Write an iterator that iterates through a run-length encoded sequence.

The iterator is initialized by RLEIterator(int[] A), where A is a run-length encoding of some sequence.  More specifically, for all even i, A[i] tells us the number of times that the non-negative integer value A[i+1] is repeated in the sequence.

The iterator supports one function: next(int n), which exhausts the next n elements (n >= 1) and returns the last element exhausted in this way.  If there is no element left to exhaust, next returns -1 instead.

For example, we start with A = [3,8,0,9,2,5], which is a run-length encoding of the sequence [8,8,8,5,5].  This is because the sequence can be read as "three eights, zero nines, two fives".

**Example 1:**

**Input:** ["RLEIterator","next","next","next","next"], [[[3,8,0,9,2,5]],[2],[1],[1],[2]]

**Output:** [null,8,8,5,-1]

**Explanation:**

RLEIterator is initialized with RLEIterator([3,8,0,9,2,5]).

This maps to the sequence [8,8,8,5,5].

RLEIterator.next is then called 4 times:

.next(2) exhausts 2 terms of the sequence, returning 8. The remaining sequence is now [8, 5, 5].

.next(1) exhausts 1 term of the sequence, returning 8. The remaining sequence is now [5, 5].

.next(1) exhausts 1 term of the sequence, returning 5. The remaining sequence is now [5].

.next(2) exhausts 2 terms, returning -1. This is because the first term exhausted was 5,

but the second term did not exist. Since the last term exhausted does not exist, we return -1.

**Note:**

1. 0 <= A.length <= 1000
2. A.length is an even integer.
3. 0 <= A[i] <= 10^9
4. There are at most 1000 calls to RLEIterator.next(int n) per test case.
5. Each call to RLEIterator.next(int n) will have 1 <= n <= 10^9.