프로그래머스 - 공 던지기

태그 IntStream.range Queue<Integer> queue flatMapToInt(IntStream::of) numbers[(k - 1) * 2 % numbers.length] queue.add(queue.poll())

자바스크립트
공 던지기
문제 설명
머쓱이는 친구들과 동그랗게 서서 공 던지기 게임을 하고 있습니다. 공은 1번부터 던지며 오른쪽으로 한 명을 건너뛰고 그다음 사람에게만 던질 수 있습니다. 친구들의 번호가 들어있는 정수 배열 numbers와 정수 K가 주어질 때, k번째로 공을 던지는 사람의 번호는 무엇인지 return 하도록 solution 함수를 완성해보세요.

제한사항
2 < numbers의 길이 < 100
0 < k < 1,000
numbers의 첫 번째와 마지막 번호는 실제로 바로 옆에 있습니다.
numbers는 1부터 시작하며 번호는 순서대로 올라갑니다.

입출력 예 #1
numbers: [1, 2, 3, 4] ,k:2 ,result:3
1번은 첫 번째로 3번에게 공을 던집니다.
3번은 두 번째로 1번에게 공을 던집니다.
입출력 예 #2
numbers: [1, 2, 3, 4, 5, 6] ,k:5 ,result:3
1번은 첫 번째로 3번에게 공을 던집니다.
3번은 두 번째로 5번에게 공을 던집니다.
5번은 세 번째로 1번에게 공을 던집니다.
1번은 네 번째로 3번에게 공을 던집니다.
3번은 다섯 번째로 5번에게 공을 던집니다.
입출력 예 #3
numbers: [1, 2, 3] ,k:3 ,result:2
1번은 첫 번째로 3번에게 공을 던집니다.
3번은 두 번째로 2번에게 공을 던집니다.
2번은 세 번째로 1번에게 공을 던집니다.

```
function solution(numbers, k) {
    let idx = 0
    for(let i = 1; i < k; i++){
        idx += 2;
        if(idx > numbers.length){
            idx %= numbers.length
        }
    }
    return numbers[idx];
}
```

```
// function solution(numbers, k) {
  let current = 0;
  for (let i = 0; i < k; i++) {
    current = (current + 2) % numbers.length;
  }
  return numbers[current];
}
```

```
function solution(numbers, k) {
    return numbers[(k - 1) * 2 % numbers.length];
}

function solution(numbers, k) {
    return numbers[(--k*2)%numbers.length];
}
```

k번째 던지는 사람의 수를 결정하는 데 다른 접근 방식을 사용합니다. 공의 던지기를 시뮬레이션하는 대신 이 솔루션은 k번째 던지는 사람의 위치를 직접 계산합니다. 다음 공식을 사용하여 이를 수행합니다. (k - 1) * 2 % numbers.length. 이 공식의 작동 방식은 다음과 같습니다. (k - 1) * 2: 이 식은 첫 번째 투구자(포지션 0에 있는)를 기준으로 k 번째 투구자의 위치를 계산합니다. 공은 한 사람에서 다음 사람에게 던지고 그 사이에 한 사람을 건너 뛰므로 k 번째 던지는 사람의 위치는 첫 번째 던지는 사람으로부터 (k - 1) * 2 위치가 됩니다. 예를 들어 k가 3이면 k번째 던지는 사람은 첫 번째 던지는 사람을 기준으로 위치 6에 있게 됩니다(3 * 2 = 6이므로). % numbers.length: 이 표현식은 숫자 배열의 끝 주변 위치를 래핑합니다. 위치가 숫자 배열의 길이보다 크거나 같으면 배열의 길이를 모듈로 줄입니다. 이는 숫자 배열이 원을 나타내므로 배열의 끝에 있는 사람이 처음에 있는 사람 옆에 있기 때문에 필요합니다.
예를 들어 첫 번째 입력 사례(숫자: [1, 2, 3, 4] ,k:2)에서 k번째 던지는 사람은 첫 번째 던지는 사람을 기준으로 위치 4에 있습니다((2 - 1) * 2 = 4이므로). 숫자 배열의 길이가 4이므로 이 위치는 0으로 둘러싸입니다(4 % 4 = 0). 그러면 함수는 위치 0(1)에 있는 요소를 반환합니다.

이 솔루션에 사용된 공식은 (--k*2)%numbers.length입니다. 이는 이전 솔루션에서 사용된 수식과 유사하지만 두 가지 차이점이 있습니다. --k: 이 표현식은 k를 1씩 감소시킵니다. 이것은 k가 실제 값보다 1 작은 것처럼 k번째 던지는 선수의 위치가 계산됨을 의미합니다. 예를 들어 k가 3이면 k번째 투구자의 위치는 k가 2인 것처럼 계산됩니다. (--k*2): 이 표현식은 감소된 후 k에 2를 곱합니다. 이것은 k번째 투구자의 위치가 k가 실제 값보다 2 작은 것처럼 계산된다는 것을 의미합니다. 예를 들어 k가 3이면 k번째 투구자의 위치는 k가 1인 것처럼 계산됩니다. 나머지 수식은 이전 솔루션과 동일한 방식으로 작동합니다. 모듈로 연산자(%)를 사용하여 숫자 배열의 끝 주위에 위치를 래핑합니다. 예를 들어 첫 번째 입력 사례(숫자: [1, 2, 3, 4] ,k:2)에서 k번째 던지는 사람은 첫 번째 던지는 사람에 비해 위치 2에 있습니다((1 - 1) * 2 = 2이므로)). 숫자 배열의 길이가 4이므로 이 위치는 2로 둘러싸입니다(2 % 4 = 2). 그러면 함수는 위치 2(3)에 있는 요소를 반환합니다.

```
import java.util.stream.IntStream;

class Solution {
    public int solution(int[] numbers, int k) {
        return IntStream.range(1, k).mapToObj(i -> numbers).flatMapToInt(IntStream::of).toArray()[2 * k - 2];
    }
}
```

```
import java.util.*;

class Solution {
    public int solution(int[] numbers, int k) {
        int answer = 0;

        Queue<Integer> queue = new LinkedList<>();
        for(int num : numbers) queue.add(num);

        int cnt = 1;
        while(cnt !=k){
            queue.add(queue.poll());
            queue.add(queue.poll());
            cnt++;
        }
        answer = queue.poll();
        return answer;
    }
}
```

이 솔루션은 Java의 Stream API를 사용하여 k번째 thrower의 수를 계산합니다.IntStream.range(1, k)는 1에서 k - 1까지의 정수 스트림을 생성합니다.mapToObj(i -> numbers)는 스트림의 요소(정수)를 정수(숫자)의 배열로 변환합니다.
.flatMapToInt(IntStream::of) 는 배열 스트림을 정수 스트림으로 병합합니다. toArray()는 스트림의 요소를 배열로 수집합니다.
[2 * k - 2]는 배열의 인덱스 2 * k - 2에 있는 요소에 액세스합니다. 결과 값은 k번째 던지는 사람의 번호입니다.예를 들어 첫 번째 입력 사례(숫자: [1, 2, 3, 4] ,k:2)에서 스트림은 [1, 2] 요소를 포함하고 배열은 [1, 2, 3, 4, 1, 2]. 이 배열의 인덱스 2 * 2 - 2에 있는 요소는 올바른 결과인 3입니다.

대기열은 숫자 배열의 요소로 초기화됩니다.

while 루프는 k번째 throw가 시뮬레이션될 때까지 계속됩니다. 루프의 각 반복에서:

queue.poll()은 대기열의 첫 번째 요소를 제거합니다. 이것은 공을 던지는 사람을 나타냅니다.

queue.add(queue.poll()) 대기열에서 다음 사람을 제거하고 대기열 끝에 추가합니다. 이것은 공을 받는 사람을 나타냅니다.

queue.add(queue.poll()) 대기열에서 다음 사람을 제거하고 대기열 끝에 추가합니다. 건너뛴 사람을 나타냅니다.

cnt는 시뮬레이트된 투구 횟수를 추적하기 위해 증가합니다.

while 루프가 끝나면 대기열의 첫 번째 요소(k번째 던질 때 공을 던질 사람)가 제거되고 반환됩니다.

예를 들어 첫 번째 입력 사례(숫자: [1, 2, 3, 4] ,k:2)에서 큐는 루프의 각 반복에서 다음과 같이 업데이트됩니다.

[2, 3, 4, 1] (1은 3에게 공을 던지고, 3은 공을 받고, 2는 스킵)
[4, 1, 2, 3] (3은 1에게 공을 던지고, 1은 공을 받고, 4는 스킵)
두 번째 반복에서 while 루프가 종료되고 대기열의 첫 번째 요소(4)가 결과로 반환됩니다.