王争的算法训练营

习题课：堆和Trie树

堆、Trie树相关题型/套路

堆相关题目有限，并非重点，经典题目不多。

1、优先级队列

2、TOP K

3、求中位数、百分位数

习题课：堆、Trie树

王争的算法训练营

配套习题：

23. 合并K个升序链表(困难) **（已讲）**

347. 前 K 个高频元素（中等） **（已讲）**

295. 数据流的中位数（困难） **（已讲）**

973. 最接近原点的 K 个点（中等）

313. 超级丑数（中等）

208. 实现 Trie (前缀树)（中等） 标准Trie树

以下为选做题目：

面试题 17.17. 多次搜索（中等） 标准AC自动机，不过写AC自动机太复杂，Trie树搞定

212. 单词搜索 II（困难）

习题课：堆和Trie

王争的算法训练营

23. 合并K个升序链表(困难) （已讲）

给你一个链表数组，每个链表都已经按升序排列。

请你将所有链表合并到一个升序链表中，返回合并后的链表。

示例 1：

```
输入：lists = [[1,4,5],[1,3,4],[2,6]]
输出：[1,1,2,3,4,4,5,6]
解释：链表数组如下：
[
  1->4->5,
  1->3->4,
  2->6
]
将它们合并到一个有序链表中得到。
1->1->2->3->4->4->5->6
```

```java
class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        if (lists == null || lists.length == 0) return null;

        int k = lists.length;
        PriorityQueue<ListNode> minQ = new PriorityQueue<>(new Comparator<ListNode>() {
            @Override
            public int compare(ListNode q1, ListNode q2) {
                return q1.val - q2.val;
            }
        });
        for (int i = 0; i < k; ++i) {
            if (lists[i] != null) {
                minQ.offer(lists[i]);
            }
        }
        ListNode head = new ListNode();
        ListNode tail = head;
        while (!minQ.isEmpty()) {
            ListNode curNode = minQ.poll();
            tail.next = curNode;
            tail = tail.next;
            if (curNode.next != null) {
                minQ.offer(curNode.next);
            }
        }

        return head.next;
    }
}
```

习题课：堆和Trie

王争的算法训练营

347. 前 K 个高频元素（中等） （已讲）

给你一个整数数组 `nums` 和一个整数 `k` ，请你返回其中出现频率前 `k` 高的元素。你可以按 **任意顺序** 返回答案。

示例 1:

```
输入: nums = [1,1,1,2,2,3], k = 2
输出: [1,2]
```

示例 2:

```
输入: nums = [1], k = 1
输出: [1]
```

提示:

- `1 <= nums.length <= 10^5`
- `k` 的取值范围是 `[1, 数组中不相同的元素的个数]`
- 题目数据保证答案唯一，换句话说，数组中前 `k` 个高频元素的集合是唯一的

进阶: 你所设计算法的时间复杂度 **必须** 优于 `O(n log n)` ，其中 `n` 是数组大小。

```java
class Solution {
    private class QElement {
        int val;
        int count;
        public QElement(int val, int count) {
            this.val = val;
            this.count = count;
        }
    }

    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> counts = new HashMap<>();
        for (int num : nums) {
            counts.put(num, counts.getOrDefault(num, 0) + 1);
        }

        PriorityQueue<QElement> queue = new PriorityQueue<>(new Comparator<QElement>() {
            public int compare(QElement e1, QElement e2) {
                return e1.count - e2.count;
            }
        });

        for (Map.Entry<Integer, Integer> entry : counts.entrySet()) {
            int num = entry.getKey();
            int count = entry.getValue();
            if (queue.size() < k)  {
                queue.offer(new QElement(num, count));
            } else {
                if (queue.peek().count < count) {
                    queue.poll();
                    queue.offer(new QElement(num, count));
                }
            }
        }
        int[] result = new int[k];
        for (int i = 0; i < k; ++i) {
            result[i] = queue.poll().val;
        }
        return result;
    }
}
```

习题课：堆和Trie

王争的算法训练营

295. 数据流的中位数（困难） **（已讲）**

中位数是有序列表中间的数。如果列表长度是偶数，中位数则是中间两个数的平均值。

例如，

[2,3,4] 的中位数是 3

[2,3] 的中位数是 (2 + 3) / 2 = 2.5

设计一个支持以下两种操作的数据结构：

- void addNum(int num) - 从数据流中添加一个整数到数据结构中。
- double findMedian() - 返回目前所有元素的中位数。

示例：

```
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
```

进阶：

1. 如果数据流中所有整数都在 0 到 100 范围内，你将如何优化你的算法?
2. 如果数据流中 99% 的整数都在 0 到 100 范围内，你将如何优化你的算法?

```java
class MedianFinder {
    private PriorityQueue<Integer> minQueue = new PriorityQueue(new Comparator<Integer>() {
        @Override
        public int compare(Integer o1, Integer o2) {
            return o1 - o2;
        }
    });
    private PriorityQueue<Integer> maxQueue = new PriorityQueue(new Comparator<Integer>() {
        @Override
        public int compare(Integer o1, Integer o2) {
            return o2 - o1;
        }
    });

    public MedianFinder() {}

    public void addNum(int num) {
        if (maxQueue.isEmpty() || num <= maxQueue.peek()) {
            maxQueue.add(num);
        } else {
            minQueue.add(num);
        }
        while (maxQueue.size() < minQueue.size()) {
            Integer e = minQueue.poll();
            maxQueue.add(e);
        }
        while (minQueue.size() < maxQueue.size()-1) {
            Integer e = maxQueue.poll();
            minQueue.add(e);
        }
    }

    public double findMedian() {
        if (maxQueue.size() > minQueue.size()) {
            return maxQueue.peek();
        } else {
            return (maxQueue.peek()+minQueue.peek())/2f;
        }
    }
}
```

习题课：堆和Trie

王争的算法训练营

973. 最接近原点的 K 个点（中等）

我们有一个由平面上的点组成的列表 `points`。需要从中找出 `K` 个距离原点 `(0, 0)` 最近的点。

（这里，平面上两点之间的距离是欧几里德距离。）

你可以按任何顺序返回答案。除了点坐标的顺序之外，答案确保是唯一的。

示例 1：

```
输入：points = [[1,3],[-2,2]], K = 1
输出：[[-2,2]]
解释：
(1, 3) 和原点之间的距离为 sqrt(10)，
(-2, 2) 和原点之间的距离为 sqrt(8)，
由于 sqrt(8) < sqrt(10)，(-2, 2) 离原点更近。
我们只需要距离原点最近的 K = 1 个点，所以答案就是 [[-2,2]]。
```

示例 2：

```
输入：points = [[3,3],[5,-1],[-2,4]], K = 2
输出：[[3,3],[-2,4]]
（答案 [[-2,4],[3,3]] 也会被接受。）
```

习题课：堆和Trie

王争的算法训练营

973. 最接近原点的 K 个点（中等）

```
class Solution {
    public int[][] kClosest(int[][] points, int K) {
        PriorityQueue<int[]> pq = new PriorityQueue<int[]>(new Comparator<int[]>() {
            public int compare(int[] array1, int[] array2) {
                return array2[0] - array1[0];
            }
        });
        for (int i = 0; i < K; ++i) {
            pq.offer(new int[]{points[i][0] * points[i][0] + points[i][1] * points[i][1], i});
        }
        int n = points.length;
        for (int i = K; i < n; ++i) {
            int dist = points[i][0] * points[i][0] + points[i][1] * points[i][1];
            if (dist < pq.peek()[0]) {
                pq.poll();
                pq.offer(new int[]{dist, i});
            }
        }
        int[][] ans = new int[K][2];
        for (int i = 0; i < K; ++i) {
            ans[i] = points[pq.poll()[1]];
        }
        return ans;
    }
}
```

习题课：堆和Trie

王争的算法训练营

313. 超级丑数（中等）

编写一段程序来查找第 `n` 个超级丑数。

超级丑数是指其所有质因数都是长度为 `k` 的质数列表 `primes` 中的正整数。

示例:

```
输入: n = 12, primes = [2,7,13,19]
输出: 32
解释: 给定长度为 4 的质数列表 primes = [2,7,13,19]，前 12 个超级丑数序列为:
[1,2,4,7,8,13,14,16,19,26,28,32] 。
```

说明:

- `1` 是任何给定 `primes` 的超级丑数。
- 给定 `primes` 中的数字以升序排列。
- $0 < k \le 100$, $0 < n \le 10^6$, $0 < primes[i] < 1000$ 。
- 第 `n` 个超级丑数确保在 32 位有符整数范围内。

```java
class Solution {
    public int nthSuperUglyNumber(int n, int[] primes) {
        // 优先级队列（小顶堆）
        PriorityQueue<Long> pq = new PriorityQueue<>(
            new Comparator<Long>() {
                public int compare(Long o1, Long o2) {
                    if (o1 < o2) return -1;
                    else if (o1 == o2) return 0;
                    else return 1;
                }
        });
        HashSet<Long> uniset = new HashSet<>();

        pq.add(1L);
        uniset.add(1L);

        int count = 0;
        long data = 1;
        while (!pq.isEmpty() && count < n) {
            data = pq.poll();
            count++;
            for (int i = 0; i < primes.length; ++i) {
                if (!uniset.contains(data*primes[i])) {
                    pq.add(data*primes[i]);
                    uniset.add(data*primes[i]);
                }
            }
        }
        return (int)data;
    }
}
```

82 / 85 个通过测试用例

状态：**超出时间限制**

提交时间：**3 分钟前**

最后执行的输入：

1000000
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,521,523,541]

习题课：堆和Trie

王争的算法训练营

208. 实现 Trie (前缀树) （中等） 标准Trie树

Trie（发音类似 "try"）或者说 **前缀树** 是一种树形数据结构，用于高效地存储和检索字符串数据集中的键。这一数据结构有相当多的应用情景，例如自动补完和拼写检查。

请你实现 Trie 类：

- `Trie()` 初始化前缀树对象。
- `void insert(String word)` 向前缀树中插入字符串 `word` 。
- `boolean search(String word)` 如果字符串 `word` 在前缀树中，返回 `true` （即，在检索之前已经插入）；否则，返回 `false` 。
- `boolean startsWith(String prefix)` 如果之前已经插入的字符串 `word` 的前缀之一为 `prefix` ，返回 `true` ；否则，返回 `false` 。

示例：

```
输入
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
输出
[null, null, true, false, true, null, true]

解释
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // 返回 True
trie.search("app");     // 返回 False
trie.startsWith("app"); // 返回 True
trie.insert("app");
trie.search("app");     // 返回 True
```

```java
class Trie {
    private class TrieNode {
        char data;
        boolean isEnding = false;
        TrieNode[] children = new TrieNode[26];
        public TrieNode(char data) {
            this.data = data;
        }
    }

    private TrieNode root;
    /** Initialize your data structure here. */
    public Trie() {
        root = new TrieNode('/');
    }
}
```

```java
/** Inserts a word into the trie. */
public void insert(String word) {
    TrieNode p = root;
    for (int i = 0; i < word.length(); i++) {
        char c = word.charAt(i);
        if (p.children[c-'a'] == null) {
            p.children[c-'a'] = new TrieNode(c);
        }
        p = p.children[c-'a'];
    }
    p.isEnding = true;
}

/** Returns if the word is in the trie. */
public boolean search(String word) {
    TrieNode p = root;
    for (int i = 0; i < word.length(); ++i) {
        char c = word.charAt(i);
        if (p.children[c-'a'] == null) return false;
        p = p.children[c-'a'];
    }
    return p.isEnding;
}

public boolean startsWith(String prefix) {
    TrieNode p = root;
    for (int i = 0; i < prefix.length(); ++i) {
        char c = prefix.charAt(i);
        if (p.children[c-'a'] == null) return false;
        p = p.children[c-'a'];
    }
    return true;
}
```

```java
class Trie {
    private class TrieNode {
        char data;
        boolean isEnding = false;
        Map<Character, TrieNode> children = new HashMap<>();
        public TrieNode(char data) {
            this.data = data;
        }
    }

    private TrieNode root;
    /** Initialize your data structure here. */
    public Trie() {
        root = new TrieNode('/');
    }

    /** Inserts a word into the trie. */
    public void insert(String word) {
        TrieNode p = root;
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (!p.children.containsKey(c)) {
                p.children.put(c, new TrieNode(c));
            }
            p = p.children.get(c);
        }
        p.isEnding = true;
    }

    /** Returns if the word is in the trie. */
    public boolean search(String word) {
        TrieNode p = root;
        for (int i = 0; i < word.length(); ++i) {
            char c = word.charAt(i);
            if (!p.children.containsKey(c)) {
                return false;
            }
            p = p.children.get(c);
        }
        return p.isEnding;
    }

    public boolean startsWith(String prefix) {
        TrieNode p = root;
        for (int i = 0; i < prefix.length(); ++i) {
            char c = prefix.charAt(i);
            if (!p.children.containsKey(c)) {
                return false;
            }
            p = p.children.get(c);
        }
        return true;
    }
}
```

提问环节

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料