LeetCode 903 — Sum of Subarray Minimums

Teerth Patel — tort.txs

1 Formulation

Let the given array be A with length N.
Define $m(i,j)$ as $\min_{i \leq k \leq j} A[k]$: the minimum of the subarray starting at index i and ending at index j.
Our goal is essentially to find

$$\sum_{j=0}^{N-1} \sum_{i=0}^{j} m(i,j),$$

which is the sum of $m(i,j)$ for all pairs of subarray endpoints (i,j).

Let us call the inner sum for a certain j value $f(j)$, where

$$f(j) = \sum_{i=0}^{j} m(i,j),$$

so the problem then becomes calculating

$$\sum_{j=0}^{N-1} f(j).$$

2 Computation of f(j)

We need to find an efficient way to compute $f(j)$, meaning we should try to reuse previous results somehow. (I found this method by thinking about the idea of "peaks" which are higher than the current element being safely ignorable)

We will tackle this problem by defining the integer n_j to be the largest integer such that $n_j < j$ and $A[n_j] \leq A[j]$.
If there is no such number, then clearly $A[j]$ must be the smallest value in between the indices 0 and j, so $m(i,j) = A[j]$ when $0 \leq i \leq j$, and thus

$$f(j) = \sum_{i=0}^{j} m(i,j) = \sum_{i=0}^{j} A[j] = (j+1) \times A[j]$$

Otherwise we can then split $f(j)$ into the sum

$$f(j) = \sum_{i=0}^{j} m(i,j) = \sum_{i=0}^{n_j} m(i,j) + \sum_{i=n_j+1}^{j} m(i,j).$$

We replace $m(i, j)$ with $\min\{m(i, n_j), m(n_j+1, j)\}$ for the left hand sum (since we can compute minimum of a range from the minimum of minimums of subranges).

$$f(j) = \sum_{i=0}^{n_j} \min\{m(i, n_j), m(n_j+1, j)\} + \sum_{i=n_j+1}^{j} m(i, j).$$

Now remember that n_j is the *maximum* number such that $n_j < j$ and $A[n_j] \leq A[j]$, so for each k where $n_j < k < j$, we must have $A[k] > A[j]$, and thus we can substitute $m(i, j) = A[j]$ when $i > n_j$.

$$f(j) = \sum_{i=0}^{n_j} \min\{m(i, n_j), A[j]\} + \sum_{i=n_j+1}^{j} A[j] = \sum_{i=0}^{n_j} \min\{m(i, n_j), A[j]\} + (j - n_j) \times A[j].$$

Clearly $m(i, n_j) \leq A[n_j]$ when $i \leq n_j$ (as the minimum of a range must be less than or equal to every element in the range); and since $A[n_j] \leq A[j]$, we know $m(i, n_j) \leq A[n_j] \leq A[j]$, so $\min\{m(i, n_j), A[j]\} = m(i, n_j)$. Hence we can write:

$$f(j) = \sum_{i=0}^{n_j} m(i, n_j) + (j - n_j) \times A[j] = f(n_j) + (j - n_j) \times A[j]$$

3 Computation of n_j and Wrap-up

We must effectively compute n_j as we go along. In order to do so, we use a *monotonically increasing stack*, which has the special property that the elements are always listed in increasing order if we maintain it correctly .

Let us call this stack S, and have it contain indices in the array A.
As we traverse over j, we should perform the operation

```
while (not S.empty) and (A[S.top] > A[j]):
    S.pop()
if not S.empty:
    i = S.top
    f(j) = f(i) + A[j] * (j - i)
else:
    f(j) = A[j] * (j + 1)
S.push(j)
```

This is because:
If $A[i] > A[j]$, then $n_j \neq i$,
If $k > i$ then $n_k \neq i$ because if $n_k = i$, then $A[i] \leq A[k]$ would be true, so $A[j] < A[i] \leq A[k]$; however as $j > i$, we see i is not the maximum possible index such that $i < k$ and $A[i] \leq A[k]$, which is a contradiction. Hence such an i is useless for current and future purposes, and thus our sequence of pops is valid

At the end of the sequence, the value on top of the stack (if any) will be n_j. This is because we have at some point pushed every number between 0 and $j - 1$, eliminating all those which are not valid candidates, and the numbers are inserted in increasing order, so the maximum index will be at the top.

We simply run the formula on this n_j if it exists, or the alternative formula if there is no n_j, and then push j onto the stack since it may be a candidate for future n_k.

And then we add $f(j)$ over $j = 0$ to $N - 1$ as specified earlier.