2주차

이런 문제들을 정확하고 빠르게 풀어내는게 중요하다~

스택 수열

스택 수열 성공

☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	98958	36839	26043	36.480%

문제

스택 (stack)은 기본적인 자료구조 중 하나로, 컴퓨터 프로그램을 작성할 때 자주 이용되는 개념이다. 스택은 자료를 넣는 (push) 입구와 자료를 뽑는 (pop) 입구가 같아 제일 나중에 들어간 자료가 제일 먼저 나오는 (LIFO, Last in First out) 특성을 가지고 있다.

1부터 n까지의 수를 스택에 넣었다가 뽑아 늘어놓음으로써, 하나의 수열을 만들 수 있다. 이때, 스택에 push하는 순서는 반드시 오름차순을 지키도록 한다고 하자. 임의의 수열이 주어졌을 때 스택을 이용해 그 수열을 만들 수 있는지 없는지, 있다면 어떤 순서로 push와 pop 연산을 수행해야 하는지를 알아낼 수 있다. 이를 계산하는 프로그램을 작성하라.

입력

첫 줄에 n (1 ≤ n ≤ 100,000)이 주어진다. 둘째 줄부터 n개의 줄에는 수열을 이루는 1이상 n이하의 정수가 하나씩 순서대로 주어진다. 물론 같은 정수가 두 번 나오는 일은 없다.

출력

입력된 수열을 만들기 위해 필요한 연산을 한 줄에 한 개씩 출력한다. push연산은 +로, pop 연산은 -로 표현하도록 한다. 불가능한 경우 NO를 출력한다.

예제 입력 1 복사

```
8
4
3
6
8
7
5
2
1
```

예제 출력 1 복사

```
+
+
+
+
-
-
+
+
-
+
```

시간 복잡도

O(N)

오답조건

예제 입력 2 복사

```
5
1
2
5
3
4
```

예제 출력 2 복사

```
NO
```

제일 큰 수 → 작은 수 → 두번째로 큰 수 발견 시 실패

```
private static void solution() {
    StringBuilder sb = new StringBuilder();

    int stackNum = 1;
    int numsIdx = 0;

    Stack<Integer> stack = new Stack<>();
    stack.add(0);
    while (numsIdx < N) {
        if (stack.peek() < nums[numsIdx]) {
            stack.add(stackNum++);
            sb.append('+').append('\n');
        } else if (stack.peek() > nums[numsIdx]) {
            System.out.println("NO");
            return;
```

```
        } else if (stack.peek() == nums[numsIdx++]) {
            stack.pop();
            sb.append('-').append('\n');
        }
    }

    System.out.println(sb.toString());
}
```

점화식이 있으면 stack 안써도 될 것 같긴한데 귀찮아서 그냥 스택 씀

프린터 큐

프린터 큐 다국어

☆ 한국어

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	50315	28383	22458	57.706%

문제

여러분도 알다시피 여러분의 프린터 기기는 여러분이 인쇄하고자 하는 문서를 인쇄 명령을 받은 '순서대로', 즉 먼저 요청된 것을 먼저 인쇄한다. 여러 개의 문서가 쌓인다면 Queue 자료구조에 쌓여서 FIFO - First In First Out - 에 따라 인쇄가 되게 된다. 하지만 상근이는 새로운 프린터기 내부 소프트웨어를 개발하였는데, 이 프린터기는 다음과 같은 조건에 따라 인쇄를 하게 된다.

1. 현재 Queue의 가장 앞에 있는 문서의 '중요도'를 확인한다.
2. 나머지 문서들 중 현재 문서보다 중요도가 높은 문서가 하나라도 있다면, 이 문서를 인쇄하지 않고 Queue의 가장 뒤에 재배치 한다. 그렇지 않다면 바로 인쇄를 한다.

예를 들어 Queue에 4개의 문서(A B C D)가 있고, 중요도가 2 1 4 3 라면 C를 인쇄하고, 다음으로 D를 인쇄하고 A, B를 인쇄하게 된다.

여러분이 할 일은, 현재 Queue에 있는 문서의 수와 중요도가 주어졌을 때, 어떤 한 문서가 몇 번째로 인쇄되는지 알아내는 것이다. 예를 들어 위의 예에서 C문서는 1번째로, A문서는 3번째로 인쇄되게 된다.

시간 복잡도

입력

첫 줄에 테스트케이스의 수가 주어진다. 각 테스트케이스는 두 줄로 이루어져 있다.

테스트케이스의 첫 번째 줄에는 문서의 개수 N(1 ≤ N ≤ 100)과, 몇 번째로 인쇄되었는지 궁금한 문서가 현재 Queue에서 몇 번째에 놓여 있는지를 나타내는 정수 M(0 ≤ M < N)이 주어진다. 이때 맨 왼쪽은 0번째라고 하자. 두 번째 줄에는 N개 문서의 중요도가 차례대로 주어진다. 중요도는 1 이상 9 이하의 정수이고, 중요도가 같은 문서가 여러 개 있을 수도 있다.

출력

각 테스트 케이스에 대해 문서가 몇 번째로 인쇄되는지 출력한다.

예제 입력 1 복사

```
3
1 0
5
4 2
1 2 3 4
6 0
1 1 9 1 1 1
```

예제 출력 1 복사

```
1
2
5
```

입력이 만약

```
100 0
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 ...
```

이면 완전탐색으로 으로 가능할까? 모르겠다.

풀이

우선순위를 저장하는 PriorityQueue 와 문서가 대기하는 Queue를 운영하자

Queue에서 나온 Document 의 priority 가 maxPriority 보다 크거나 같으면 출력

```
private static void solution() {
    PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
    Queue<Document> q = new LinkedList<>();

    for (int i = 0; i < N; i++) {
        pq.add(priorities[i]);
        q.add(new Document(i, priorities[i]));
    }

    while (!q.isEmpty()) {
        Document d = q.peek();
        int maxPriority = pq.peek();
        if (d.priority >= maxPriority) {
            q.poll();
            pq.poll();

            if (d.idx == K) {
                break;
            }
        } else if (d.priority < maxPriority) {
            q.add(q.poll());
        }

    }

    sb.append(N - q.size()).append('\n');
}
```