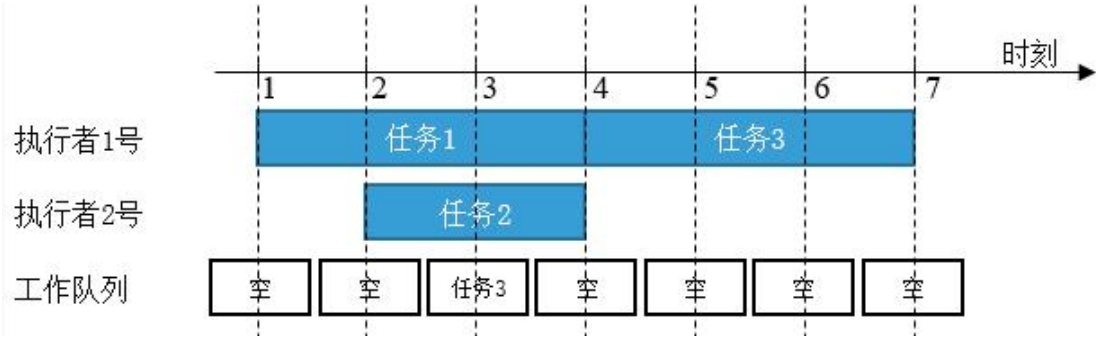
Java-队列-让我们来模拟一个工作队列的运作
题目描述：
让我们来模拟一个工作队列的运作，有一个任务提交者和若干任务执行者，执行者从 1 开始编号。

提交者会在给定的时刻向工作队列提交任务，任务有执行所需的时间，执行者取出任务的时刻加上执行时间即为任务完成的时刻。
执行者完成任务变为空闲的时刻会从工作队列中取最老的任务执行，若这一时刻有多个空闲的执行者，其中优先级最高的会执行这个任务。编号小的执行者优先级高。初始状态下所有执行者都空闲。
工作队列有最大长度限制，当工作队列满而有新的任务需要加入队列时，队列中最老的任务会被丢弃。
特别的，在工作队列满的情况下，当执行者变为空闲的时刻和新的任务提交的时刻相同时，队列中最老的任务被取出执行，新的任务加入队列。

输入描述：
输入为两行。第一行为 2N 个正整数，代表提交者提交的 N 个任务的时刻和执行时间。第一个数字是第一个任务的提交时刻，第二个数字是第一个任务的执行时间，以此类推。用例保证提交时刻不会重复，任务按提交时刻升序排列。

第二行为两个数字，分别为工作队列的最大长度和执行者的数量。

两行的数字都由空格分隔。N 不超过 20，数字为不超过 1000 的正整数。
输出描述：
输出两个数字，分别为最后一个任务执行完成的时刻和被丢弃的任务的数量，数字由空格分隔。
补充说明：
示例 1
输入：
1 3 2 2 3 3
3 2
输出：
7 0
说　　明　　：



有两个执行者，执行者 1 号在时刻 1 取出了第 1 个任务，该任务需时为 3，故 1 号会在时刻 4 完成任务；执行者 2 号在时刻 2 取出了第 2 个任务，该任务需时为 2，故 2 号也会在时刻 4 完成任务；第 3 个任务在时刻 3 提交，但此时没有空闲的执行者，故缓存在队列中，直到时刻 4 两个执行者都变为空闲，此时执行者 1 号会取出这个任务，该任务需时为 3，故会在时刻 7 完成任务。期间没有任务被丢弃。

示例 2

输入：

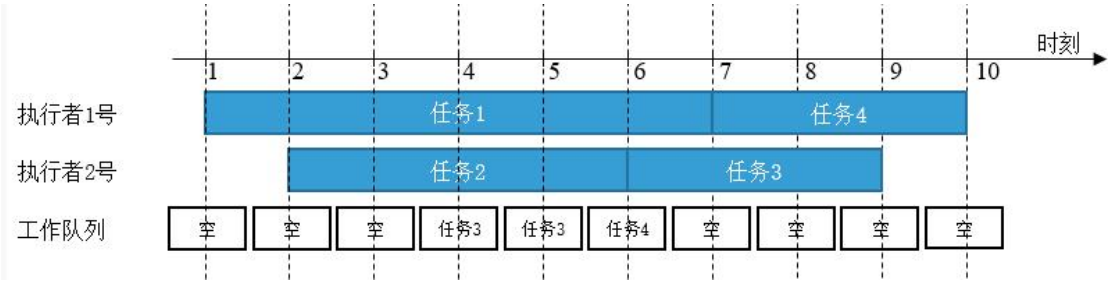
1 6 2 4 4 3 6 3

1 2

输出：

10 0

说明：



有两个执行者，执行者 1 号在时刻 1 取出了第 1 个任务，该任务需时为 6，故 1 号会在时刻 7 完成任务；执行者 2 号在时刻 2 取出了第 2 个任务，该任务需时为 4，故 2 号会在时刻 6 完成任务；第 3 个任务在时刻 4 提交，但此时没有空闲的执行者，故缓存在队列中。时刻 6 执行者 2 号变为空闲，同时第 4 个任务被提交，此时队列已满。按特别说明，第 4 个任务加入队列，队列中最老的第 3 个任务被取出执行并在时刻 9 完成。最后执行者 1 号在时刻 7 取出第 4 个任务并在时刻 10 完成，期间没有任务被丢弃。

示例 3

输入：

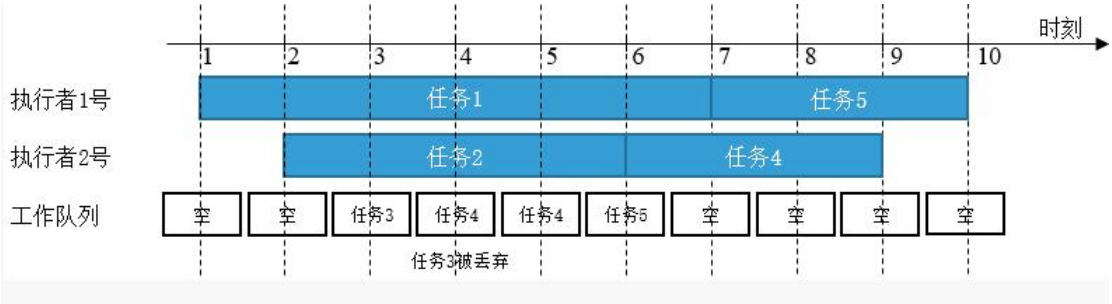
1 6 2 4 3 3 4 3 6 3

1 2

输出：

10 1

说明：



大致的流程和示例 2 类似，只是第 4 个任务在时刻 4 被提交时，没有空闲的执行者，而且工作队列已经缓存了第 3 个任务满了，故第 3 个任务被丢弃。

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        Integer[] tasks = Arrays.stream(in.nextLine().split(" ")).map(
                                    Integer::parseInt).toArray(Integer[]::new);
        int length = in.nextInt();
        int num = in.nextInt();
        ArrayList<Integer[]> execute = new ArrayList<>(num);
        // 初始化所有执行者的任务为 null
        for (int i = 0; i < num; i++) {
            execute.add(null);
        }
        Queue<Integer[]> queue = new LinkedList<>();
        int t = 0;
        int index = 0;
        int wasted = 0;
        // 反正数字不会超过 1000，就 1 秒 1 秒处理呗
        while (true) {
            // 判断任务是否执行完成,若完成，及时清理，并取出队列中的老任务继续执行
            for (int i = 0; i < execute.size(); i++) {
                Integer[] task = execute.get(i);
                // 执行者没有任务，不着急指派，因为下方在执行完任务时会立刻从队列取，为 null 表示队列为空，至于新来的任务会在下个阶段处理
                if (task == null) continue;
                if (task[0] + task[1] == t) {
                    if (! queue.isEmpty()) {
                        // 先进先出，肯定是最老的，但是开始时间要改成现在，然后丢给执行者
                        Integer[] oldTask = queue.poll();
                        oldTask[0] = t;
                        execute.set(i, oldTask);
                        // 队列空的话，就只能给执行者放假了
                    } else execute.set(i, null);
                }
```

```
                }
                // 如果有新任务进来
                if (index < tasks.length && tasks[index] == t) {
                    Integer[] newTask = new Integer[] {tasks[index], tasks[index + 1]};
                    boolean flag = false;
                    // 检查是否有空闲执行者，若有，直接给他
                    for (int i = 0; i < execute.size(); i++) {
                        if (execute.get(i) == null) {
                            execute.set(i, newTask);
                            flag = true;
                            break;
                        }
                    }
                    // 没有空闲执行者，检查队列是否有空位，有则加入，无则先丢了最老的，再加入新的
                    if (! flag) {
                        if (queue.size() < length) queue.add(newTask);
                        else {
                            queue.poll();
                            queue.add(newTask);
                            wasted++;
                        }
                    }
                    // 指向下一个任务
                    index += 2;
                }
                // 没有新任务，队列全空，执行者全员空闲，则结束处理
                if (index >= tasks.length && queue.isEmpty() && isAllNull(execute)) break;
                t++;
            }
            System.out.println(t + " " + wasted);
        }
        private static boolean isAllNull(ArrayList<Integer[]> execute) {
            for (Integer[] integers : execute) {
                if (integers != null) return false;
            }
            return true;
        }
}
```