2110. Number of Smooth Descent Periods of a Stock

Solved

Medium | Topics | Companies | Hint

You are given an integer array `prices` representing the daily price history of a stock, where `prices[i]` is the stock price on the `i^th` day.

A **smooth descent period** of a stock consists of **one or more contiguous** days such that the price on each day is **lower** than the price on the **preceding day** by **exactly** `1`. The first day of the period is exempted from this rule.

Return *the number of* ***smooth descent periods***.

Example 1:

```
Input: prices = [3,2,1,4]
Output: 7
Explanation: There are 7 smooth descent periods:
[3], [2], [1], [4], [3,2], [2,1], and [3,2,1]
Note that a period with one day is a smooth descent period by the
definition.
```

Example 2:

> **Input:** prices = [8,6,7,7]
> **Output:** 4
> **Explanation:** There are 4 smooth descent periods: [8], [6], [7], and [7]
> Note that [8,6] is not a smooth descent period as 8 - 6 ≠ 1.

Example 3:

> **Input:** prices = [1]
> **Output:** 1
> **Explanation:** There is 1 smooth descent period: [1]

Constraints:

- $1 <= prices.length <= 10^5$
- $1 <= prices[i] <= 10^5$

Python:

```
class Solution:
    def getDescentPeriods(self, prices: List[int]) -> int:
        ans = cnt = 0
        prev = -math.inf
        for cur in prices:
            if prev - cur == 1:
                cnt += 1
            else:
                cnt = 1
            ans += cnt
            prev = cur
        return ans
```

JavaScript:

```
function getDescentPeriods(prices) {
    let ans = 0;
    const n = prices.length;
    for (let i = 0, j = 0; i < n; i = j) {
        j = i + 1;
        while (j < n && prices[j - 1] - prices[j] === 1) {
```

```
            ++j;
        }
        const cnt = j - i;
        ans += Math.floor(((1 + cnt) * cnt) / 2);
    }
    return ans;
}
```

Java:

```
class Solution {
    public long getDescentPeriods(int[] prices) {
        int i=0;
        int j=1;
        long ans=1;
        while(j<prices.length){
            if( prices[j-1]-prices[j]==1){
                                //It means that j(current element) can be part of previous subarrays (j-i)
                                //and can also start a subarray from me (+1). So add (j-i+1) in total
Subarrays
                int count=j-i+1;
                ans+=count;
            }else{
                                //It means that j cannot be part of previous subarrays but can start
subarray from me. So, ans+=1
                i=j;
                ans+=1;
            }
            j++;
        }
        return ans;
    }
}
```