宫水三叶的刷题日记

多路归并

Author ：宫水三叶
Date ：2021/10/07
QQ Group: 703311589
WeChat ：oaoaya

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

噔噔噔噔，这是公众号「宫水三叶的刷题日记」的原创专题「多路归并」合集。

本合集更新时间为 2021-10-07，大概每 2-4 周会集中更新一次。关注公众号，后台回复「多路归并」即可获取最新下载链接。

💡下面介绍使用本合集的最佳使用实践：

学习算法：

1. 打开在线目录（Github 版 & Gitee 版）；
2. 从侧边栏的类别目录找到「多路归并」；
3. 按照「推荐指数」从大到小进行刷题，「推荐指数」相同，则按照「难度」从易到难进行刷题‘
4. 拿到题号之后，回到本合集进行检索。

维持熟练度：

1. 按照本合集「从上往下」进行刷题。

学习过程中遇到任何困难，欢迎加入「每日一题打卡 QQ 群：703311589」进行交流 🍭🍭🍭

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

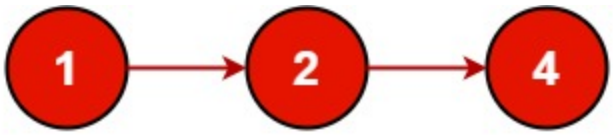
这是 LeetCode 上的 **21. 合并两个有序链表** ，难度为 **简单**。

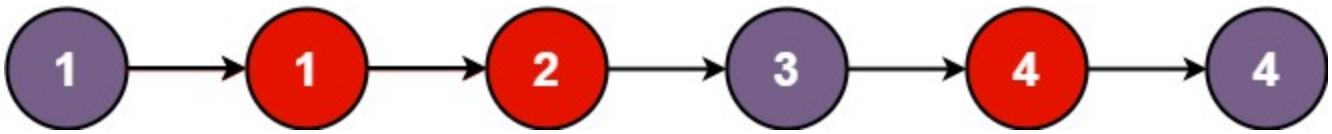
Tag : 「多路归并」、「链表」

将两个升序链表合并为一个新的 升序 链表并返回。

新链表是通过拼接给定的两个链表的所有节点组成的。

示例 1：






```
输入：l1 = [1,2,4], l2 = [1,3,4]

输出：[1,1,2,3,4,4]
```

示例 2：

```
输入：l1 = [], l2 = []

输出：[]
```

示例 3：

```
输入：l1 = [], l2 = [0]

输出：[0]
```

提示：

- 两个链表的节点数目范围是 [0, 50]
- -100 <= Node.val <= 100
- l1 和 l2 均按 非递减顺序 排列

多路归并（哨兵技巧）

哨兵技巧我们之前在「2. 两数相加」讲过啦，让三叶来帮你回忆一下：

做有关链表的题目，有个常用技巧：添加一个虚拟头结点（哨兵），帮助简化边界情况的判断。

由于两条链表本身就是有序的，只需要在遍历过程中进行比较即可：

代码：

```
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;

        ListNode dummy = new ListNode(0);
        ListNode cur = dummy;
        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                cur.next = l1;
                cur = cur.next;
                l1 = l1.next;
            } else {
                cur.next = l2;
                cur = cur.next;
                l2 = l2.next;
            }
        }

        while (l1 != null) {
            cur.next = l1;
            cur = cur.next;
            l1 = l1.next;
        }
        while (l2 != null) {
            cur.next = l2;
            cur = cur.next;
            l2 = l2.next;
        }

        return dummy.next;
    }
}
```

- 时间复杂度：对两条链表扫描一遍。复杂度为 $O(n)$

- 空间复杂度：$O(1)$

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **264. 丑数 II** ，难度为 **中等**。

Tag：「多路归并」、「堆」、「优先队列」

给你一个整数 n ，请你找出并返回第 n 个 丑数 。

丑数 就是只包含质因数 2、3 和 5 的正整数。

示例 1：

```
输入：n = 10

输出：12

解释：[1, 2, 3, 4, 5, 6, 8, 9, 10, 12] 是由前 10 个丑数组成的序列。
```

示例 2：

```
输入：n = 1

输出：1

解释：1 通常被视为丑数。
```

提示：

- 1 <= n <= 1690

基本思路

根据丑数的定义，我们有如下结论：

- 1 是最小的丑数。

- 对于任意一个丑数 x，其与任意的质因数（2、3、5）相乘，结果（$2x$、$3x$、$5x$）仍为丑数。

优先队列（小根堆）解法

有了基本的分析思路，一个简单的解法是使用优先队列：

1. 起始先将最小丑数 1 放入队列
2. 每次从队列取出最小值 x，然后将 x 所对应的丑数 $2x$、$3x$ 和 $5x$ 进行入队。
3. 对步骤 2 循环多次，第 n 次出队的值即是答案。

为了防止同一丑数多次进队，我们需要使用数据结构 Set 来记录入过队列的丑数。

代码：

```
class Solution {
    int[] nums = new int[]{2,3,5};
    public int nthUglyNumber(int n) {
        Set<Long> set = new HashSet<>();
        Queue<Long> pq = new PriorityQueue<>();
        set.add(1L);
        pq.add(1L);
        for (int i = 1; i <= n; i++) {
            long x = pq.poll();
            if (i == n) return (int)x;
            for (int num : nums) {
                long t = num * x;
                if (!set.contains(t)) {
                    set.add(t);
                    pq.add(t);
                }
            }
        }
        return -1;
    }
}
```

- 时间复杂度：从优先队列中取最小值为 $O(1)$，往优先队列中添加元素复杂度为 $O(\log n)$。整体复杂度为 $O(n \log n)$
- 空间复杂度：$O(n)$

多路归并（多指针）解法

从解法一中不难发现，我们「往后产生的丑数」都是基于「已有丑数」而来（使用「已有丑数」乘上「质因数」2、3、5）。

因此，如果我们所有丑数的有序序列为 $a1, a2, a3, ..., an$ 的话，序列中的每一个数都必然能够被以下三个序列（中的至少一个）覆盖：

- 由丑数 * 2 所得的有序序列：$1 * 2$、$2 * 2$、$3 * 2$、$4 * 2$、$5 * 2$、$6 * 2$、$8 * 2$ ...
- 由丑数 * 3 所得的有序序列：$1 * 3$、$2 * 3$、$3 * 3$、$4 * 3$、$5 * 3$、$6 * 3$、$8 * 3$ ...
- 由丑数 * 5 所得的有序序列：$1 * 5$、$2 * 5$、$3 * 5$、$4 * 5$、$5 * 5$、$6 * 5$、$8 * 5$ ...

举个🌰，假设我们需要求得 $[1, 2, 3, ..., 10, 12]$ 丑数序列 arr 的最后一位，那么该序列可以看作以下三个有序序列归并而来：

- $1 * 2, 2 * 2, 3 * 2, ..., 10 * 2, 12 * 2$ ，将 2 提出，即 $arr * 2$
- $1 * 3, 2 * 3, 3 * 3, ..., 10 * 3, 12 * 3$ ，将 3 提出，即 $arr * 3$
- $1 * 5, 2 * 5, 3 * 5, ..., 10 * 5, 12 * 5$ ，将 5 提出，即 $arr * 5$

因此我们可以使用三个指针来指向目标序列 arr 的某个下标（下标 0 作为哨兵不使用，起始都为 1），使用 $arr[下标] * 质因数$ 代表当前使用到三个有序序列中的哪一位，同时使用 idx 表示当前生成到 arr 哪一位丑数。

代码：

```java
class Solution {
    public int nthUglyNumber(int n) {
        // ans 用作存储已有丑数（从下标 1 开始存储，第一个丑数为 1）
        int[] ans = new int[n + 1];
        ans[1] = 1;
        // 由于三个有序序列都是由「已有丑数」*「质因数」而来
        // i2、i3 和 i5 分别代表三个有序序列当前使用到哪一位「已有丑数」下标（起始都指向 1）
        for (int i2 = 1, i3 = 1, i5 = 1, idx = 2; idx <= n; idx++) {
            // 由 ans[iX] * X 可得当前有序序列指向哪一位
            int a = ans[i2] * 2, b = ans[i3] * 3, c = ans[i5] * 5;
            // 将三个有序序列中的最小一位存入「已有丑数」序列，并将其下标后移
            int min = Math.min(a, Math.min(b, c));
            // 由于可能不同有序序列之间产生相同丑数，因此只要一样的丑数就跳过（不能使用 else if ）
            if (min == a) i2++;
            if (min == b) i3++;
            if (min == c) i5++;
            ans[idx] = min;
        }
        return ans[n];
    }
}
```

- 时间复杂度：$O(n)$
- 空间复杂度：$O(n)$

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **313. 超级丑数** ，难度为 **中等**。

Tag :「优先队列」、「多路归并」

超级丑数 是一个正整数，并满足其所有质因数都出现在质数数组 primes 中。

给你一个整数 n 和一个整数数组 primes ，返回第 n 个 超级丑数 。

题目数据保证第 n 个 超级丑数 在 32-bit 带符号整数范围内。

示例 1：

```
输入：n = 12, primes = [2,7,13,19]

输出：32

解释：给定长度为 4 的质数数组 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] 。
```

示例 2：

```
输入：n = 1, primes = [2,3,5]

输出：1

解释：1 不含质因数，因此它的所有质因数都在质数数组 primes = [2,3,5] 中。
```

提示：

- 1 <= n <= 10^6
- 1 <= primes.length <= 100
- 2 <= primes[i] <= 1000
- 题目数据 保证 primes[i] 是一个质数
- primes 中的所有值都 互不相同 ，且按 递增顺序 排列

基本分析

类似的题目在之前的每日一题也出现过。

本题做法与 264. 丑数 II 类似，相关题解在 这里。

回到本题，根据丑数的定义，我们有如下结论：

- 1 是最小的丑数。
- 对于任意一个丑数 x，其与任意给定的质因数 $primes[i]$ 相乘，结果仍为丑数。

优先队列（堆）

有了基本的分析思路，一个简单的解法是使用优先队列：

1. 起始先将最小丑数 1 放入队列
2. 每次从队列取出最小值 x，然后将 x 所对应的丑数 $x * primes[i]$ 进行入队。
3. 对步骤 2 循环多次，第 n 次出队的值即是答案。

为了防止同一丑数多次进队，我们需要使用数据结构 Set 来记录入过队列的丑数。

代码：

```
class Solution {
    public int nthSuperUglyNumber(int n, int[] primes) {
        Set<Long> set = new HashSet<>();
        PriorityQueue<Long> q = new PriorityQueue<>();
        q.add(1L);
        set.add(1L);
        while (n-- > 0) {
            long x = q.poll();
            if (n == 0) return (int)x;
            for (int k : primes) {
                if (!set.contains(k * x)) {
                    set.add(k * x);
                    q.add(k * x);
                }
            }
        }
        return -1; // never
    }
}
```

- 时间复杂度：令 $primes$ 长度为 m，需要从优先队列（堆）中弹出 n 个元素，每次弹出最多需要放入 m 个元素，堆中最多有 $n * m$ 个元素。复杂度为 $O(n * m \log (n * m))$
- 空间复杂度：$O(n * m)$

多路归并

从解法一中不难发现，我们「往后产生的丑数」都是基于「已有丑数」而来（使用「已有丑数」乘上「给定质因数」$primes[i]$）。

因此，如果我们所有丑数的有序序列为 $a1, a2, a3, ..., an$ 的话，序列中的每一个数都必然能够被以下三个序列（中的至少一个）覆盖（这里假设 $primes = [2, 3, 5]$）：

- 由丑数 * 2 所得的有序序列：$1 * 2$、$2 * 2$、$3 * 2$、$4 * 2$、$5 * 2$、$6 * 2$、$8 * 2$ ...
- 由丑数 * 3 所得的有序序列：$1 * 3$、$2 * 3$、$3 * 3$、$4 * 3$、$5 * 3$、$6 * 3$、$8 * 3$ ...
- 由丑数 * 5 所得的有序序列：$1 * 5$、$2 * 5$、$3 * 5$、$4 * 5$、$5 * 5$、$6 * 5$、$8 * 5$ ...

我们令这些有序序列为 arr，最终的丑数序列为 ans。

如果 $primes$ 的长度为 m 的话，我们可以使用 m 个指针来指向这 m 个有序序列 arr 的当前下标。

显然，我们需要每次取 m 个指针中值最小的一个，然后让指针后移（即将当前序列的下一个值放入堆中），不断重复这个过程，直到我们找到第 n 个丑数。

当然，实现上，我们并不需要构造出这 m 个有序序列。

我们可以构造一个存储三元组的小根堆，三元组信息为 (val, i, idx)：

- val ：为当前列表指针指向具体值；
- i ：代表这是由 $primes[i]$ 构造出来的有序序列；
- idx：代表丑数下标，存在关系 $val = ans[idx] * primes[i]$。

起始时，我们将所有的 $(primes[i], i, 0)$ 加入优先队列（堆）中，每次从堆中取出最小元素，那么下一个该放入的元素为 $(ans[idx + 1] * primes[i], i, idx + 1)$。

另外，由于我们每个 arr 的指针移动和 ans 的构造，都是单调递增，因此我们可以通过与当前最后一位构造的 $ans[x]$ 进行比较来实现去重，而无须引用常数较大的 `Set` 结构。

代码：

```java
class Solution {
    public int nthSuperUglyNumber(int n, int[] primes) {
        int m = primes.length;
        PriorityQueue<int[]> q = new PriorityQueue<>((a,b)->a[0]-b[0]);
        for (int i = 0; i < m; i++) {
            q.add(new int[]{primes[i], i, 0});
        }
        int[] ans = new int[n];
        ans[0] = 1;
        for (int j = 1; j < n; ) {
            int[] poll = q.poll();
            int val = poll[0], i = poll[1], idx = poll[2];
            if (val != ans[j - 1]) ans[j++] = val;
            q.add(new int[]{ans[idx + 1] * primes[i], i, idx + 1});
        }
        return ans[n - 1];
    }
}
```

- 时间复杂度：需要构造长度为 n 的答案，每次构造需要往堆中取出和放入元素，堆中有 m 个元素，起始时，需要对 $primes$ 进行遍历，复杂度为 $O(m)$。整体复杂度为 $O(\max(m, n\log m))$
- 空间复杂度：存储 n 个答案，堆中有 m 个元素，复杂度为 $O(n+m)$

💡更新 Tips：本专题更新时间为 2021-10-07，大概每 2-4 周 集中更新一次。

最新专题合集资料下载，可关注公众号「宫水三叶的刷题日记」，回台回复「多路归并」获取下载链接。

觉得专题不错，可以请作者吃糖 🍭🍭🍭 ：

"给作者手机充个电"

YOLO 的赞赏码