拉勾教育
—互联网人实战大学—

队列：最近的请求次数

简单/数组+双指针、队列

学习目标

拉勾教育
—互联网人实战大学—

- 了解算法题的解题思路
- 数组 + 双指针
- 队列的特点

题目描述

最近的请求次数：在 RecentCounter 类中有一个方法：ping(int t)，t 代表某个时间（毫秒），返回从 3000 毫秒前（时间范围：[t - 3000, t] ）到现在的 ping 数

- 保证每次对 ping 的调用都使用比之前更大的 t 值，1 <= t <= 10^9
- 每个测试用例会使用严格递增的 t 值来调用ping，最多调用 10000 次 ping

输入：inputs = [[1],[100],[3001],[3002]]
输出：[1,2,3,3]

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

1. 在[1-10^9]时间范围内的任意一个（整数）时间点都有可能发生请求
2. 下一次请求时间一定大于上一次请求时间，可以看作一个递增数列
3. 每次请求都会执行 ping(int t)方法，该方法返回 t 时间点及此前3000毫秒内的请求总次数（包含 t 时间点和 t-3000 时间点）

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

1. 题目主要含义

把每次请求的时间点看成一个递增数列，求指定元素到此前某个元素之间的总个数

2. 附加信息

请求时间取值范围在 int 类型内

测试用例最多调用10000次，即最多产生10000个时间点

二. Choose 数据结构及算法思维选择

方案一：遍历数组（暴力解法）

- 把请求时间点看成是递增数组
  - 数据结构：数组
- 每次请求统计 t 时间点到此前3000毫秒之间到请求次数
  - 算法思维：遍历

三. Code 基本解法及编码实现

解法一：暴力解法思路分析

1. 创建数组，存放所有的请求

   整型数组，存放10000个元素

2. 把当前请求存入数组

   记录最后一次存入的索引，从0开始

3. 统计距离此次请求前3000毫秒之间的请求次数

   从最后一次存放位置倒序遍历

```
// 1. 创建数组，存放所有的请求
int[] array = new int[10000];

public int ping(int t) {
    // 2.把当前请求存入数组
    int end = 0;

    // 3.统计前3000毫秒之间的请求次数
    int count = 0;
    while () {
        end--;
    }

    return count;
}
```

三. Code 基本解法及编码实现

解法一：暴力解法边界和细节问题

1. 边界问题

   数组越界：最多存放10000个元素不会越界

2. 其它细节

   存入数组的元素都 > 0（整型默认值）

   记录最后一次存入的索引，倒序遍历每个元素，直到元素小于 t-3000

```
// 1. 创建数组，存放所有的请求
int[] array = new int[10000];

public int ping(int t) {
    // 2.把当前请求存入数组
    int end = 0;

    // 3.统计前3000毫秒之间的请求次数
    int count = 0;
    while () {
        end--;
    }

    return count;
}
```

三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

```
class RecentCounter {
    // 1. 创建数组，存放所有的请求
    int[] array = new int[10000];
    public int ping(int t) {
        int end = 0; // 最近一次请求存放的索引，从0开始
        // 2.把当前请求存入数组
        for (int i = 0; i < 10000; i++) {
            if (array[i] == 0) { // 细节：数组元素为0，则该位置没有存过请求
                array[i] = t;
                end = i; // 记录最近一次请求存放的索引
                break; // 存放操作完成
            }
        }
        // 3.统计前3000毫秒之间的请求次数
        int count = 0; // 计数器
        while (array[end] >= t - 3000) { // 数组元素在符合要求的范围内
            count++;
            if (--end < 0) { // 倒序遍历，防止越界
                break;
            }
        }
        return count;
    }
}
```

时间复杂度：O(n^2)

- 存入数组，每次遍历n-1个位置：O(n-1)
- 统计请求次数，每次遍历1~n个位置：O(n)
- ping方法会调用n次：n(n-1+n)，即：O(n^2)

空间复杂度：O(1)

- 固定10000长度的数组开销空间复杂度：O(1)
- 从绝对空间消耗来说消耗非常大

执行耗时:400 ms,击败了5.71% 的Java用户
内存消耗:47.2 MB,击败了64.72% 的Java用户

三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

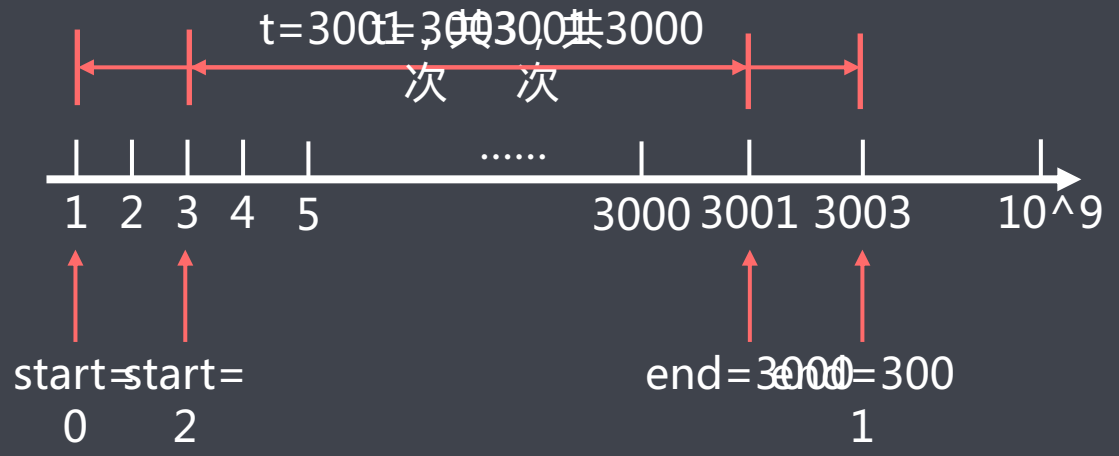
解法二：优化解法**分析**

1. 最多有多少次符合要求的请求？

   t 请求次 + 前3000次 = 3001次

2. 统计请求次数，是否有必要遍历已发生的所有请求？

   记录当前请求时间点对应的符合要求的起止索引，下次缩小检查范围



三. Code 基本解法及编码实现

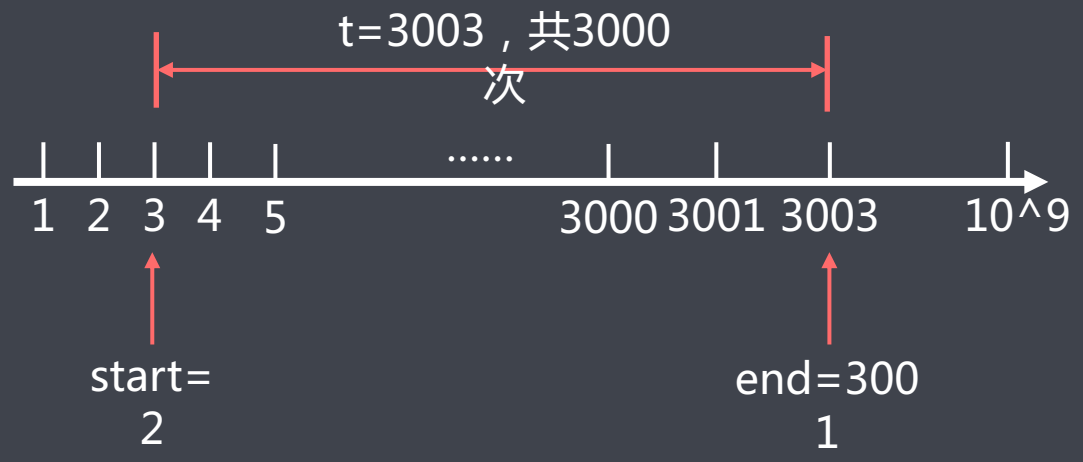
拉勾教育
—互联网人实战大学—

解法二：优化解法**思路**

1. 创建数组存放请求：int[3002]
2. 额外定义开始指针

   start=0，end=0，记录起止索引
3. 存放请求后，更新起止索引

   end++; 从上次的开始索引（start）向后查找　　直到新的合法的起始位置
4. 通过end与start差值计算请求次数



三. Code 基本解法及编码实现

解法二：优化解法边界和细节问题

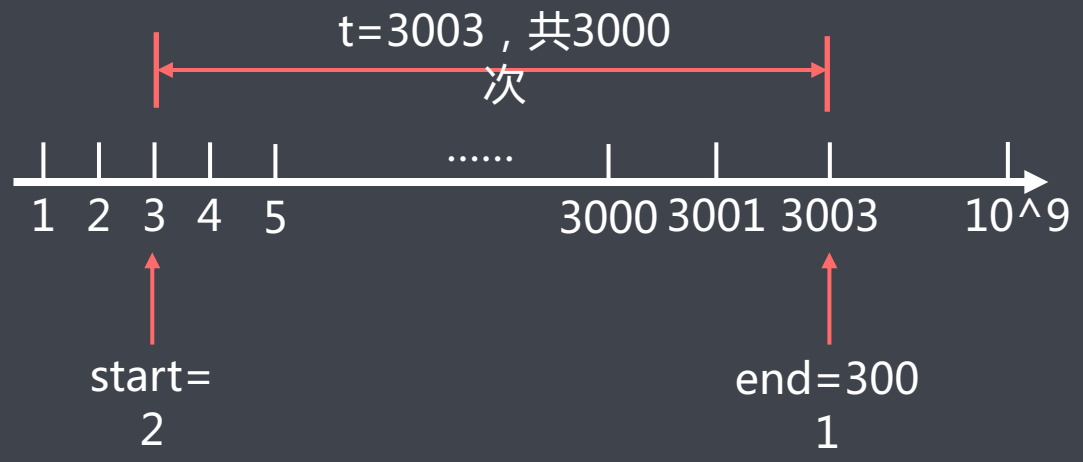
1. 边界问题

   计算控制start和end，请求次数超过数组容量则越界

2. 其它细节

   end指针溢出重新指向：0，start同理

   计算请求次数时，包含start和end



三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

```
class RecentCounter {
    // 1. 创建数组存放请求，最大合法请求次数为3001次（双闭区间）
    final int length = 3002; // 增加一个额外空间
    // 2. 记录起止索引，从0开始
    int start = 0, end = 0;
    int[] array = new int[length];
    public int ping(int t) {
        // 3. 存放请求后，更新起止索引
        array[end++] = t; // 存放最近一次请求，结束索引加1
        end = end == length ? 0 : end; // 越界后，从头开始
        // 从start位置正向查找符合要求的请求次数
        while (array[start] < t - 3000) { // 过滤所有不符合要求的数据
            start++; // 开始指针后移一位，继续循环检测下一个位置的元素
            start = start == length ? 0 : start; // 越界后，从头开始
        }
        // 4. 通过end与start差值计算请求次数
        if (start > end) { // 请求次数超过数组容量，发生了溢出
            return length - (start - end);
        }
        // 此时，end为最新一次请求+1的索引，start是3000毫秒前的第一次合法请求的索引
        return end - start;
    }
}
```

时间复杂度：O(n)

- 存入数组：O(1)
- 过滤不符合要求的元素：1~3002，即O(1)
- ping方法调用n次，最终时间复杂度：O(n)

空间复杂度：O(1)

- 固定3002长度的数组开销，空间复杂度：O(1)
- 额外两个指针：O(1)
- 从绝对空间消耗来说，消耗较大

执行耗时:24 ms,击败了99.91% 的Java用户
内存消耗:46.7 MB,击败了97.76% 的Java用户

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

1. 剔除无效代码或优化空间消耗
   - 创建成千上万个容量的数组比较浪费空间
     能否动态扩展容器的容量？
   - 是否有其它更方便的数据结构？
2. 寻找更好的算法思维
   - 精准计算首尾指针的索引容易出错
     能否做到不计算就获取首尾？
   - 借鉴其它算法

五. Code 最优解思路及编码实现

关键知识点：队列

1. 数据结构选择：队列

   始终在一端插入数据，另一端删除数据

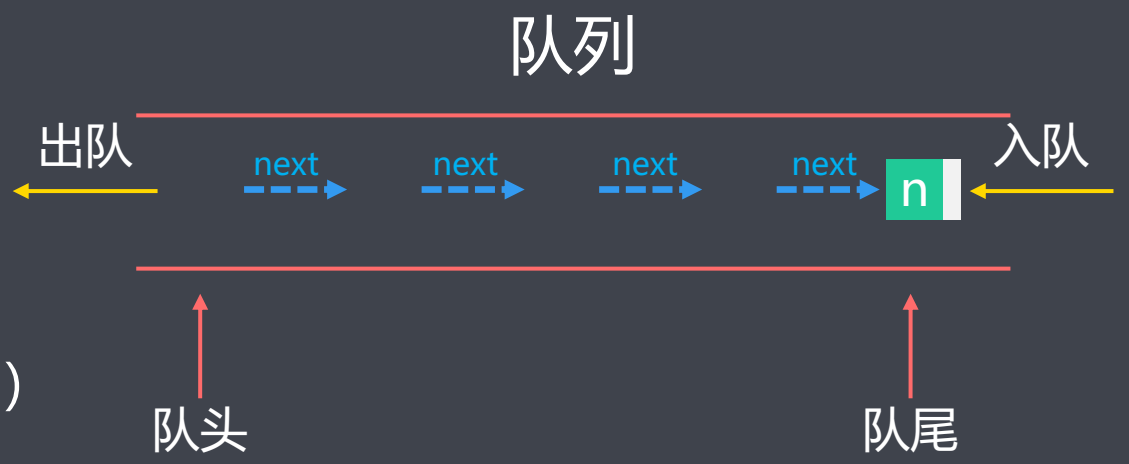
   先进先出（FIFO，First Input First Output）

   插入和删除时间复杂度：O(1)

2. 基本操作

   入队：向队列尾部添加节点，size++

   出队：从队列头部删除节点，size--



五. Code 最优解思路及编码实现

拉勾教育
—互联网人实战大学—

最优解：队列解法思路分析

1. 使用链表实现一个队列

   定义属性：队头-head、队尾-tail、长度-size

   定义方法：添加节点-add(int)、移除节点-poll()、队列长度-size()

   定义内部类：Node，封装每次入队的请求数据和指向下一个节点的指针

```
class Queue {
    Node head;
    Node tail;
    int size = 0;
    public void add(int x) { // 向尾部添加一个节点
        size++;
    }
    public int poll() { // 从头部移除一个节点
        size--;
    }
    public int size() {
        return size;
    }
}
```

```
class Queue {
    class Node {
        int val;
        Node next;
        Node(int x) {
            val = x;
        }
        int getVal() {
            return val;
        }
    }
}
```

五. Code 最优解思路及编码实现

最优解：队列解法思路分析

2. 每次请求向队列尾部追加节点
3. 循环检查队头数据是否合法

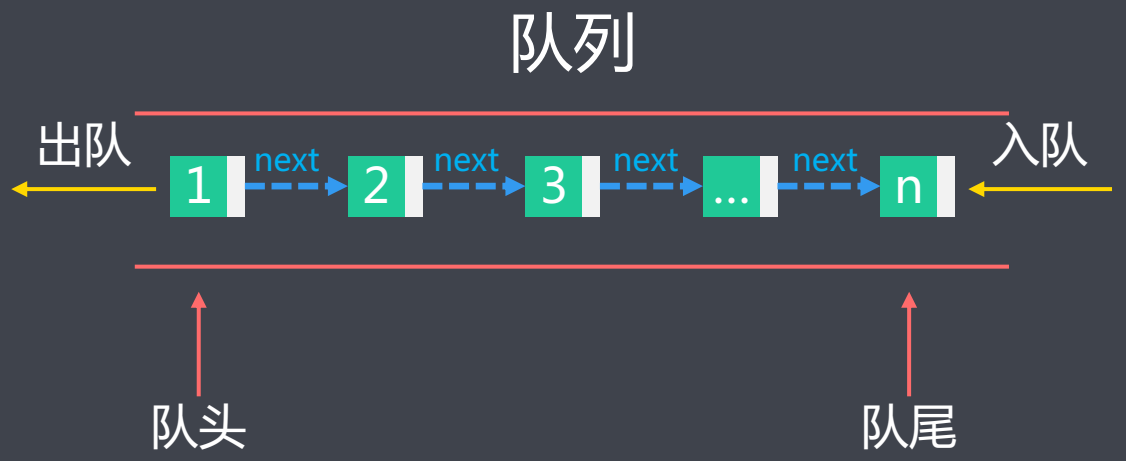
   不合法则移除该节点点
4. 返回队列长度

```
class RecentCounter {
    Queue q;
    public RecentCounter() {
        q = new Queue();
    }
    public int ping(int t) {
        // 2.每次请求向队列尾部追加节点
        q.add(t);
        // 3. 循环检查队头数据是否合法
        while (q.head.getVal() < t - 3000)
            q.poll(); // 移除队头节点
        // 4. 返回队列长度
        return q.size();
    }
}
```

五. Code 最优解思路及编码实现

最优解：边界、细节问题和复杂度分析

- 边界问题
  - 入队和出队始终关注首尾节点指针
- 细节问题
  - 第一次添加节点，首尾指针都是null
  - 每次追加尾节点，size++，重置tail
  - 每次移除头结点，size--，重置head



五. Code 最优解思路及编码实现


```
class RecentCounter {
    Queue q = new Queue();
    public int ping(int t) {
        q.add(t);
        while (q.head.getVal() < t - 3000)
            q.poll();
        return q.size();
    }
    class Queue {
        Node head;
        Node tail;
        int size = 0;
        public void add(int x) {} // 向尾部添加一个节点
        public int poll() {} // 从头部移除一个节点
        public int size() {return size;}
        class Node {
            int val;
            Node next;
            Node(int x) {val = x;}
            int getVal() {return val;}
        }
    }
}
```

```
public void add(int x) { // 向尾部添加一个节点
    Node last = tail; // 获取原来的尾节点
    Node newNode = new Node(x); // 创建新节点，封装数据
    tail = newNode; // 尾指针指向新节点
    if (last == null) { // 第一次添加数据
        head = newNode; // 头节点为新节点
        tail = newNode;
    } else {
        last.next = newNode; // 前一个节点指向新节点
    }
    size++; // 每添加一个节点，队列长度+1
}
public int poll() { // 从头部移除一个节点
    int headVal = head.val; // 获取头节点的数据
    Node next = head.next; // 获取头节点的下一个节点
    head.next = null; // 断开队列链接，help GC
    head = next; // 头指针指向下一个节点
    if (next == null) { // 队列中的最后一个元素
        tail = null; // 处理尾指针
    }
    size--; // 每移除一个节点，队列长度-1
    return headVal;
}
```

执行耗时:31 ms,击败了43.43% 的Java用户
内存消耗:46.7 MB,击败了97.76% 的Java用户

五. Code 最优解思路及编码实现

拉勾教育
—互联网人实战大学—

最优解：边界、细节问题和复杂度分析

- 边界问题
  - 入队和出队始终关注首尾节点指针
- 细节问题
  - 第一次添加节点，首尾指针都是null
  - 每次追加尾节点，size++，重置tail
  - 每次移除头结点，size--，重置head
- 时间复杂度：O(1)
  - 添加或删除元素，都是O(1)
- 空间复杂度：O(1)
  - 每次添加都是O(1)
  - 最多保留3001个元素，所以空间复杂度为：1~3001，即O(1)

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

- （练习）数组 + 双指针模式也是队列的另一种存在形式

延伸扩展

- 实际编码中常常初始化队列时创建一个空的Node对象作为head节点，同时，tail也指向这个Node对象，即：head = tail = new Node();减少对头指针的非空判断。

本题来源：

- leetcode 933 https://leetcode-cn.com/problems/number-of-recent-calls/

总结

拉勾教育
—互联网人实战大学—

6C解题法

队列的特点

- 始终在一端插入数据，另一端删除数据
- 先进先出（FIFO，First Input First Output）
- 插入和删除时间复杂度：O(1)

队列的应用

- 生产者消费者模式：CPU调度多线程；消息队列，作为缓冲区提高效率

课后练习

1. 用栈实现队列 (Leetcode 232/简单)
2. 用两个栈实现队列（剑指 Offer 09/简单）
3. 用队列实现栈（Leetcode 225 /简单）
4. 和至少为 K 的最短子数组（Leetcode 862/困难）

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容