Question: https://leetcode.com/problems/minimum-number-of-taps-to-open-to-water-a-garden/

The comments are self explanatory!

Code:

class Solution {

public int minTaps(int n, int[] ranges) {

// create an array where startToEnd[i] points to the max

// possible range of a tap starting at position i.

int[] startToEnd = new int[n + 1];

// in the first loop, set the start and end ranges

// within the startToEnd array, we can exit early here

// if we come across a range that covers all the way

// from 0 to n (there is only 1 tap needed in this case).

for(int i = 0; i <= n; i++){

int start = Math.max(0, i - ranges[i]);

int end = Math.min(n, i + ranges[i]);

if(start == 0 && end == n)

return 1;

startToEnd[start] = Math.max(startToEnd[start], end);

}

// Now the general case of this problem is to do 2 things:

//

// 1) validate that we can cover the entire garden with taps.

// 2) provide the minimum number of taps to provide full coverage.

//

// both of these cases can be achieved by starting at position 0

// in the garden, and moving forward till the end of the max position

// at the start, all the while gathering a potential next endpoint

// (the largest in the current range we are iterating through)

// We will do this until there are no more next endpoints to reach

// if the final endpoint we reach is n, return the total number

// of times we have changed endpoints (this is garunteed to be the

// minimum number of taps to fill the garden). If the final endpoint

// we reached was not n, then return -1 as we have proven that it is

// impossible to cover the entire garden with the given taps.

int currentEnd = startToEnd[0];

int count = 0;

int nextEnd = currentEnd;

int index = 0;

while(index <= currentEnd){

nextEnd = Math.max(nextEnd, startToEnd[index]);

if(index == currentEnd){

currentEnd = nextEnd;

count++;

}

index++;

}

return currentEnd == n ? count : -1;

}

}

Github Link :<https://lnkd.in/ecwtJeaz>