징검다리 건너기

문제 설명

[본 문제는 정확성과 효율성 테스트 각각 점수가 있는 문제입니다.]

카카오 초등학교의 "니니즈 친구들"이 "라이언" 선생님과 함께 가을 소풍을 가는 중에 **징검다리**가 있는 개울을 만나서 건너편으로 건너려고 합니다. "라이언" 선생님은 "니니즈 친구들"이 무사히 징검다리를 건널 수 있도록 다음과 같이 규칙을 만들었습니다.

- 징검다리는 일렬로 놓여 있고 각 징검다리의 디딤돌에는 모두 숫자가 적혀 있으며 디딤돌의 숫자는 한 번 밟을 때마다 1씩 줄어듭니다.
- 디딤돌의 숫자가 0이 되면 더 이상 밟을 수 없으며 이때는 그 다음 디딤돌로 한번에 여러 칸을 건너 뛸 수 있습니다.
- 단, 다음으로 밟을 수 있는 디딤돌이 여러 개인 경우 무조건 가장 가까운 디딤돌로만 건너뛸 수 있습니다.

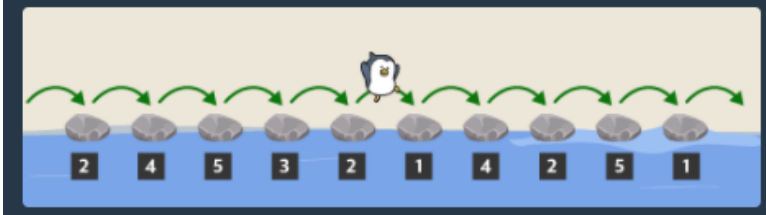
"니니즈 친구들"은 개울의 왼쪽에 있으며, 개울의 오른쪽 건너편에 도착해야 징검다리를 건넌 것으로 인정합니다.
"니니즈 친구들"은 한 번에 한 명씩 징검다리를 건너야 하며, 한 친구가 징검다리를 모두 건넌 후에 그 다음 친구가 건너기 시작합니다.

디딤돌에 적힌 숫자가 순서대로 담긴 배열 stones와 한 번에 건너뛸 수 있는 디딤돌의 최대 칸수 k가 매개변수로 주어질 때, 최대 몇 명까지 징검다리를 건널 수 있는지 return 하도록 solution 함수를 완성해주세요.

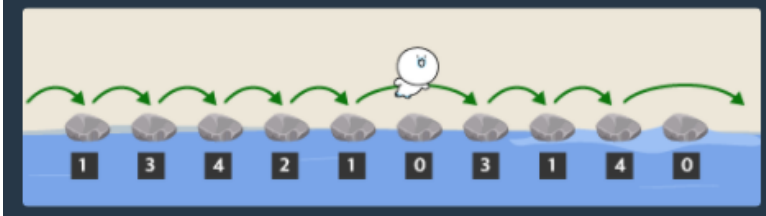
[제한사항]

- 징검다리를 건너야 하는 니니즈 친구들의 수는 무제한 이라고 간주합니다.
- stones 배열의 크기는 1 이상 200,000 이하입니다.
- stones 배열 각 원소들의 값은 1 이상 200,000,000 이하인 자연수입니다.
- k는 1 이상 stones의 길이 이하인 자연수입니다.

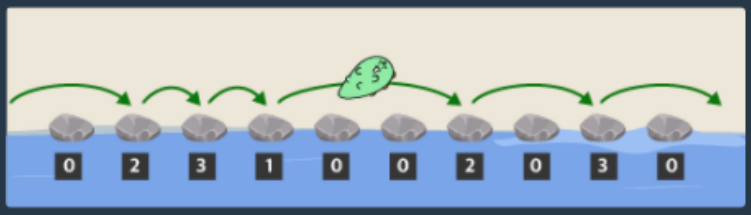
첫 번째 친구는 다음과 같이 징검다리를 건널 수 있습니다.



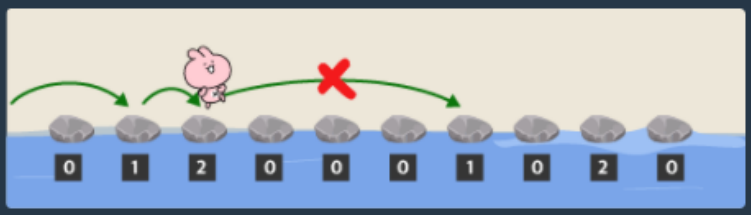
첫 번째 친구가 징검다리를 건넌 후 디딤돌에 적힌 숫자는 아래 그림과 같습니다.
두 번째 친구도 아래 그림과 같이 징검다리를 건널 수 있습니다.



두 번째 친구가 징검다리를 건넌 후 디딤돌에 적힌 숫자는 아래 그림과 같습니다.
세 번째 친구도 아래 그림과 같이 징검다리를 건널 수 있습니다.



세 번째 친구가 징검다리를 건넌 후 디딤돌에 적힌 숫자는 아래 그림과 같습니다.
네 번째 친구가 징검다리를 건너려면, 세 번째 디딤돌에서 일곱 번째 디딤돌로 네 칸을 건너뛰어야 합니다. 하지만 k = 3 이므로 건너뛸 수 없습니다.



따라서 최대 3명이 디딤돌을 모두 건널 수 있습니다.

초기 아이디어

```
1. 친구 한명이 지나갈 때, 징검다리의 값을 하나씩 줄이면서 진행한다.

2. 끝까지 도달한 경우 카운터를 1 올린다.

3. 실패 할 때 까지 위를 반복한다.

4. 실패 했을 때 카운터를 정답으로 도출

-> 시간 초과
```

해결 아이디어

이분 탐색 (건널 수 있다고 생각되는 친구들의 수를 기준)

전체 돌의 최소값과 최대값을 순회하며 구한다.

건널 수 있는 친구의 수는 최소값 <= 친구 수 <= 최대 값 사이에 존재하게 된다.

1. (최소 + 최대) / 2 값의 친구가 건널 수 있다고 가정하고 유효성 검사를 한다.

2. 유효 하다면 더 큰 값으로 가능한지 해본다.

3. 유효 하지 않다면 더 작은 값으로 가능한지 시도해본다.

* 유효성 검사

=> 돌의 적힌 숫자에서 - 타겟 값을 뺐을 때, 0보다 작아지면 이 돌은 실패할 가능성이 있는돌

=> 가능성이 있는 돌이 k 번 반복 된다면,

   타겟보다 많이 밟히는 돌이 있다는 뜻이므로,

=> 너무 많은 친구의 수 라는 뜻

이분 탐색을 고려해봐야 할 문제 내의 지문

답이 여러개 발생 할 수 있는 상황에서, 그 최대 혹은 최소 값을 찾는 지문

1. 답이 나올 수 있는 범위를 한정

2. 중간 값으로 유효한 지를 판별하며 값을 찾는다.

3. 탐색 시간을 N -> logN 으로 줄일 수 있다.

코드

```
class Solution {
  private static boolean isOk(int mid, int k, int[] stones) {
    int fail = 0;
```

```
    // 실패하는 경우
    for(int i = 0; i < stones.length; i++) {
      // 3    4 3  1 0 0 0
      if(stones[i] - mid < 0) {
        fail++; // 실패 가능성
      }
      else {
        fail = 0;
      }

      if(fail == k ) {
        return false;
      }
    }
    return true;
  }
    public static int solution(int[] stones, int k) {
      int answer = 0;

      int max = Integer.MIN_VALUE;
      int min = Integer.MAX_VALUE;
      for(int i = 0; i < stones.length; i++) {
        max = Math.max(max, stones[i]);
        min = Math.min(min, stones[i]);
      }

      int mid = 0;

      while(max >= min) {
        mid = ( max + min ) / 2;
        // 중간값의 친구만큼 이동이 가능?
        if(isOk(mid, k , stones)) {
          // 할 수 있으면 작은값을 mid 보다 1 크게 가능?
          answer = mid;
          min = mid + 1;
        }
        else {
          // 못하면 큰 값을 mid 보다 1작게 가능?
          max = mid - 1;
        }
      }
        return answer;
    }
}
```