135. Candy

There are n children standing in a line. Each child is assigned a rating value given in the integer array ratings.

You are giving candies to these children subjected to the following requirements:

- Each child must have at least one candy.
- Children with a higher rating get more candies than their neighbors.

Return *the minimum number of candies you need to have to distribute the candies to the children*.

Example 1:

Input: ratings = [1,0,2]

Output: 5

Explanation: You can allocate to the first, second and third child with 2, 1, 2 candies respectively.

Example 2:

Input: ratings = [1,2,2]

Output: 4

Explanation: You can allocate to the first, second and third child with 1, 2, 1 candies respectively.

The third child gets 1 candy because it satisfies the above two conditions.

SOLUTION

CODE

```
class Solution {
public:
    int candy(vector<int>& ratings) {
        int n = ratings.size(), result = 1;
        int increment = 1;
        int descend = 0;

        int pre = 1;
        for (int i=1; i<n; i++) {
            if(ratings[i]>ratings[i-1]){
                // one more candy than former
                descend = 0;
```

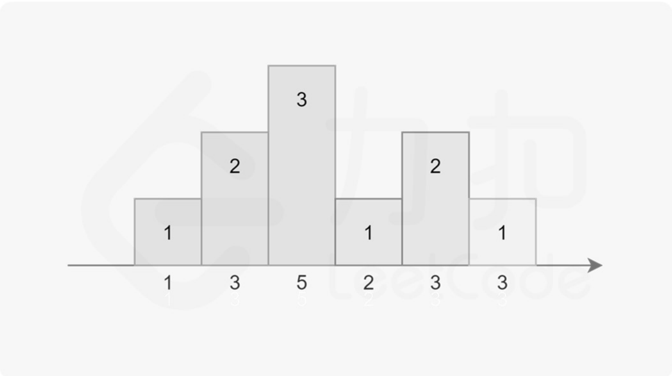
```
                pre = pre + 1;
                result += pre;
                increment = pre;
            }
            else if (ratings[i]<ratings[i-1]){
                descend++;
                if(descend == increment) {
                    descend++;
                }
                result += descend;
                pre = 1;
            }
            else {
                descend = 0;
                pre = 1;
                result++;
                increment = 1;
            }
        }
        return result;
    }
};
```

ANALYSIS

Time: O(n), Space: O(1).

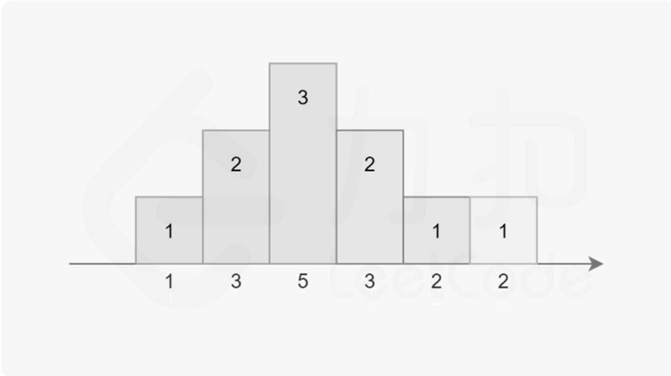
Each $ratings[i]$ could be bigger than, less than or equal to $ratings[i-1]$. When bigger, we just give $child[i]$ one more candy than $child[i-1]$ and then the settings could be satisfied. But when not bigger, it is more complex.

We can use variables $descend$ and $increment$ to record the length of current continuous decreasing and increasing sequences.

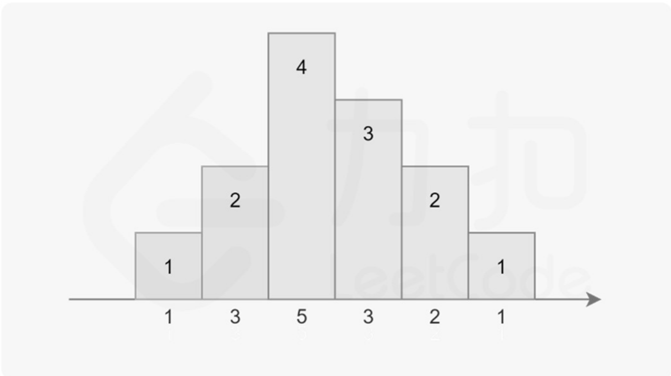


Every increasing sequence starts from 1 for less total amount of candies. In this way, when the increasing is interrupted, we always try to only give one to $child[i]$.

However, it’s not always reasonable.



In this condition, candy given to child[3] could not be 1, as it must exceeds which to child[4]. When continuous descent happens, we can use `result += descend` to solve this problem. Then it’s the circulus of this decreasing sequence.



Another example. As the length of decreasing sequence is bigger than that of increasing sequence, candy amount to `child[2]` must be changed to ensure each child could get at least one candy. There is a intriguing way: when `descend == increment`, we just add another `descend` to result and `descend` itself. In this case, when the last 1 was added to the program, descend would be 4 from 2 and the result would be 13(1+2+3+1+2+4) from 9(1+2+3+1+2). It results in expected effect with easier statement, although it is not the way we think of this problem.