力扣 LeetCode × 拉勾 LAGOU.COM

第十一课

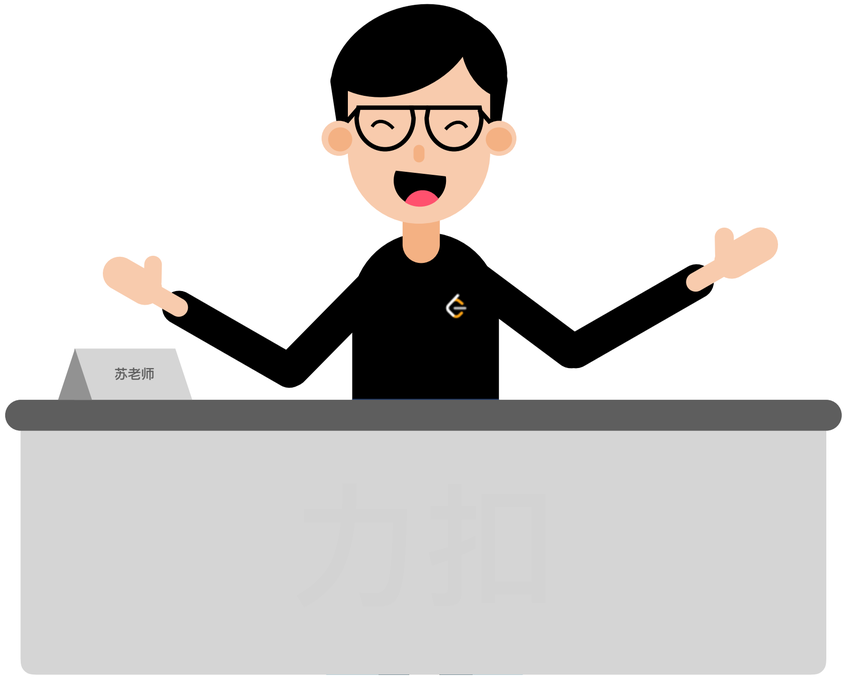
剖析大厂算法面试真题 - 难题精讲（二）

11.0 简介 / Introduction

力扣 LeetCode × 拉勾 LAGOU.COM

难题精讲（二)

- 回文对
- 至多包含 K 个不同字符的最长子串
- 接雨水 II



力扣 LeetCode × 拉勾 LAGOU.COM

336. 回文对

给定一组**唯一**的单词，找出所有**不同**的索引对 `[i,j]`，使得列表中的两个单词， `word[i] + word[j]`，可拼接成回文串。

示例 1：

输入：
`["abcd","dcba","lls","s","sssll"]`
输出： `[[0,1],[1,0],[3,2],[2,4]]`
解释： 可拼接成的回文串为
`["dcbaabcd","abcddcba","slls","llssssll"]`。

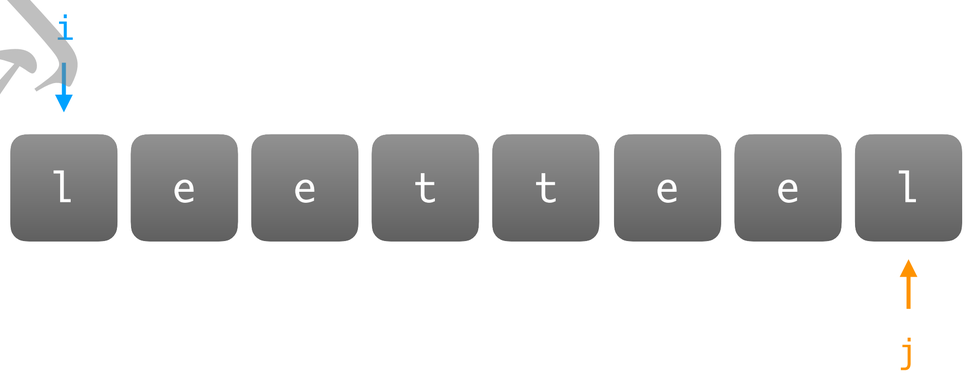
示例 2：

输入： ["bat","tab","cat"]
输出： `[[0,1],[1,0]]`
解释： 可拼接成的回文串为 `["battab","tabbat"]`。

336. 回文对

解题思路

- 回文：正读和反读都一样的字符串
- 检查字符串是否回文
  - 翻转给定字符串后，对比原字符串，
    需要 $O(n)$ 的空间复杂度
  - 定义指针 `i` 指向字符串的头，定义指针 `j` 指向字符串的尾，
    从两头开始检查，发现不相等表明不是回文，
    一直检查到两个指针相遇为止



```
boolean isPalindrome(String word, int i, int j) {
  while (i < j) {
    if (word.charAt(i++) != word.charAt(j--))
      return false;
  }

  return true;
}
```

336. 回文对

暴力法

- 找出所有的两两组合
- 对每种组合进行排查，寻找可以构成回文的组合
- 复杂度分析
  - 假设有 n 个单词，每个单词平均长度为 k，从 n 个单词中选两个组合在一起排列，共有 $P(n,2) = n \times (n-1)$ 种
  - 对每个组合在一起的字符串进行回文检查，需要 $2k$ 的时间复杂度，最终时间复杂度为 $O(n^2 \times k)$

336. 回文对

暴力法

- 当对两个组合在一起的字符串进行回文检查时，假设两个字符串长度分别为 k_1, k_2，共有以下三种情况：
  - $k_1 = k_2$

336. 回文对

暴力法

- 当对两个组合在一起的字符串进行回文检查时，假设两个字符串长度分别为 k_1, k_2，共有以下三种情况：
  - $k_1 = k_2$
  - $k_1 > k_2$

336. 回文对

暴力法

- 当对两个组合在一起的字符串进行回文检查时，假设两个字符串长度分别为 k_1, k_2，共有以下三种情况：
  - $k_1 = k_2$
  - $k_1 > k_2$
  - $k_1 < k_2$

  一旦发现两字符不对，或剩下的字符串不是回文时，则不能构成回文串

336. 回文对

暴力法

- 暴力法慢的原因：要对所有情况进行检查
- `s1 = "abcd"` 时，如果 `s1 + s2` 的组合构成回文：
  - s2 长度小于 2 时，最后一个字符必须为 a
  - s2 长度大于等于 2 时，最后两个字符必须为 ba
- 如何快速得知哪些字符串以 a 结尾，哪些字符串以 ba 结尾?

336. 回文对

暴力法

- 如何快速得知哪些字符串以 a 结尾，哪些字符串以 ba 结尾？

  -> 如何快速找出所有以 a 开头，或以 ba 开头的字符串？

  ★ Trie

  - 避免不必要的组合以及检查
  - 需要对每个字符串反着构建 Trie
  - **复习：**一个 Trie 一般由很多个 TrieNode 节点构成

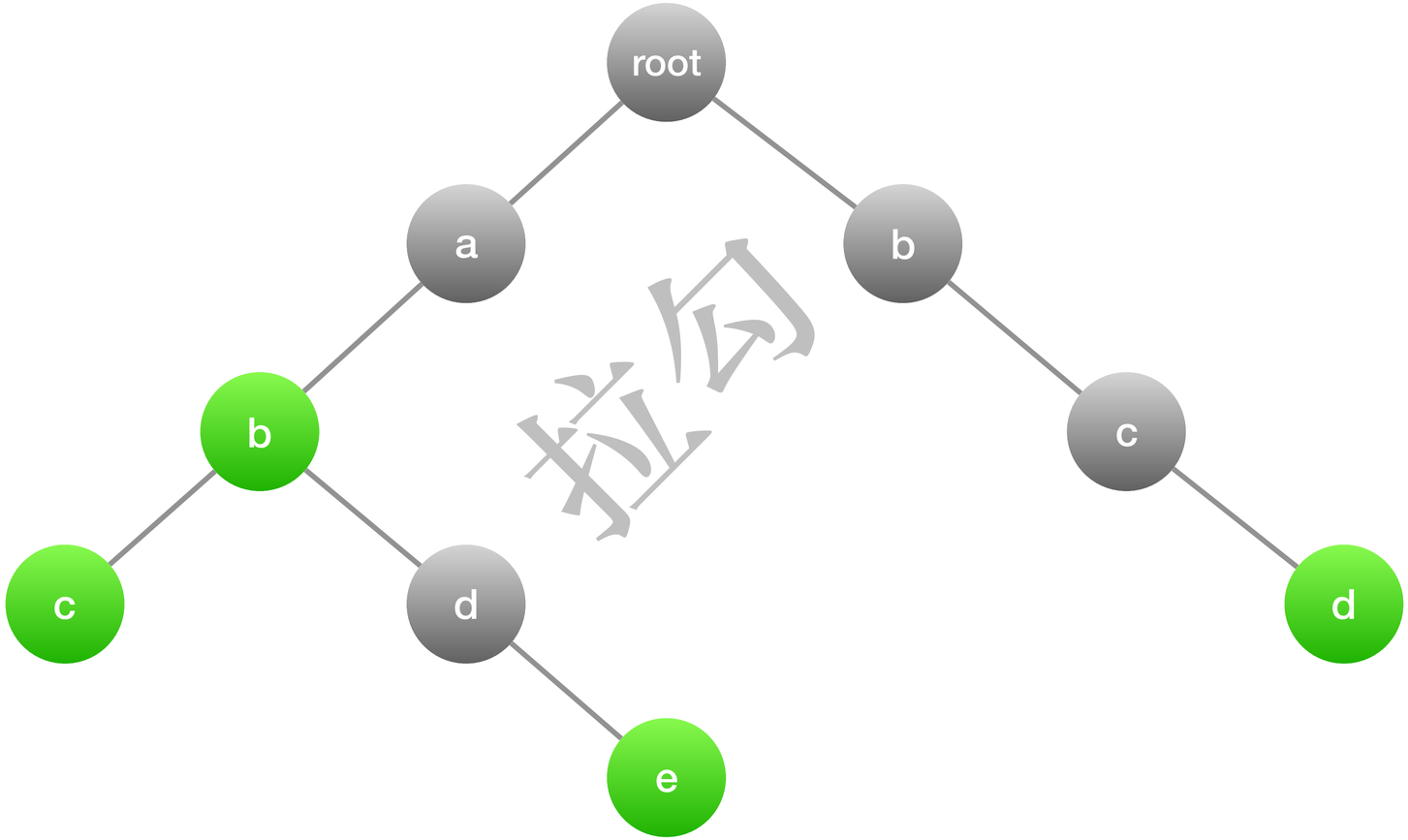
```
class TrieNode {
  boolean isEnd;
  HashMap<Character, TrieNode> children;

  TrieNode() {
    isEnd = false;
    children = new HashMap<>();
  }
}
```

- `children`:
  数组或者集合，罗列出每个分支中包含的所有字符
- `isEnd`:
  布尔值，表示该节点是否为某字符串的结尾

11.1 难题精讲（二） / 回文对

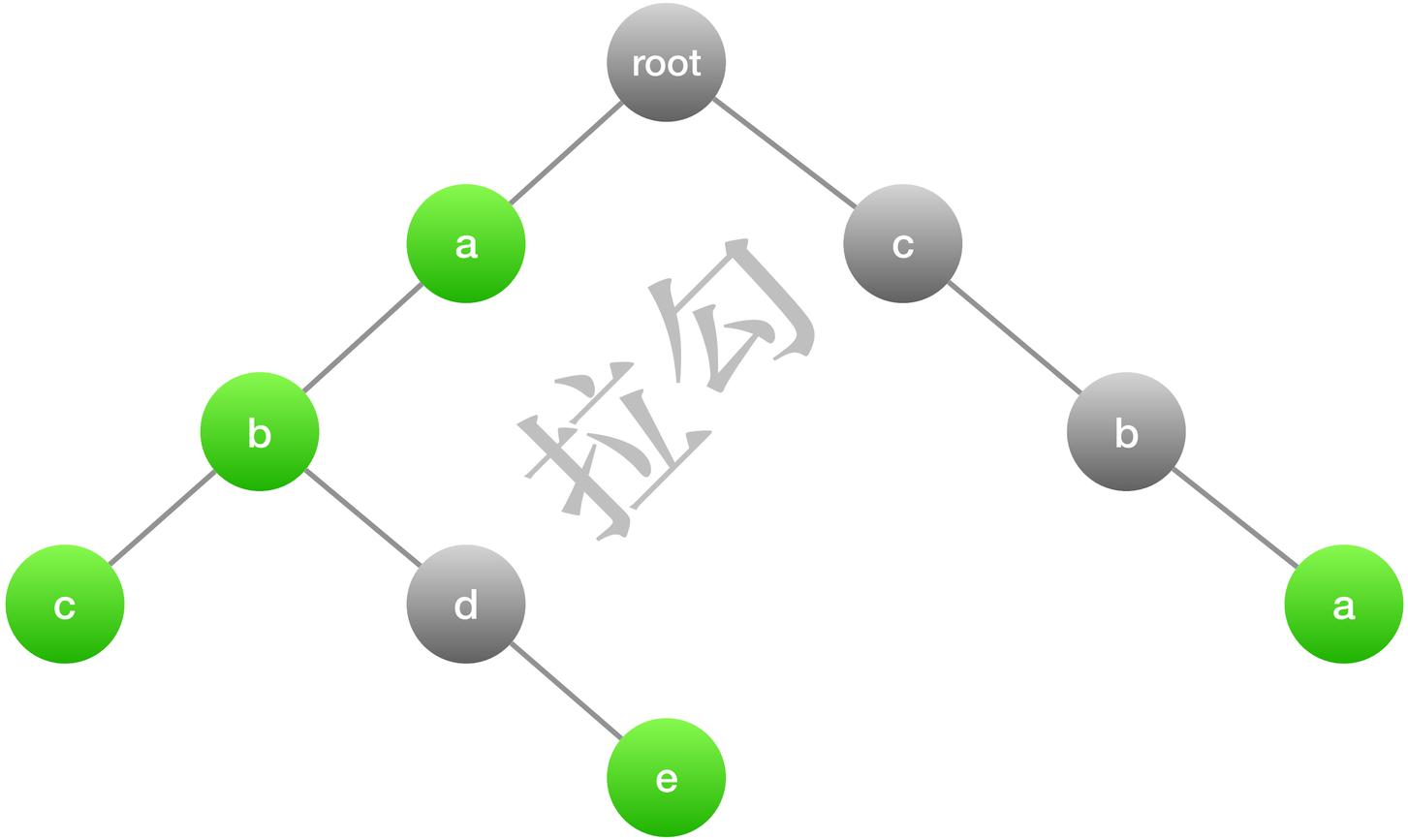
力扣 LeetCode × 拉勾 LAGOU.COM

isEnd = true



"ba" "cba" "edba" "dcb""abc"

11.1 难题精讲（二） / 回文对



"a"　"abc"　"ba"　"cba"　"edba"

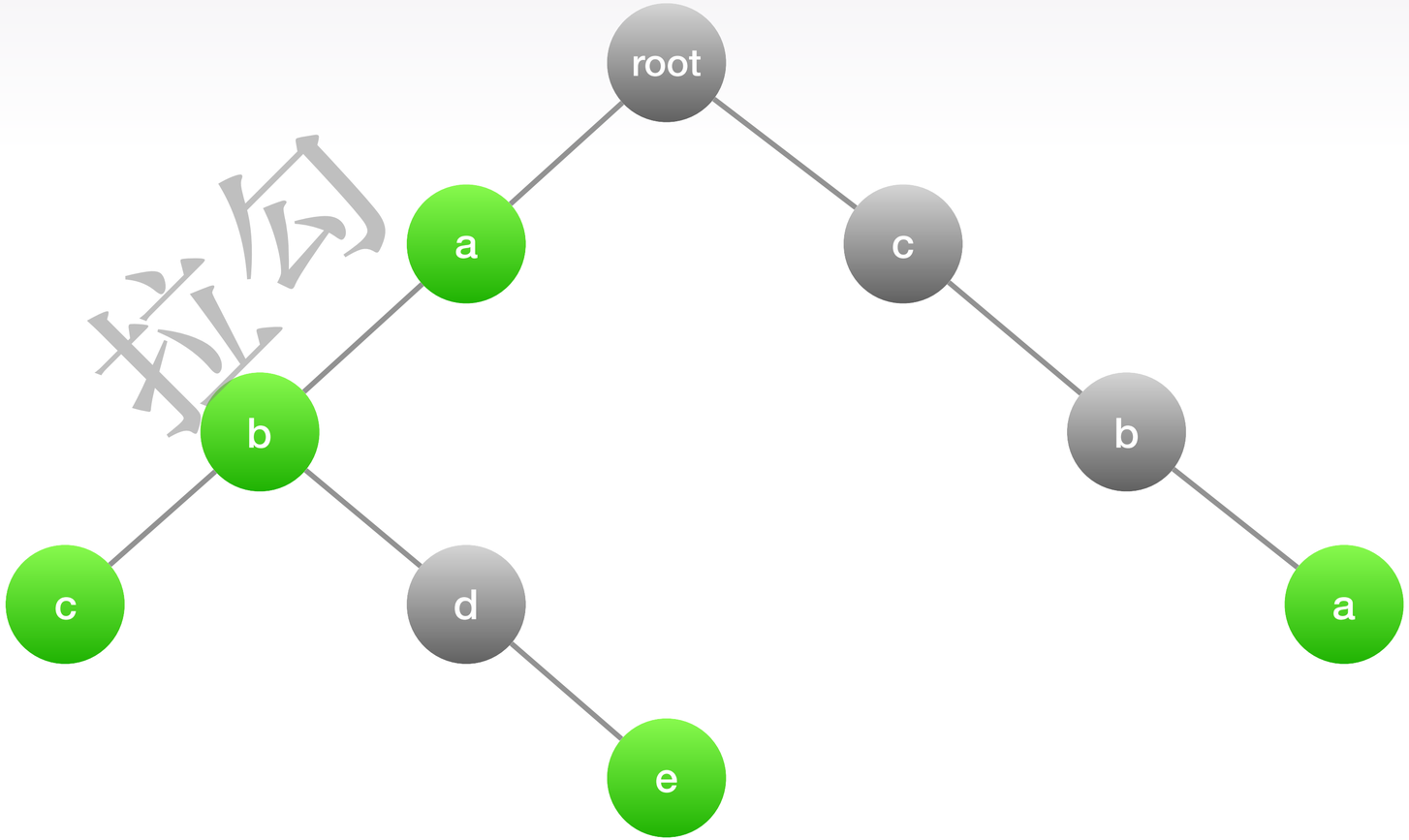
力扣 LeetCode × 拉勾 LAGOU.COM

336. 回文对

- $k_1 = k_2$

  - s1 = "abc", s2 = "cba"
    s1 + s2 = "abccba"

s1 a b c



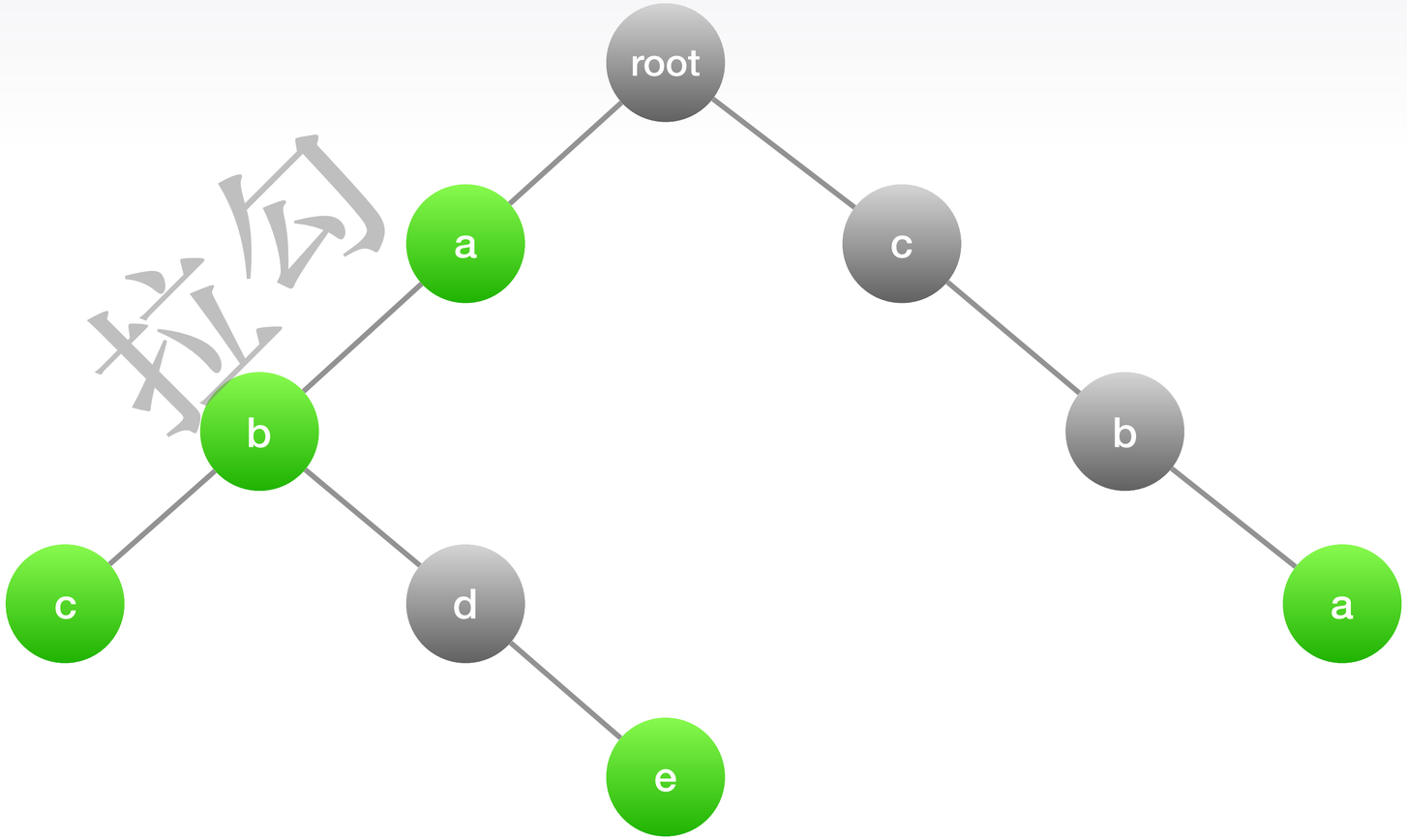
力扣 LeetCode × 拉勾 LAGOU.COM

336. 回文对

- $k_1 > k_2$

  - s1 = "abc", s2 = "ba"
    s1 + s2 = "abcba"

s1 a b c



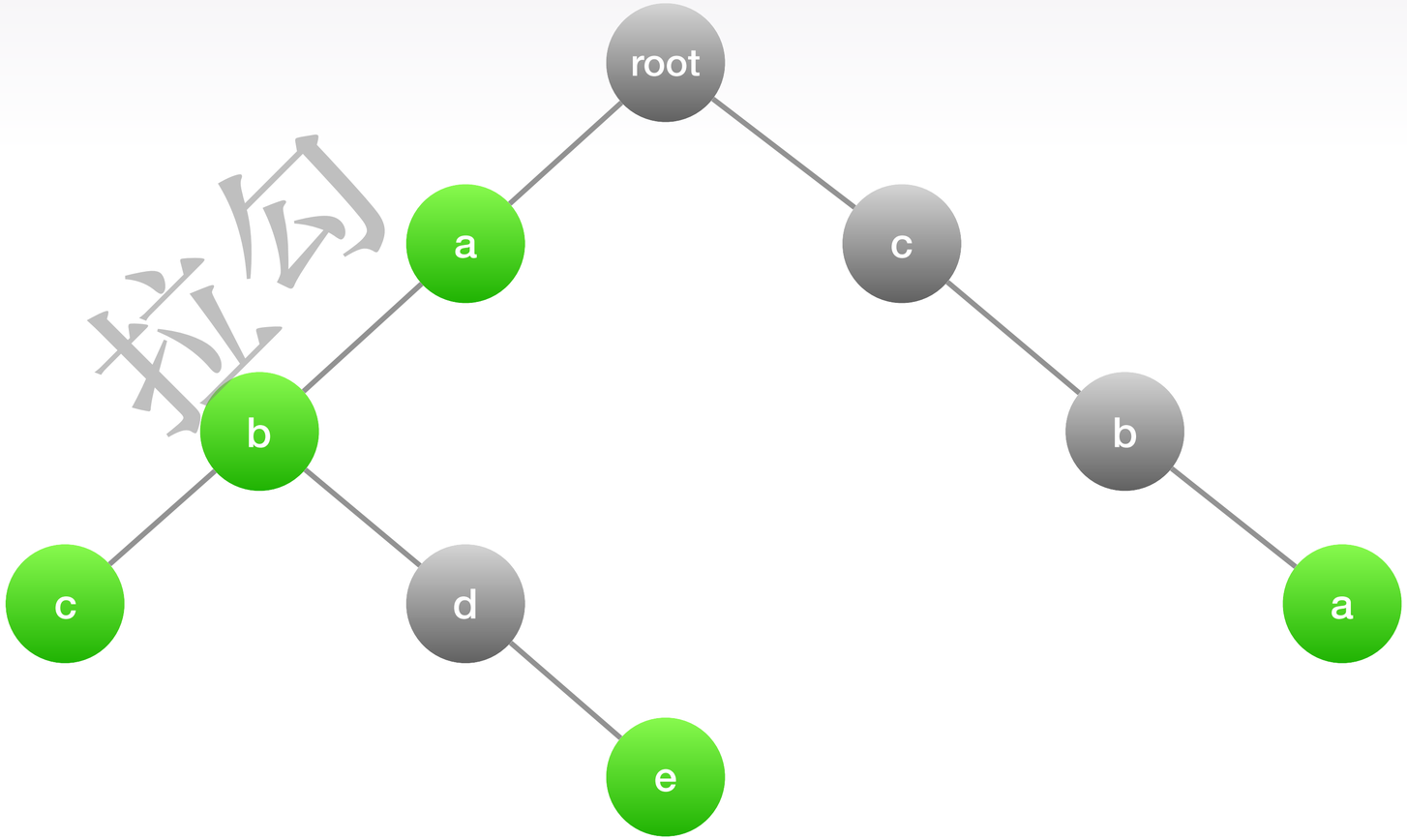
力扣 LeetCode × 拉勾 LAGOU.COM

336. 回文对

- $k_1 < k_2$

  - s1 = "a", s2 = "ba"
    s1 + s2 = "aba"

s1 a



- 对于第一、三种情况：
  - s1 字符串一定会被遍历完毕
  - 遍历完毕后，在 Trie 中对应的节点：
    - s2 中最后一个字符
    - s2 的剩余字符
      - 只要剩余字符本身是回文，就可以给这个节点添加一个数组
      - 用来记录从该节点向后所有剩余能构成回文字符串的字符下标即可
- 对于第二种情况：
  - 在 Trie 中，当遇到某个绿色节点，而它表示了某个字符串的结束，只要 s1 剩余字符能构成回文即可
  - 如何迅速从 Trie 中得知该 s2 的下标？可以用 index 代替 isEnd
    - 当 index 为 -1 时，表示不是字符串的结束位置
    - 当表示字符串的结束时，用 index 记录输入字符串的下标即可

11.1 难题精讲（二） / 回文对

力扣 LeetCode × 拉勾 LAGOU.COM

```
class TrieNode {
  int index;
  List<Integer> palindromes;
  HashMap<Character, TrieNode> children;

  TrieNode() {
    index = -1;
    children = new HashMap<>();
    palindromes = new ArrayList<>();
  }
}
```

- 修改 `TrieNode` 结构，用 `index` 替换 `isEnd`
- 添加一个 `palindromes` 列表，
  用来记录从该节点往下的字符串，
  能构成回文的所有输入字符串的下标

11.1 难题精讲（二） / 回文对

```
List<List<Integer>> palindromePairs(String[] words) {
  List<List<Integer>> res = new ArrayList<>();

  TrieNode root = new TrieNode();

  for (int i = 0; i < words.length; i++) {
      addWord(root, words[i], i);
  }

  for (int i = 0; i < words.length; i++) {
      search(words, i, root, res);
  }

  return res;
}
```

- 主体函数比较简单
- 定义一个空列表，用来记录找到的配对
- 定义一个 `Trie` 的根节点 `root`
- 创建 `Trie`
- 利用 `Trie`，找出所有的配对
- 最后返回结果

11.1 难题精讲（二） / 回文对

力扣 LeetCode × 拉勾 LAGOU.COM

```
void addWord(TrieNode root, String word, int index) {
  for (int i = word.length() - 1; i >= 0; i--) {
    char ch = word.charAt(i);

    if (!root.children.containsKey(ch)) {
      root.children.put(ch, new TrieNode());
    }

    if (isPalindrome(word, 0, i)) {
      root.palindromes.add(index);
    }

    root = root.children.get(ch);
  }

  root.palindromes.add(index);
  root.index = index;
}
```

- 创建 Trie 时，从每个字符串的末尾开始遍历
- 对于每个当前字符，
  如果他还没有被添加到 children 哈希表中，
  就创建一个新的节点
- 难点：如果该字符串从头开始到当前位置能成为回文，
  则把该字符串的下标添加到这个 Trie 节点的回文列表中
  例如：字符串“aaaba”
  由于从后向前遍历，当遍历到字符 b 时，
  发现 aaa 是回文，于是需更新一下 b 指向的节点，
  这个节点往下有一个字符能构成回文
- 当对该字符串创建完 Trie，
  将字符串的下标添加到回文列表中，
  并赋给 index，表明这里是字符串的结束

11.1 难题精讲（二） / 回文对

力扣 LeetCode × 拉勾 LAGOU.COM

```
void search(String[] words, int i, TrieNode root,
List<List<Integer>> res) {
  for (int j = 0; j < words[i].length(); j++) {
    if (root.index >= 0 && root.index != i &&
        isPalindrome(words[i], j, words[i].length() - 1))
    {
        res.add(Arrays.asList(i, root.index));
    }

    root = root.children.get(words[i].charAt(j));
      if (root == null) return;
  }

  for (int j : root.palindromes) {
    if (i == j) continue;
    res.add(Arrays.asList(i, j));
  }
}
```

- 最后来看看如何处理查找
- 处理查找时，从头遍历每个字符串
  然后尝试从 `Trie` 中找到匹配的字符串
- 当第二种情况 $k_1 > k_2$，并且 `s1` 剩下的字符能构成回文
  则把这对组合添加到结果中
- 然后不断往下继续查找
- 最后，当遇到第一种和第三种情况时，
  只需把回文列表中的字符串都与 `s1` 组合在一起即可

336. 回文对

复杂度分析

- 利用 Trie，在创建和查找的过程中最多会遇到 $n \times k$ 个节点，而且会进行回文检查整体的时间复杂度为 $O(n \times k \times k)$
- 如果字符串的字符个数在一定范围之内，这个问题可以优化成近乎一个线性问题

340. 至多包含 K 个不同字符的最长子串

给定一个字符串 s，找出至多包含 K 个不同字符的最长子串 T。

示例 1:

输入: s = "eceba", k = 2
输出: 3
解释: 则 T 为 “ece”，所以长度为 3。

示例 2:

输入: s = "aa", k = 1
输出: 2
解释: 则 T 为 “aa”，所以长度为 2。

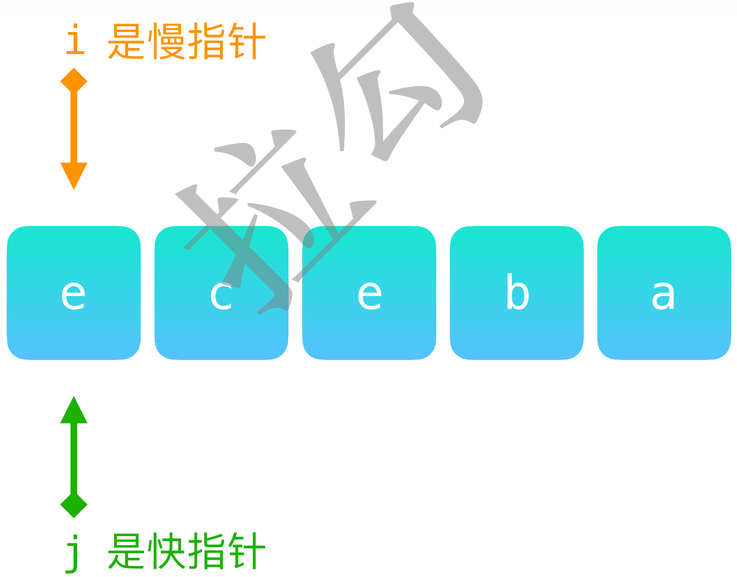
340. 至多包含 K 个不同字符的最长子串

暴力法

- 找出所有的子串，注意检查是否最多包含 k 个不同的字符
- 检查的方法是用一个哈希表或哈希集合去统计
- 算法复杂度为 $O(n^2)$
  - 第 8 课：找出无重复的最长子串 -> 时间复杂度为 $O(n)$

```
s = “eceba”, k = 2
```

340. 至多包含 K 个不同字符的最长子串

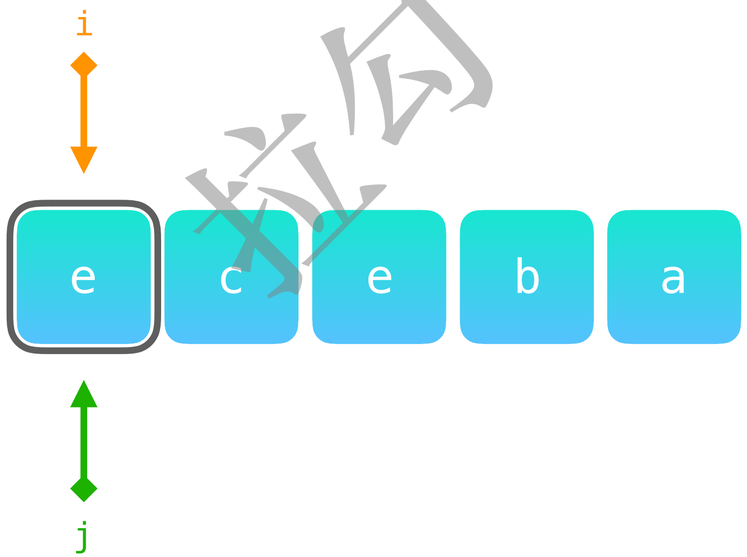


需要一个哈希表来统计每个字符出现的个数，同时也用来统计不同字符的个数

map: {}

(size = 0)

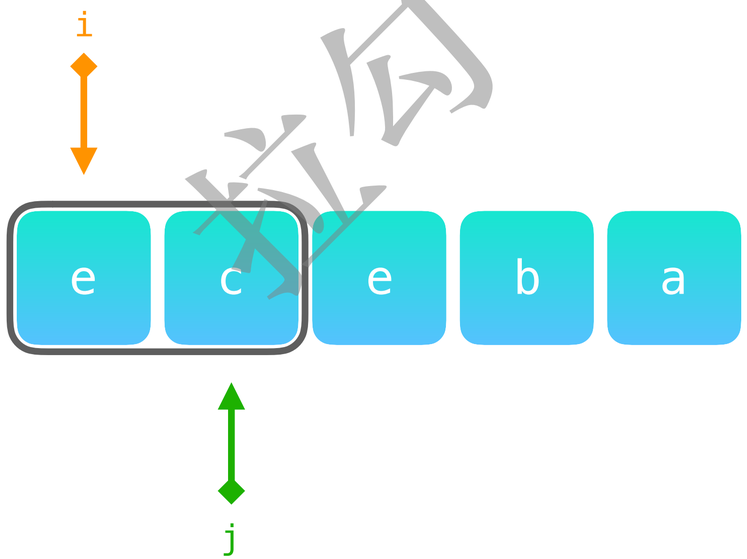
340. 至多包含 K 个不同字符的最长子串



map: {e:1}

(size = 1)

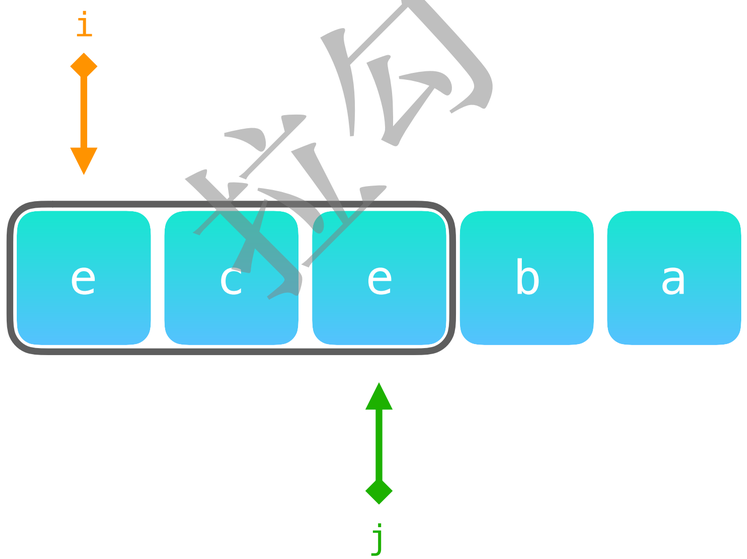
340. 至多包含 K 个不同字符的最长子串



map: {e:1, c:1}

(size = 2)

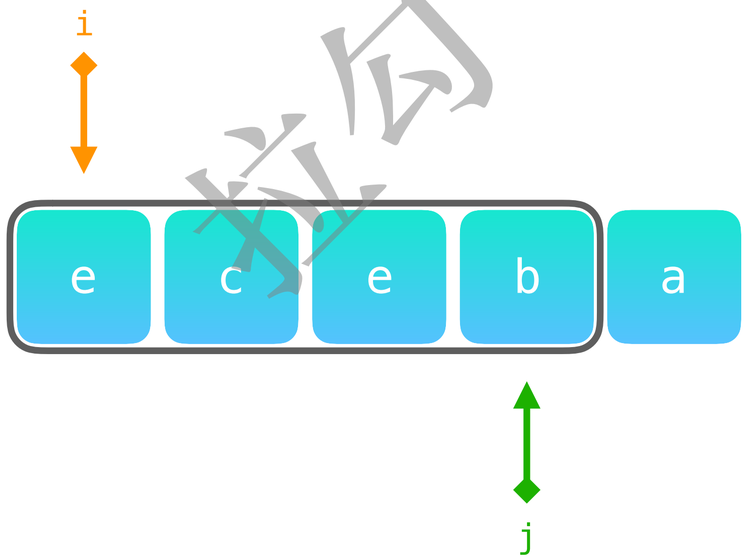
340. 至多包含 K 个不同字符的最长子串



map: {e:2, c:1}

(size = 2)

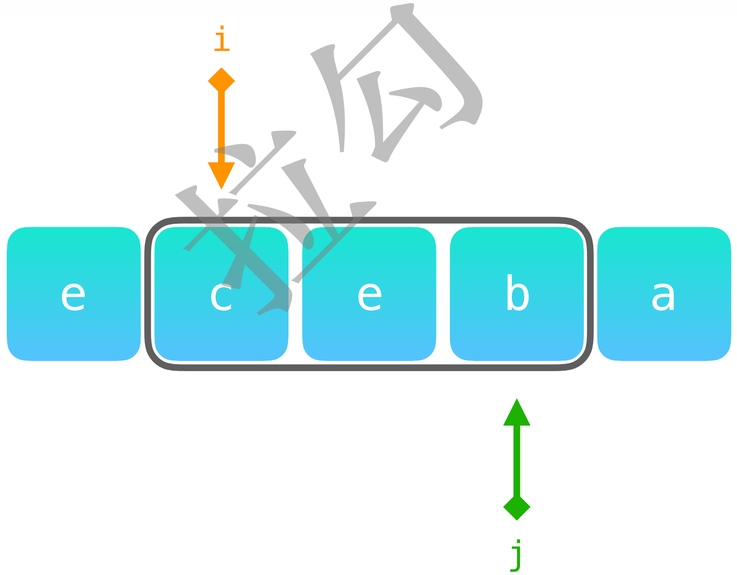
340. 至多包含 K 个不同字符的最长子串



map: {e:2, c:1, b:1}

(size = 3)

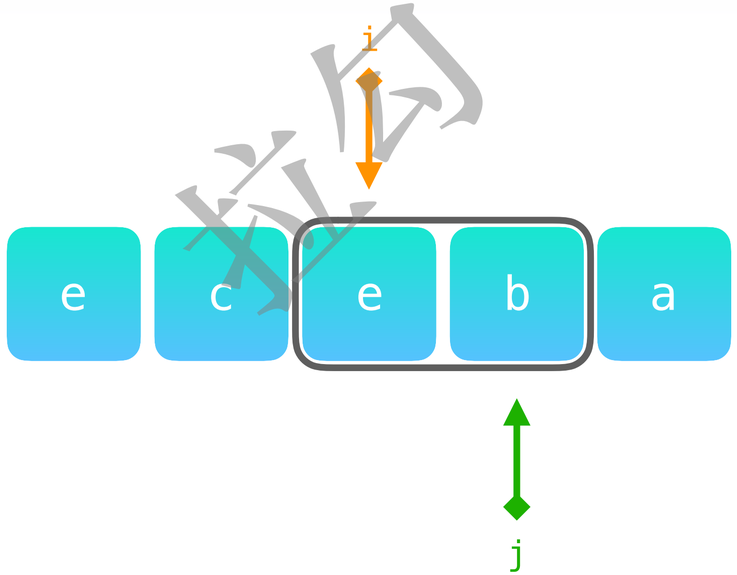
340. 至多包含 K 个不同字符的最长子串



map: {e:1, c:1, b:1}

(size = 3)

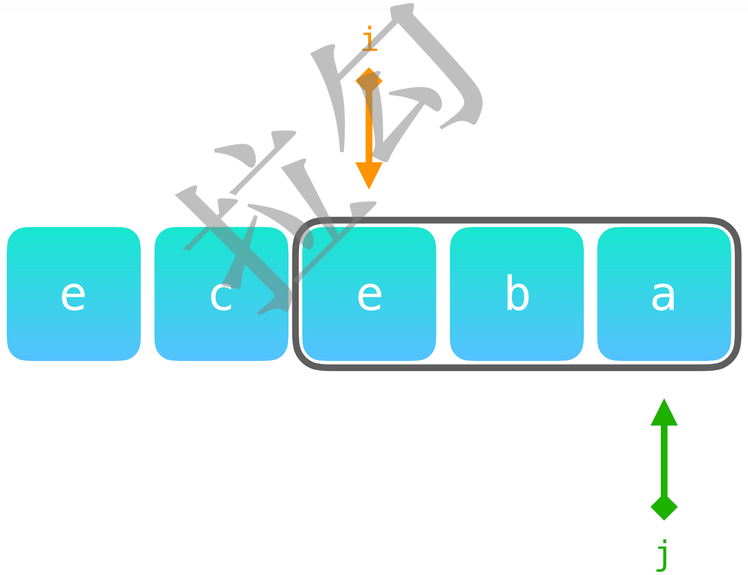
340. 至多包含 K 个不同字符的最长子串



map: {e:1, b:1}

(size = 2)

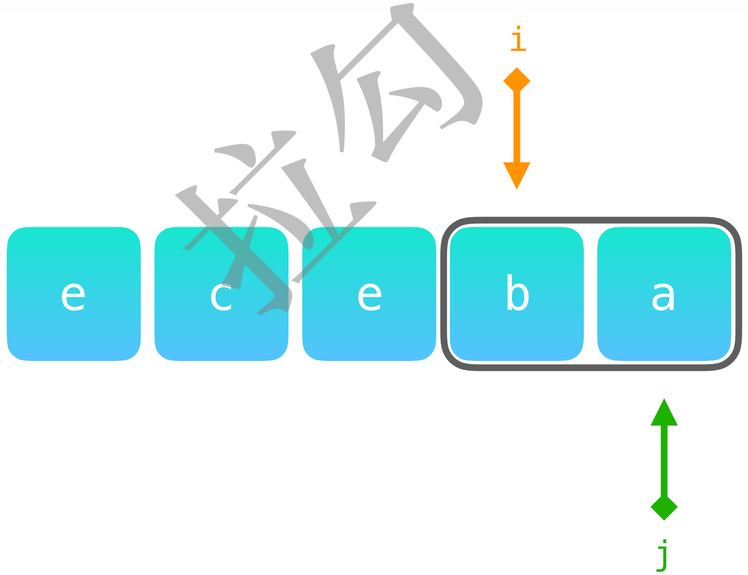
340. 至多包含 K 个不同字符的最长子串



map: {c:1, b:1, a:1}

(size = 3)

340. 至多包含 K 个不同字符的最长子串



map: {b:1, a:1}

(size = 2)

11.2 难题精讲（二） / 至多包含 K 个不同字符的最长子串

力扣 LeetCode × 拉勾 LAGOU.COM

```
int lengthOfLongestSubstringKDistinct(String s, int k) {
  HashMap<Character, Integer> map = new
HashMap<>();
  int max = 0;

  for (int i = 0, j = 0; j < s.length(); j++) {
    char cj = s.charAt(j);

    // Step 1. count the character
    map.put(cj, map.getOrDefault(cj, 0) + 1);
```

- 初始化一个哈希表 map，用来统计出现的不同字符
- 用 max 变量记录最长的子串，其中最多包括 k 个不同字符
- 用快慢指针遍历字符串
- 将快指针指向的字符加入到 map 中，统计字符出现的次数

11.2 难题精讲（二） / 至多包含 K 个不同字符的最长子串

力扣 LeetCode × 拉勾 LAGOU.COM

```
    // Step 2. clean up the map if condition doesn't match
    while (map.size() > k) {
      char ci = s.charAt(i);

      map.put(ci, map.get(ci) - 1);

      if (map.get(ci) == 0) {
        map.remove(ci); // that character count has become 0
      }
      i++;
    }

    // Step 3. condition matched, now update the result
    max = Math.max(max, j - i + 1);
  }

  return max;
}
```

- 如果发现 map 大小超过了 k，
  则开始不断将慢指针指向的字符从 map 中清除掉
- 首先获取当前慢指针指向的字符
- 将它在 map 中的计数减一
- 一旦它的统计次数变成了 0，就可以把它从 map 中删掉了
- 接下来，慢指针继续往前走
- 当 map 的大小恢复正常，统计一下当前子串的长度
- 最后返回最大的子串长度
- **复杂度分析：**
  - 快慢指针遍历字符串一遍，时间复杂度是 $O(n)$
  - 运用了一个 map 来统计，空间复杂度是 $O(n)$

407. 接雨水 II

给定一个 m × n 的矩阵，其中的值均为正整数，代表二维高度图每个单元的高度，请计算图中形状最多能接多少体积的水。

说明：

m 和 n 都是小于 110 的整数。每一个单位的高度都大于 0 且小于 20000。

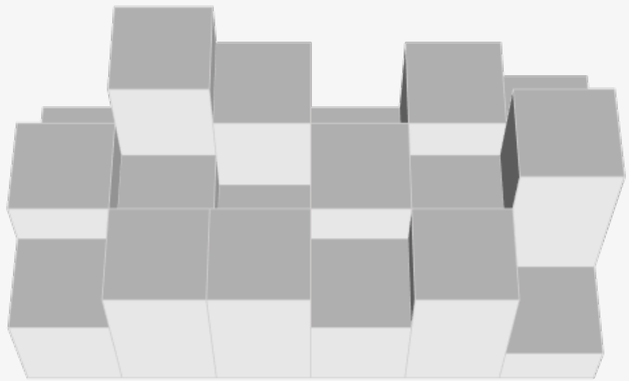
示例：

给出如下 3×6 的高度图：

```
[
    [1, 4, 3, 1, 3, 2],
    [3, 2, 1, 3, 2, 4],
    [2, 3, 3, 2, 3, 1]
]
```

返回：4

这是下雨前的高度图状态。



407. 接雨水 II

给定一个 m × n 的矩阵，其中的值均为正整数，代表而为高度图每个单元的高度，请计算图中形状最多能接多少体积的水。

说明：

m 和 n 都是小于 110 的整数。每一个单位的高度都大于 0 且小于 20000。

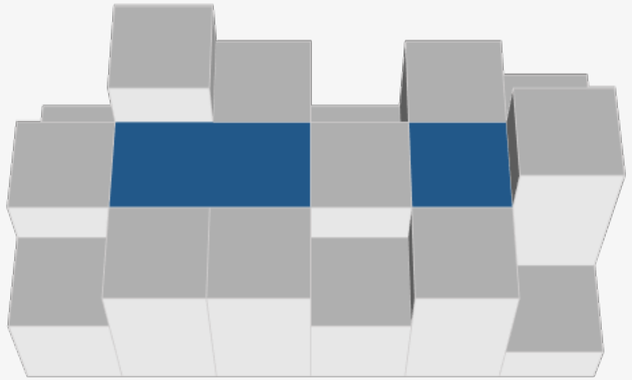
示例：

给出如下 3×6 的高度图：

```
[
    [1, 4, 3, 1, 3, 2],
    [3, 2, 1, 3, 2, 4],
    [2, 3, 3, 2, 3, 1]
]
```

返回：4

下雨后，雨水将会存储在这些方块中。总的接雨量是 4.



407. 接雨水 II

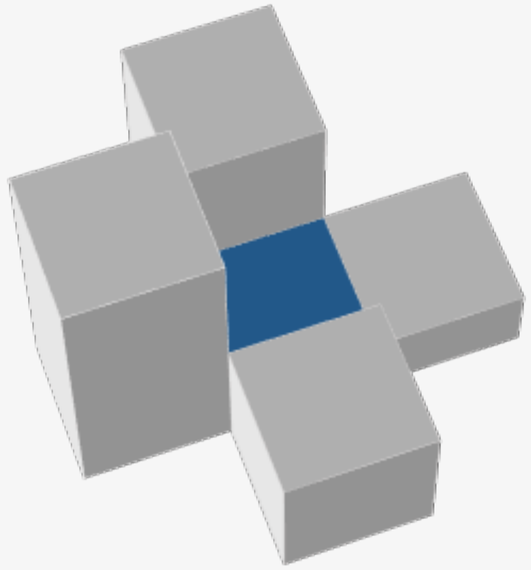
解题思路

假设：

1

2 0 3

4



407. 接雨水 II

解题思路

假设：

		8		
	5	1	6	
4	2	**0**	3	5
	7	4	8	
		9		

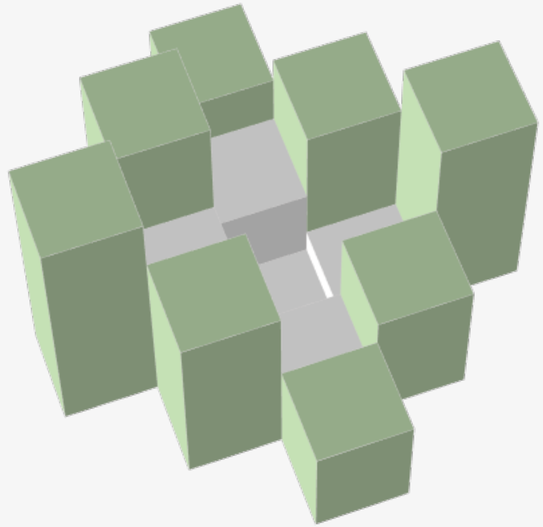
- 时间复杂度：$O(n^3)$

407. 接雨水 II

解题思路

假设：

		8		
	5	1	6	
4	2	0	3	5
	7	4	8	
		9		

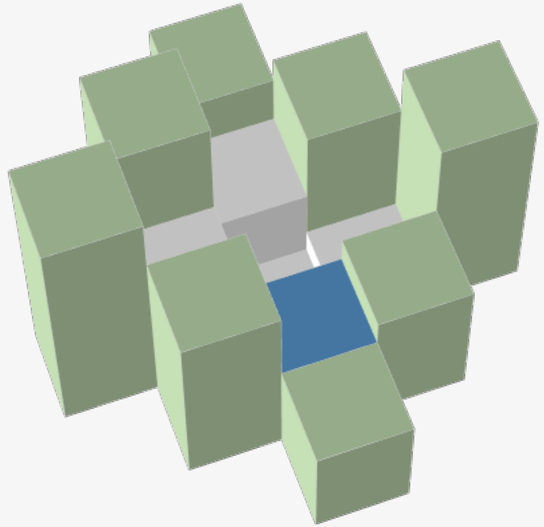


407. 接雨水 II

解题思路

假设：

		8		
	5	1	6	
4	4	0	3	5
	7	4	8	
		9		

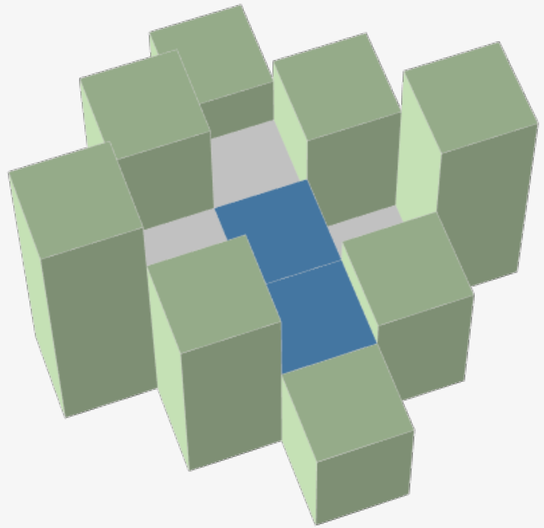


407. 接雨水 II

解题思路

假设：

		8		
	5	1	6	
4	4	4	3	5
	7	4	8	
		9		

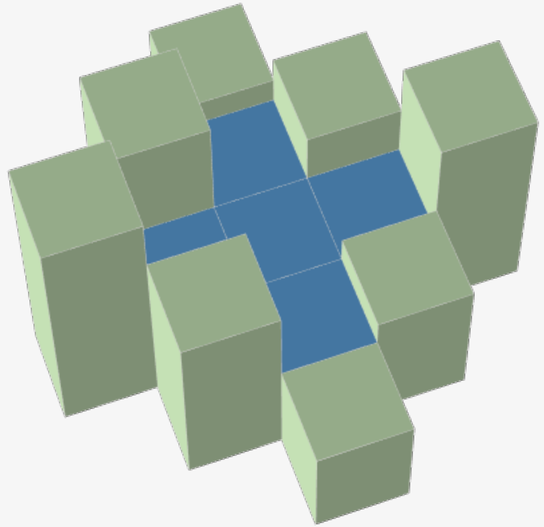


407. 接雨水 II

解题思路

假设：

```
        8
    5   4   6
4   4   4   4   5
    7   4   8
        9
```

11.3 难题精讲（二） / 接雨水 II

力扣 LeetCode × 拉勾 LAGOU.COM

```
class Cell {
  int row;
  int col;
  int height;

  public Cell(int row, int col, int height) {
    this.row = row;
    this.col = col;
    this.height = height;
  }
}
```

- 为了配合优先队列的操作，我们定义一个 Cell 类，用来保存每个方块的坐标以及接了雨水后的高度

11.3 难题精讲（二） / 接雨水 II

```
public int trapRainWater(int[][] heights) {
  // Sanity check
  if (heights == null || heights.length == 0 ||
heights[0].length == 0) {
      return 0;
  }

  int m = heights.length;
  int n = heights[0].length;

  PriorityQueue<Cell> queue = new PriorityQueue(new
Comparator<Cell>() {
    public int compare(Cell a, Cell b) { return a.height -
b.height; }
  });
  boolean[][] visited = new boolean[m][n];
```

- 首先对输入进行一些基本判断
- 用变量 m 和 n 分别表示输入矩阵的行数和列数
- 定义一个优先队列或者最小堆，
  按照每个方块接了雨水后的高度排列

```
// Initially, add all the Cells which are on borders to the queue.
for (int i = 0; i < m; i++) {
  visited[i][0] = true;
  visited[i][n - 1] = true;
  queue.offer(new Cell(i, 0, heights[i][0]));
  queue.offer(new Cell(i, n - 1, heights[i][n - 1]));
}

for (int j = 0; j < n; j++) {
  visited[0][j] = true;
  visited[m - 1][j] = true;
  queue.offer(new Cell(0, j, heights[0][j]));
  queue.offer(new Cell(m - 1, j, heights[m - 1][j]));
}
```

- 初始化优先队列时，
  把矩形的外围四个边上的方块都加入到优先队列中

11.3 难题精讲（二） / 接雨水 II

```
  // From the borders, pick the shortest cell visited and
check its
  // neighbors:
  // If the neighbor is shorter, collect the water it can trap
and update
  // its height as its height plus the water trapped.
  // Add all its neighbors to the queue.
  int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

  int total = 0;

  while (!queue.isEmpty()) {
    Cell cell = queue.poll();

    for (int[] dir : dirs) {
      int row = cell.row + dir[0];
      int col = cell.col + dir[1];
```

- 进入 `while` 循环，开始进行 BFS
- 每次，从优先队列中取出最爱的方块
- 从四个方向扩散

```
        if (row >= 0 && row < m && col >= 0 && col < n && !
visited[row][col])
        {
          visited[row][col] = true;
          total += Math.max(0, cell.height - heights[row][col]);
          queue.offer(
            new Cell(row, col, Math.max(heights[row][col],
cell.height))
          );
        }
      }
    }

    return total;
  }
```

- 该方向上的相邻方块能接多少雨水
  取决于它是否低于当前方块
- 同时，将新方块加入到优先队列中
- 最后返回承接雨水总量

407. 接雨水 II

复杂度分析

- 假设一共有 m 行 n 列，则共有 $m \times n$ 个方块
- 对于每个方块，我们都有可能会进行优先队列的操作
- 而优先队列的大小为 $m + n$，加上初始化优先队列的操作时间，整体时间复杂度为：

$$O(m+n) + O(m \times n \times log(m+n)) = O(m \times n \times log(m+n))$$

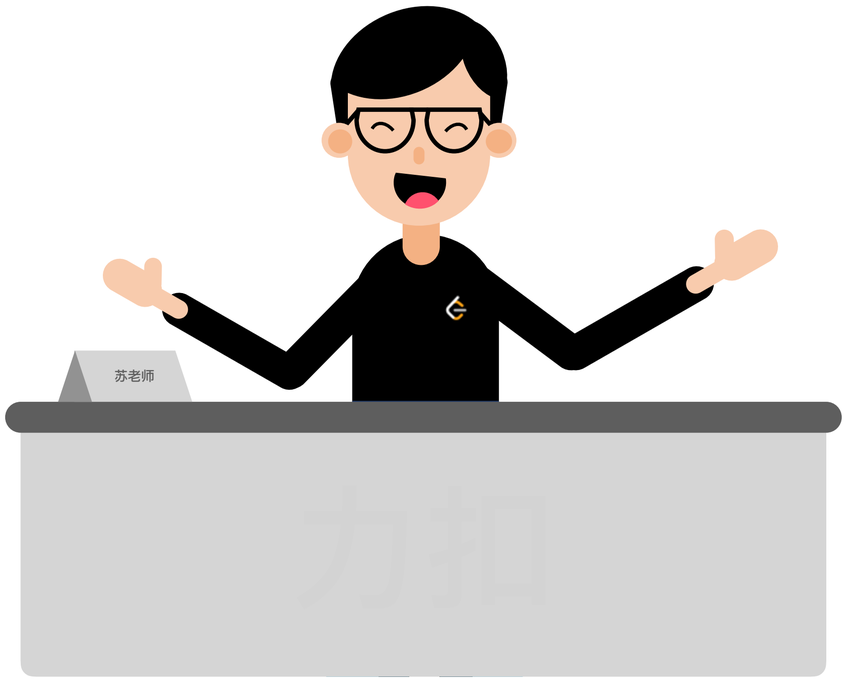
- 复杂度降低了一个维度
- BFS 中运用“农村包围城市”策略：「力扣 417」太平洋大西洋水流问题

11.4 结束 / Ending

力扣 LeetCode × 拉勾 LAGOU.COM

难题精讲（二)

- 回文对
- 至多包含 K 个不同字符的最长子串
- 接雨水 II



力扣 LeetCode × 拉勾 LAGOU.COM

Next：课时 12《冲刺》

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