

```

Action[] record = { Action.No };
int step = 0;
while (rS.size() != num + 1) {
    step += fStackTotStack(record, Action.MToL, Action.LToM, lS, mS,
        left, mid);
    step += fStackTotStack(record, Action.LToM, Action.MToL, mS, lS,
        mid, left);
    step += fStackTotStack(record, Action.RToM, Action.MToR, mS, rS,
        mid, right);
    step += fStackTotStack(record, Action.MToR, Action.RToM, rS, mS,
        right, mid);
}
return step;
}

public static int fStackTotStack(Action[] record, Action preNoAct,
    Action nowAct, Stack<Integer> fStack, Stack<Integer> tStack,
    String from, String to) {
    if (record[0] != preNoAct && fStack.peek() < tStack.peek()) {
        tStack.push(fStack.pop());
        System.out.println("Move " + tStack.peek() + " from " + from + "
to " + to);
        record[0] = nowAct;
        return 1;
    }
    return 0;
}
}

```

生成窗口最大值数组

【题目】

有一个整型数组 `arr` 和一个大小为 `w` 的窗口从数组的最左边滑到最右边，窗口每次向右边滑一个位置。

例如，数组为[4,3,5,4,3,3,6,7]，窗口大小为 3 时：

[4 3 5]	4 3 3 6 7	窗口中最大值为 5
4 [3 5 4]	3 3 6 7	窗口中最大值为 5
4 3 [5 4 3]	3 6 7	窗口中最大值为 5
4 3 5 [4 3 3]	6 7	窗口中最大值为 4
4 3 5 4 [3 3 6]	7	窗口中最大值为 6
4 3 5 4 3 [3 6 7]		窗口中最大值为 7

如果数组长度为 n ，窗口大小为 w ，则一共产生 $n-w+1$ 个窗口的最大值。

请实现一个函数。

- 输入：整型数组 arr ，窗口大小为 w 。
- 输出：一个长度为 $n-w+1$ 的数组 res ， $res[i]$ 表示每一种窗口状态下的最大值。

以本题为例，结果应该返回 $\{5,5,5,4,6,7\}$ 。

【难度】

尉 ★★☆☆

【解答】

如果数组长度为 N ，窗口大小为 w ，如果做出时间复杂度 $O(N \times w)$ 的解法是不能让面试官满意的，本题要求面试者想出时间复杂度 $O(N)$ 的实现。

本题的关键在于利用双端队列来实现窗口最大值的更新。首先生成双端队列 $qmax$ ， $qmax$ 中存放数组 arr 中的下标。

假设遍历到 $arr[i]$ ， $qmax$ 的放入规则为：

1. 如果 $qmax$ 为空，直接把下标 i 放进 $qmax$ ，放入过程结束。
2. 如果 $qmax$ 不为空，取出当前 $qmax$ 队尾存放的下标，假设为 j 。
 - 1) 如果 $arr[j] > arr[i]$ ，直接把下标 i 放进 $qmax$ 的队尾，放入过程结束。
 - 2) 如果 $arr[j] \leq arr[i]$ ，把 j 从 $qmax$ 中弹出，继续 $qmax$ 的放入规则。

假设遍历到 $arr[i]$ ， $qmax$ 的弹出规则为：

如果 $qmax$ 队头的下标等于 $i-w$ ，说明当前 $qmax$ 队头的下标已过期，弹出当前对头的下标即可。

根据如上的放入和弹出规则， $qmax$ 便成了一个维护窗口为 w 的子数组的最大值更新的结构。下面举例说明题目给出的例子。

1. 开始时 $qmax$ 为空， $qmax = \{\}$
2. 遍历到 $arr[0] = 4$ ，将下标 0 放入 $qmax$ ， $qmax = \{0\}$ 。
3. 遍历到 $arr[1] = 3$ ，当前 $qmax$ 的队尾下标为 0，又有 $arr[0] > arr[1]$ ，所以将下标 1 放入 $qmax$ 的尾部， $qmax = \{0, 1\}$ 。
4. 遍历到 $arr[2] = 5$ ，当前 $qmax$ 的队尾下标为 1，又有 $arr[1] \leq arr[2]$ ，所以将下标 1 从 $qmax$ 的尾部弹出， $qmax$ 变为 $\{0\}$ 。当前 $qmax$ 的队尾下标为 0，又有 $arr[0] \leq arr[2]$ ，所以将下标 0 从 $qmax$ 尾部弹出， $qmax$ 变为 $\{\}$ 。将下标 2 放入 $qmax$ ， $qmax = \{2\}$ 。此时已经遍历到下标 2 的位置，窗口 $arr[0..2]$ 出现，当前 $qmax$ 队头的下标为 2，所以窗口 $arr[0..2]$

的最大值为 $\text{arr}[2]$ (即 5)。

5. 遍历到 $\text{arr}[3]=4$, 当前 qmax 的队尾下标为 2, 又有 $\text{arr}[2]>\text{arr}[3]$, 所以将下标 3 放入 qmax 尾部, $\text{qmax}=\{2,3\}$ 。窗口 $\text{arr}[1..3]$ 出现, 当前 qmax 队头的下标为 2, 这个下标还没有过期, 所以窗口 $\text{arr}[1..3]$ 的最大值为 $\text{arr}[2]$ (即 5)。

6. 遍历到 $\text{arr}[4]=3$, 当前 qmax 的队尾下标为 3, 又有 $\text{arr}[3]>\text{arr}[4]$, 所以将下标 4 放入 qmax 尾部, $\text{qmax}=\{2,3,4\}$ 。窗口 $\text{arr}[2..4]$ 出现, 当前 qmax 队头的下标为 2, 这个下标还没有过期, 所以窗口 $\text{arr}[2..4]$ 的最大值为 $\text{arr}[2]$ (即 5)。

7. 遍历到 $\text{arr}[5]=3$, 当前 qmax 的队尾下标为 4, 又有 $\text{arr}[4]\leq\text{arr}[5]$, 所以将下标 4 从 qmax 的尾部弹出, qmax 变为 $\{2,3\}$ 。当前 qmax 的队尾下标为 3, 又有 $\text{arr}[3]>\text{arr}[5]$, 所以将下标 5 放入 qmax 尾部, $\text{qmax}=\{2,3,5\}$ 。窗口 $\text{arr}[3..5]$ 出现, 当前 qmax 队头的下标为 2, 这个下标已经过期, 所以从 qmax 的头部弹出, qmax 变为 $\{3,5\}$ 。当前 qmax 队头的下标为 3, 这个下标没有过期, 所以窗口 $\text{arr}[3..5]$ 的最大值为 $\text{arr}[3]$ (即 4)。

8. 遍历到 $\text{arr}[6]=6$, 当前 qmax 的队尾下标为 5, 又有 $\text{arr}[5]\leq\text{arr}[6]$, 所以将下标 5 从 qmax 的尾部弹出, qmax 变为 $\{3\}$ 。当前 qmax 的队尾下标为 3, 又有 $\text{arr}[3]\leq\text{arr}[6]$, 所以将下标 3 从 qmax 的尾部弹出, qmax 变为 $\{\}$ 。将下标 6 放入 qmax , $\text{qmax}=\{6\}$ 。窗口 $\text{arr}[4..6]$ 出现, 当前 qmax 队头的下标为 6, 这个下标没有过期, 所以窗口 $\text{arr}[4..6]$ 的最大值为 $\text{arr}[6]$ (即 6)。

9. 遍历到 $\text{arr}[7]=7$, 当前 qmax 的队尾下标为 6, 又有 $\text{arr}[6]\leq\text{arr}[7]$, 所以将下标 6 从 qmax 的尾部弹出, qmax 变为 $\{\}$ 。将下标 7 放入 qmax , $\text{qmax}=\{7\}$ 。窗口 $\text{arr}[5..7]$ 出现, 当前 qmax 队头的下标为 7, 这个下标没有过期, 所以窗口 $\text{arr}[5..7]$ 的最大值为 $\text{arr}[7]$ (即 7)。

10. 依次出现的窗口最大值为 $[5,5,5,4,6,7]$, 在遍历过程中收集起来, 最后返回即可。

上述过程中, 每个下标值最多进 qmax 一次, 出 qmax 一次。所以遍历的过程中进出双端队列的操作是时间复杂度为 $O(N)$, 整体的时间复杂度也为 $O(N)$ 。具体过程参看如下代码中的 `getMaxWindow` 方法。

```
public int[] getMaxWindow(int[] arr, int w) {
    if (arr == null || w < 1 || arr.length < w) {
        return null;
    }
    LinkedList<Integer> qmax = new LinkedList<Integer>();
    int[] res = new int[arr.length - w + 1];
    int index = 0;
    for (int i = 0; i < arr.length; i++) {
        while (!qmax.isEmpty() && arr[qmax.peekLast()] <= arr[i]) {

```

```

        qmax.pollLast();
    }
    qmax.addLast(i);
    if (qmax.peekFirst() == i - w) {
        qmax.pollFirst();
    }
    if (i >= w - 1) {
        res[index++] = arr[qmax.peekFirst()];
    }
}
return res;
}

```

构造数组的 MaxTree

【题目】

定义二叉树节点如下：

```

public class Node {
    public int value;
    public Node left;
    public Node right;

    public Node(int data) {
        this.value = data;
    }
}

```

一个数组的 MaxTree 定义如下。

- 数组必须没有重复元素。
- MaxTree 是一棵二叉树，数组的每一个值对应一个二叉树节点。
- 包括 MaxTree 树在内且在其中的每一棵子树上，值最大的节点都是树的头。

给定一个没有重复元素的数组 `arr`，写出生成这个数组的 MaxTree 的函数，要求如果数组长度为 N ，则时间复杂度为 $O(N)$ 、额外空间复杂度为 $O(N)$ 。

【难度】

校 ★★★★★