Python O(n) - slightly easier to grasp solution (explained)

sedim ★ 171 Last Edit: September 18, 2021 7:38 PM 5.4K VIEWS

165

Given the input sequence `A=[3,1,2,5,4]` we can cycle through array and write out the all the subarrays **ending** with th element.

```
[3]
[3,1], [1]
[3,1,2], [1,2], [2]
[3,1,2,5], [1,2,5], [2,5], [5]
[3,1,2,5,4], [1,2,5,4], [2,5,4], [5,4], [4]
```

We can denote by `result[i]` the sum of min values of those subarrays (ending with i-th element). Here are they:

```
3
1 + 1
1 + 1 + 2
1 + 1 + 2 + 5
1 + 1 + 2 + 4 + 4
result = [3,2,4,9,12]
```

We can notice a pattern

If `A[i-1] <= A[i]` then `result[i] = result[i-1] + A[i]`

It's easy to see why this happens: our subarrays ending with i-th value are basically same subarrays for (i-1)-th value w extra element A[i] added to each one of them and plus one extra subarray consisting of singular value A[i]. Adding sam or bigger value to subarrays doesn't change their minimal values. Thus we can reuse previous sum and account for tha extra singular subarray, thus `result[i] = result[i-1] + A[i]`

We can generalize

If we find previous less or equal value `A[j] <= A[i] (j<i)` then `result[i] = result[j] + A[i]*(i-j)`

Let's consider our previous example to see why. Let's take `i=4` and look at subarrays for the element `A[i]=4` : `[3,1,2,5,4], [1,2,5,4], [2,5,4], [5,4], [4]`

First part of these subarrays look similar to subarrays generated previous less value `2` at `j=2` as if we took those subarrays (`[3,1,2], [1,2], [2]`) and added elements `[5,4]` to each of them. Sum of those subarrays equals `result[j]`.

The rest of our i-th subarrays consist of elements after `j` and up to `i` : `[5,4], [4]` . They are not less then our element `4` so their sum equals to `(i-j)*A[i]`

Together these two observations give us formula `result[i] = result[j] + A[i]*(i-j)`

Solution

This forms the basis of the solution: we build monotonously incressing (well, strictly speaking - non-decreasing) stack. And then find previous less or equal value and reuse it's sum:

(trick: we add zeros to A and stack to avoid dealing with empty stack)