

括号匹配检验:

假设表达式中允许包含两种括号:圆括号与方括号,其嵌套顺序随意,即([]()) 或者[([][])]都是正确的.而这[(]或者(()])或者([()) 都是不正确的格式. 检验括号是否匹配的方法可用”期待的急迫程度"这个概念来描述.例如,考虑以下括号的判断: [([ ] [ ])]



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

20. Valid Parentheses

课程研发:CC老师
课程授课:CC老师

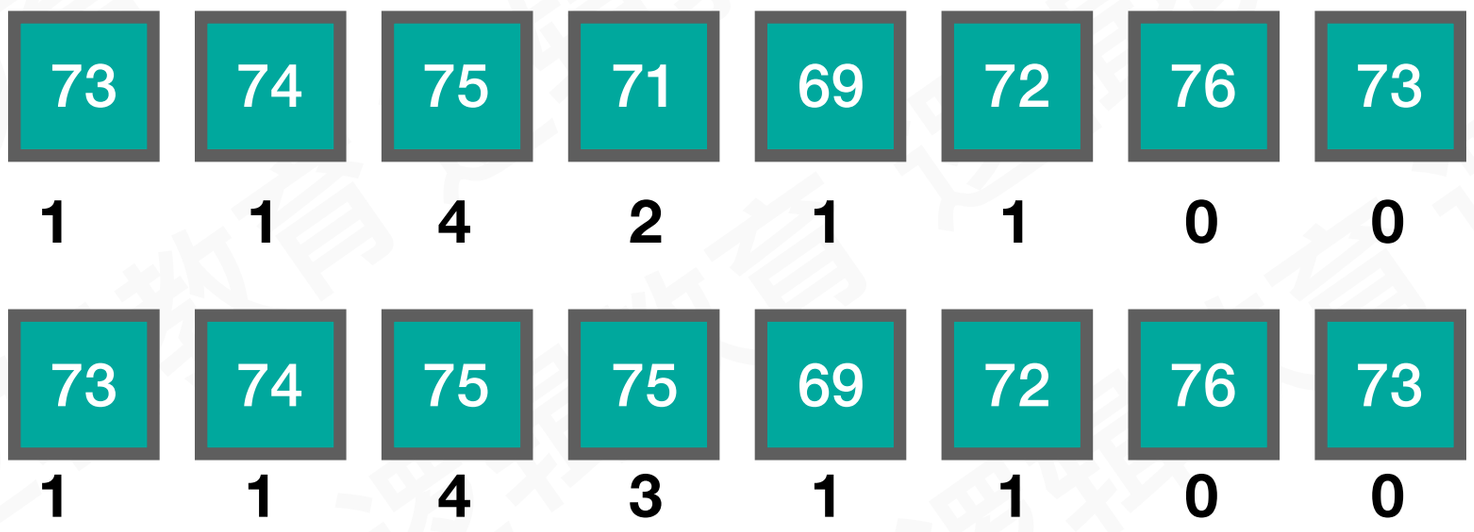
逻辑教育
Logic education

算法题目(LeetCode 中级难度):

每日气温:

题目: 根据每日气温列表，请重新生成一个列表，对应位置的输入是你需要再等待多久温度才会升高超过该日的天数。如果之后都不会升高，请在该位置0来代替。例如，给定一个列表 temperatures = [73, 74, 75, 71, 69, 72, 76, 73]，你的输出应该是 [1, 1, 4, 2, 1, 1, 0, 0]。

提示：气温 列表长度的范围是 [1, 30000]。每个气温的值的均为华氏度，都是在 [30, 100] 范围内的整数。



题目解析: 暴力破解(1)

每日气温:

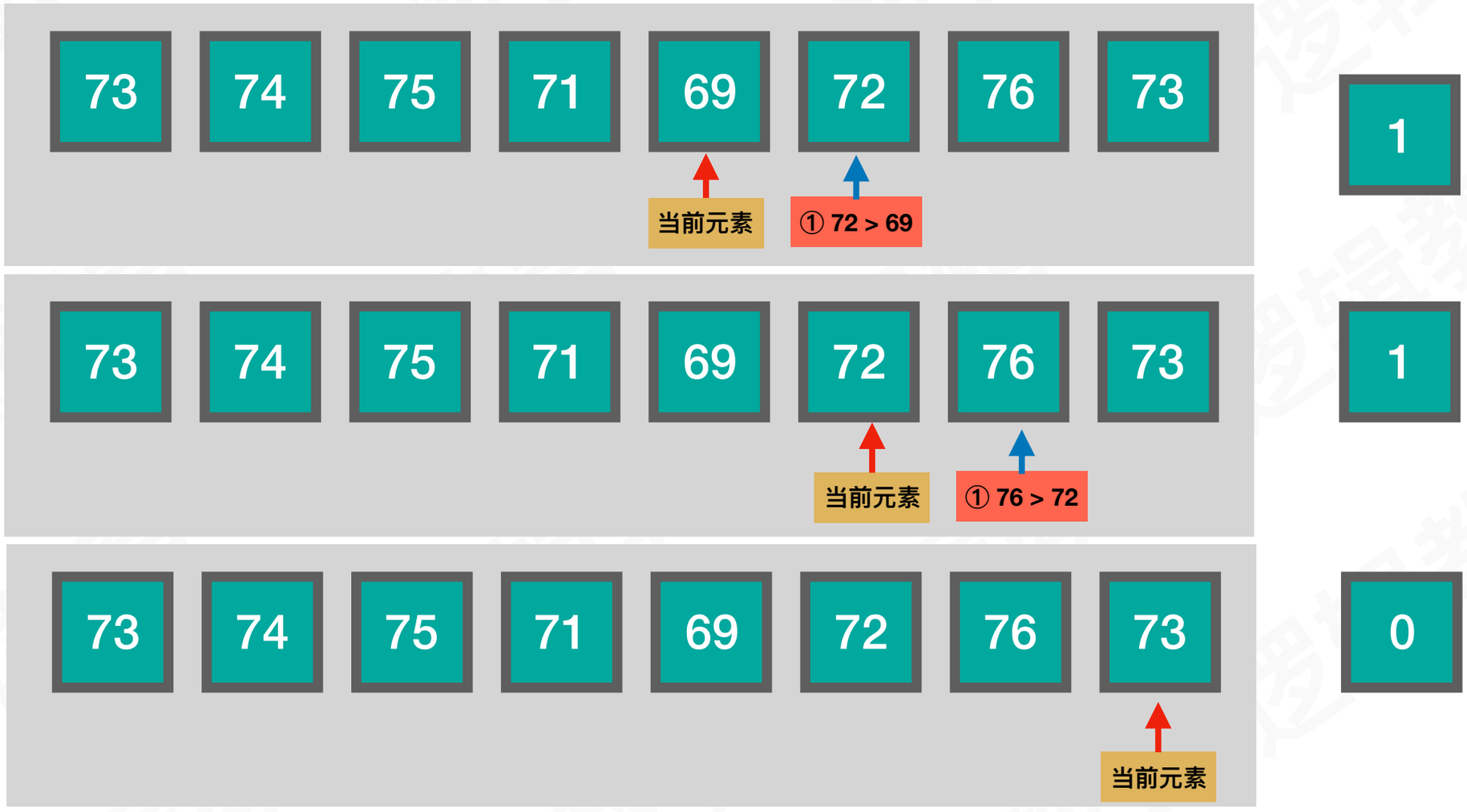
解题关键: 实际上就是找当前元素 从[i,TSize] 找到大于该元素时. 数了几次. 首先最后一个元素默认是0,因为它后面已经没有元素了.

课程研发:CC老师
课程授课:CC老师

题目解析: 暴力破解(1)

课程研发:CC老师
课程授课:CC老师

题目解析: 暴力破解(1)



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

题目解析: 暴力破解(1)

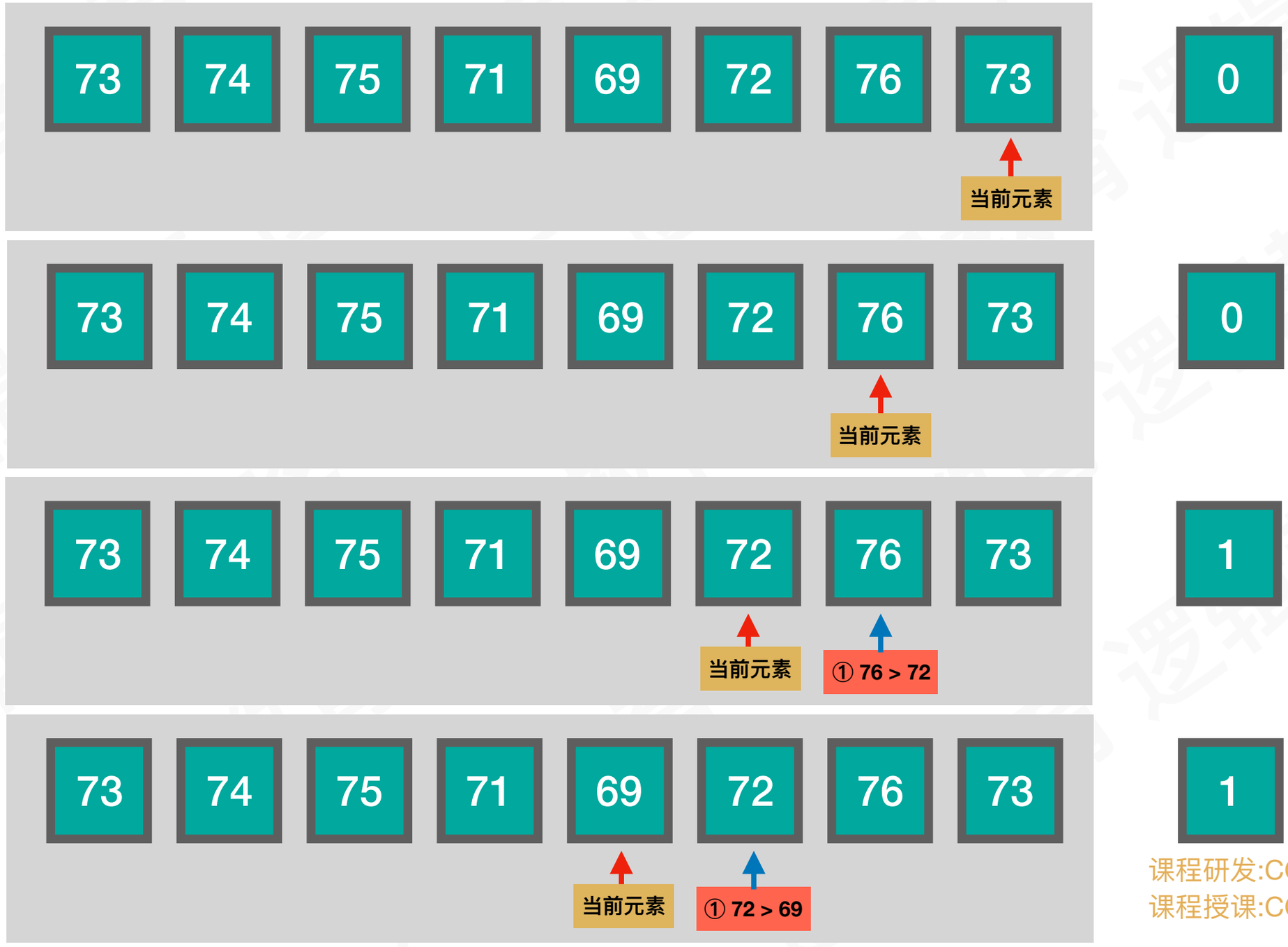
代码实现!

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

题目解析: 跳跃对比

思考: 怎样减少遍历次数?

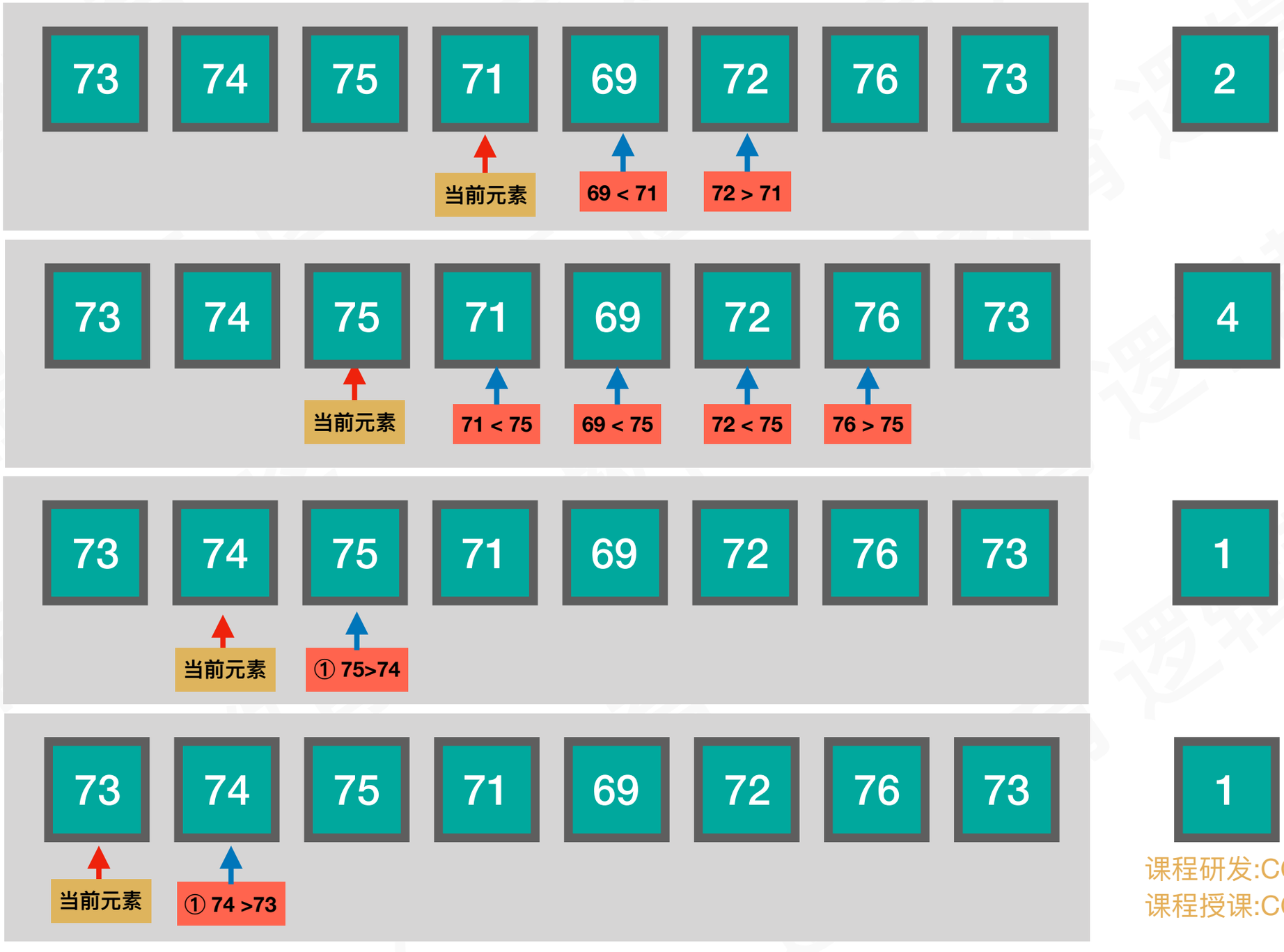


课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

题目解析: 跳跃对比

思考: 怎样减少遍历次数?



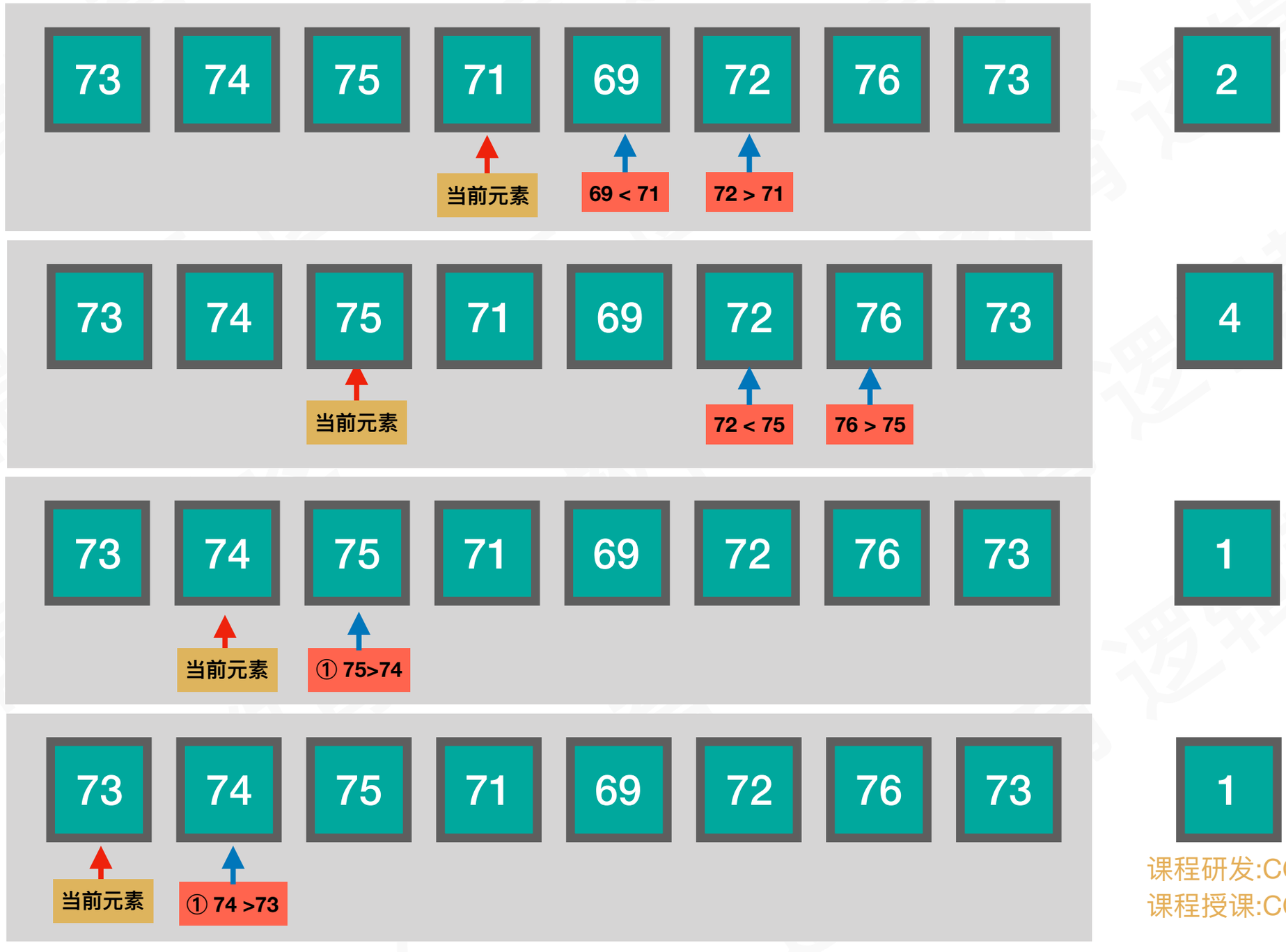
课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

从右到左的计算,那么如果遇到计算过的位置则不需要重复计算.
例如,当计算75时, 遍历到71时,则直接使用计算好的71上对应的值2. 那么我就直接跳2步再进行比较. 利用已有的结果,减少遍历次数.

题目解析: 跳跃对比

思考: 怎样减少遍历次数?



课程研发:CC老师
课程授课:CC老师

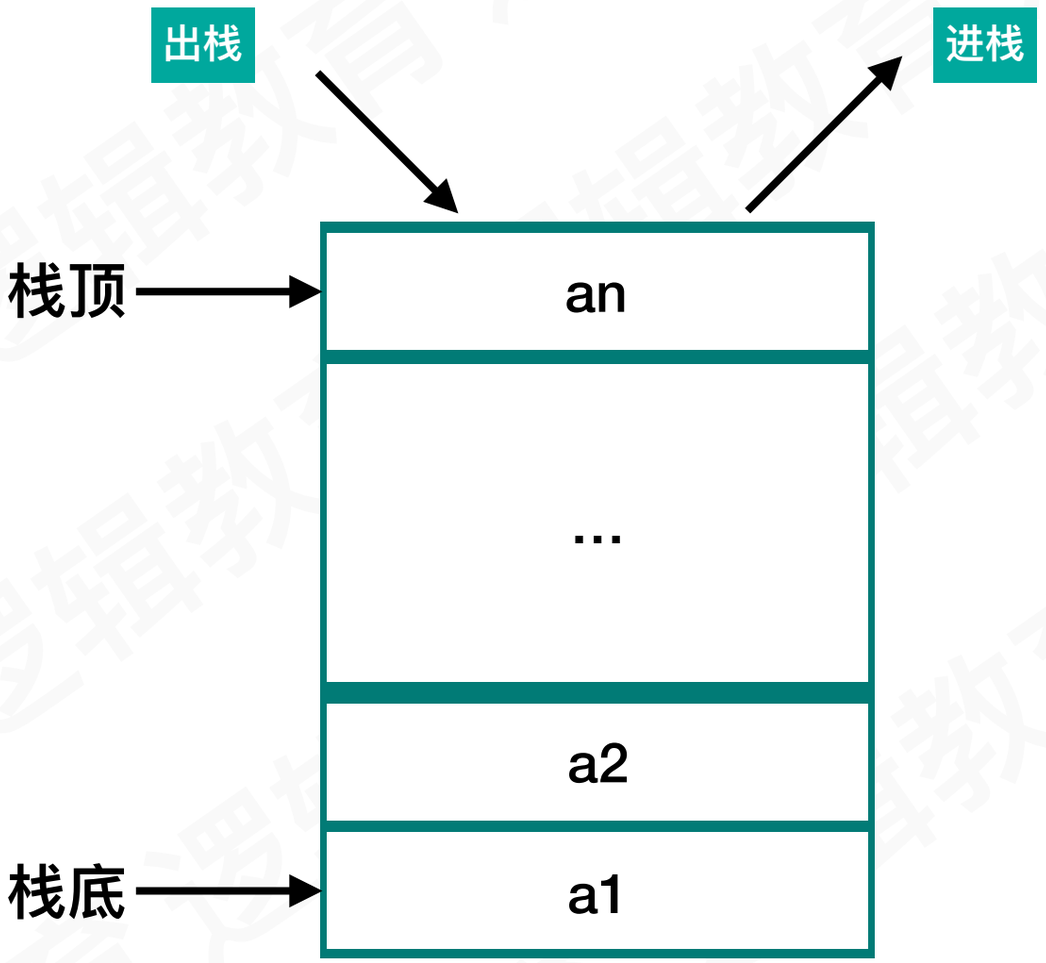
逻辑教育
Logic education

题目解析: 跳跃对比

代码实现!

课程研发:CC老师
课程授课:CC老师

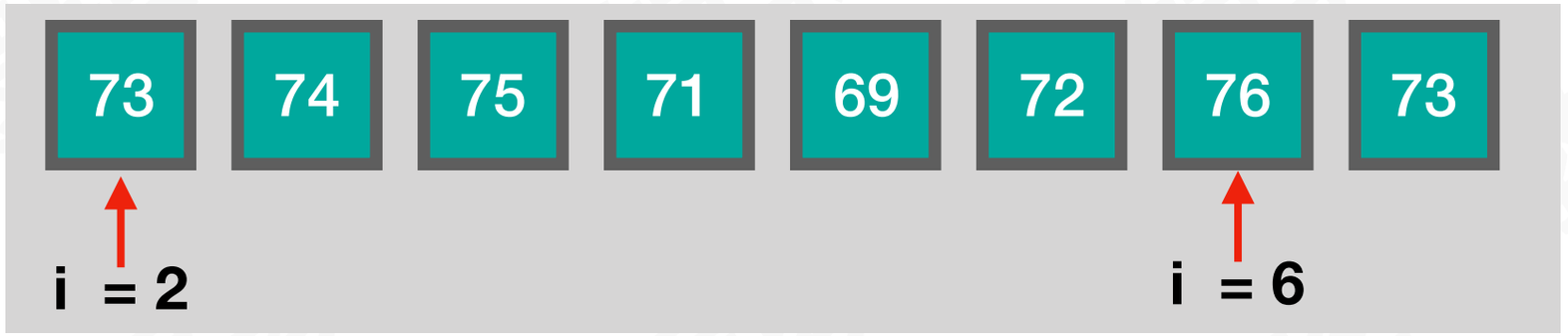
栈结构

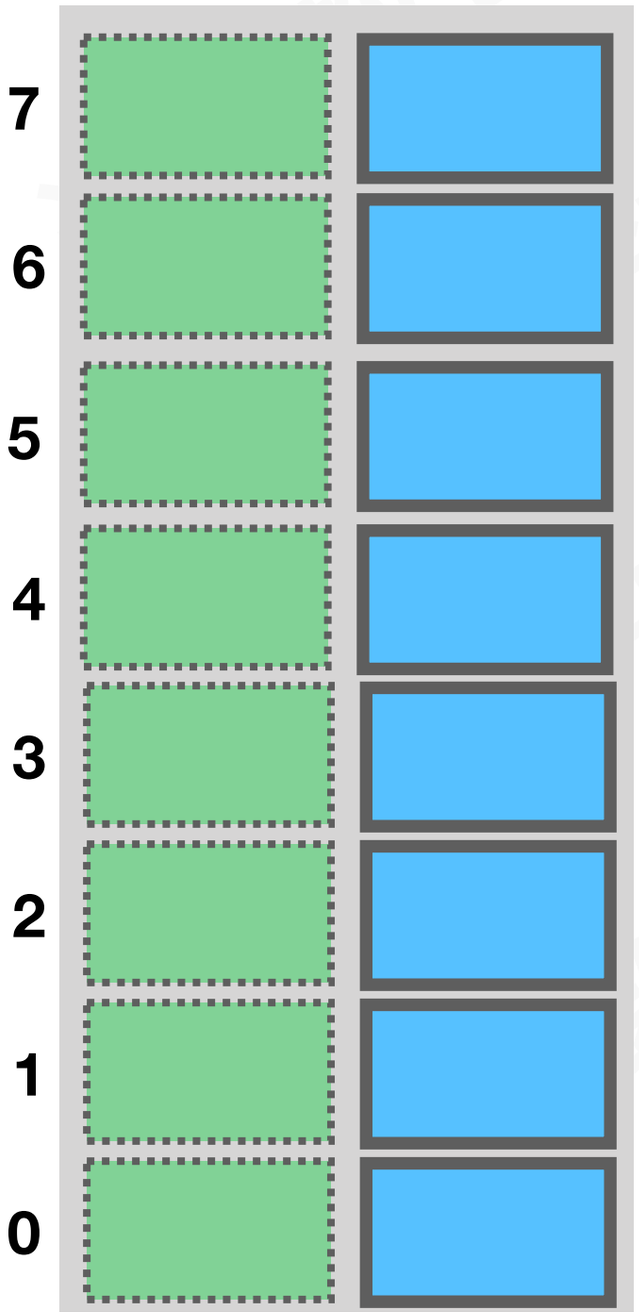


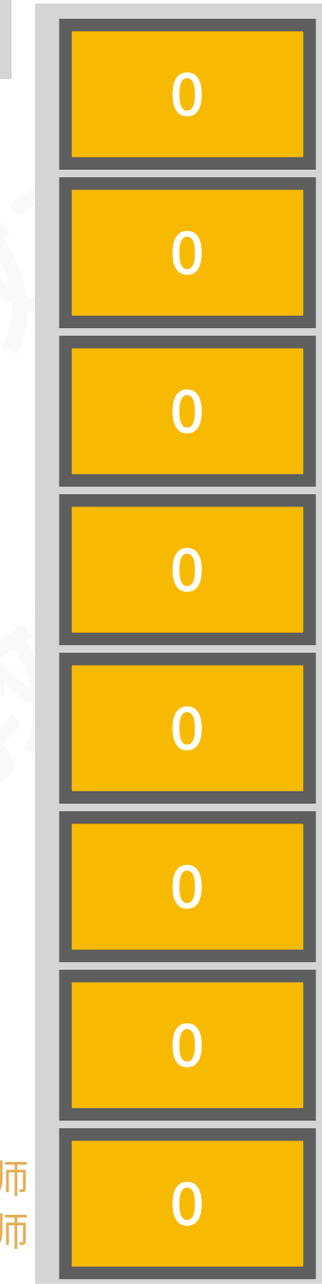
栈的示意图

1. 栈中存储的是元素的索引值index；
2.将当前元素和栈顶元素比较；
如果栈为空，那么直接将当前元素索引index 存储到栈中；
如果栈顶元素>当前元素，则将当前元素索引index 存储到栈中；
如果栈顶元素<当前元素，则将当前元素索引index-栈顶元素index，计算完毕则将当前栈顶元素移除，将当前元素索引index 存储到栈中

题目解析: 栈

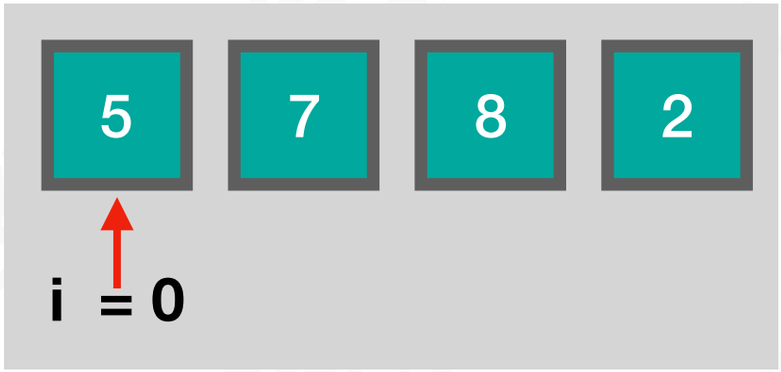


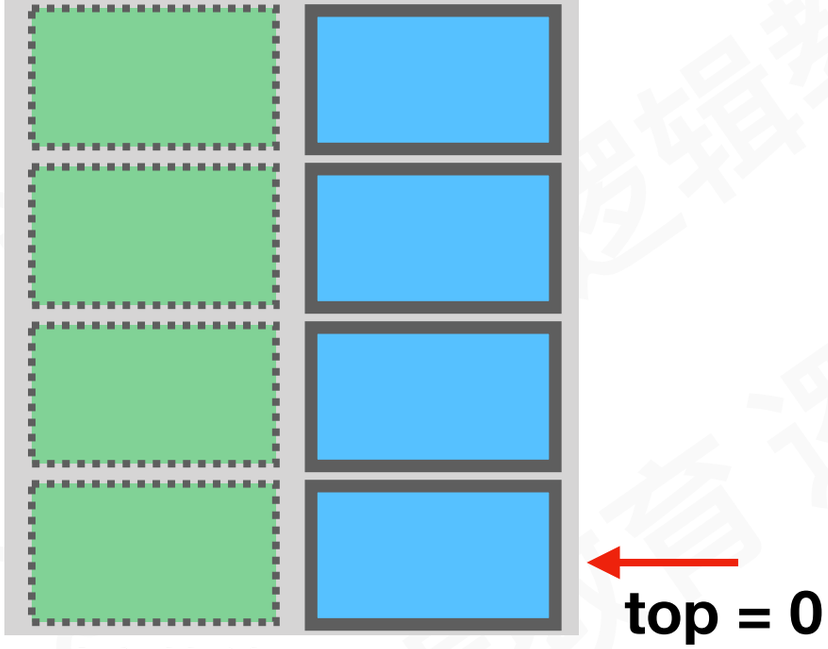


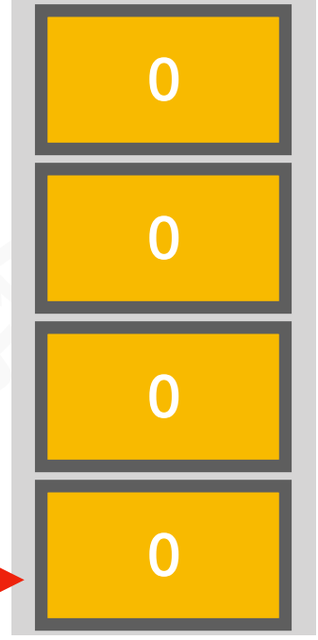


1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈



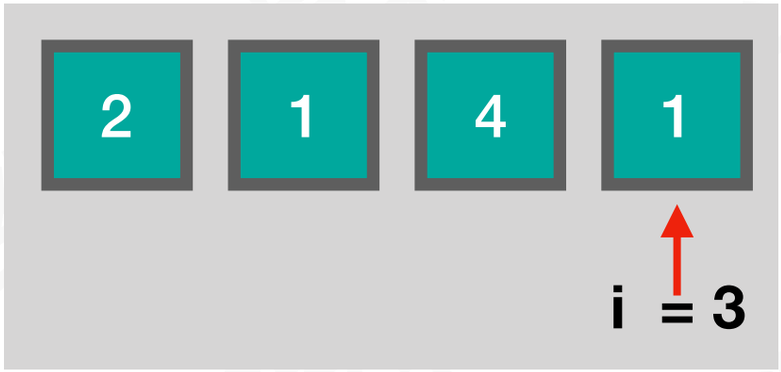


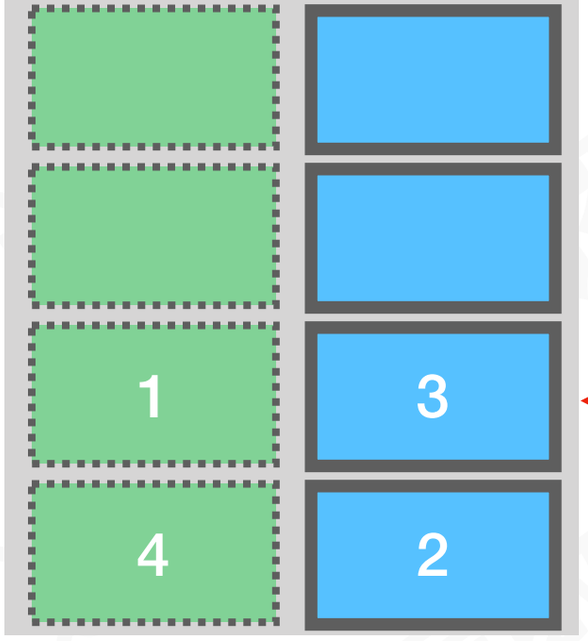


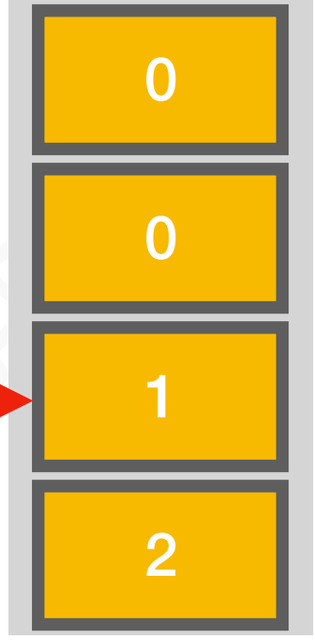
课程研发:CC老师
课程授课:CC老师

1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈



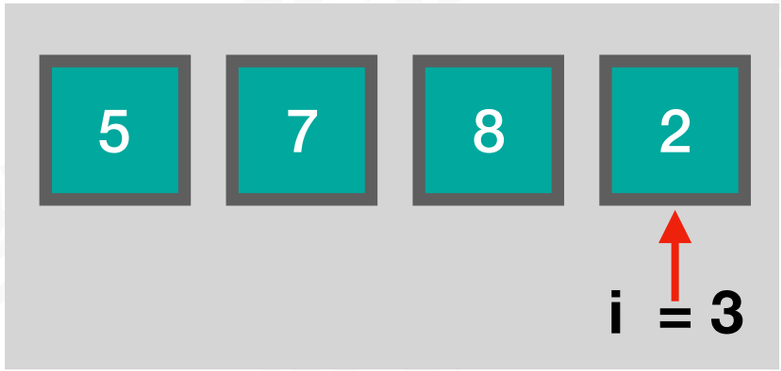


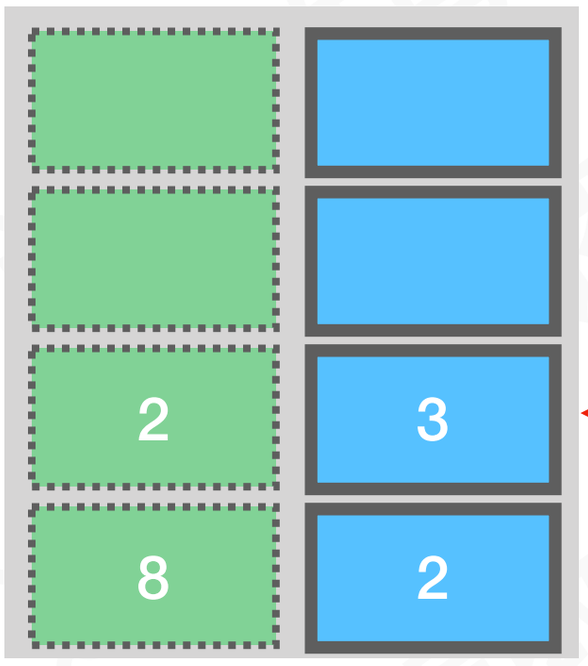


课程研发:CC老师
课程授课:CC老师

1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈

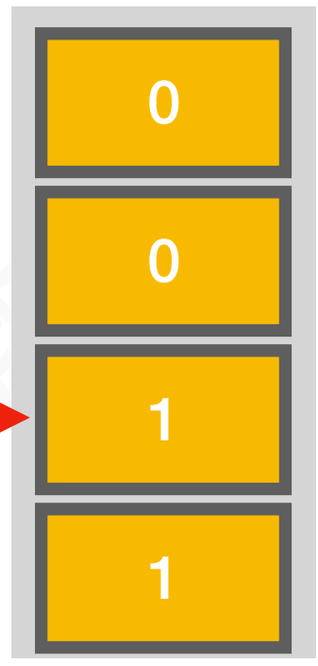




top = 1

```
循环第0次,i = 0
i= 0; top = 1, StackIndex[1] = 0

循环第1次,i = 1
temp = 0; result[0] = 1, top = 0
i= 1; top = 1, StackIndex[1] = 0

循环第2次,i = 2
temp = 1; result[1] = 1, top = 0
i= 2; top = 1, StackIndex[1] = 0

循环第3次,i = 3
i= 3; top = 2, StackIndex[2] = 268834412
```

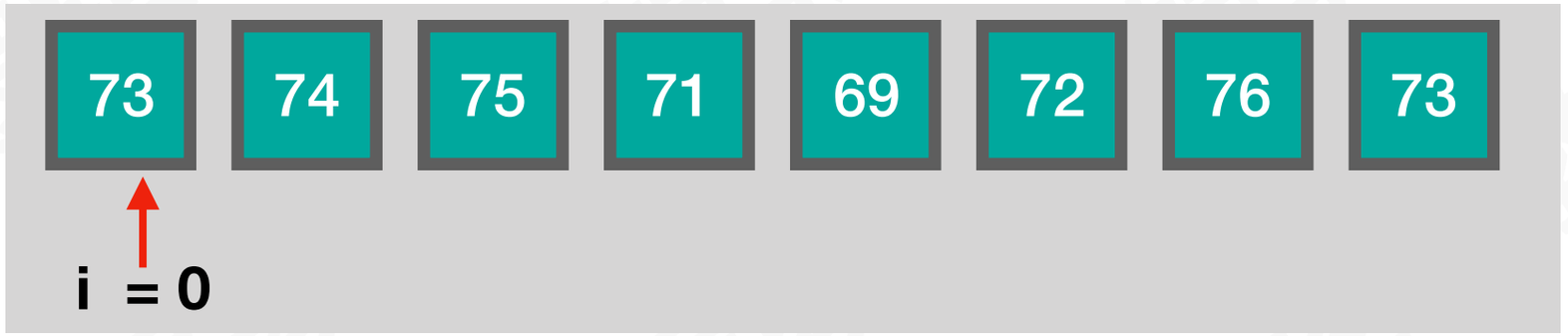
temp = 1

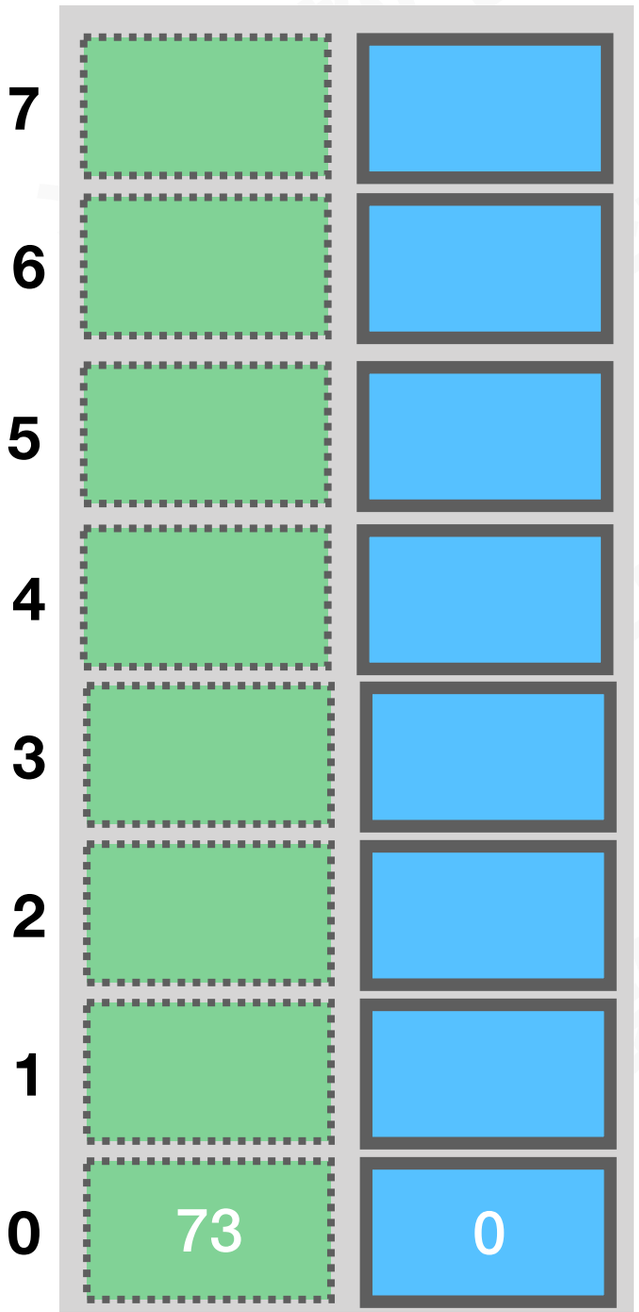


课程研发:CC老师
课程授课:CC老师

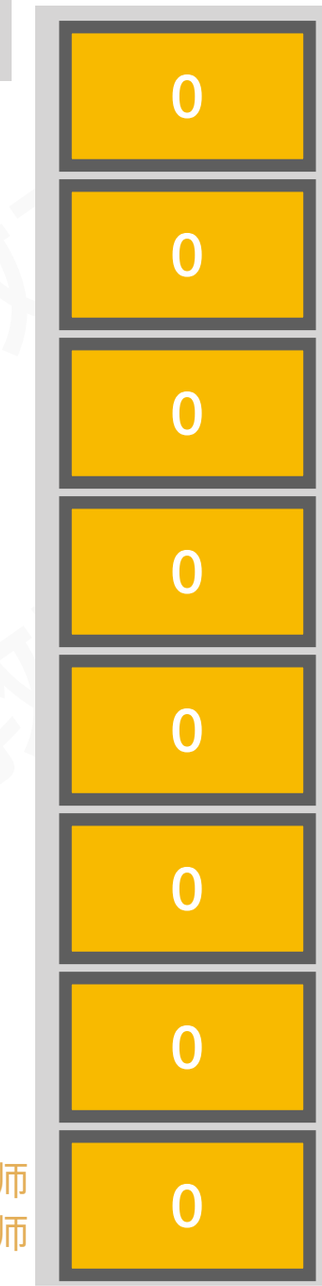
1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈



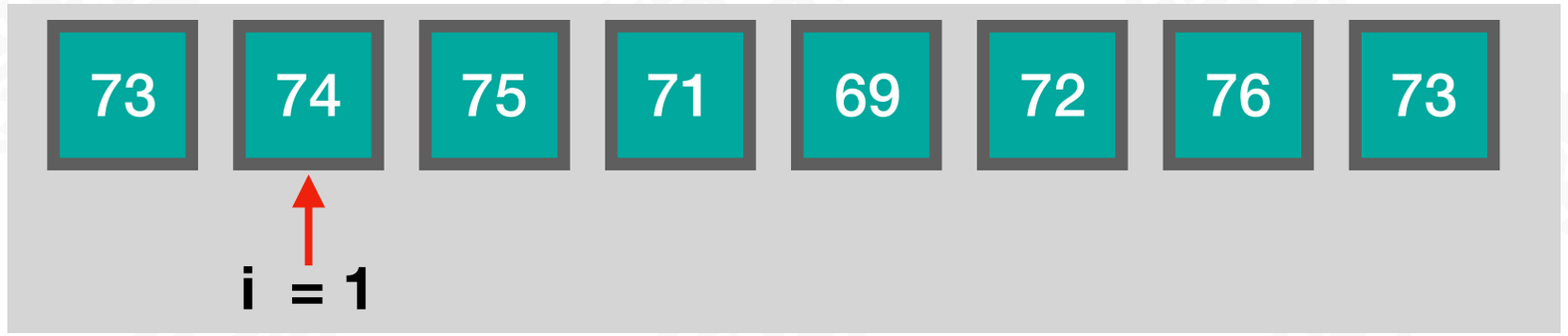


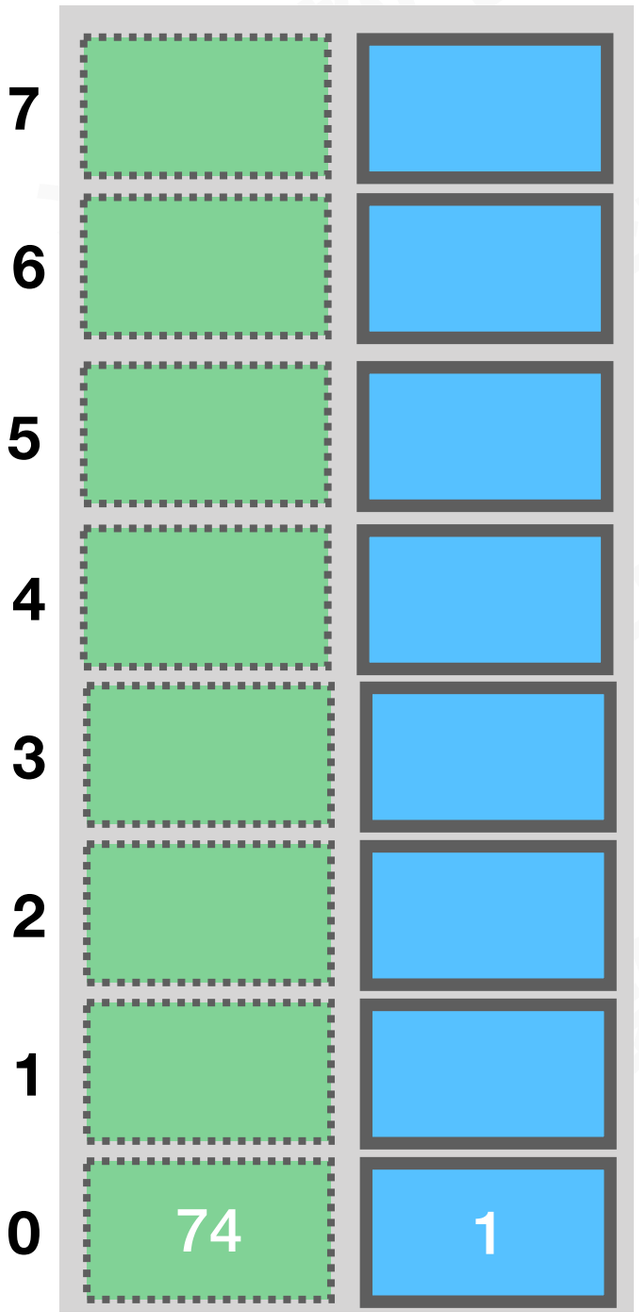
循环第0次,i = 0
i= 0; StackIndex[0] = 0 top = 1



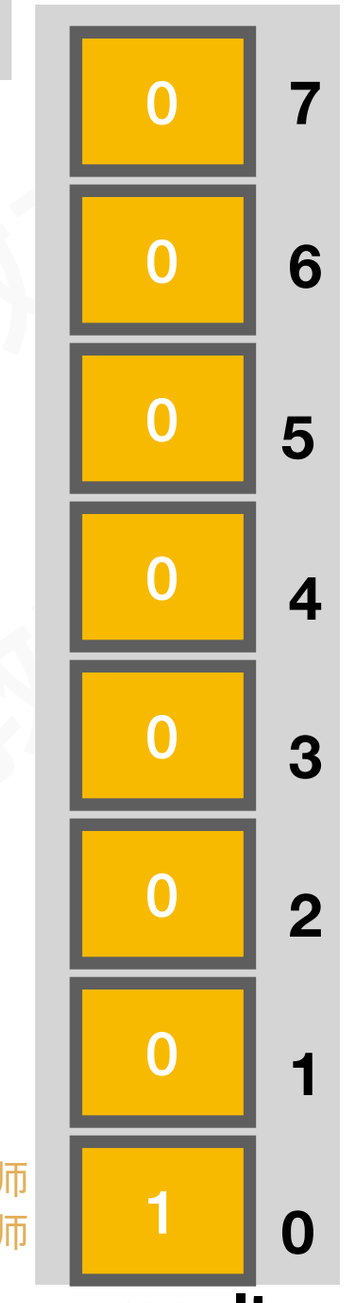
1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈



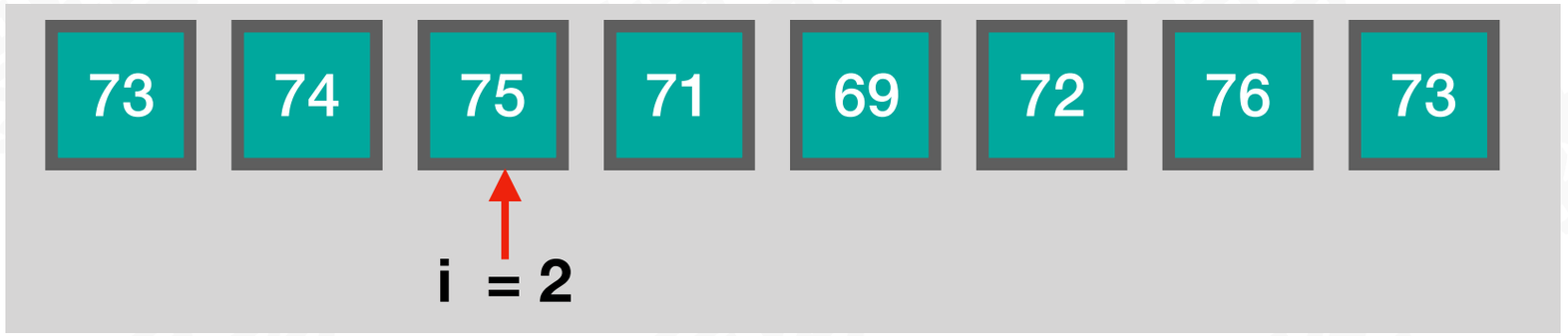


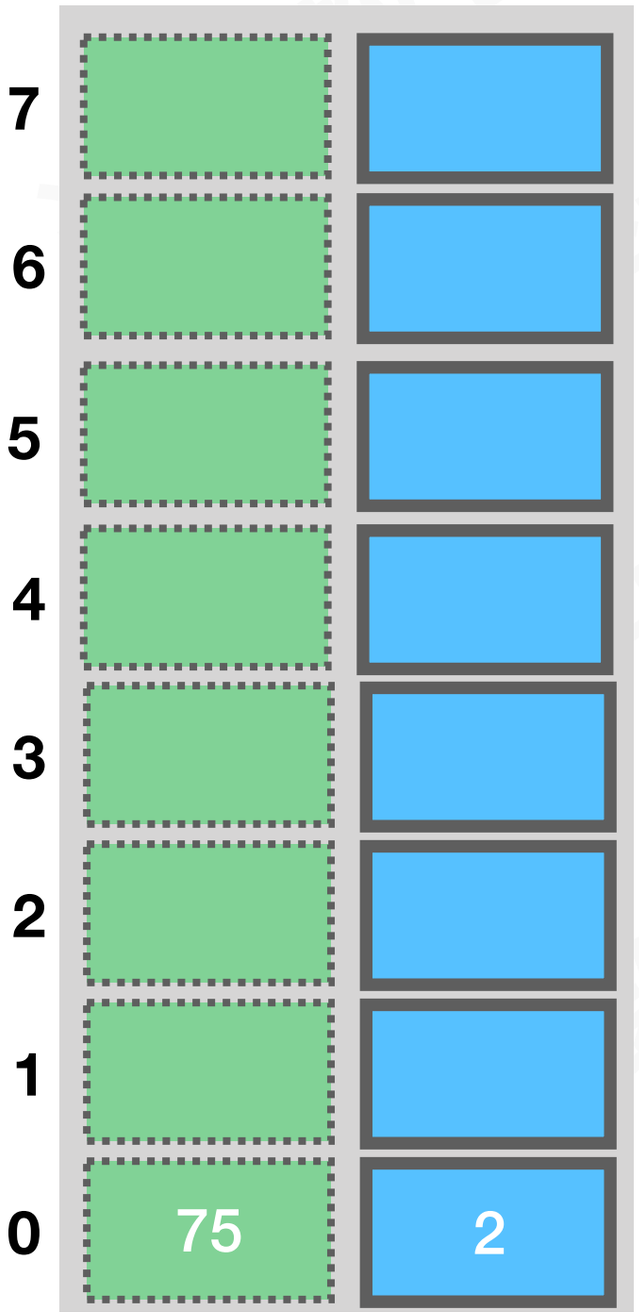

```
循环第0次,i = 0
i= 0;  StackIndex[0] = 0  top = 1

循环第1次,i = 1
temp = 0; result[0] = 1, top = 0
i= 1;  StackIndex[0] = 1  top = 1
```



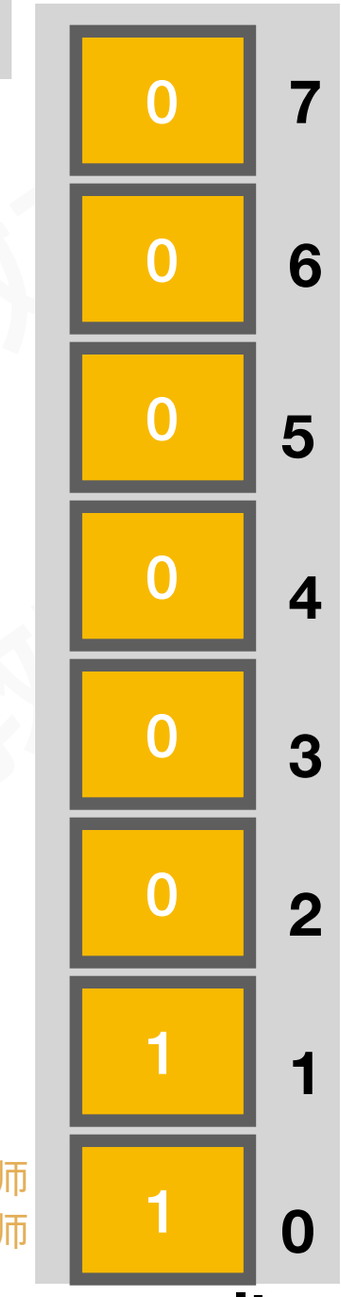
1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈



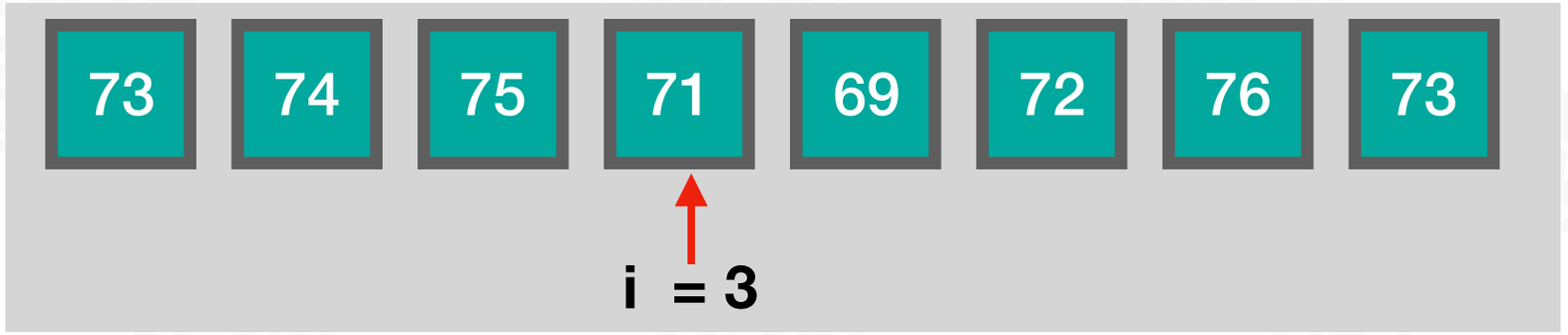


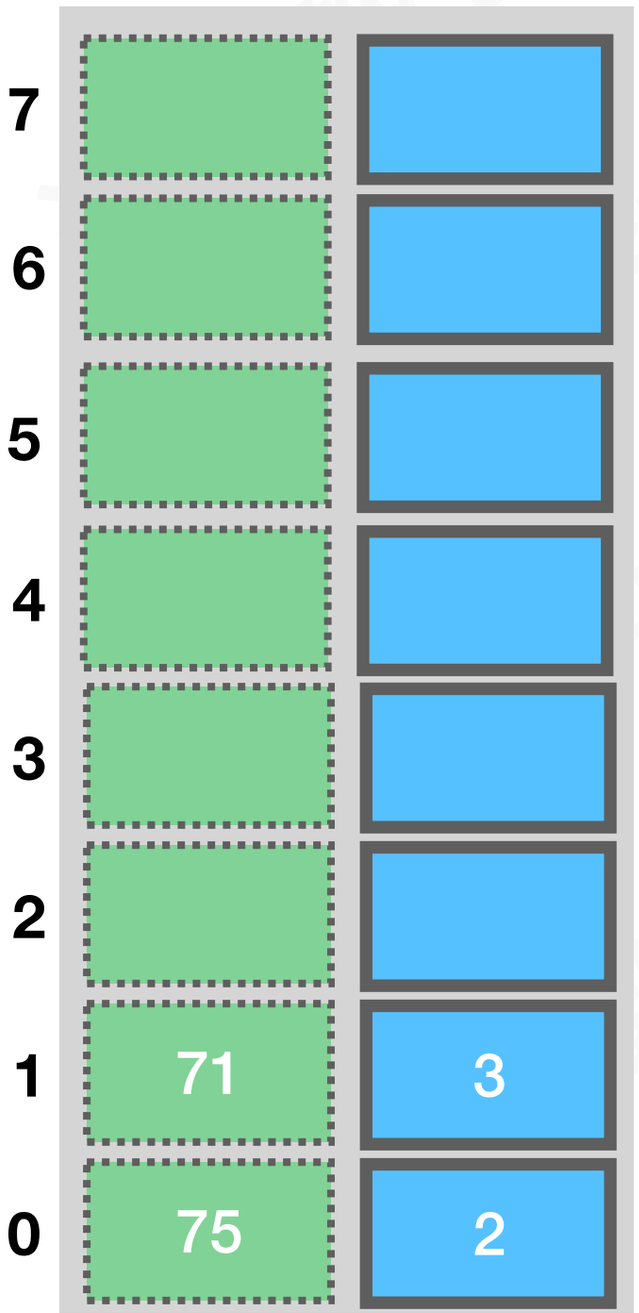

```
循环第0次,i = 0
i= 0;  StackIndex[0] = 0  top = 1

循环第1次,i = 1
temp = 0; result[0] = 1, top = 0
i= 1;  StackIndex[0] = 1  top = 1


循环第2次,i = 2
temp = 1; result[1] = 1, top = 0
i= 2;  StackIndex[0] = 2  top = 1
```



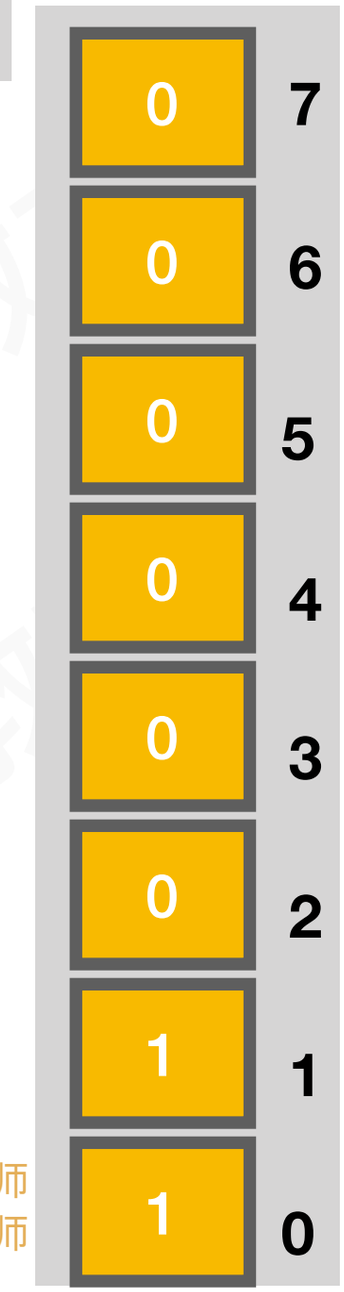
1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈



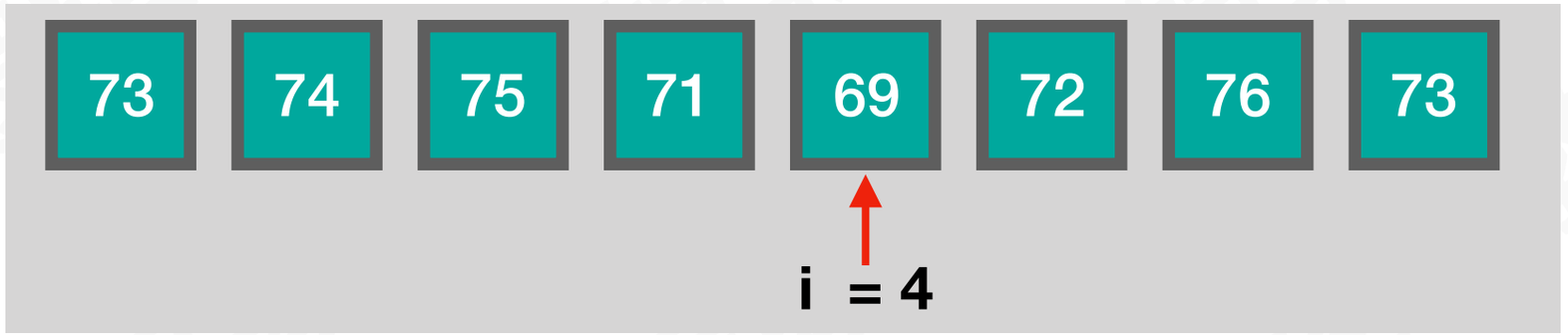


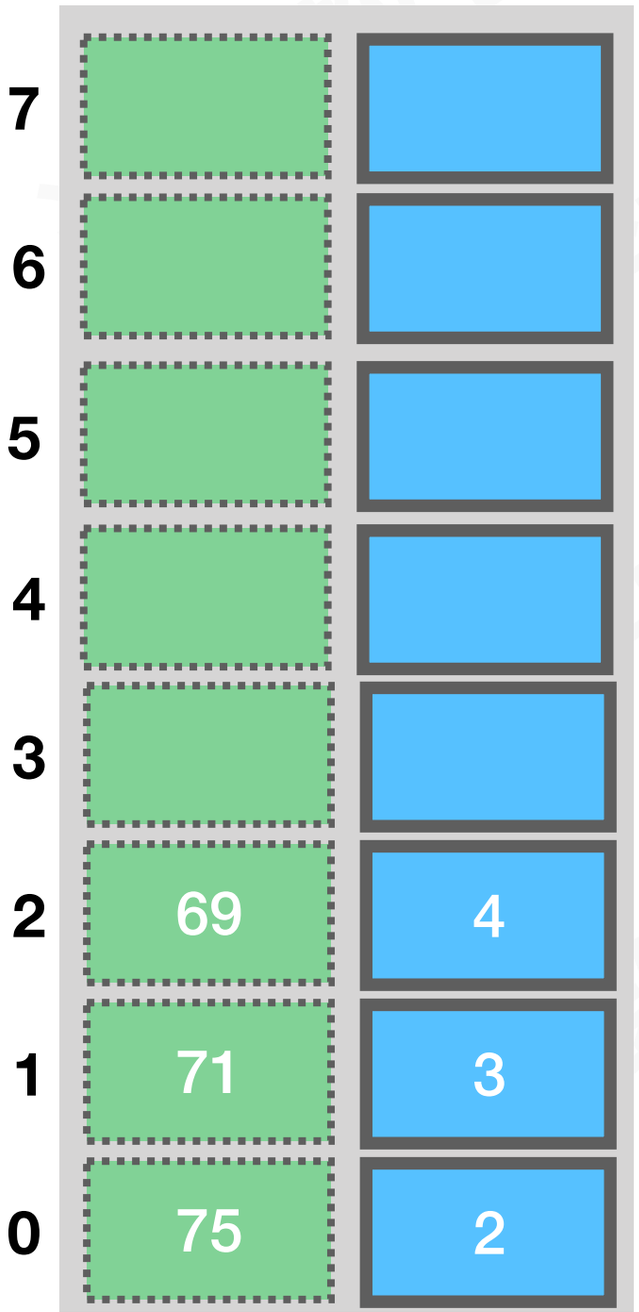

```
循环第0次,i = 0
i= 0;  StackIndex[0] = 0  top = 1

循环第1次,i = 1
temp = 0; result[0] = 1, top = 0
i= 1;  StackIndex[0] = 1  top = 1

循环第2次,i = 2
temp = 1; result[1] = 1, top = 0
i= 2;  StackIndex[0] = 2  top = 1

循环第3次,i = 3
i= 3;  StackIndex[1] = 3  top = 2
```



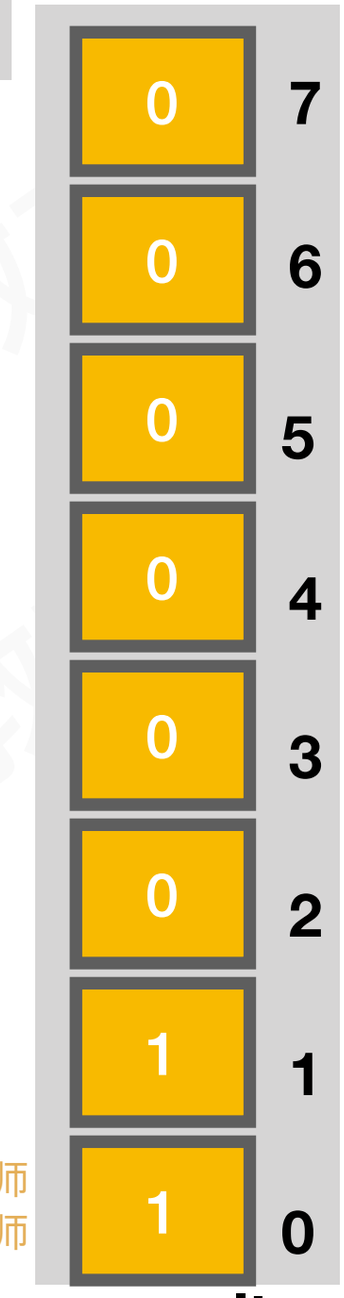
1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈



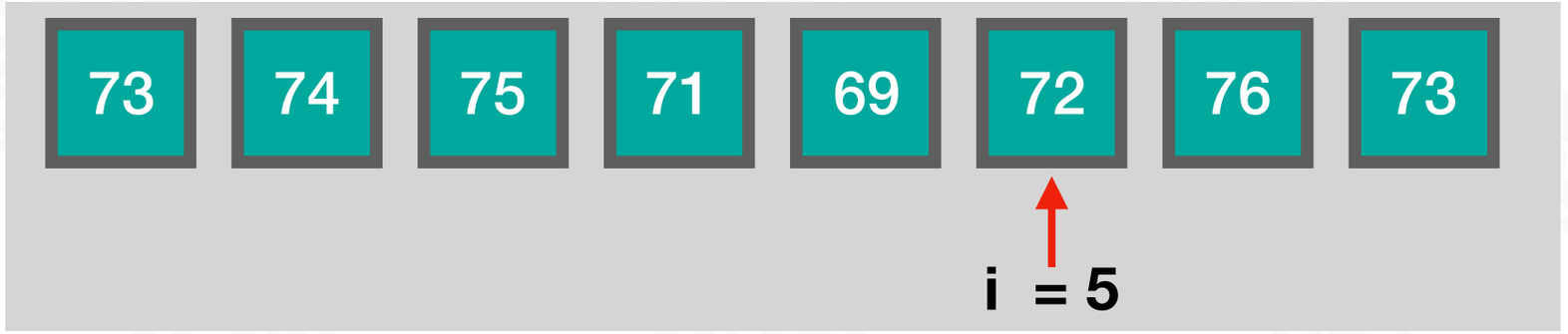


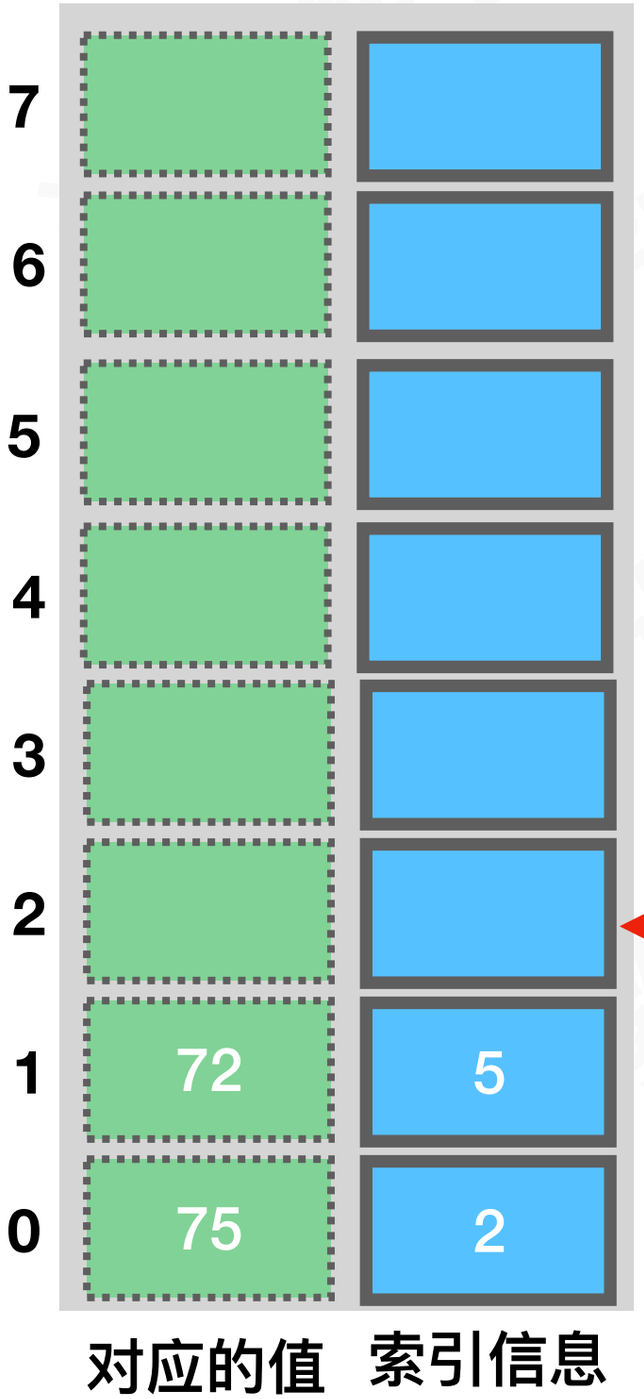

```
循环第0次,i = 0
i= 0;  StackIndex[0] = 0  top = 1

循环第1次,i = 1
temp = 0; result[0] = 1, top = 0
i= 1;  StackIndex[0] = 1  top = 1


循环第2次,i = 2
temp = 1; result[1] = 1, top = 0
i= 2;  StackIndex[0] = 2  top = 1

循环第3次,i = 3
i= 3;  StackIndex[1] = 3  top = 2

循环第4次,i = 4
i= 4;  StackIndex[2] = 4  top = 3
```



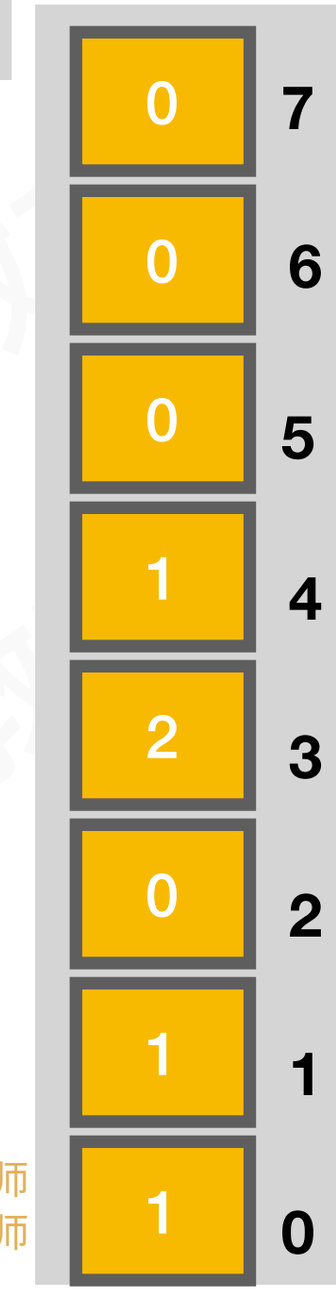
1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈



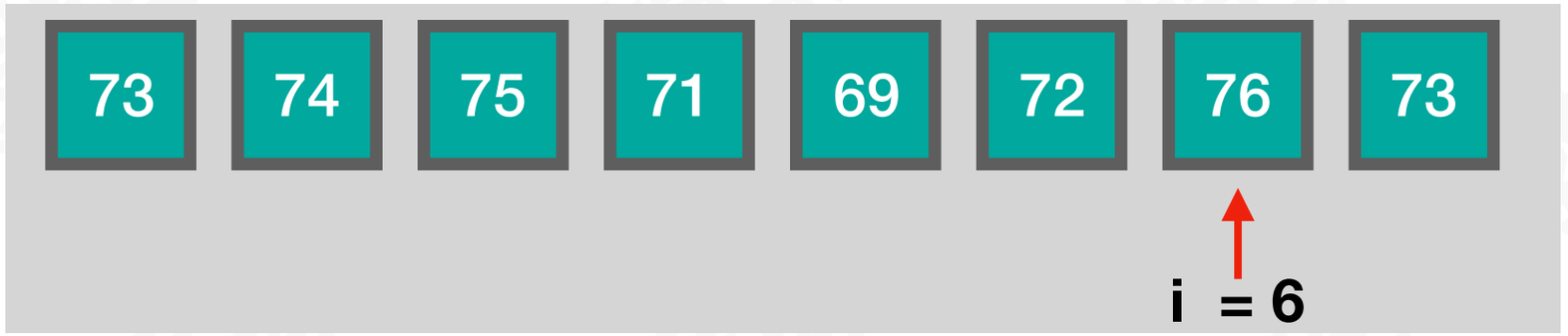


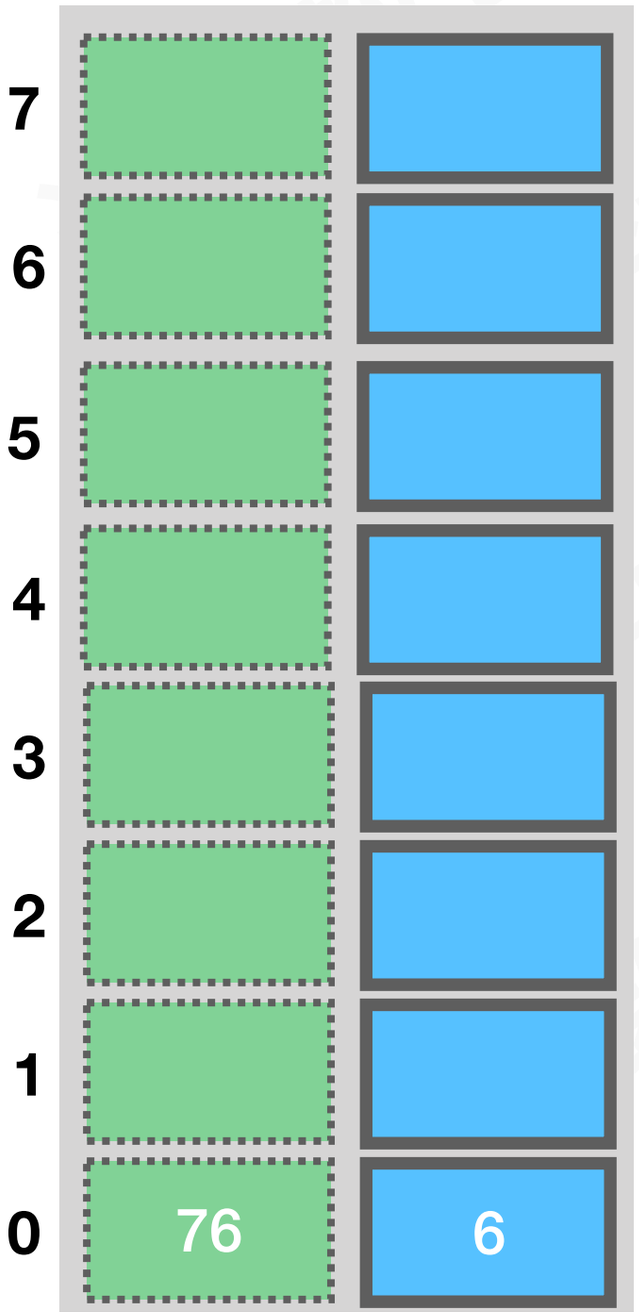

```
循环第0次,i = 0
i= 0;  StackIndex[0] = 0  top = 1

循环第1次,i = 1
temp = 0; result[0] = 1, top = 0
i= 1;  StackIndex[0] = 1  top = 1

循环第2次,i = 2
temp = 1; result[1] = 1, top = 0
i= 2;  StackIndex[0] = 2  top = 1

循环第3次,i = 3
i= 3;  StackIndex[1] = 3  top = 2

循环第4次,i = 4
i= 4;  StackIndex[2] = 4  top = 3

循环第5次,i = 5
temp = 4; result[4] = 1, top = 2
temp = 3; result[3] = 2, top = 1
i= 5;  StackIndex[1] = 5  top = 2
```



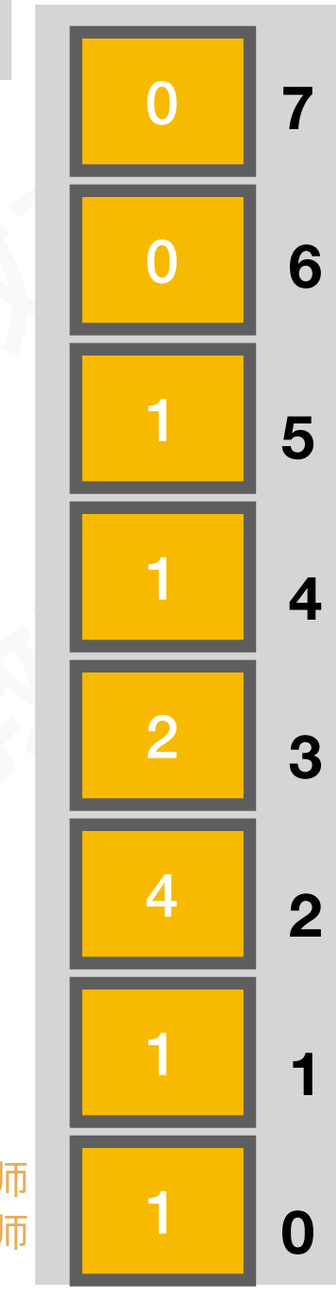
1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈



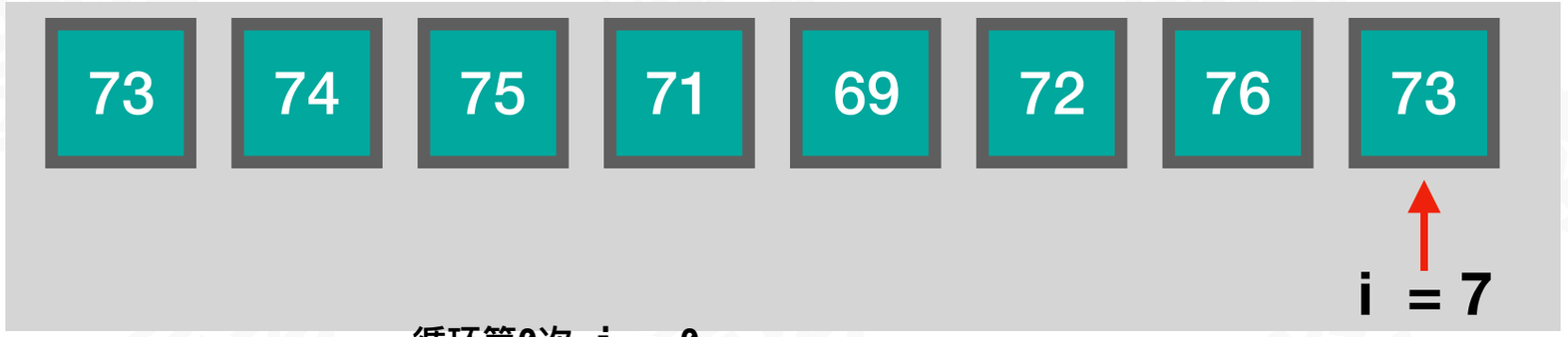


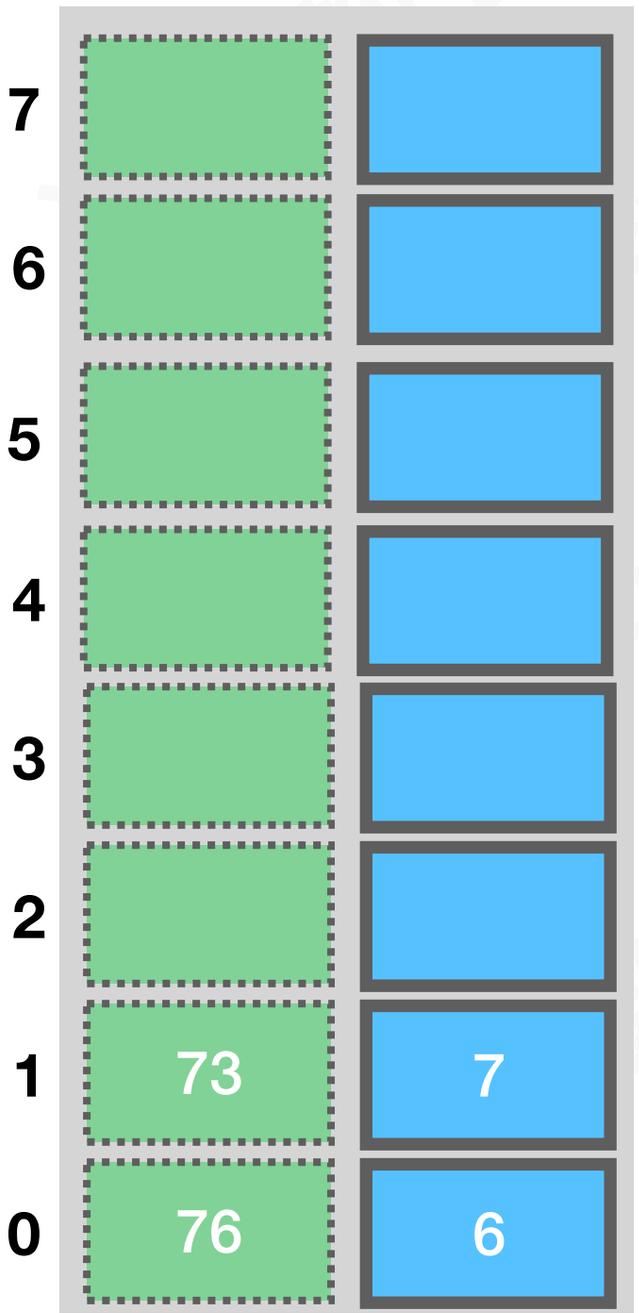

```
循环第0次,i = 0
i= 0;  StackIndex[0] = 0  top = 1

循环第1次,i = 1
temp = 0; result[0] = 1, top = 0
i= 1;  StackIndex[0] = 1  top = 1


循环第2次,i = 2
temp = 1; result[1] = 1, top = 0
i= 2;  StackIndex[0] = 2  top = 1

循环第3次,i = 3
i= 3;  StackIndex[1] = 3  top = 2

循环第4次,i = 4
i= 4;  StackIndex[2] = 4  top = 3

循环第5次,i = 5
temp = 4; result[4] = 1, top = 2
temp = 3; result[3] = 2, top = 1
i= 5;  StackIndex[1] = 5  top = 2

循环第6次,i = 6
temp = 5; result[5] = 1, top = 1
temp = 2; result[2] = 4, top = 0
i= 6;  StackIndex[0] = 6  top = 1
```



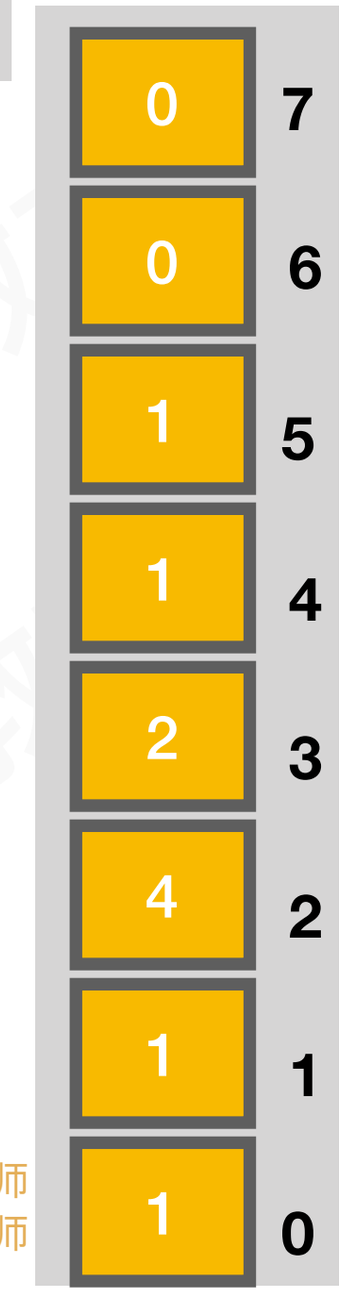
1. 栈中存储的是元素的索引值index;
2.将当前元素和栈顶元素比较;
如果栈为空,那么直接将当前元素索引index 存储到栈中;
如果栈顶元素>当前元素,则将当前元素索引index 存储到栈中;
如果栈顶元素<当前元素,则将当前元素索引index-栈顶元素index,计算完毕则将当前栈顶元素移除,将当前元素索引index 存储到栈中

题目解析: 栈






```
循环第0次,i = 0
i= 0;  StackIndex[0] = 0  top = 1

循环第1次,i = 1
temp = 0; result[0] = 1, top = 0
i= 1;  StackIndex[0] = 1  top = 1

循环第2次,i = 2
temp = 1; result[1] = 1, top = 0
i= 2;  StackIndex[0] = 2  top = 1

循环第3次,i = 3
i= 3;  StackIndex[1] = 3  top = 2

循环第4次,i = 4
i= 4;  StackIndex[2] = 4  top = 3

循环第5次,i = 5
temp = 4; result[4] = 1, top = 2
temp = 3; result[3] = 2, top = 1
i= 5;  StackIndex[1] = 5  top = 2

循环第6次,i = 6
temp = 5; result[5] = 1, top = 1
temp = 2; result[2] = 4, top = 0
i= 6;  StackIndex[0] = 6  top = 1

循环第7次,i = 7
i= 7;  StackIndex[1] = 7  top = 2
```



逻辑教育
Logic education

算法题目(LeetCode 经典题):

	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

课程研发:CC老师
课程授课:CC老师

算法题目(LeetCode 中等难度):

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢? 注意：给定 n 是一个正整数

示例1:

输入: 2
输初: 2
解释: 有2种方法可以爬到楼顶
1. 1阶+1阶
2. 2阶

示例2:

输入: 3
输初: 3
解释: 有3种方法可以爬到楼顶
1. 1阶+1阶+1阶
2. 1阶+2阶
3. 2阶+1阶

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

算法技巧—动态规划

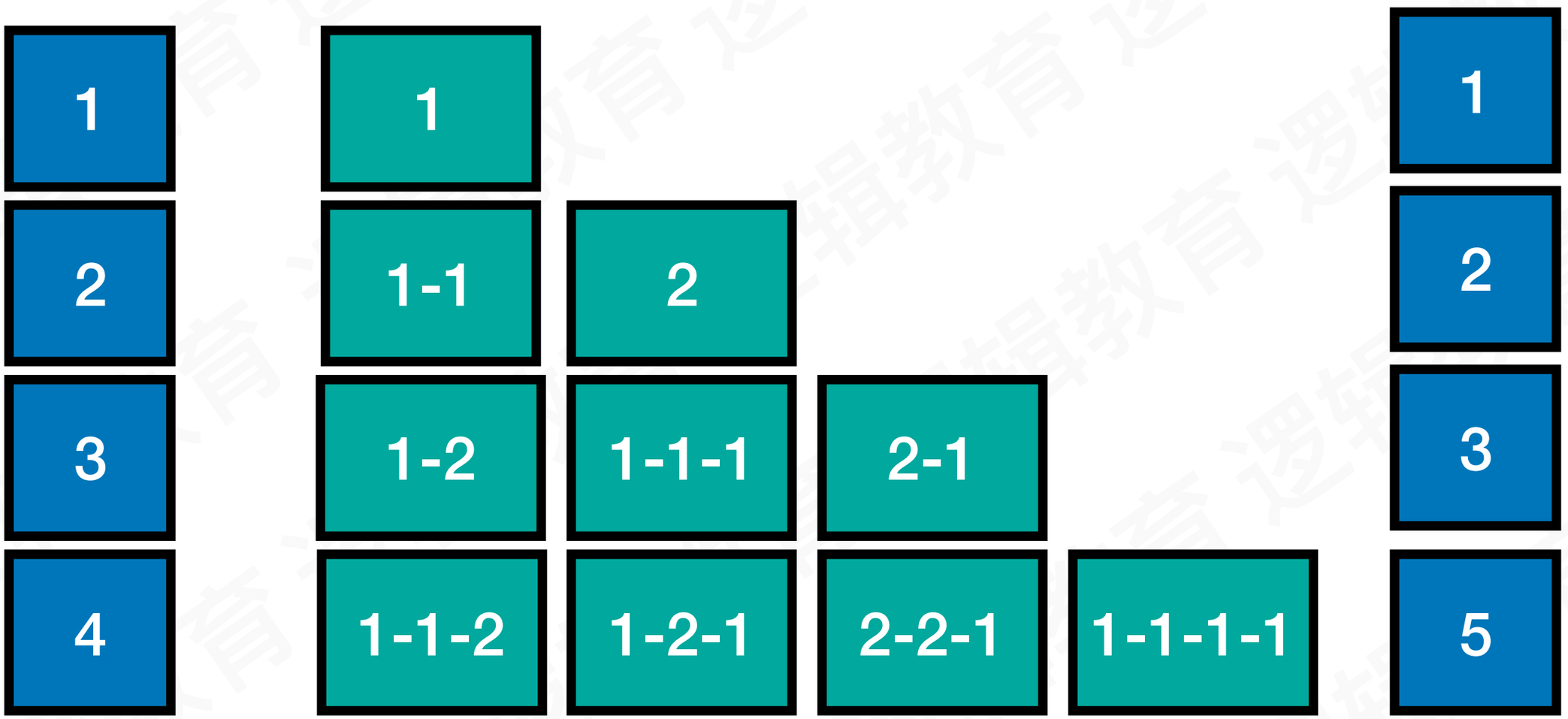
动态规划（英语：Dynamic programming，简称 DP）是一种在数学、管理科学、计算机科学、经济学和生物信息学中使用的，通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法。

动态规划常常适用于有重叠子问题和最优子结构性质的问题，动态规划方法所耗时间往往远少于朴素解法。

动态规划背后的基本思想非常简单。大致上，若要解一个给定问题，我们需要解其不同部分（即子问题），再根据子问题的解以得出原问题的解。动态规划往往用于优化递归问题，例如斐波那契数列，如果运用递归的方式来求解会重复计算很多相同的子问题，利用动态规划的思想可以减少计算量。

通常许多子问题非常相似，为此动态规划法试图仅仅解决每个子问题一次，具有天然剪枝的功能，从而减少计算量：一旦某个给定子问题的解已经算出，则将其记忆化存储，以便下次需要同一个子问题解之时直接查表。这种做法在重复子问题的数目关于输入的规模呈指数增长时特别有用。

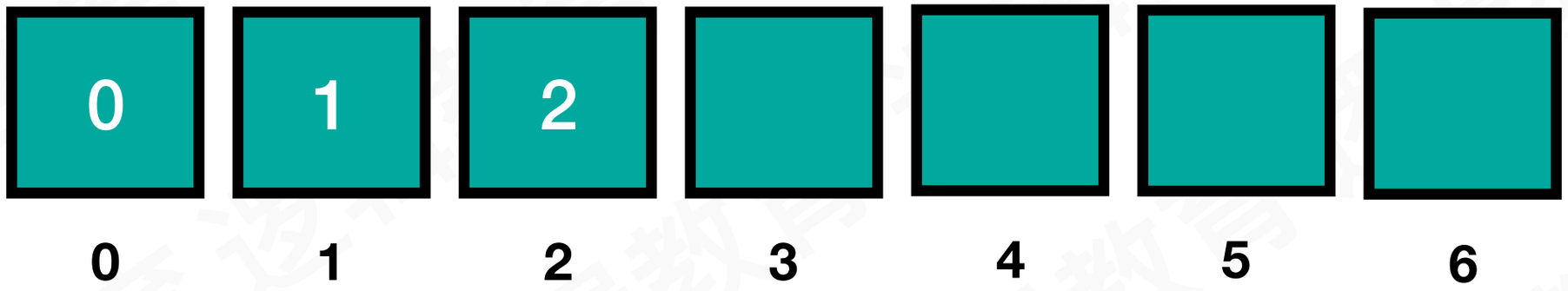
课程研发:CC老师
课程授课:CC老师



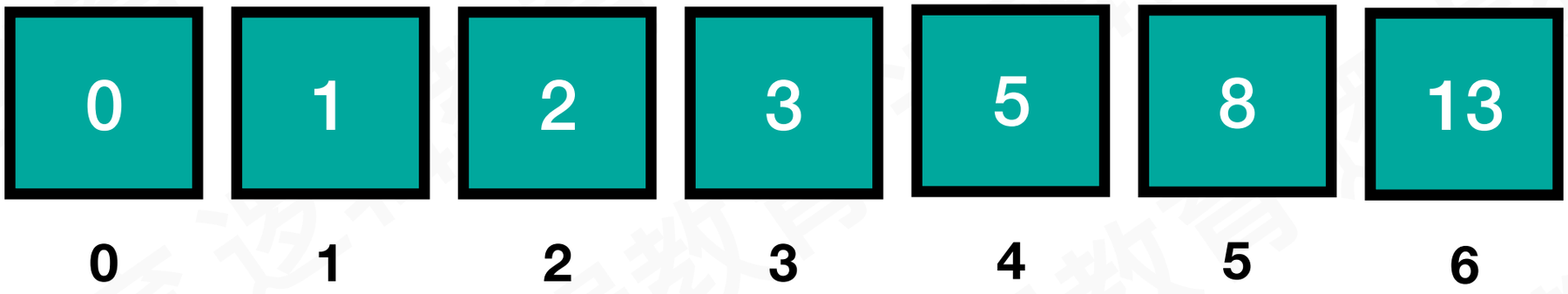
这个问题可以被分解为一些包含最优子结构的子问题，即它的最优解可以从其子问题的最优解来有效地构建，我们可以使用动态规划来解决这一问题。

课程研发:CC老师
课程授课:CC老师

假设爬 n 个台阶有f(n)个可能:
1. 假设先爬1阶, 剩下n-1阶有f(n-1)种可能
2.假设先爬2阶,剩下n-2阶有f(n-2)种可能
因此爬n阶可以转化成2种爬n-1问题的和. f(n) = f(n-1) + f(n-2)



假设爬 n 个台阶有f(n)个可能:
1. 假设先爬1阶, 剩下n-1阶有f(n-1)种可能
2.假设先爬2阶,剩下n-2阶有f(n-2)种可能
因此爬n阶可以转化成2种爬n-1问题的和. f(n) = f(n-1) + f(n-2)



逻辑教育
Logic education

394- 字符串编码 (LeetCode - 中等)

给定一个经过编码的字符串，返回它解码后的字符串。
编码规则为: k[encoded_string]，表示其中方括号内部的 encoded_string 正好重复 k 次。注意 k 保证为正整数。你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。此外，你可以认为原始数据不包含数字，所有的数字只表示重复的次数 k ，例如不会出现像 3a 或 2[4] 的输入。
例如:
s = "3[a]2[bc]", 返回 "aaabcbc".
s = "3[a2[c]]", 返回 "accaccacc".
s = "2[abc]3[cd]ef", 返回 "abcabccdcdcdef".

课程研发:CC老师
课程授课:CC老师

394- 字符串编码 (LeetCode - 中等)

s = "12[a]", 返回 "aaaaaaaaaaaa".

下标	0	1	2	3
S =	3	[	a	]

栈: 用来记录数字

下标	0	1	2	3	4	5	6	7	8	9	10
strOfInt											

栈: 用来处理编码的过程

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Stack																

top = -1

栈: 用来记录需要生成多份的字母

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
temp																

topTemp = -1

逻辑教育
Logic education

394- 字符串编码 (LeetCode - 中等)

执行结果： 通过 显示详情 >

执行用时： 0 ms ，在所有 C 提交中击败了 100.00% 的用户

内存消耗： 5.7 MB ，在所有 C 提交中击败了 100.00% 的用户

炫耀一下：

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

316. 去除重复字母(LeetCode - 困难)

给你一个仅包含小写字母的字符串，请你去除字符串中重复的字母，使得每个字母只出现一次。
需保证返回结果的字典序最小（要求不能打乱其他字符的相对位置）

例1:
输入: “bcabc"
输出: “abc"

例2:
输入: "cbacdcbc"
输出: "acdb"

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

316. 去除重复字母(LeetCode - 困难)

执行结果： 通过 显示详情 >

执行用时： 4 ms ，在所有 C 提交中击败了 68.22% 的用户

内存消耗： 5.6 MB ，在所有 C 提交中击败了 100.00% 的用户

炫耀一下：

课程研发:CC老师
课程授课:CC老师