力扣 LeetCode × 拉勾 LAGOU.COM

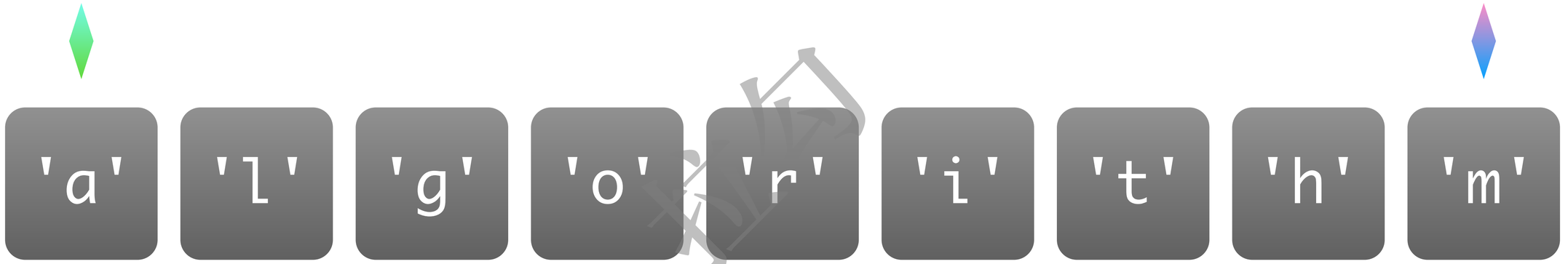
第一章

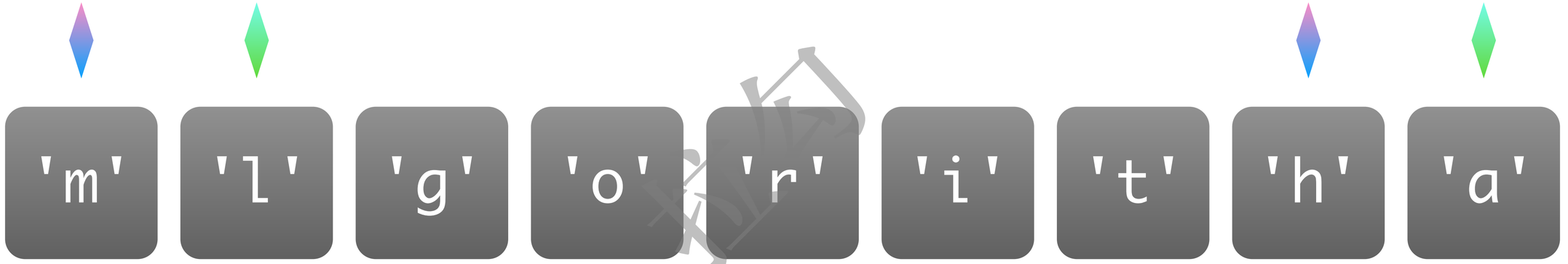
掌握实用的数据结构

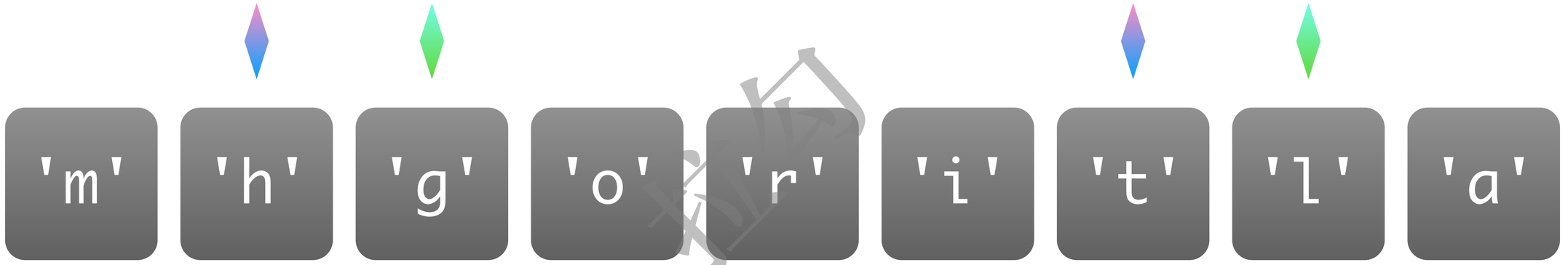
1.1 常用数据结构和技巧 / Common Data Structure

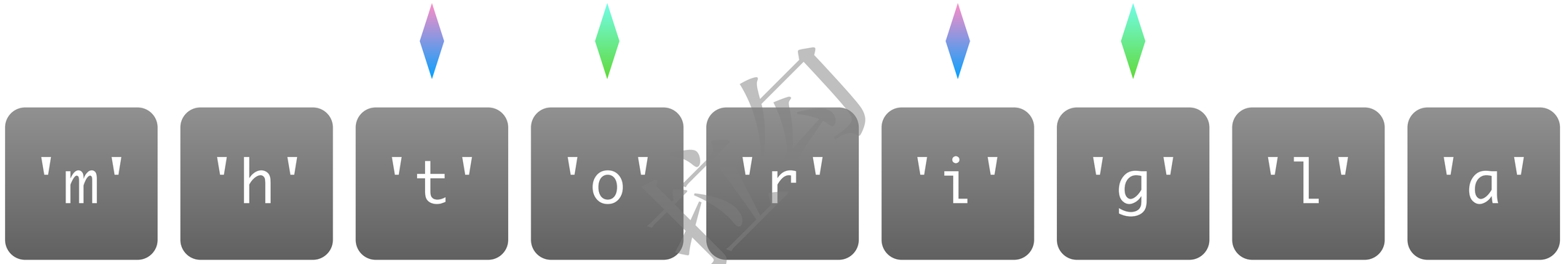
力扣 LeetCode × 拉勾 LAGOU.COM

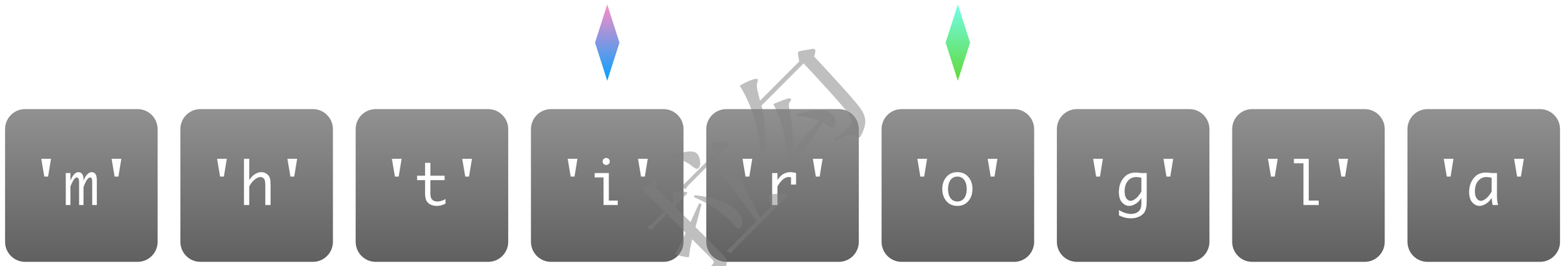
- 数组、字符串 / Array & String
- 链表 / Linked-list
- 栈 / Stack
- 队列 / Queue
- 双端队列 / Deque
- 树 / Tree


'a'
'l'
'g'
'o'
'r'
'i'
't'
'h'
'm'




'm'
'h'
'g'
'o'
'r'
'i'
't'
'l'
'a'




'm'
'h'
't'
'i'
'r'
'o'
'g'
'l'
'a'

优点

构建一个数组非常简单

能让我们在 O(1) 的时间里根据数组的下标（index）查询某个元素

缺点

构建时必须分配一段连续的空间

查询某个元素是否存在时需要遍历整个数组，耗费 O(n) 的时间（其中，n 是元素的个数）

删除和添加某个元素时，同样需要耗费 O(n) 的时间

力扣 LeetCode × 拉勾 LAGOU.COM

242. 有效的字母异位词

给定两个字符串 s 和 t，编写一个函数来判断 t 是否是 s 的字母异位词。

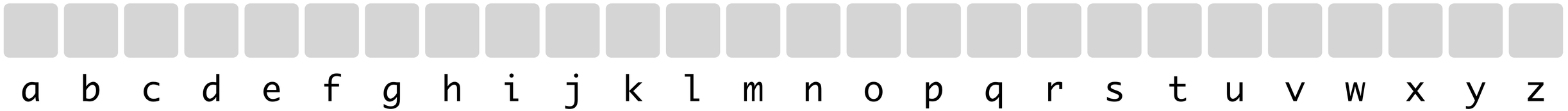
说明：
你可以假设字符串只包含小写字母。

示例 1

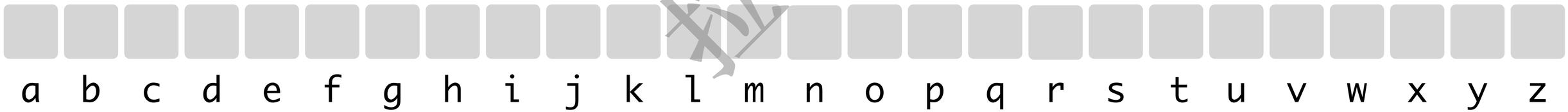
```
输入: s = "anagram", t = "nagaram"
输出: true
```

示例 2

```
输入: s = "rat", t = "car"
输出: false
```

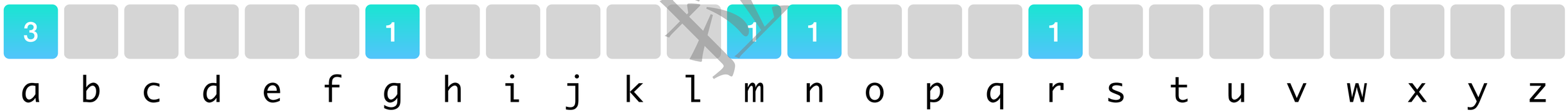


242. 有效的字母异位词



s = "anagram"　　t = "nagaram"

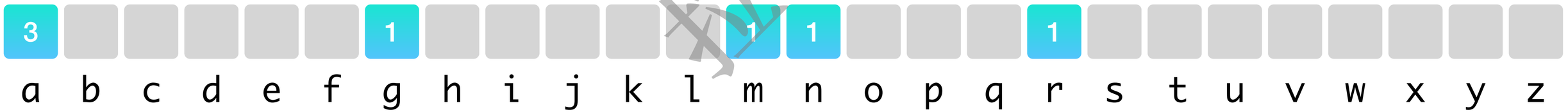
242. 有效的字母异位词



s = "anagram"

t = "nagaram"

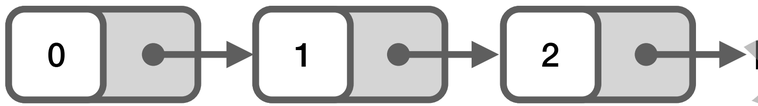
242. 有效的字母异位词



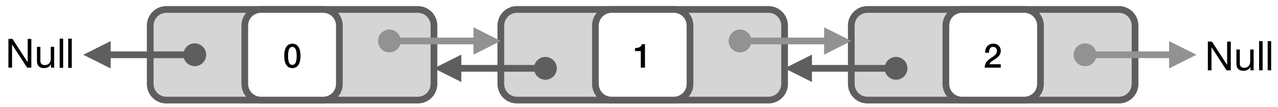
s = "anagram"

t = "nagaram"

单链表： 链表中的每个元素实际上是一个单独的对象，而所有对象都通过每个元素中的引用字段链接在一起。



双链表： 与单链表不同的是，双链表的每个结点中都含有 **两个引用字段**。



1.3 链表 / Linked List

力扣 LeetCode × 拉勾 LAGOU.COM

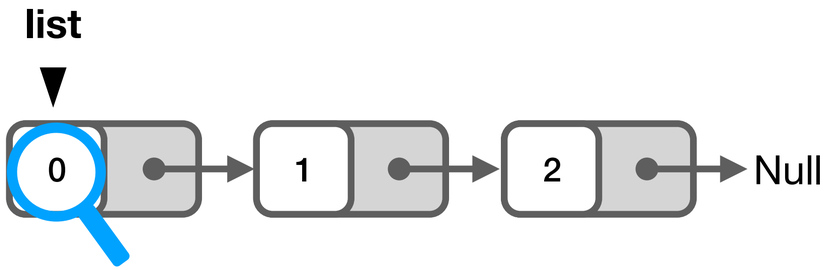
优点

灵活地分配内存空间

能在 O(1) 时间内删除或者添加元素

缺点

查询元素需要 O(n) 时间



1.3 链表 / Linked List

力扣 LeetCode × 拉勾 LAGOU.COM

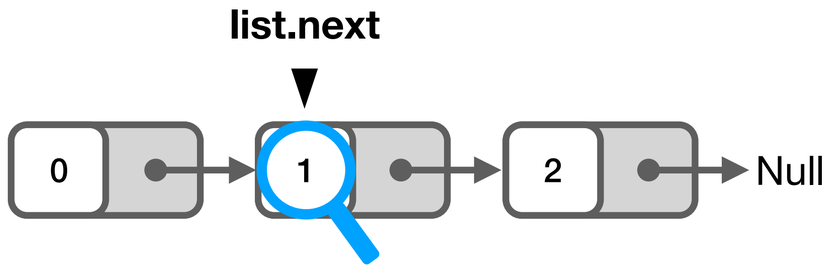
优点

灵活地分配内存空间

能在 O(1) 时间内删除或者添加元素

缺点

查询元素需要 O(n) 时间



1.3 链表 / Linked List

力扣 LeetCode × 拉勾 LAGOU.COM

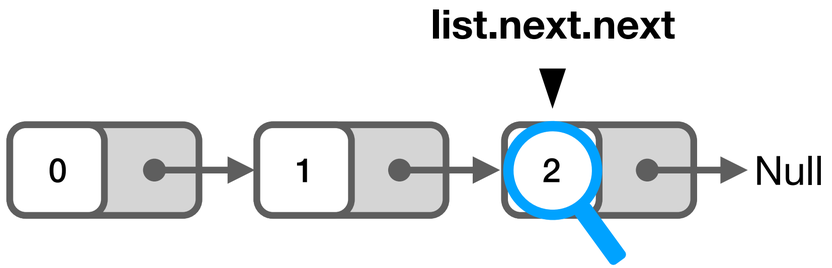
优点

灵活地分配内存空间

能在 O(1) 时间内删除或者添加元素

缺点

查询元素需要 O(n) 时间

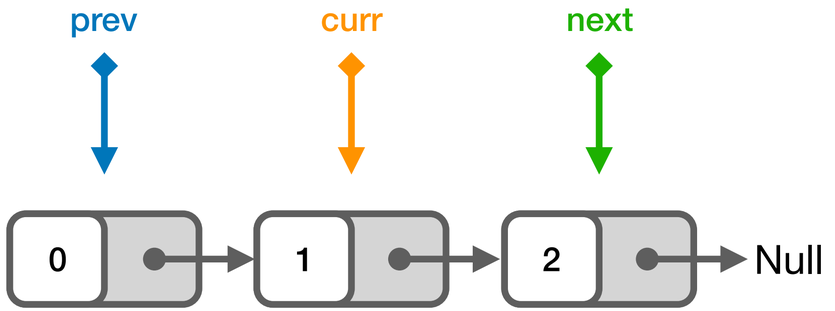


力扣 LeetCode × 拉勾 LAGOU.COM

解题技巧

利用快慢指针（有时候需要用到三个指针）

构建一个虚假的链表头



例如

两个排序链表，进行整合排序

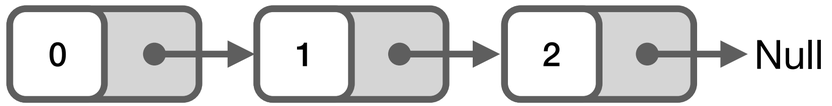
将链表的奇偶数按原定顺序分离，生成前半部分为奇数，后半部分为偶数的链表

力扣 LeetCode × 拉勾 LAGOU.COM

解题技巧

利用快慢指针（有时候需要用到三个指针）

构建一个虚假的链表头



如何训练该技巧

在纸上或者白板上画出节点之间的相互关系

画出修改的方法

25. K 个一组翻转链表

给你一个链表，每 k 个节点一组进行翻转，请你返回翻转后的链表。

k 是一个正整数，它的值小于或等于链表的长度。
如果节点总数不是 k 的整数倍，那么请将最后剩余的节点保持原有顺序。

说明：
你的算法只能使用常数的额外空间。
你不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。

示例：
给定这个链表：1->2->3->4->5
当 k = 2 时，应当返回：2->1->4->3->5
当 k = 3 时，应当返回：3->2->1->4->5

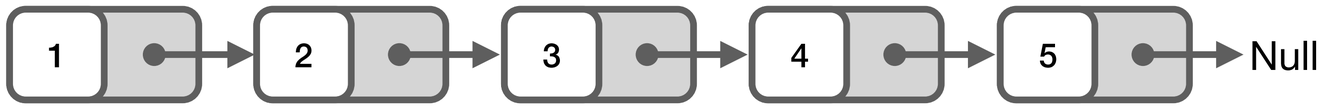
力扣 LeetCode × 拉勾 LAGOU.COM

25. K 个一组翻转链表

给你一个链表，每 k 个节点一组进行翻转，请你返回翻转后的链表。

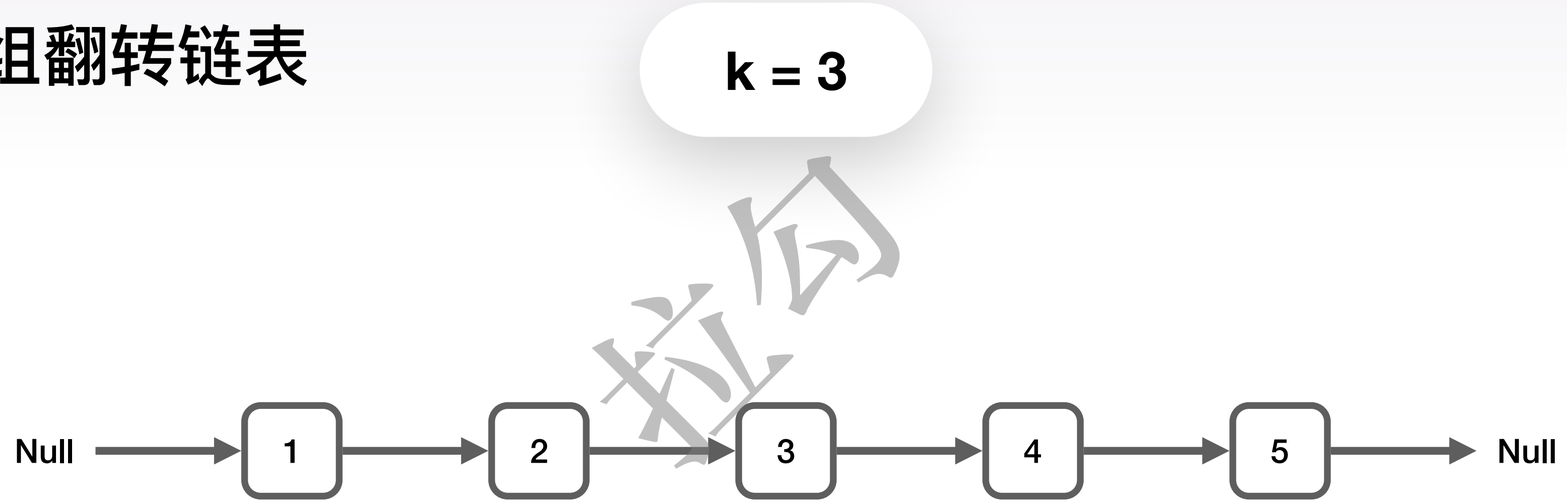
k 是一个正整数，它的值小于或等于链表的长度。
如果节点总数不是 k 的整数倍，那么请将最后剩余的节点保持原有顺序。

示例：
给定这个链表：1->2->3->4->5
当 k = 2 时，应当返回：2->1->4->3->5
当 k = 3 时，应当返回：3->2->1->4->5

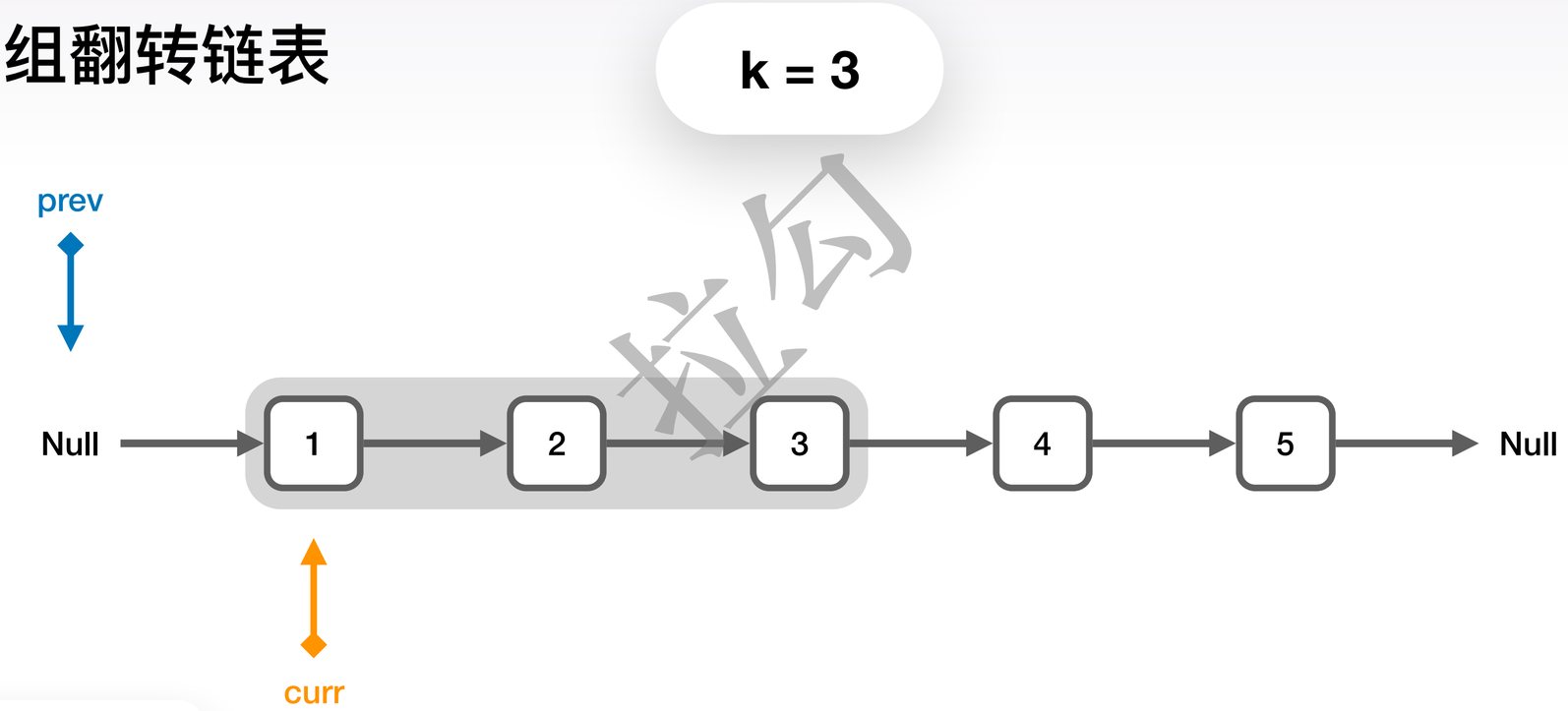


力扣 LeetCode × 拉勾 LAGOU.COM

25. K 个一组翻转链表



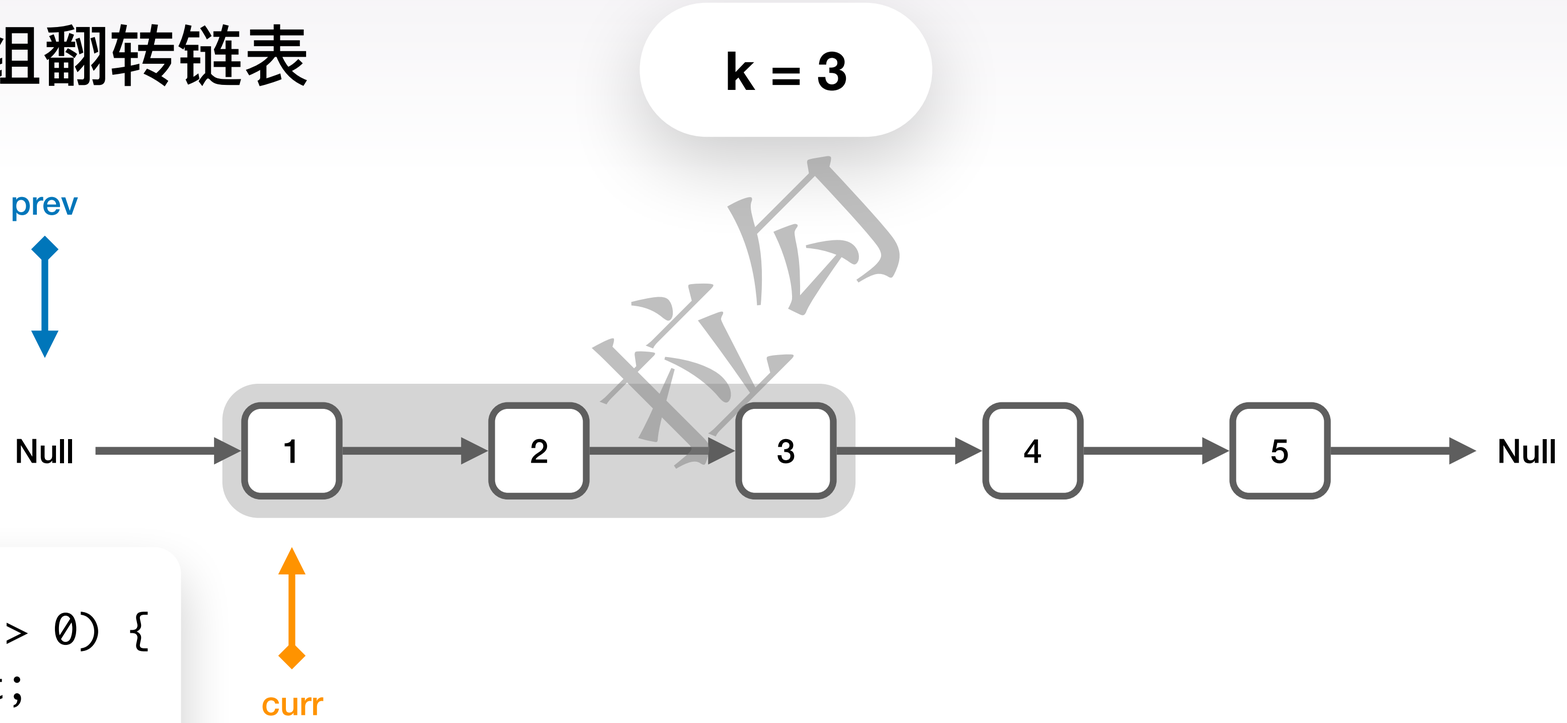
25. K 个一组翻转链表




```
prev = null;
curr = head;
n = k;
```

力扣 LeetCode × 拉勾 LAGOU.COM

25. K 个一组翻转链表

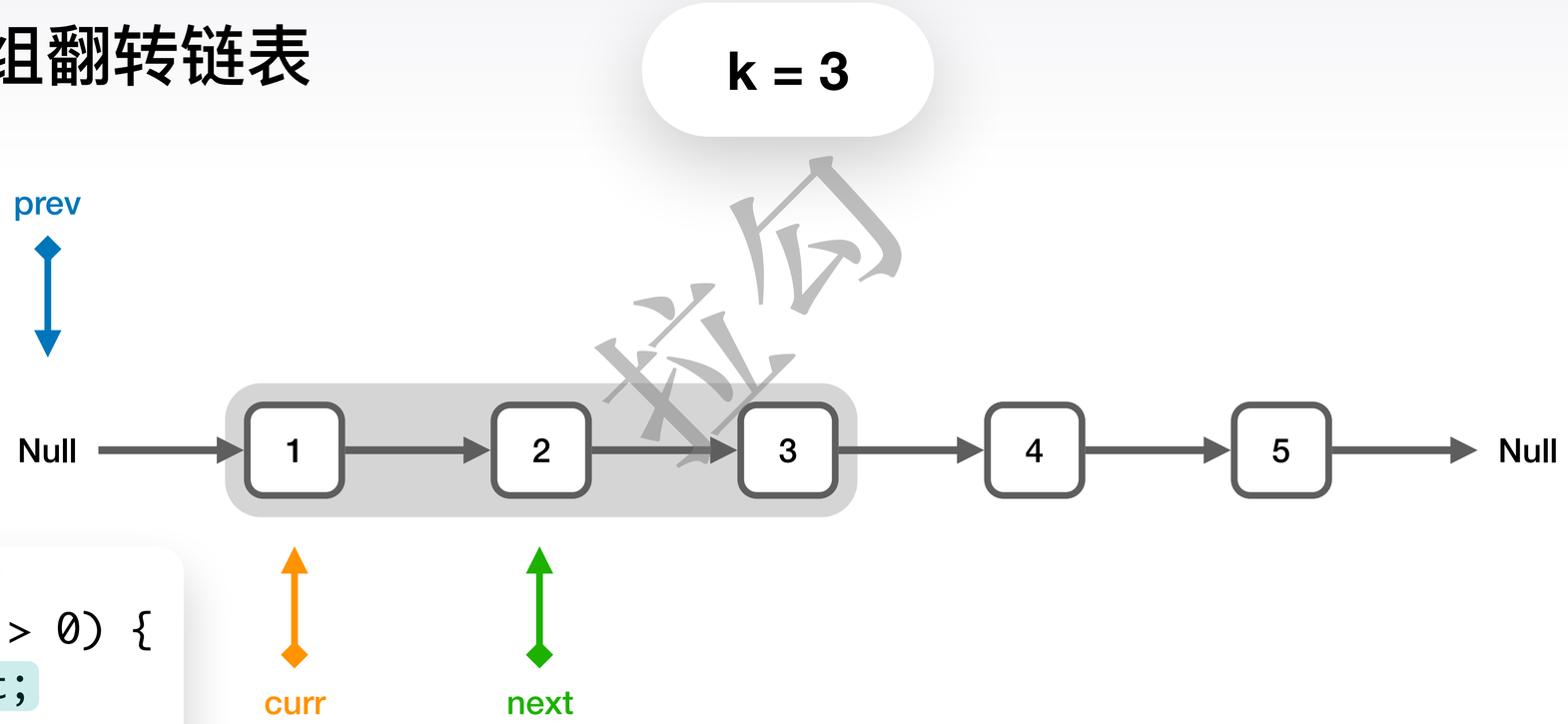



```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

3

力扣 LeetCode × 拉勾 LAGOU.COM

25. K 个一组翻转链表

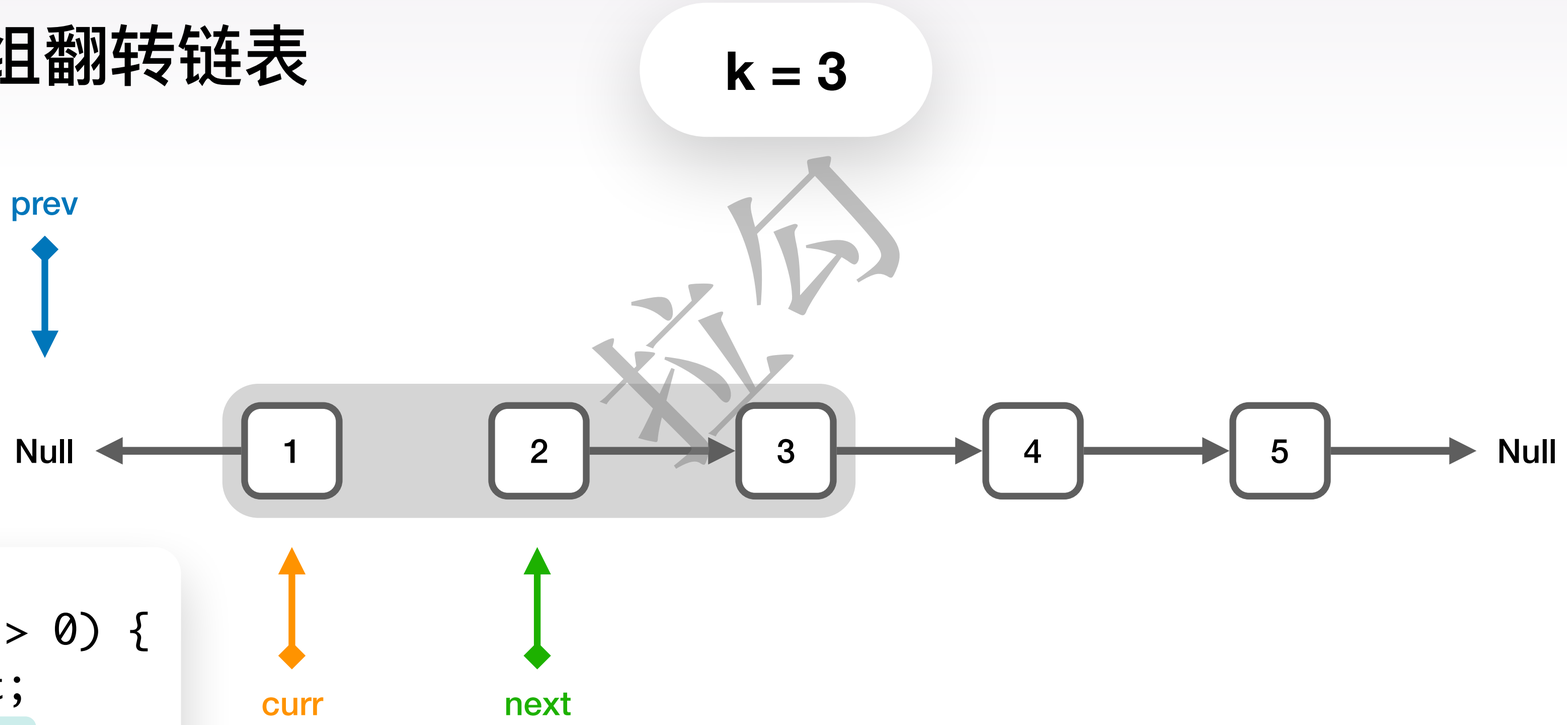



```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

2

力扣 LeetCode × 拉勾 LAGOU.COM

25. K 个一组翻转链表

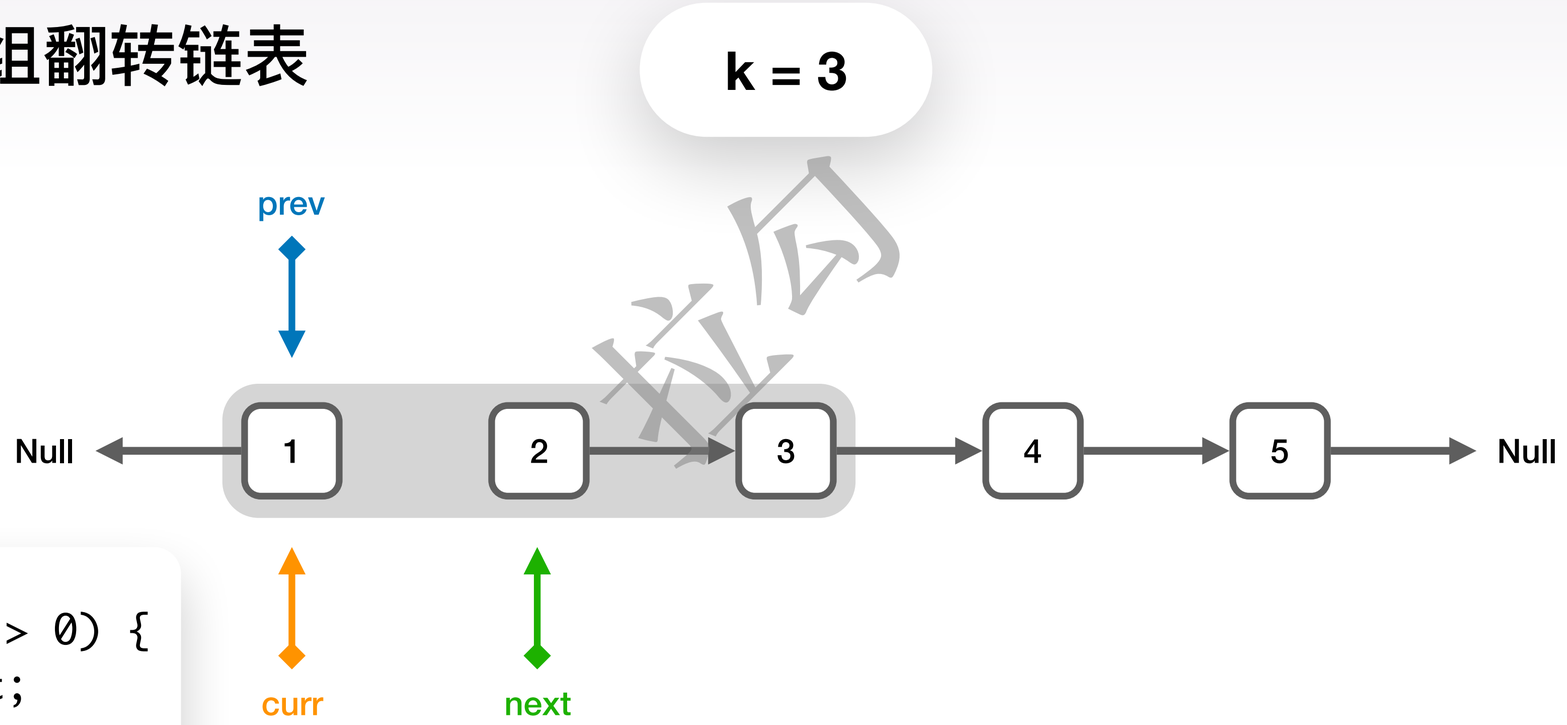


2

```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

力扣 LeetCode × 拉勾 LAGOU.COM

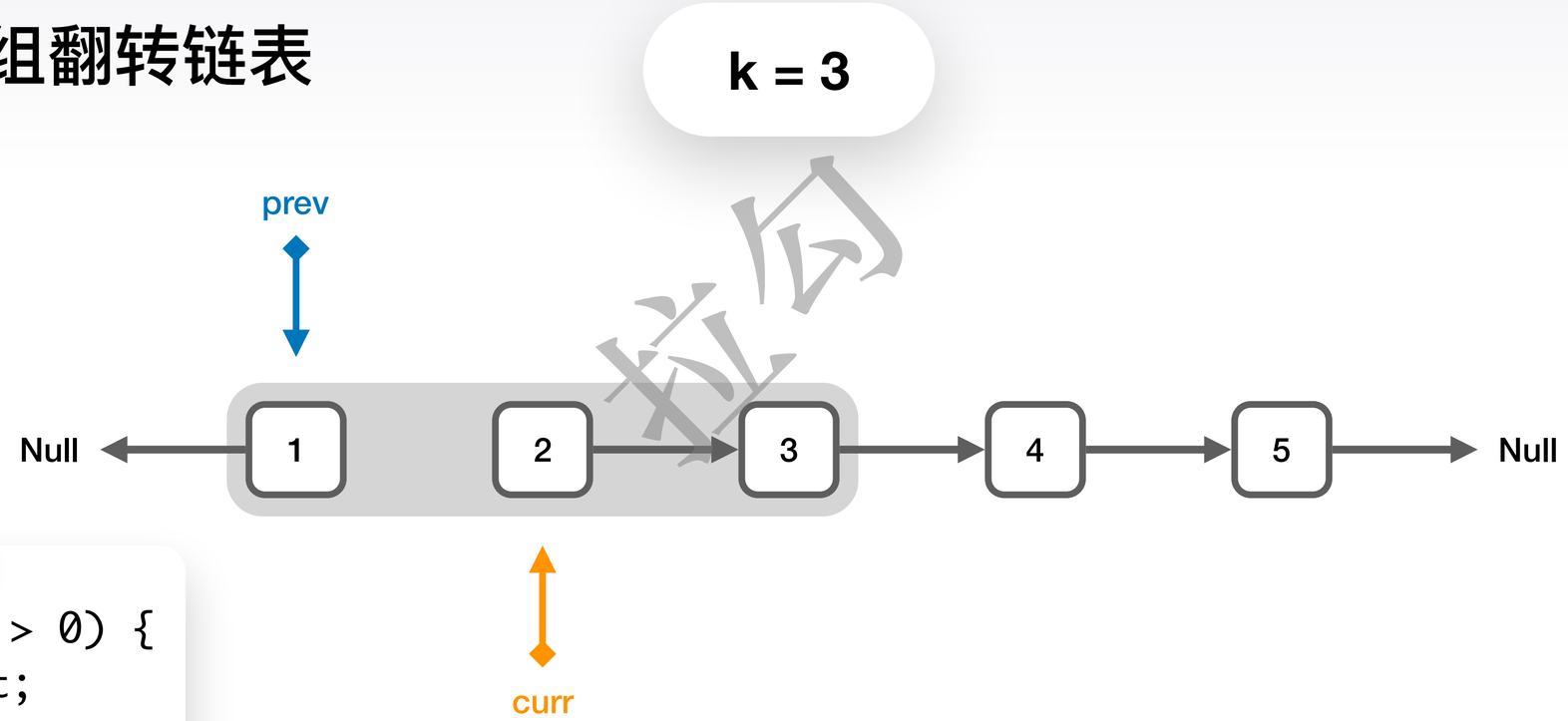
25. K 个一组翻转链表



2

```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

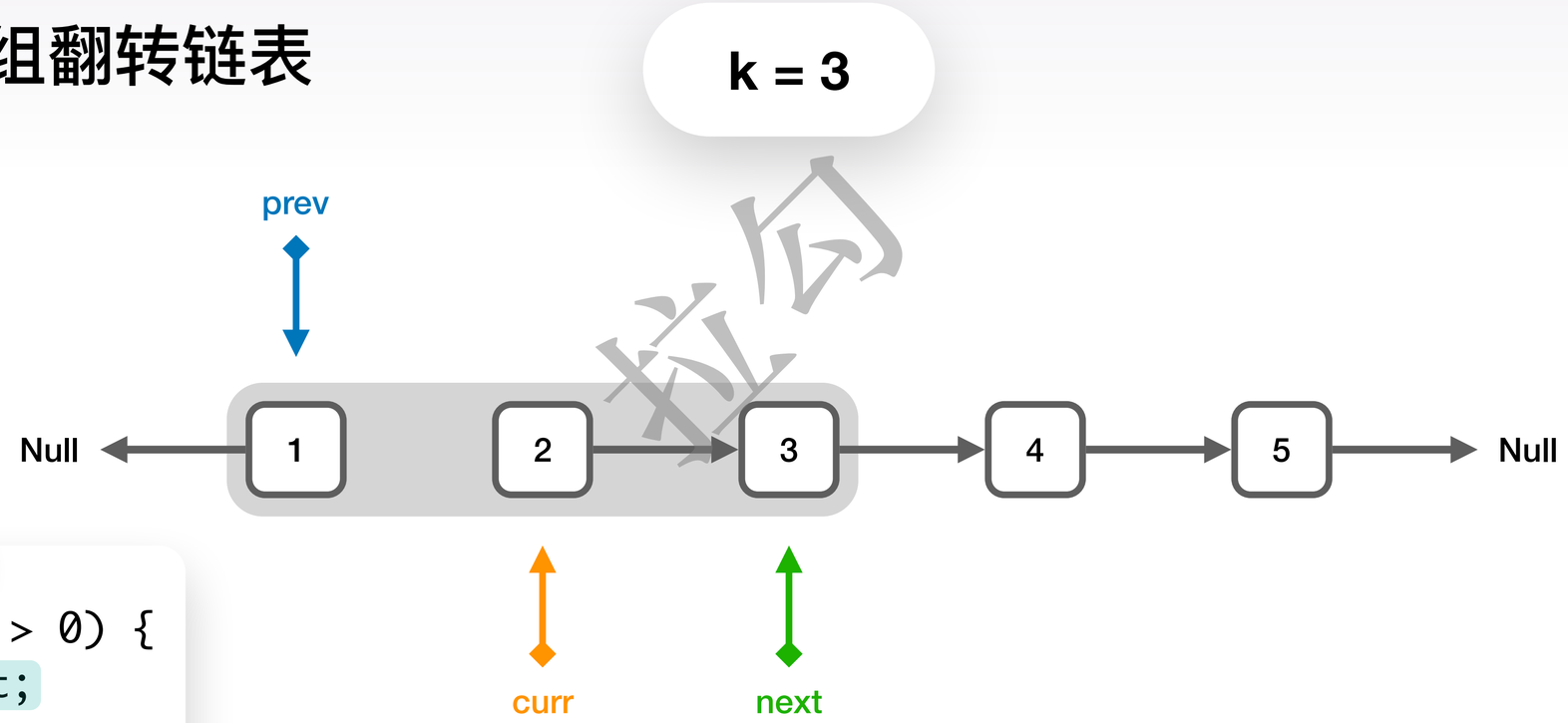
25. K 个一组翻转链表




```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

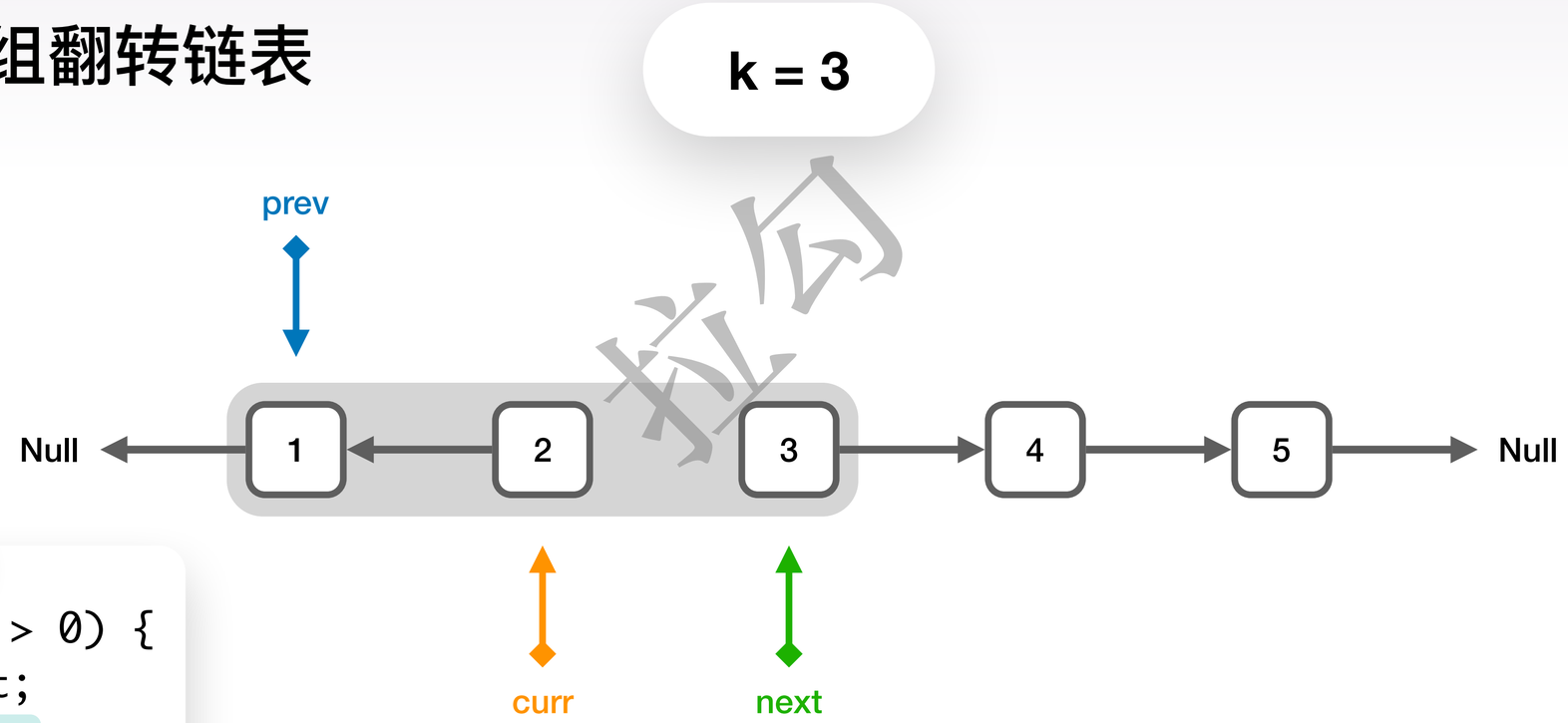
2

25. K 个一组翻转链表




```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

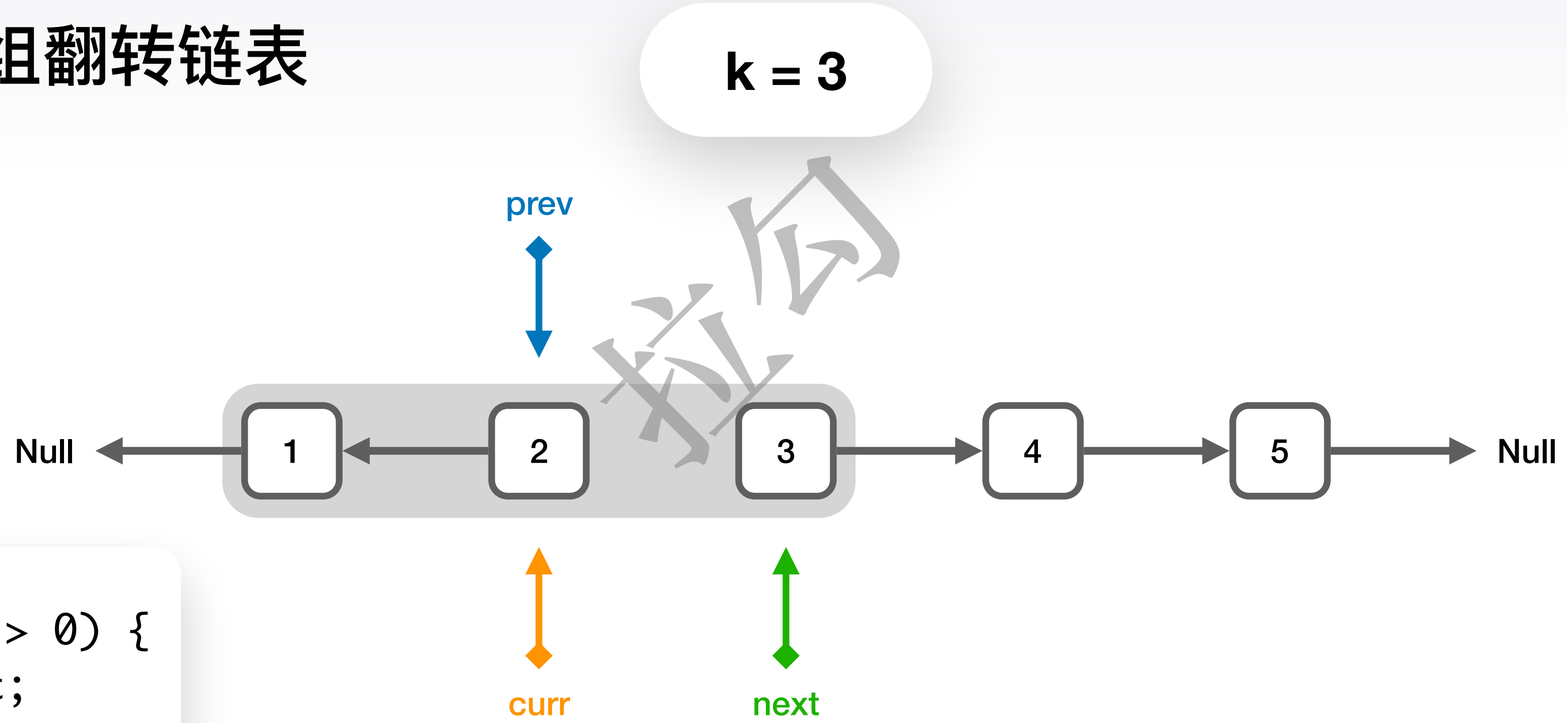
25. K 个一组翻转链表




```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

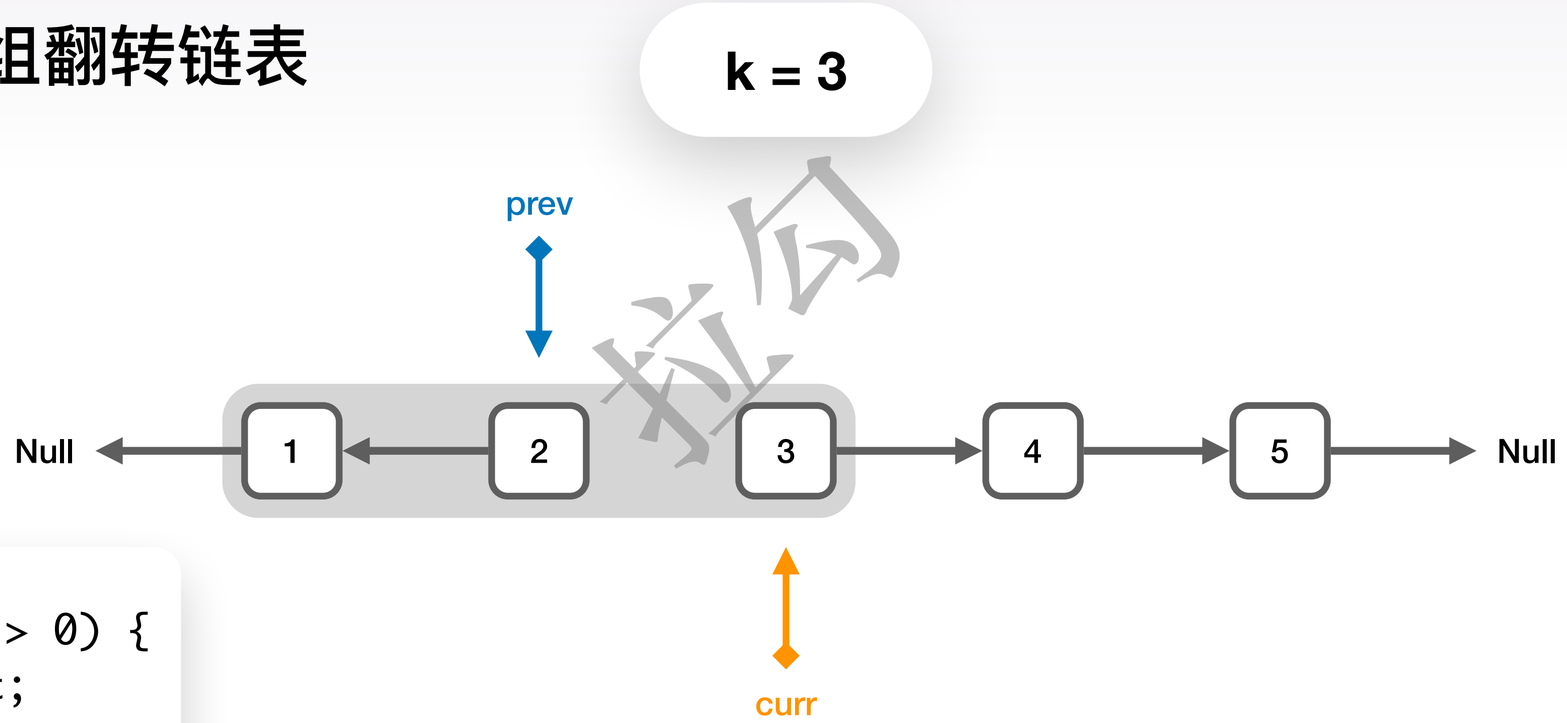
力扣 LeetCode × 拉勾 LAGOU.COM

25. K 个一组翻转链表




```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

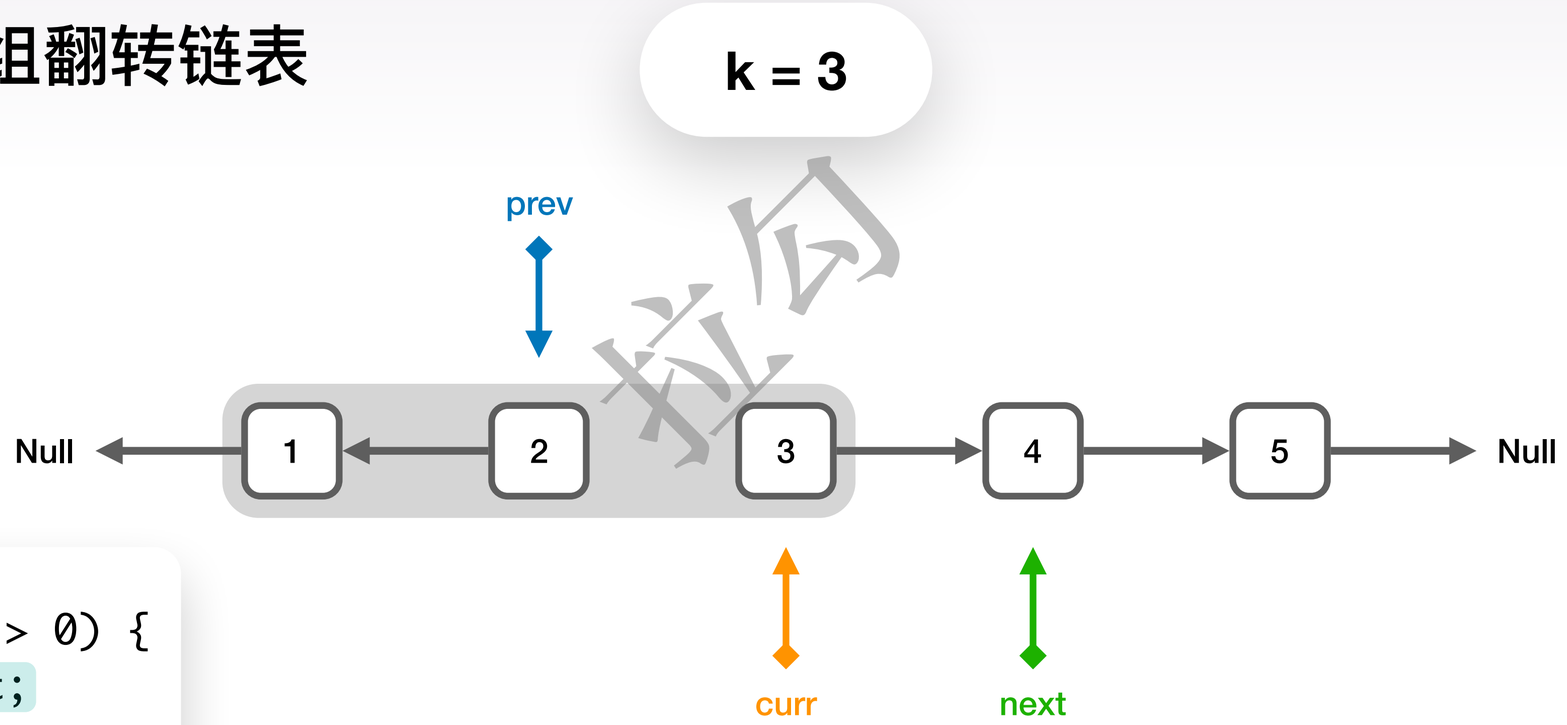
25. K 个一组翻转链表




```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

1

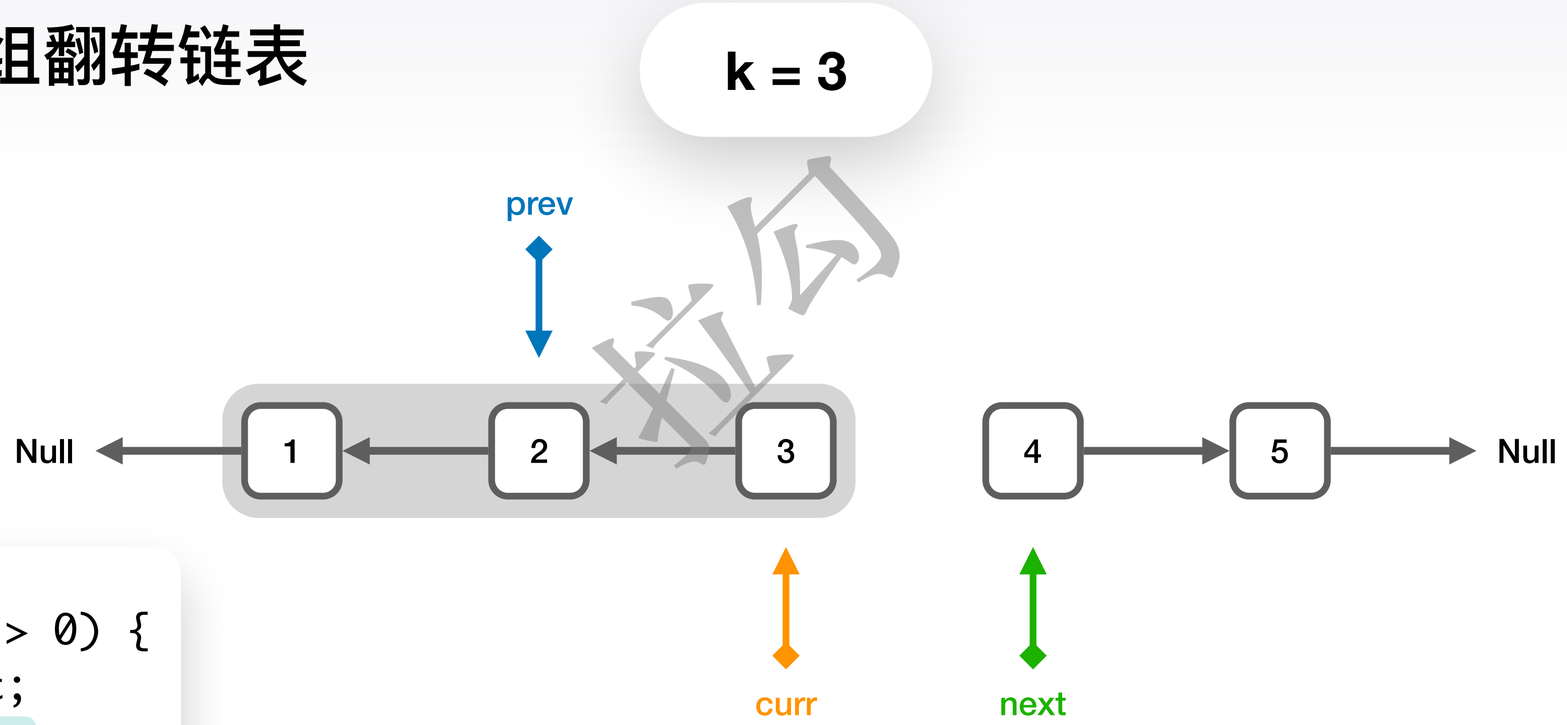
25. K 个一组翻转链表




```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

0

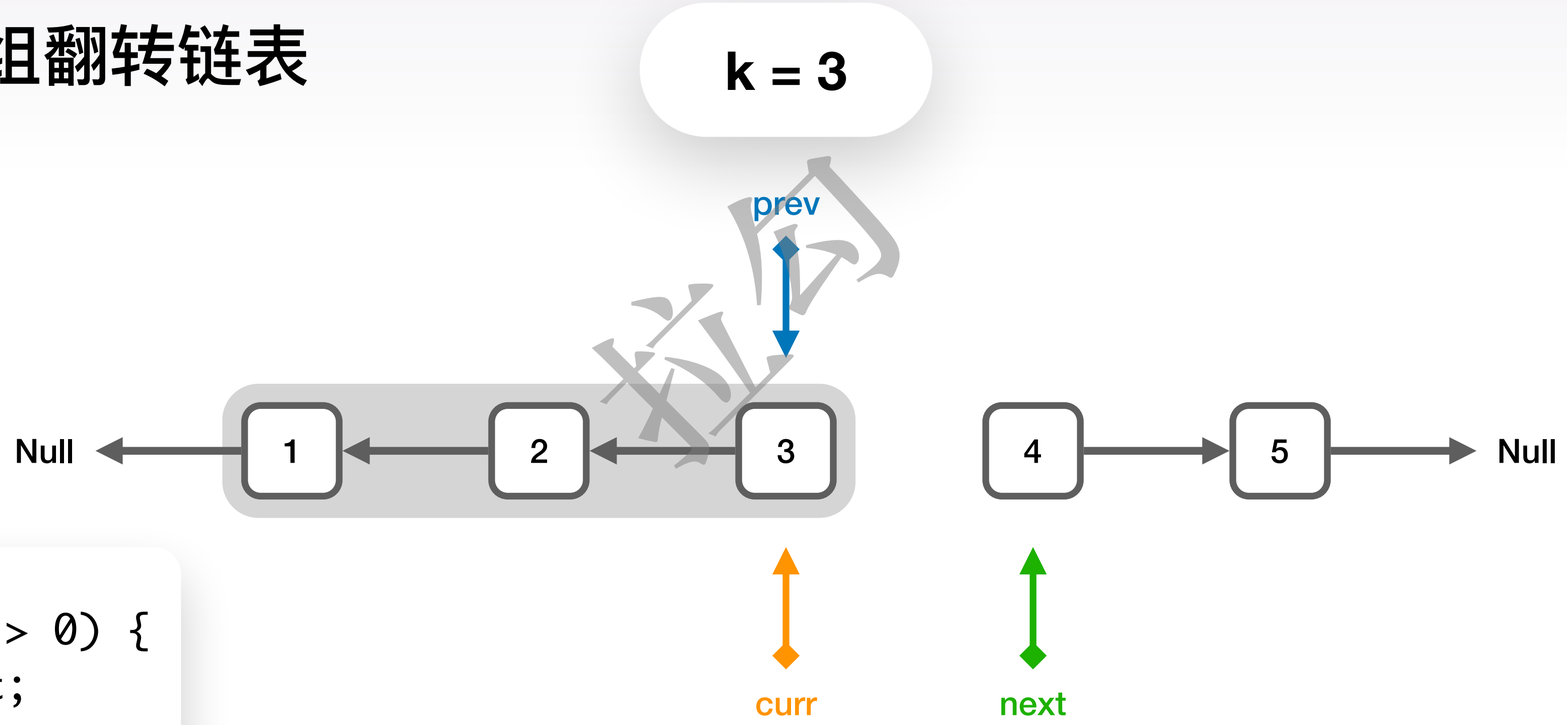
25. K 个一组翻转链表




```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

0

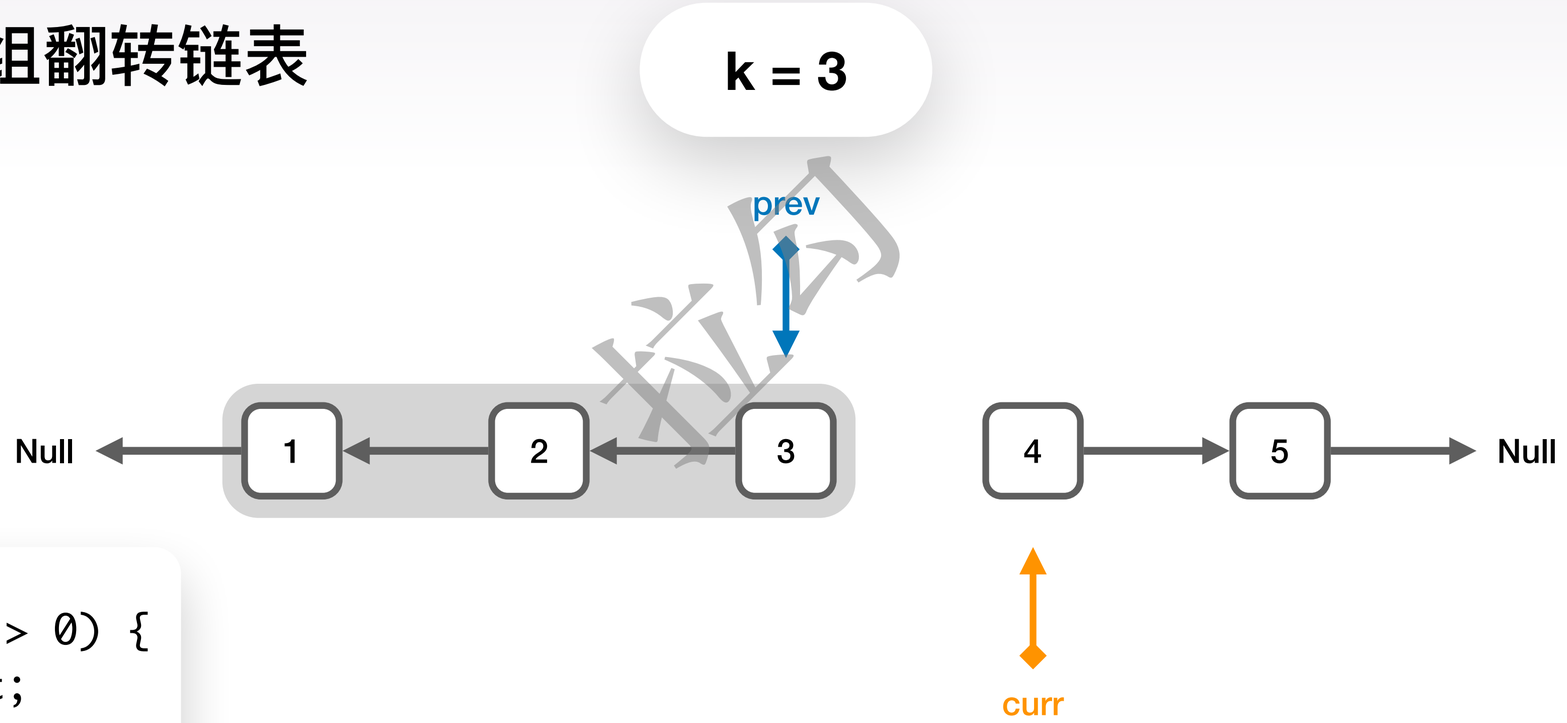
25. K 个一组翻转链表




```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

0

25. K 个一组翻转链表




```
while(curr && n-- > 0) {
  next = curr.next;
  curr.next = prev;
  prev = curr;
  curr = next;
}
```

0

1.4 栈 / Stack

力扣 LeetCode × 拉勾 LAGOU.COM

特点

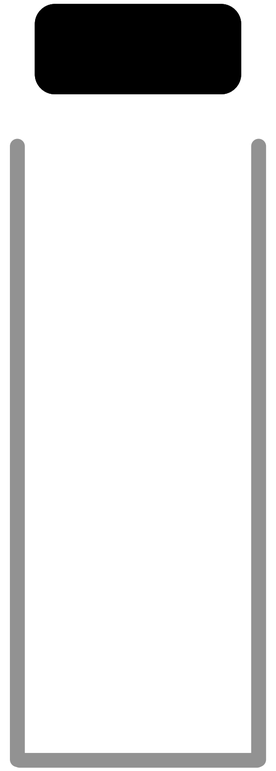
后进先出（LIFO）

算法基本思想

可以用一个单链表来实现

只关心上一次的操作

处理完上一次的操作后，能在 O(1) 时间内查找到更前一次的操作



压弹栈

20. 有效的括号

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串，判断字符串是否有效。

有效字符串需满足：

1. 左括号必须用相同类型的右括号闭合。
2. 左括号必须以正确的顺序闭合。

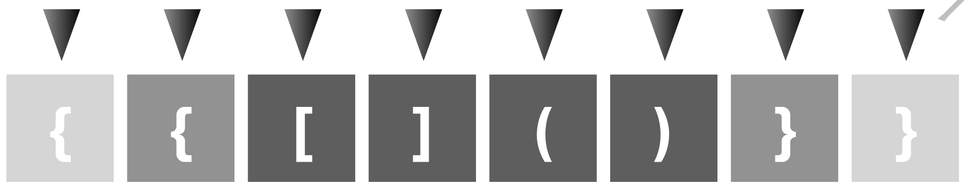
* 注意空字符串可被认为是有效字符串。

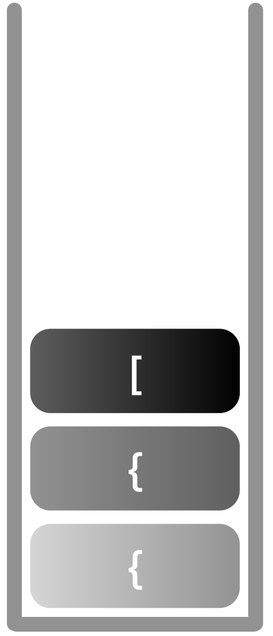
示例 1：
输入： `"()"`
输出： `true`

示例 2：
输入： `"(]"`
输出： `false`

力扣 LeetCode × 拉勾 LAGOU.COM

20. 有效的括号





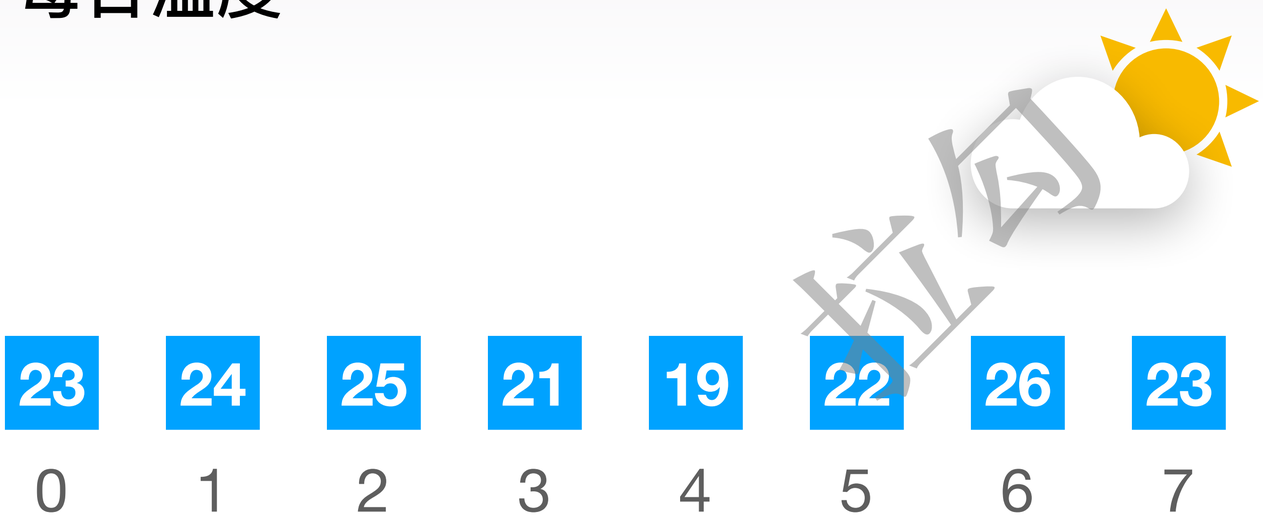
739. 每日温度

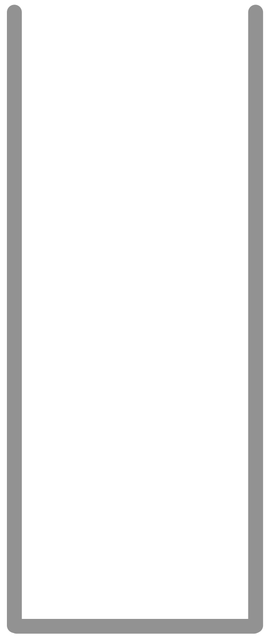
根据每日气温列表，请重新生成一个列表，对应位置的输入是你需要再等待多久温度才会升高超过该日的天数。如果之后都不会升高，请在该位置用 0 来代替。

提示：
气温列表 `temperatures` 长度的范围是 `[1, 30000]`。

示例：
`temperatures = [23, 24, 25, 21, 19, 22, 26, 23]`
输出：`[1, 1, 4, 2, 1, 1, 0, 0]`

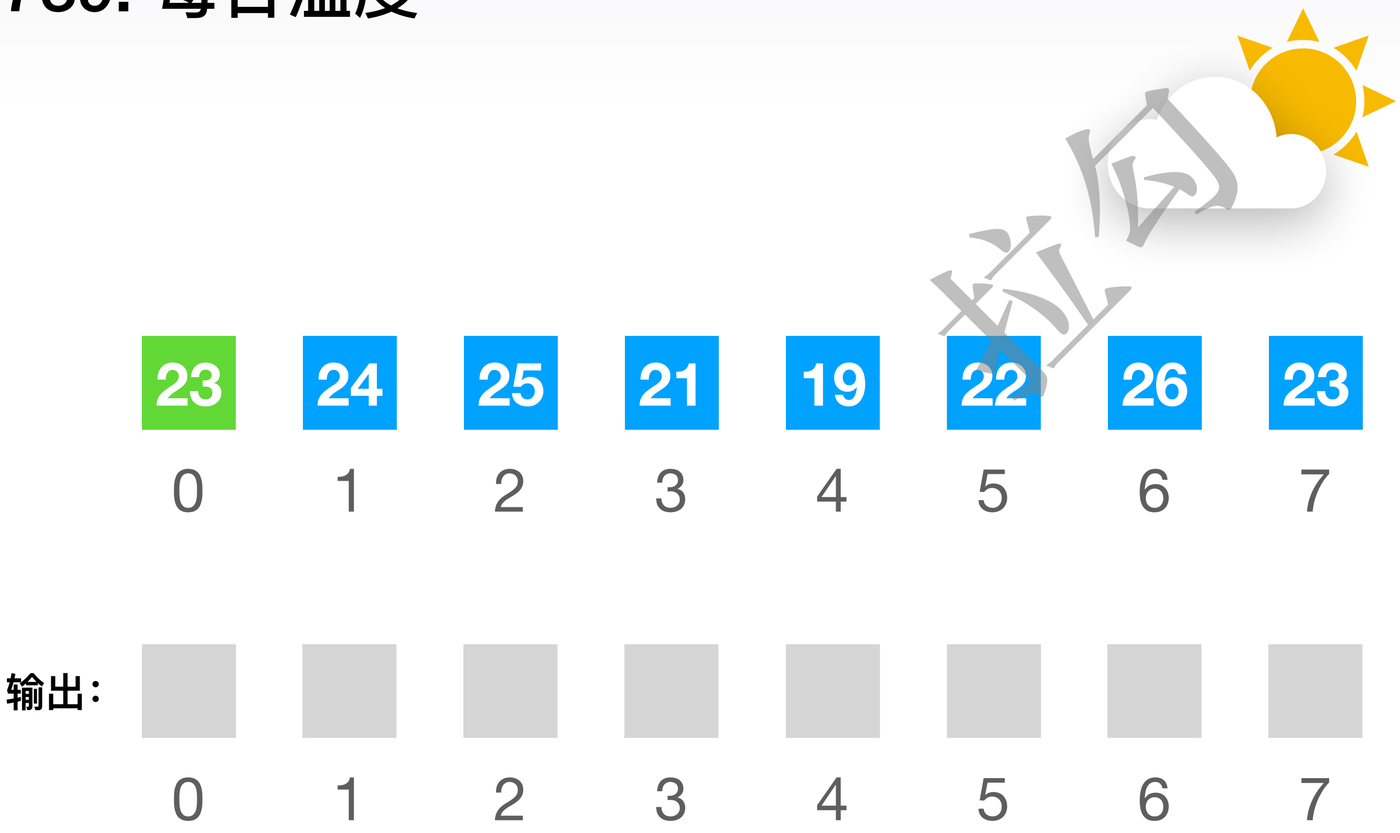
739. 每日温度

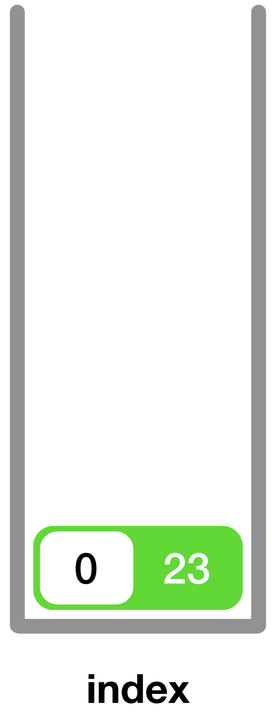




力扣 LeetCode × 拉勾 LAGOU.COM

739. 每日温度





739. 每日温度

739. 每日温度

力扣 LeetCode × 拉勾 LAGOU.COM

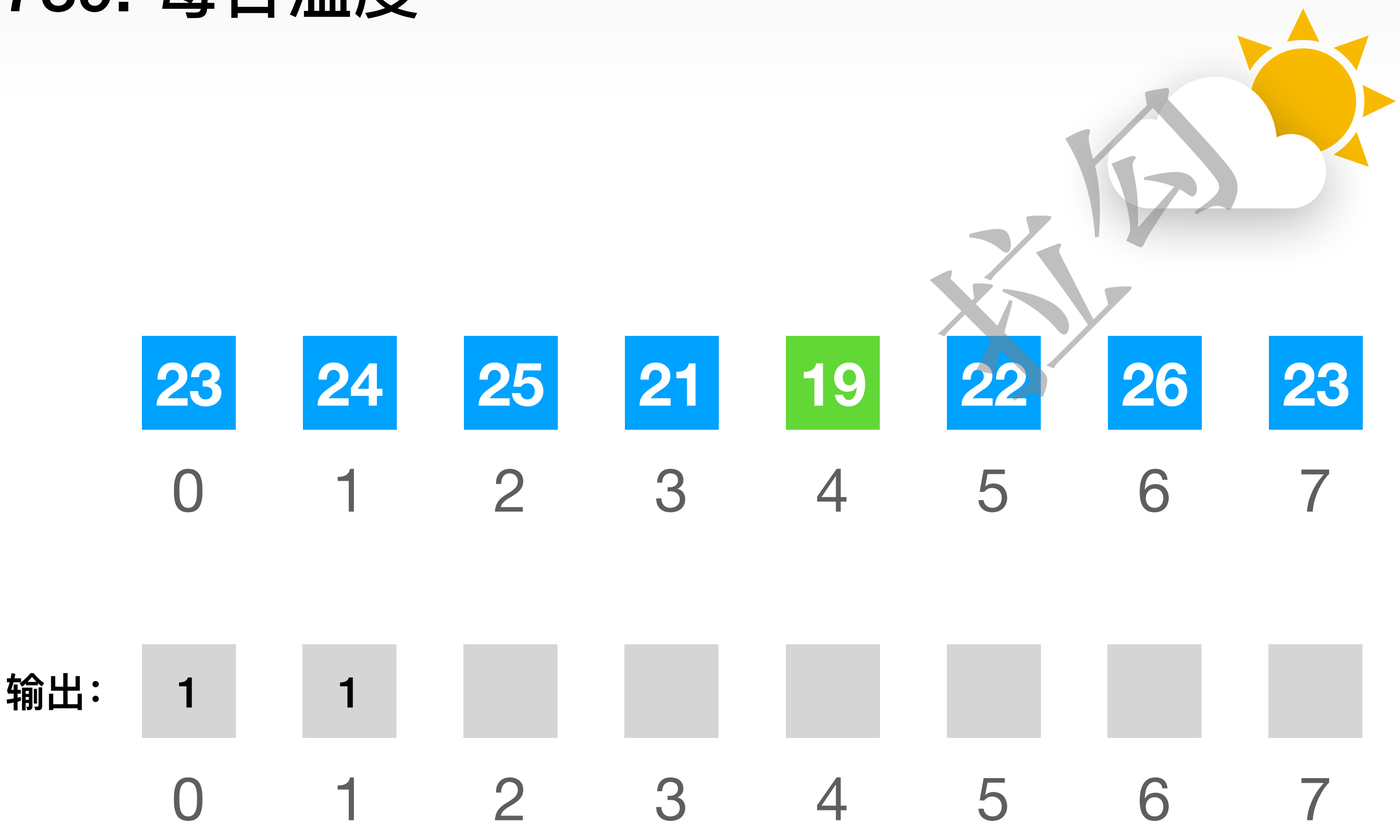
739. 每日温度

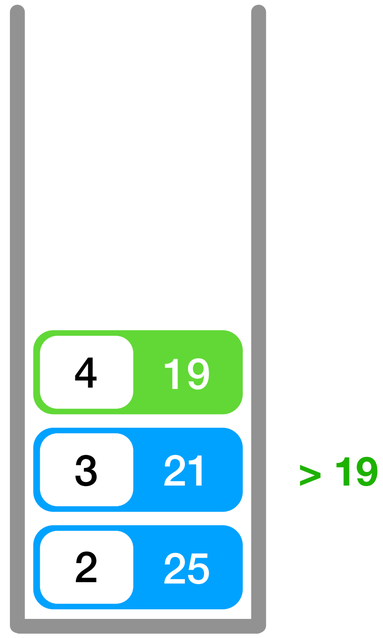
739. 每日温度

739. 每日温度

力扣 LeetCode × 拉勾 LAGOU.COM

739. 每日温度





739. 每日温度


23 24 25 21 19 22 26 23
0 1 2 3 4 5 6 7
输出：
1 1
0 1 2 3 4 5 6 7
4 19
3 21
2 25
< 22
index

力扣 LeetCode × 拉勾 LAGOU.COM

739. 每日温度

739. 每日温度


23 24 25 21 19 22 26 23
0 1 2 3 4 5 6 7
输出：
1 1 1
0 1 2 3 4 5 6 7
3 21
2 25
< 22
index

739. 每日温度


23 24 25 21 19 22 26 23
0 1 2 3 4 5 6 7
输出：
1 1 2 1
0 1 2 3 4 5 6 7
3 21
2 25
< 22
index

力扣 LeetCode × 拉勾 LAGOU.COM

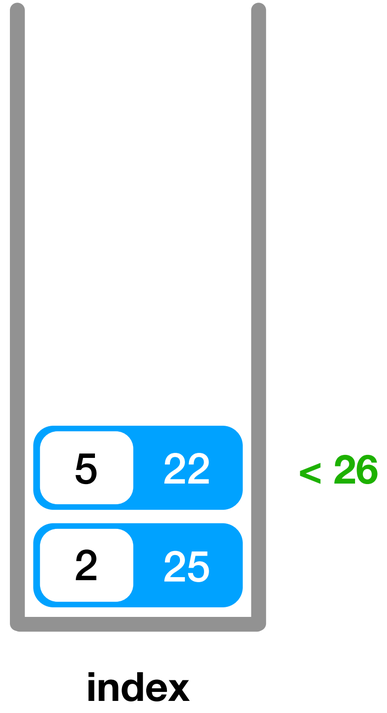
739. 每日温度

力扣 LeetCode × 拉勾 LAGOU.COM

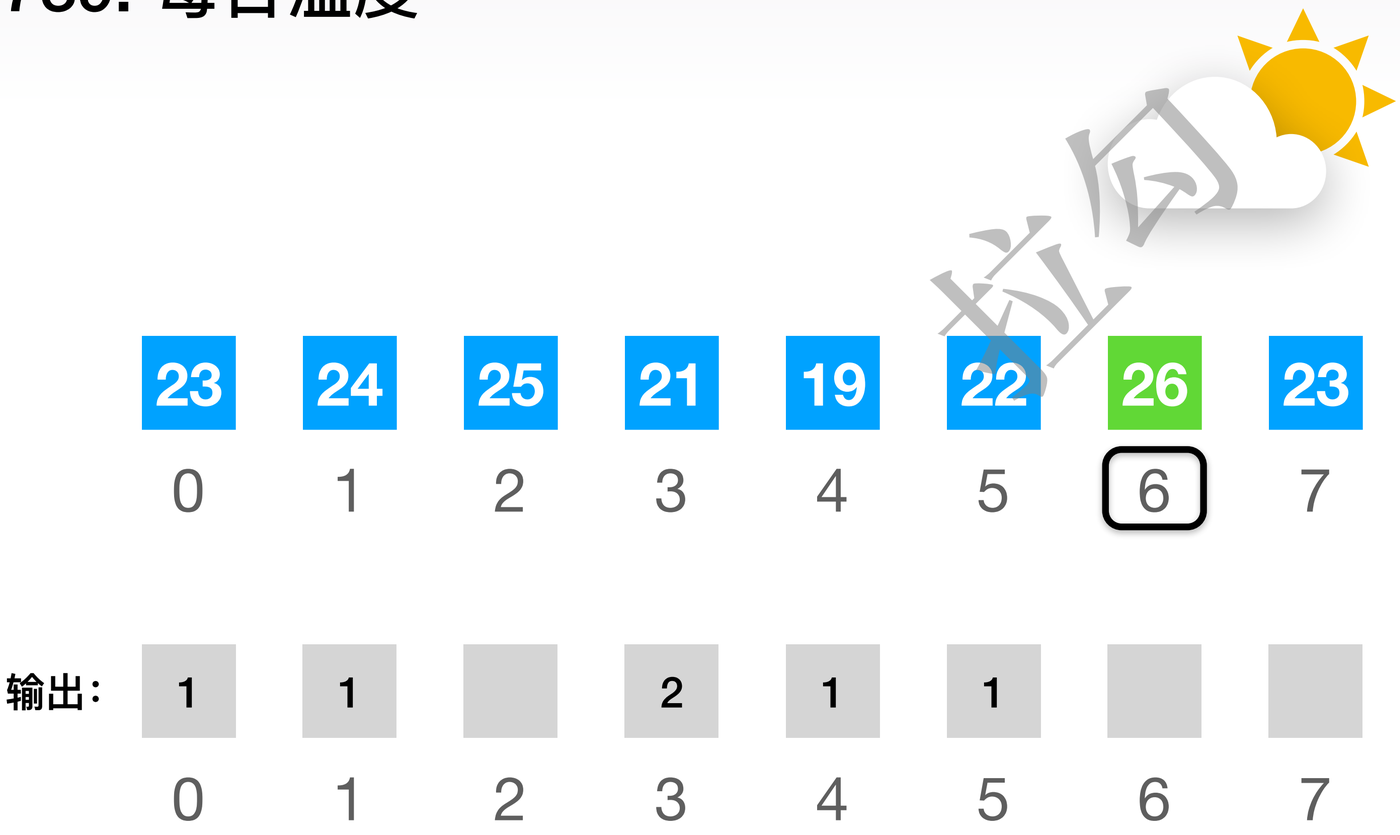
739. 每日温度

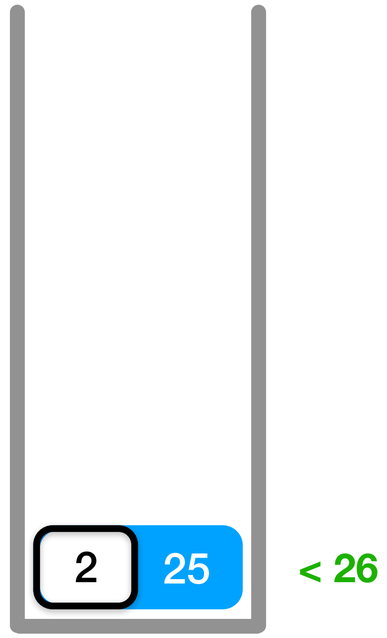
力扣 LeetCode × 拉勾 LAGOU.COM

739. 每日温度



739. 每日温度



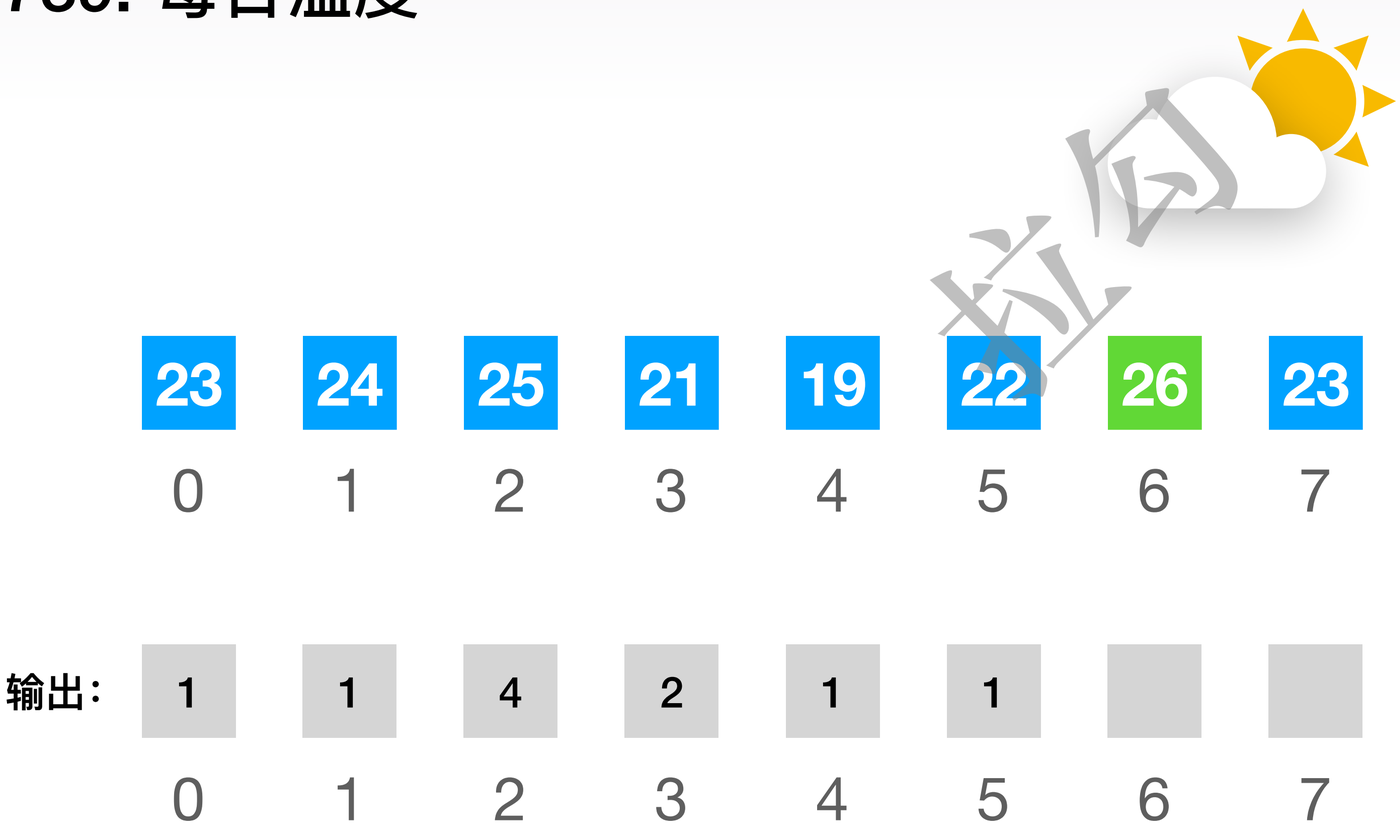


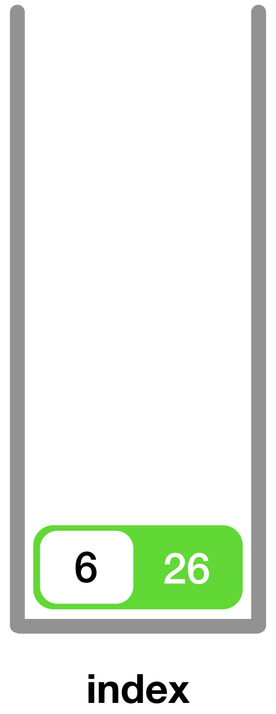
739. 每日温度


23 24 25 21 19 22 26 23
0 1 2 3 4 5 6 7
输出：
1 1 4 2 1 1
0 1 2 3 4 5 6 7
2 25
< 26
index

力扣 LeetCode × 拉勾 LAGOU.COM

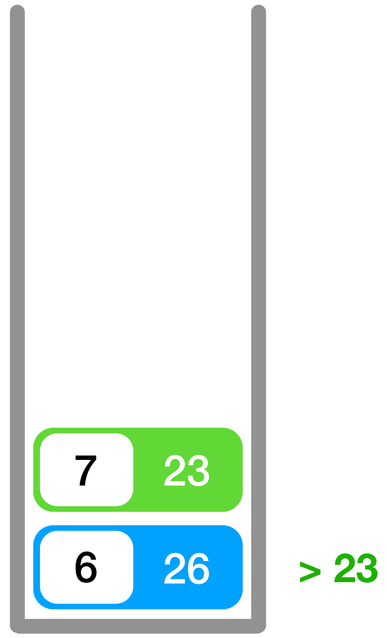
739. 每日温度





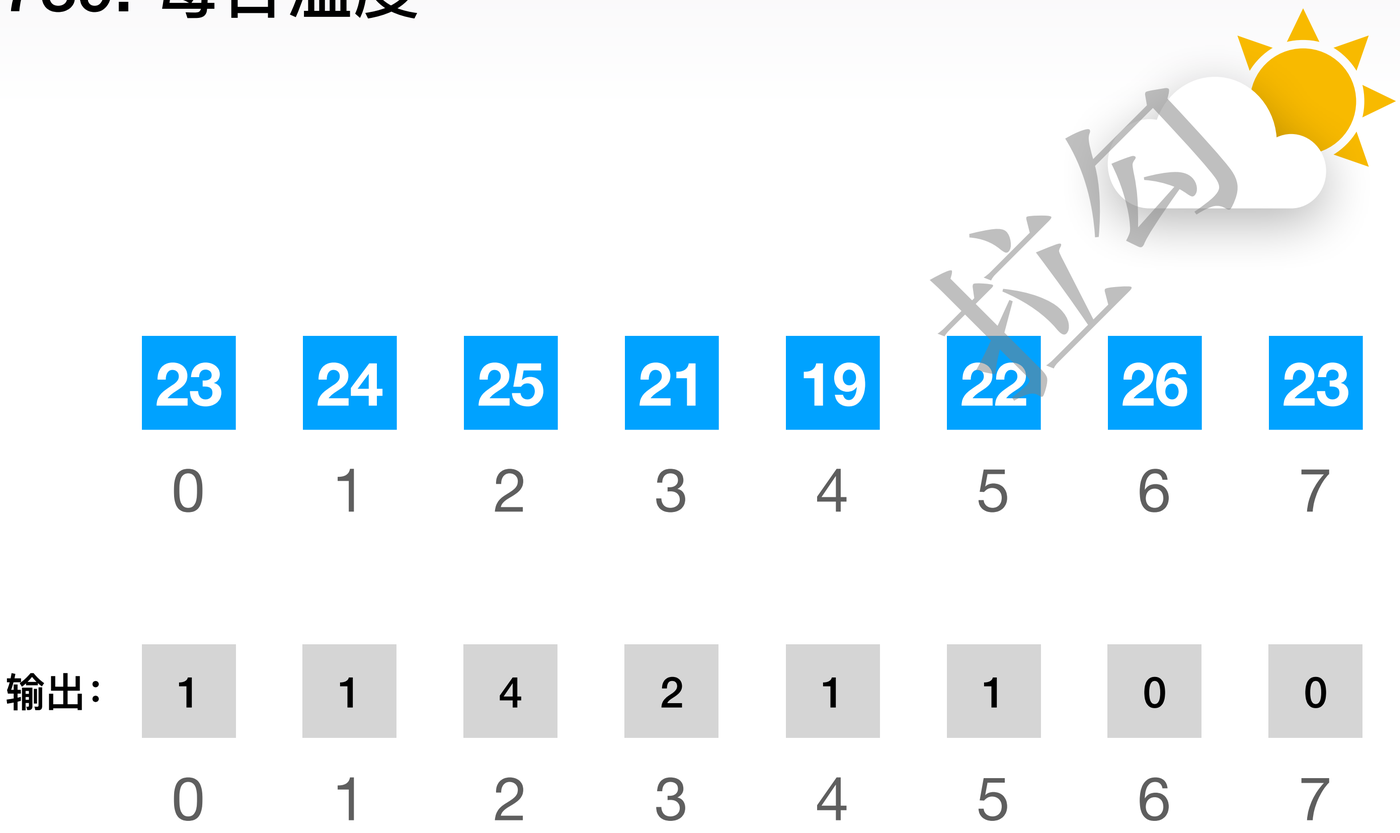
739. 每日温度

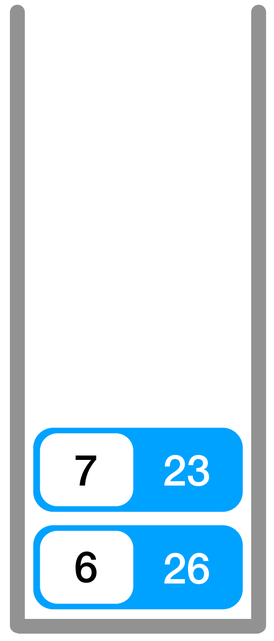

23 24 25 21 19 22 26 23
0 1 2 3 4 5 6 7
输出：
1 1 4 2 1 1
0 1 2 3 4 5 6 7


7 23
6 26
> 23
index

力扣 LeetCode × 拉勾 LAGOU.COM

739. 每日温度



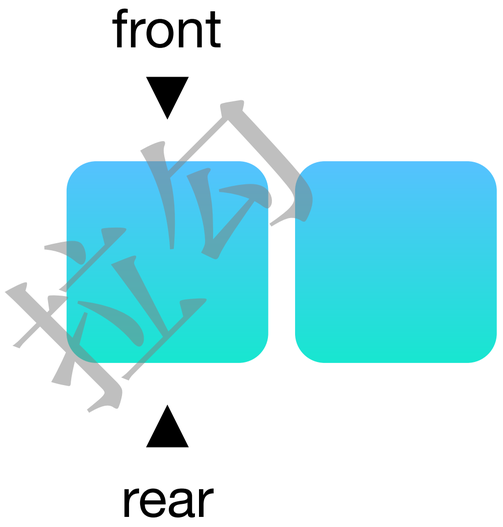


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

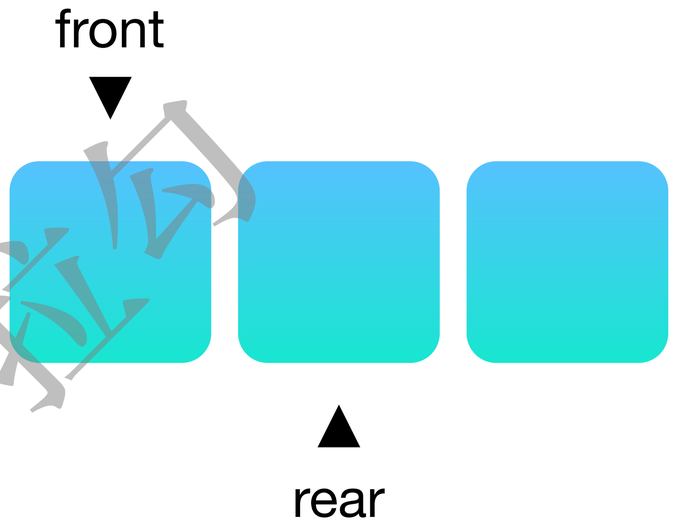


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

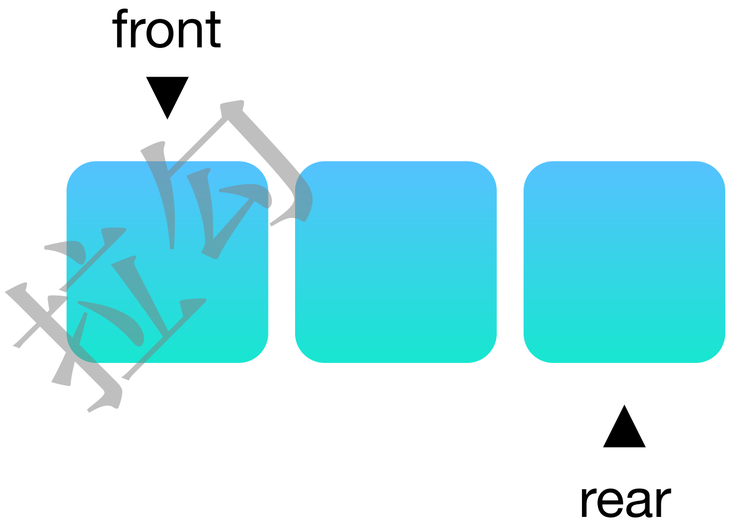


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

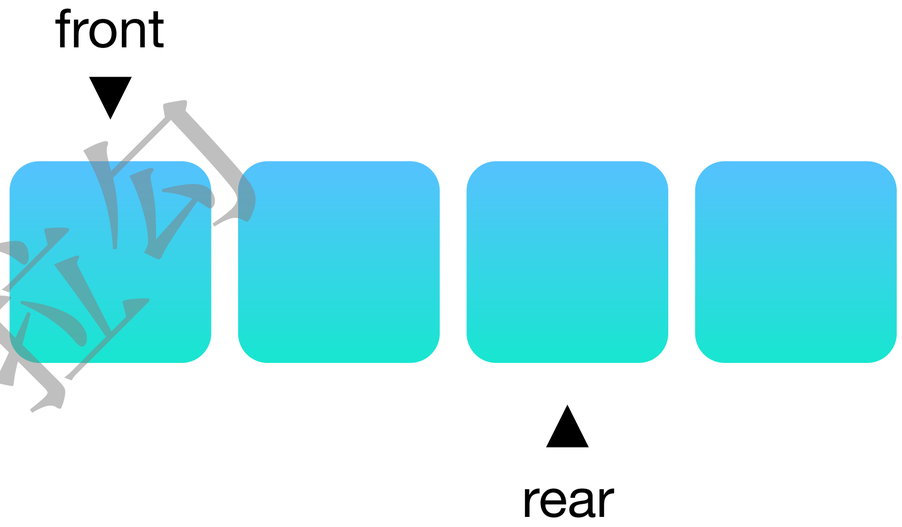


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

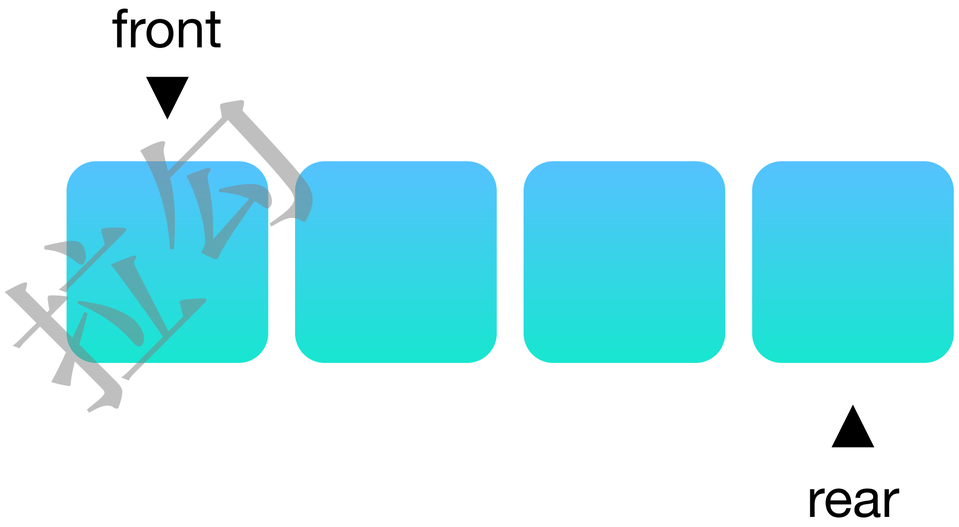


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

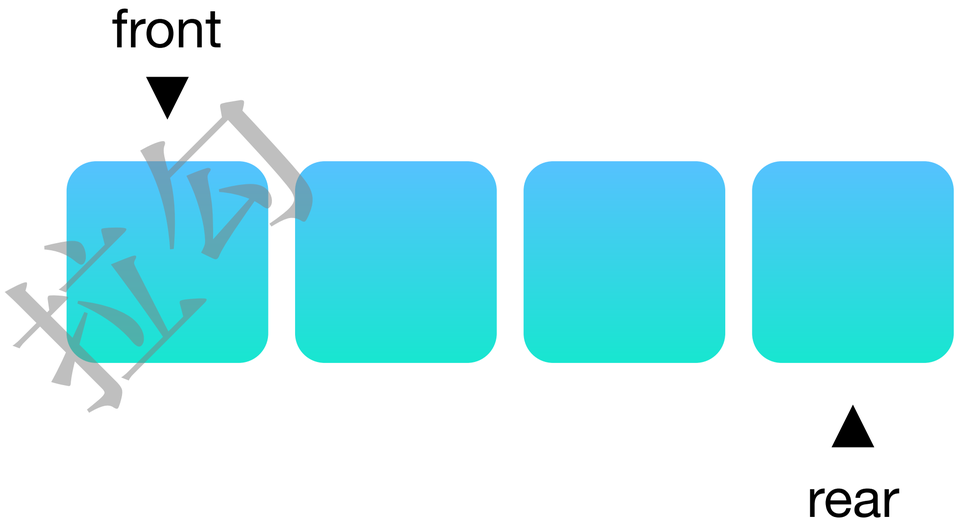


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

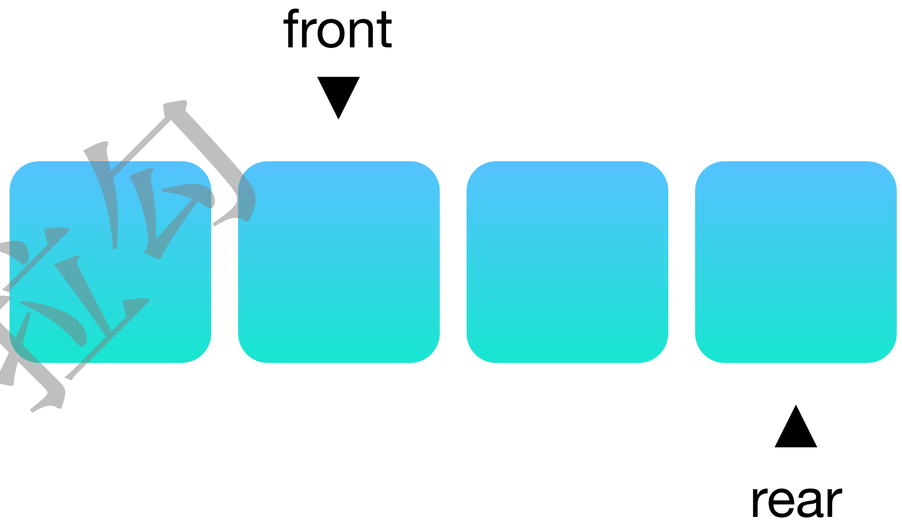


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

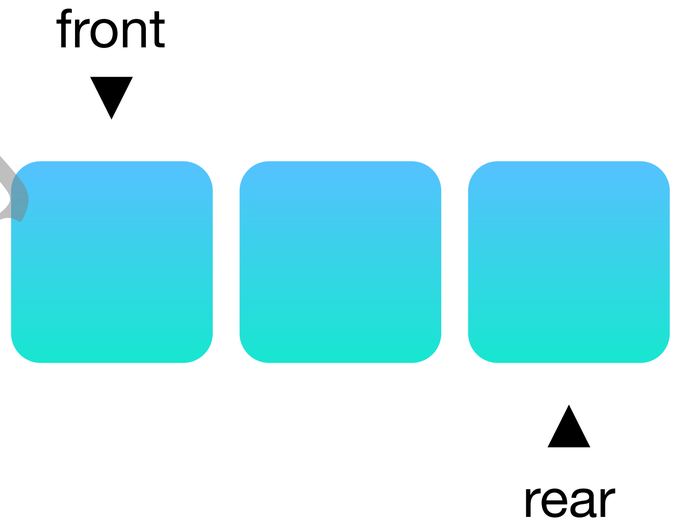


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

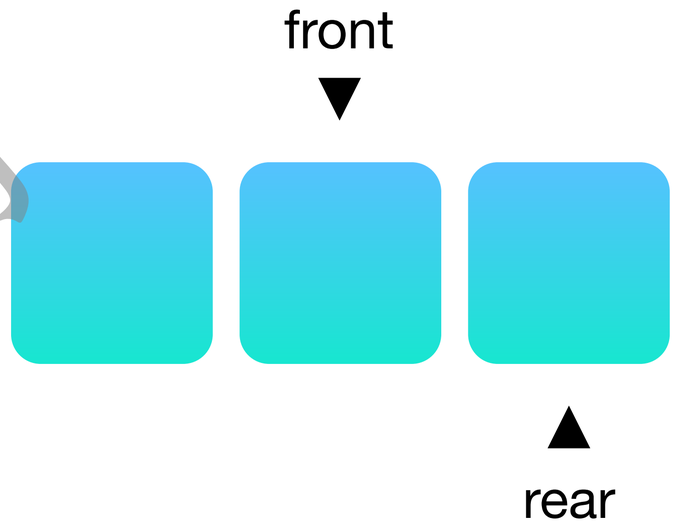


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

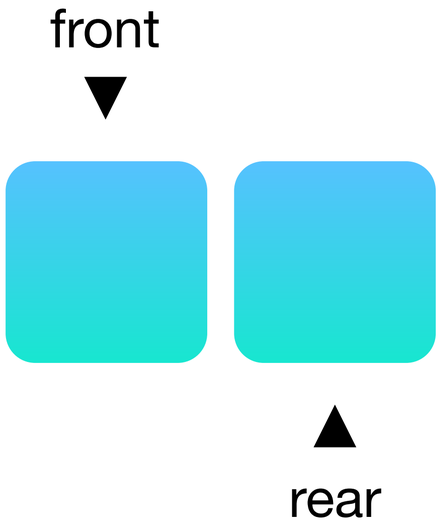


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

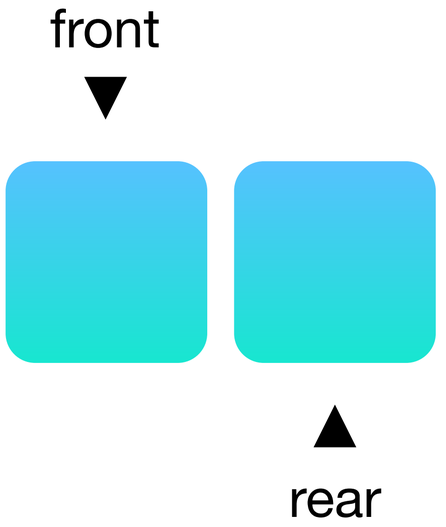


1.5 队列 / Queue

力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）



1.5 队列 / Queue

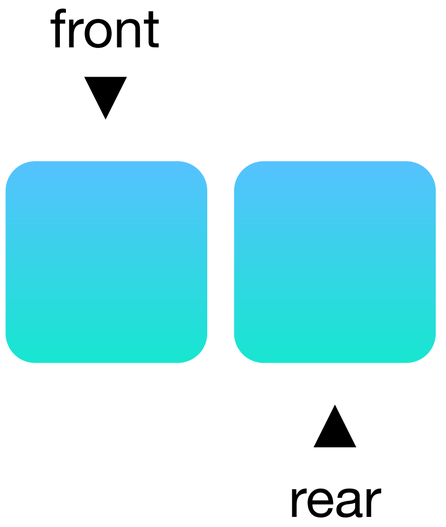
力扣 LeetCode × 拉勾 LAGOU.COM

特点

先进先出（FIFO）

常用的场景

广度优先搜索



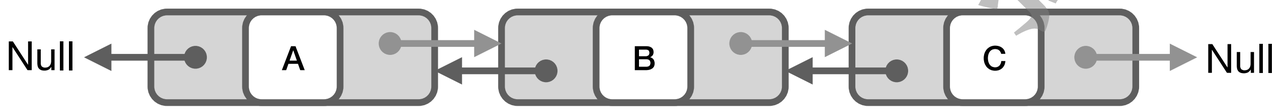
1.6 双端队列 / Deque

力扣 LeetCode × 拉勾 LAGOU.COM

基本实现

可以利用一个双链表

队列的头尾两端能在 O(1) 的时间内进行数据的查看、添加和删除



常用的场景

实现一个长度动态变化的窗口或者连续区间

力扣 LeetCode × 拉勾 LAGOU.COM

239. 滑动窗口最大值

给定一个数组 nums，有一个大小为 k 的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口 k 内的数字，滑动窗口每次只向右移动一位。
返回滑动窗口最大值。

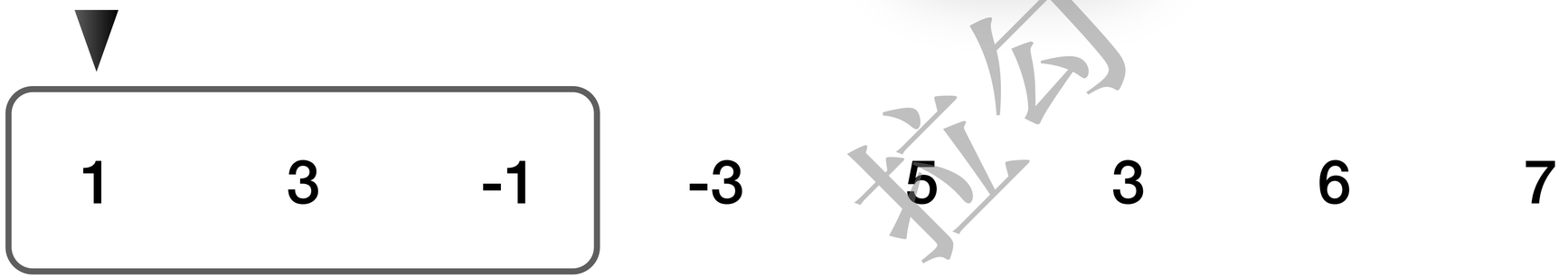
注意：
你可以假设 k 总是有效的，1 ≤ k ≤ 输入数组的大小，且输入数组不为空。

示例：
输入: nums = [1,3,-1,-3,5,3,6,7], 和 k = 3
输出: [3,3,5,5,6,7]

力扣 LeetCode × 拉勾 LAGOU.COM

239. 滑动窗口最大值

k = 3

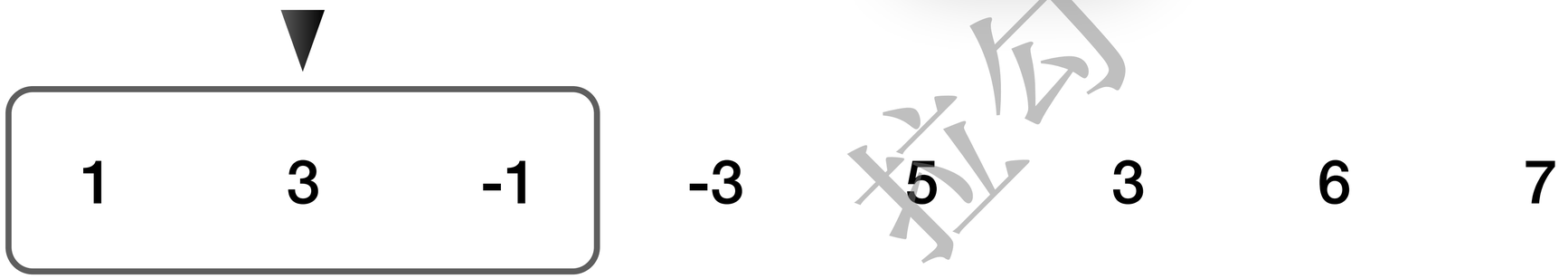


1

力扣 LeetCode × 拉勾 LAGOU.COM

239. 滑动窗口最大值

k = 3

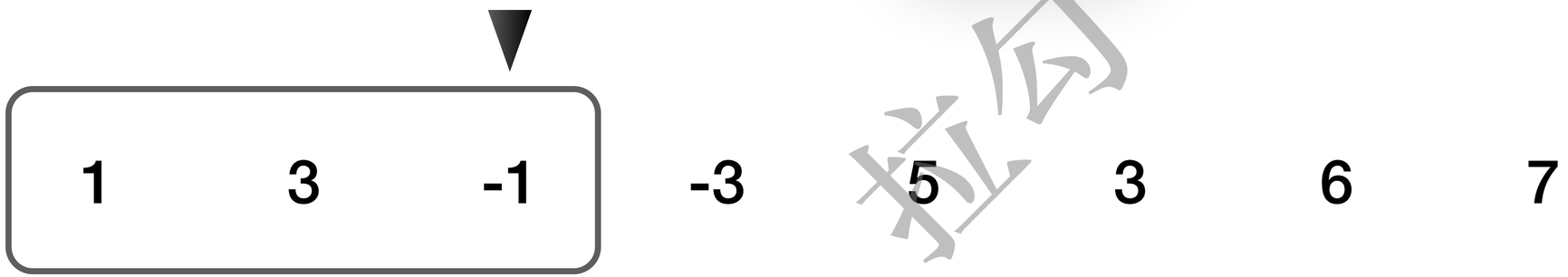


3

力扣 LeetCode × 拉勾 LAGOU.COM

239. 滑动窗口最大值

k = 3



结果：

3

力扣 LeetCode × 拉勾 LAGOU.COM

239. 滑动窗口最大值

结果：

3 , 3

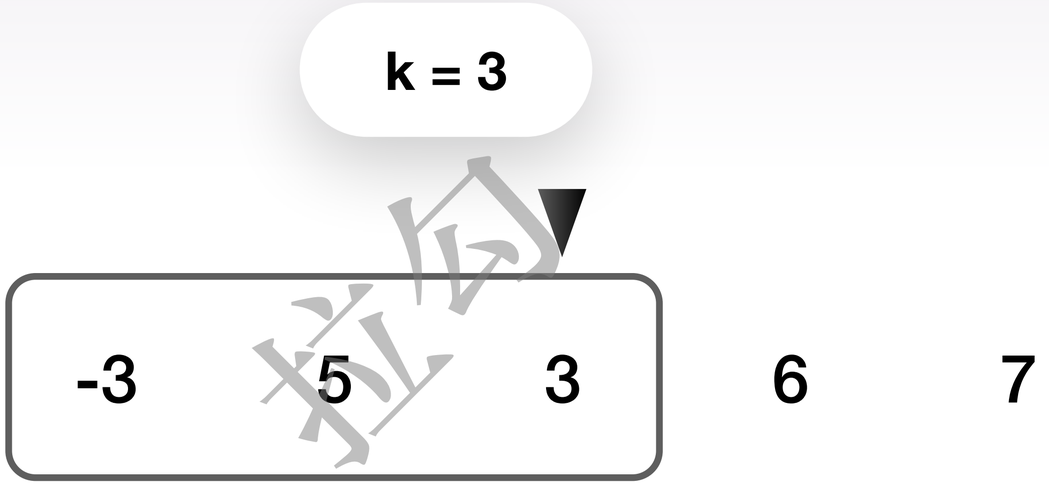
力扣 LeetCode × 拉勾 LAGOU.COM

239. 滑动窗口最大值

结果：

3 , 3 , 5

239. 滑动窗口最大值

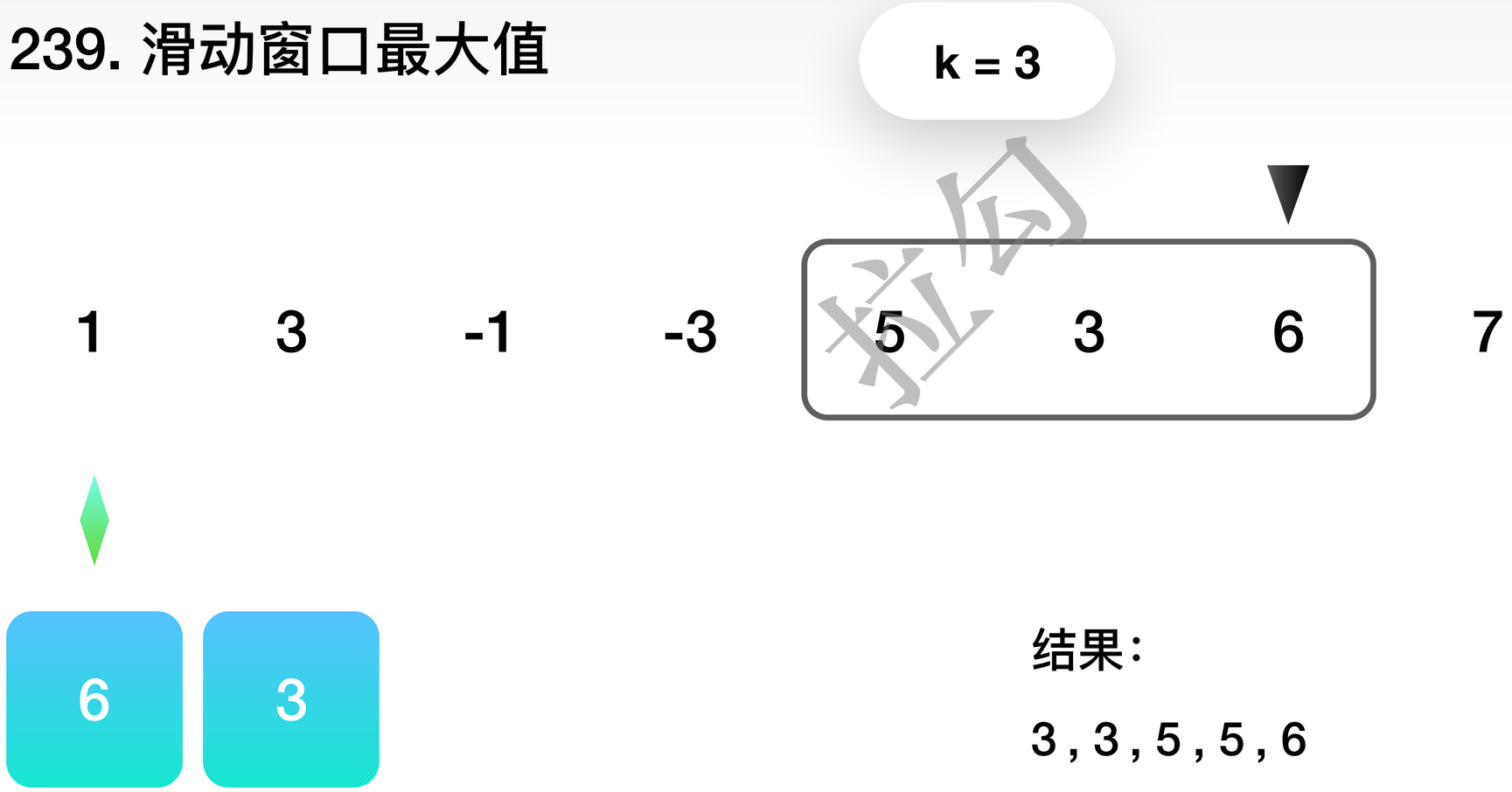


结果：

3 , 3 , 5 , 5

力扣 LeetCode × 拉勾 LAGOU.COM

239. 滑动窗口最大值

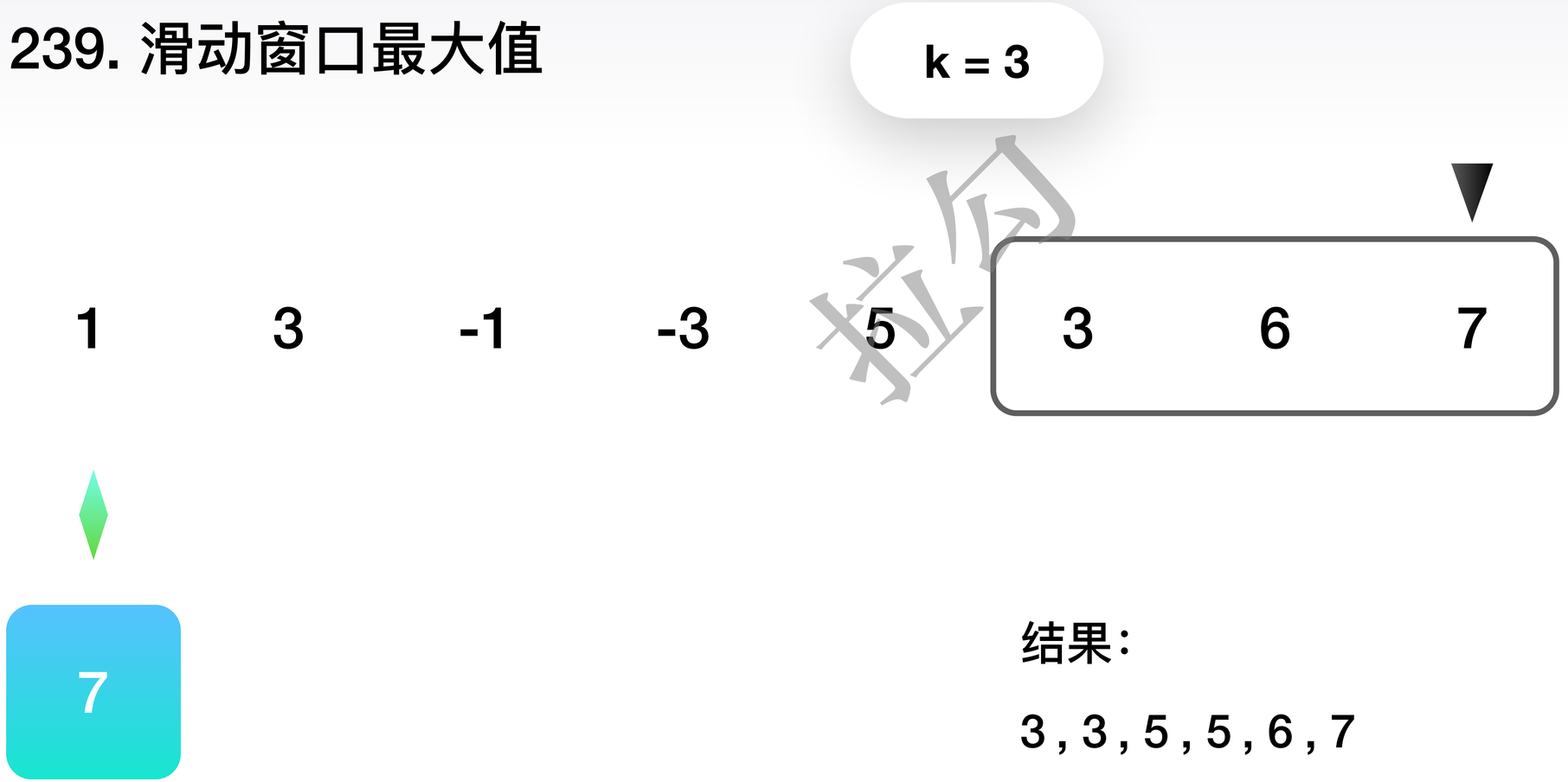


结果：

3 , 3 , 5 , 5 , 6

力扣 LeetCode × 拉勾 LAGOU.COM

239. 滑动窗口最大值



结果：

3 , 3 , 5 , 5 , 6 , 7

1.7 树 / Tree

力扣 LeetCode × 拉勾 LAGOU.COM

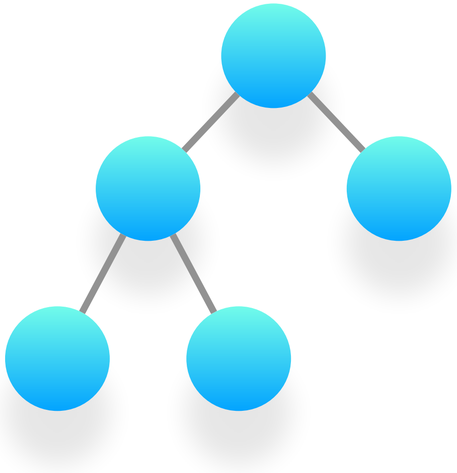
树的共性

结构直观

通过树问题来考察 **递归算法** 掌握的熟练程度

面试中常考的树的形状有

普通二叉树



1.7 树 / Tree

力扣 LeetCode × 拉勾 LAGOU.COM

树的共性

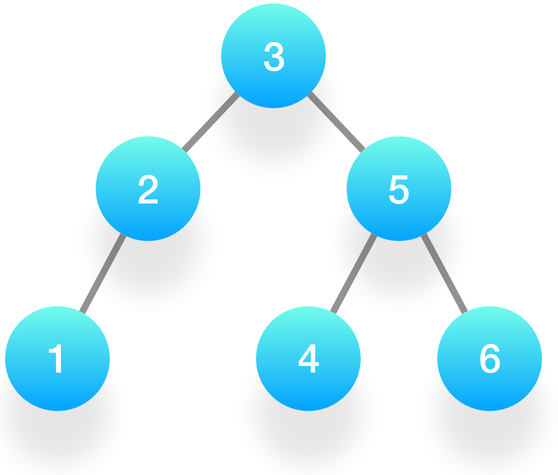
结构直观

通过树问题来考察 **递归算法** 掌握的熟练程度

面试中常考的树的形状有

普通二叉树

平衡二叉树



1.7 树 / Tree

力扣 LeetCode × 拉勾 LAGOU.COM

树的共性

结构直观

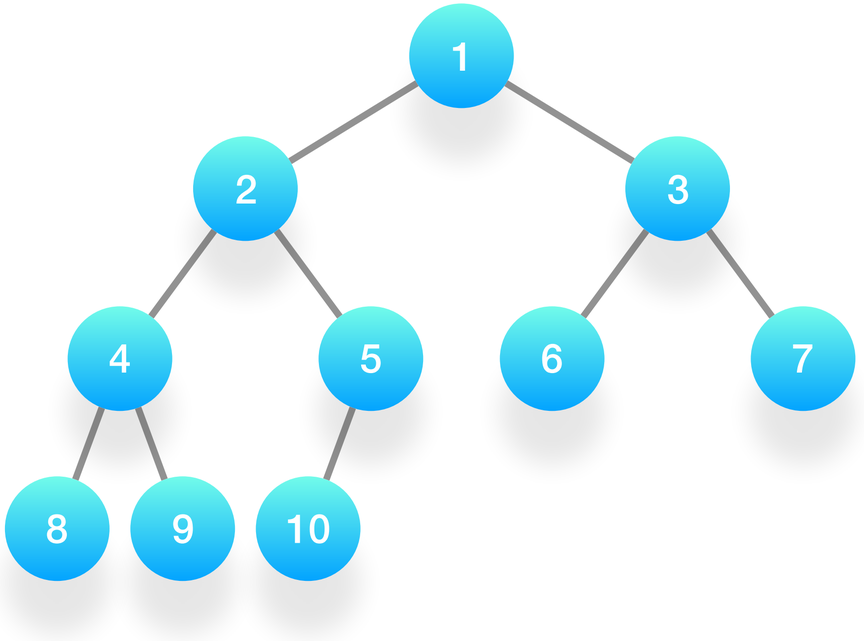
通过树问题来考察 **递归算法** 掌握的熟练程度

面试中常考的树的形状有

普通二叉树

平衡二叉树

完全二叉树



1.7 树 / Tree

力扣 LeetCode × 拉勾 LAGOU.COM

树的共性

结构直观

通过树问题来考察 **递归算法** 掌握的熟练程度

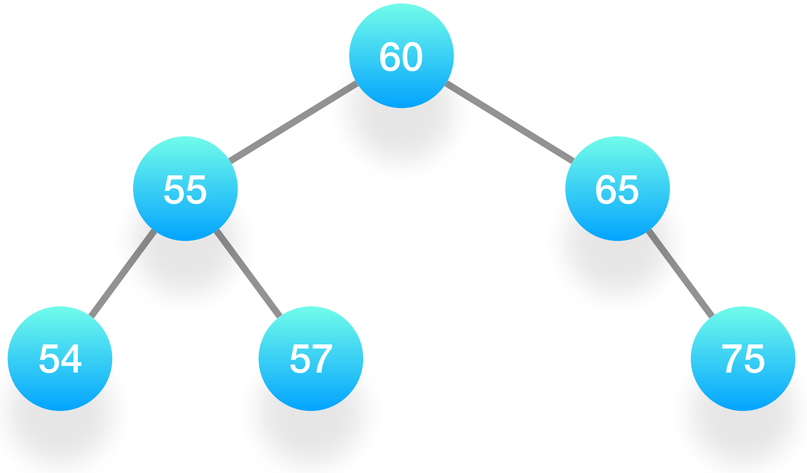
面试中常考的树的形状有

普通二叉树

平衡二叉树

完全二叉树

二叉搜索树



1.7 树 / Tree

力扣 LeetCode × 拉勾 LAGOU.COM

树的共性

结构直观

通过树问题来考察 **递归算法** 掌握的熟练程度

面试中常考的树的形状有

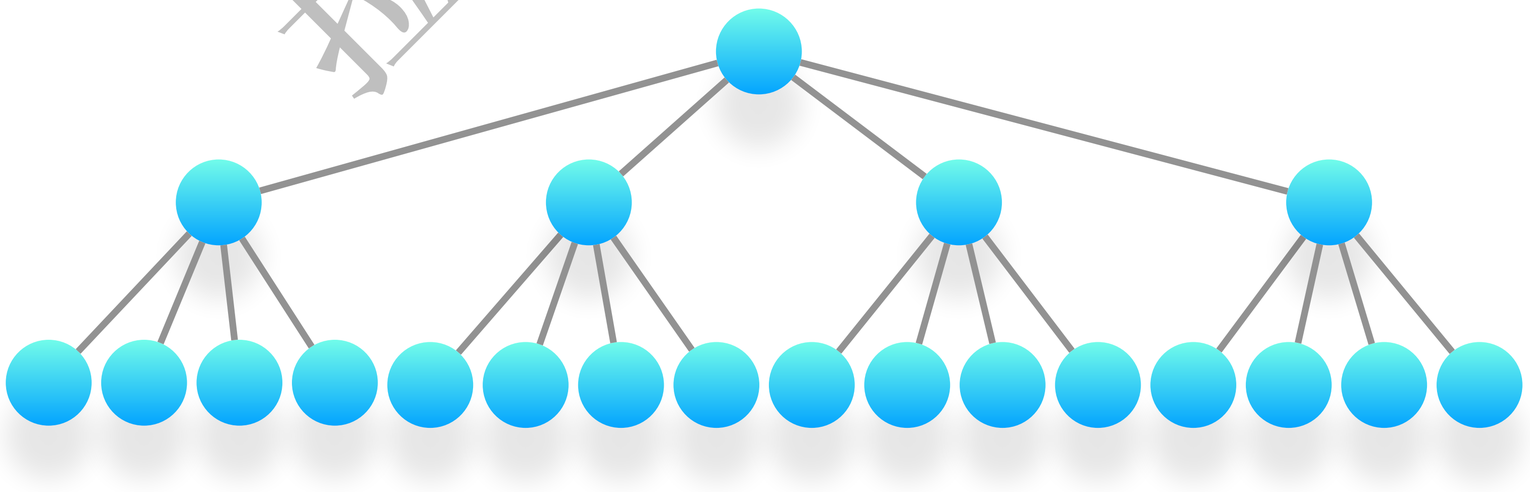
普通二叉树

平衡二叉树

完全二叉树

二叉搜索树

四叉树



1.7 树 / Tree

力扣 LeetCode × 拉勾 LAGOU.COM

树的共性

结构直观

通过树问题来考察 **递归算法** 掌握的熟练程度

面试中常考的树的形状有

普通二叉树

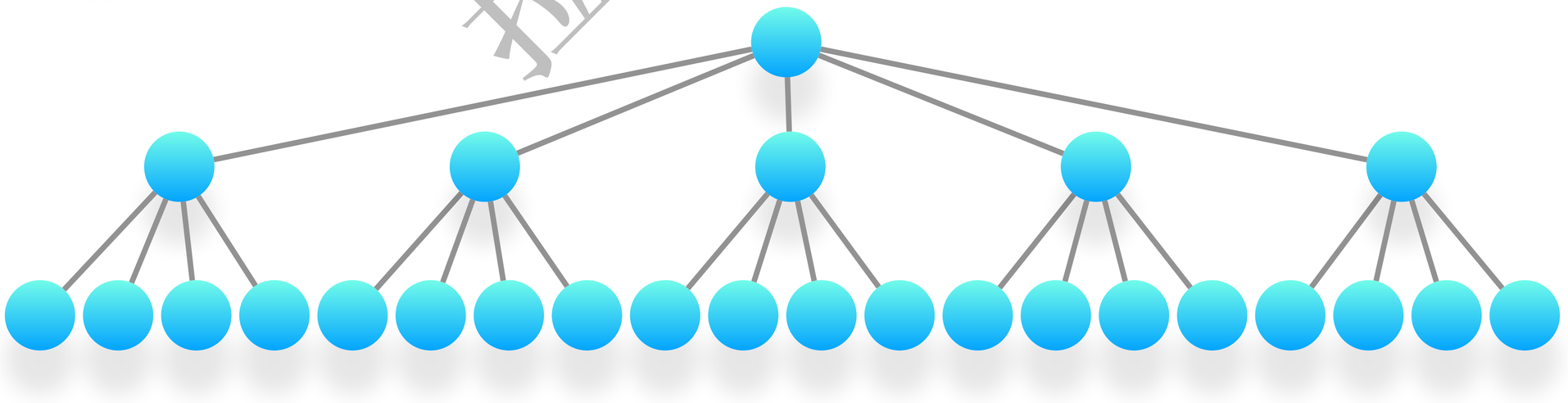
平衡二叉树

完全二叉树

二叉搜索树

四叉树

多叉树



1.7 树 / Tree

力扣 LeetCode × 拉勾 LAGOU.COM

树的共性

结构直观

通过树问题来考察 **递归算法** 掌握的熟练程度

面试中常考的树的形状有

普通二叉树

平衡二叉树

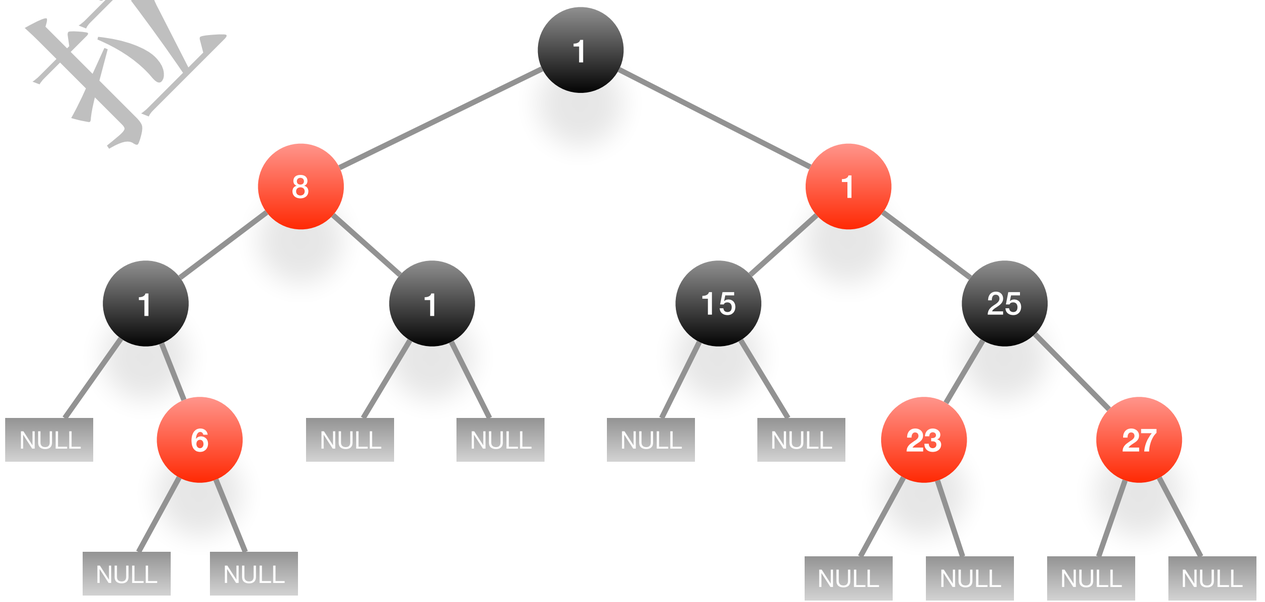
完全二叉树

二叉搜索树

四叉树

多叉树

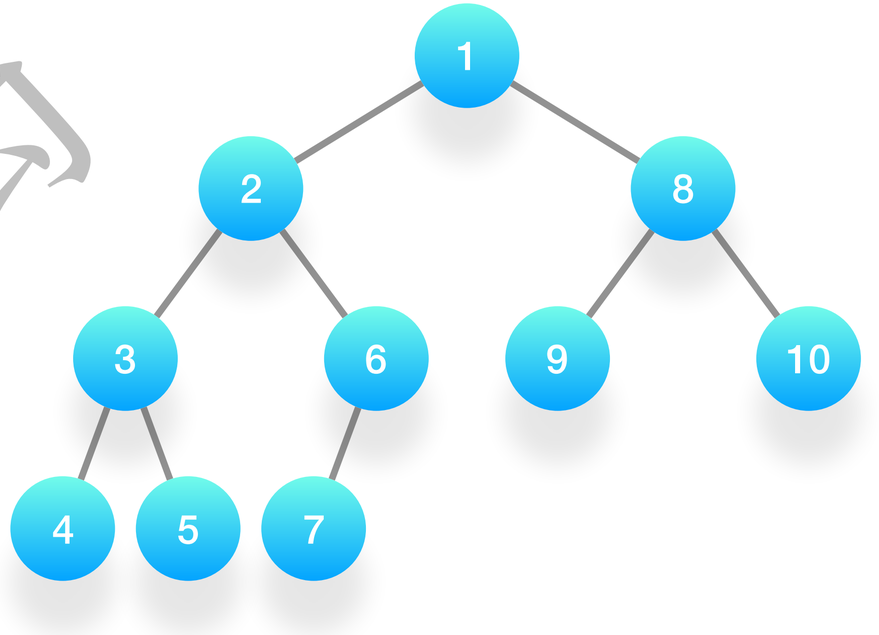
特殊的树： 红黑树、自平衡二叉搜索树



力扣 LeetCode × 拉勾 LAGOU.COM

遍历

前序遍历（Preorder Traversal）

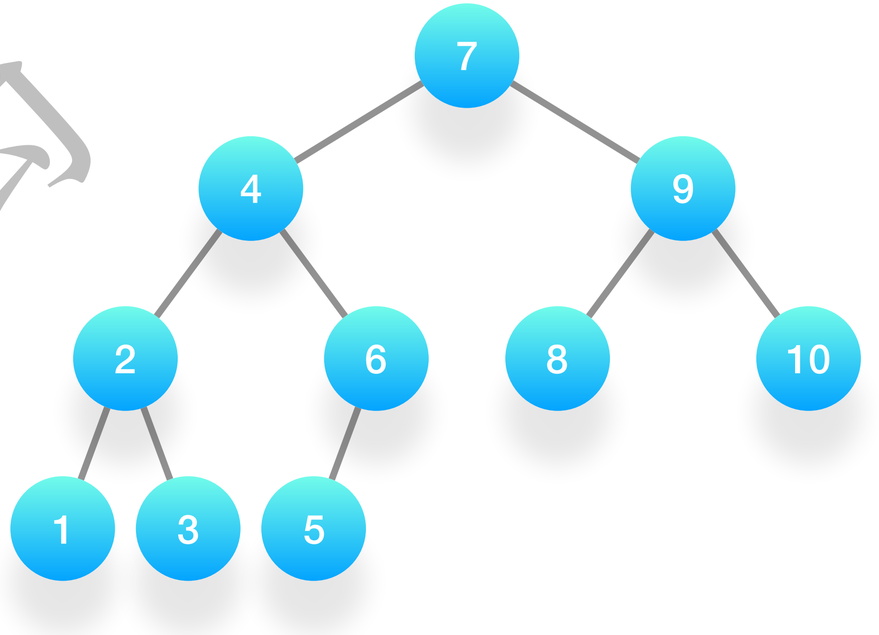


力扣 LeetCode × 拉勾 LAGOU.COM

遍历

前序遍历（Preorder Traversal）

中序遍历（Inorder Traversal）



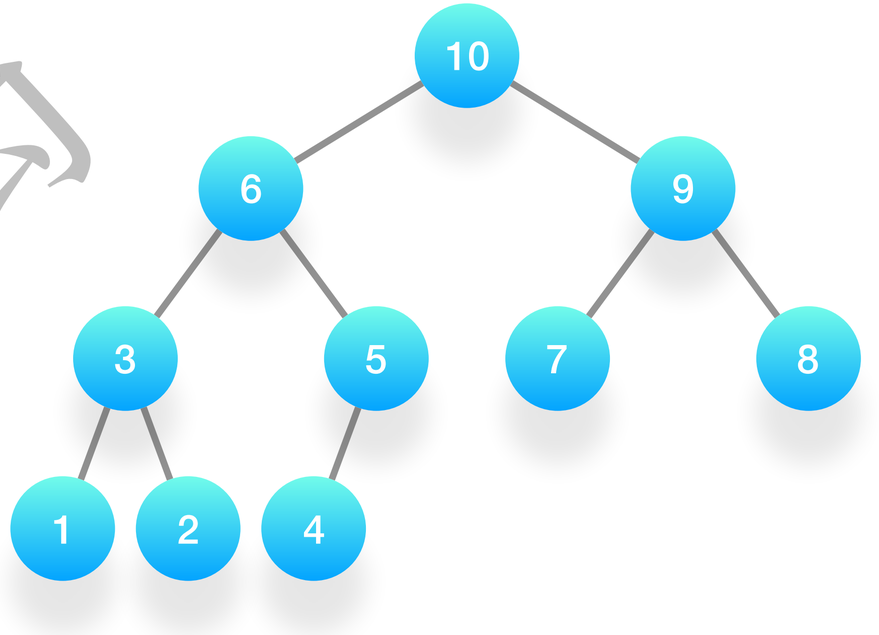
力扣 LeetCode × 拉勾 LAGOU.COM

遍历

前序遍历（Preorder Traversal）

中序遍历（Inorder Traversal）

后序遍历（Postorder Traversal）

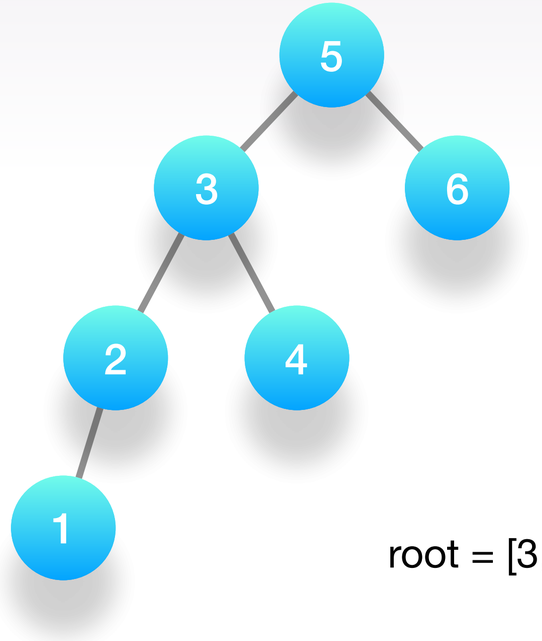


力扣 LeetCode × 拉勾 LAGOU.COM

230. 二叉搜索中第 K 小的元素

给定一个二叉搜索树，编写一个函数 `kthSmallest` 来查找其中第 **k** 个最小的元素。

说明：
你可以假设 k 总是有效的，1 ≤ k ≤ 二叉搜索树元素个数。



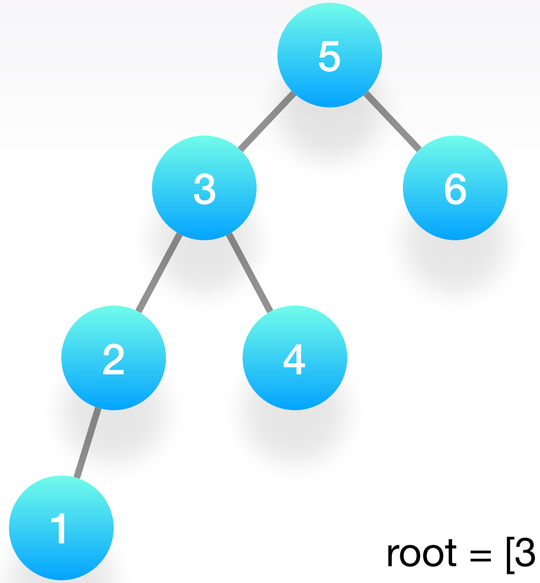
root = [3, 1, 4, null, 2]

力扣 LeetCode × 拉勾 LAGOU.COM

230. 二叉搜索中第 K 小的元素

给定一个二叉搜索树，编写一个函数 `kthSmallest` 来查找其中第 **k** 个最小的元素。

说明：
你可以假设 k 总是有效的，1 ≤ k ≤ 二叉搜索树元素个数。



root = [3, 1, 4, null, 2]

力扣 LeetCode × 拉勾 LAGOU.COM

Next：课时 2《高级数据结构》

记得多加练习，才能更好地巩固知识点。

关注 “拉勾教育”
学习技术干货

关注 “LeetCode力扣”
获得算法技术干货