Editorial: Serval and Toxel's Arrays - 1789C

Avighna

June 2024

We're asked to find the sum of the values of all pairs (i, j) where $0 \leq i < j \leq m$. Let's rewrite this as the sum of the values of all pairs with $i = 0$ plus the sum of the values of all pairs with $i = 1$, and so on till $i = m - 1$.

If we're able to find a way to calculate the first sum, and then a way to quickly transition between the sum when $i = x$ and the sum when $i = x + 1$, then we can add all these sums to obtain our answer.

Let the sum of the values of all pairs (i, j) with $i < j \leq m$ for some fixed value of i be denoted by s_i. As stated previously, our answer is $\sum_{i=0}^{m-1} s_i$. s_i itself is the sum of the values of the pairs $(i, i+1), (i, i+2), ..., (i, m)$. Let $v_{(a,b)}$ denote the value of the pair (a, b). Now let's create an array called x_i where its j^{th} element (denoted by $x_{i,j}$) is equal to $v_{(i,i+j)}$ ($x_i = [v_{(i,i+1)}, v_{(i,i+2)}, ..., v_{(i,m)}]$). Let's insert the element n behind the first element of this array: the first element is $x_{i,1}$, so $x_{i,0} = n$. Now let's find this array's adjacent difference, call this Δ_i. $\Delta_{i,j} = x_{i,j} - x_{i,j-1}\ \forall j \in [1, m - i]$.

Here's where all these definitions become useful. $s_i = \sum_{j=1}^{m-1} x_{i,j} = \sum_{j=1}^{m-1} \sum_{k=0}^{j} \Delta_{i,k}$ (by definition), so $\Delta_{i,j}$ represents the change in the number of unique values when the array formed by the concatenation of A_i and A_{j-1} morphs to the array formed by the concatenation of A_i and A_j.

Described ahead is an algorithm that computes $\Delta_{0,j}$. Imagine there being 'slots' which you use to swap out one number for another when going from A_0 to A_1, or from A_1 to A_2, for example. We say that a slot is unused if we haven't already used it to swap a number.

$$A_0 = [1, 0, 3, 4, 5] \rightarrow A_1 = [1, 6, 3, 4, 5]$$

In the above example, we've replaced a 0 with a 6 via slot 2, which was previously unused. If we then did

$$A_1 = [1, 6, 3, 4, 5] \rightarrow A_2 = [1, 9, 3, 4, 5]$$

then we'd have replaced a 6 with a 9 via the already used slot 2.

To calculate the change in the number of unique elements when the array we're concatenating A_0 with goes from A_0 to A_1, then A_1 to A_2, then A_2 to A_3 and so on; in this case, from A_0 to A_i we need to notice that if we swap out a to b and b isn't present in A_0, we add another element to the concatenation. So we add 1 to $\Delta_{0,i}$. If we add this element from a used slot, this means that we'd also added an element previously that we're now removing, so we need to make sure that we subtract 1 from $\Delta_{0,j}$ if this element was also absent from A_0.

Let's now digress and look at what happens when you find the prefix sum of the prefix sum of an array (since s_i is defined this way in terms of Δ_i)

$[a, b, c, d]$ is the original array
$[a, a + b, a + b + c, a + b + c + d]$ is the first prefix sum

$[a, 2a+b, 3a+2b+c, 4a+3b+2c+d]$ is the second prefix sum

Using this notion, adding x to the concatenation of A_0 and A_i can be thought of as x contributing a value of $m-i+1$ to s_i when it is absent from A_0. Thus s_i is equal to the sum of mn and the contributions of all numbers from 0 to $m+n$ (why are we adding mn? Just adding the contributions of all numbers doesn't account for the fact that $\Delta_{0,0} = n$ (actually, $\Delta_{i,0} = n \; \forall i \in [0, m-1]$). Δ_i has $m+1$ elements, and we're summing up $x_i[1...m]$, not $x_i[0...m]$, so we're subtracting n from the sum of $x_i[0...m] = m(n+1)$).

Now that we've found s_0, let's use its value to find s_1, then use s_1's value to find s_2 and so on, adding them along the way. To do this, we have to find a way to (implicitly) obtain Δ_i from Δ_{i-1} (in reality we're just adding and subtracting things from s_0) and then repeat the 'second prefix sum' summing procedure to subtract and add values to s_0.

Here's an example of a transition:

0.1 Test case

```
10 10
4 6 9 12 16 20 2 10 19 7
1 3
5 4
2 17
2 18
6 11
7 1
8 17
5 5
5 5
2 2
```

$$\Delta_0 = \begin{bmatrix} 10, & \\ 1, & \text{Insert 3 in slot 1} \\ 0, & \text{Insert 4 in slot 5} \\ 1, & \text{Insert 17 in slot 2} \\ -1+1, & \text{Remove 17 from slot 2, insert 18 in slot 2} \\ 1, & \text{Insert 11 in slot 6} \\ 1, & \text{Insert 1 in slot 7} \\ 1, & \text{Insert 17 in slot 8} \\ -0+1, & \text{Remove 4 from slot 5, insert 5 in slot 5} \\ -1+1, & \text{Remove 5 from slot 5, insert 5 in slot 5} \\ -1+0 & \text{Remove 18 from slot 2, insert 2 in slot 2} \end{bmatrix}$$

We implicitly convert this to Δ_1 by removing the first element, adjusting the second element to be n, and subtracting $m-i+1$ from the contribution of the element added (adding for the element removed) in the second row, since in Δ_1, our original array is no longer A_0, it's A_1 (so it's no longer contributing, it's literally part of the array). Now adjust s_0 by subtracting the contributions of the two elements (or one, if you're not removing anything) that change during the transition if they're present, update

it so that the value you just replaced (in our case, we replaced 4 at the first position of our array with 3; we went from $[4, 6, 9, 12, 16, 20, 2, 10, 19, 7]$ to $[3, 6, 9, 12, 16, 20, 2, 10, 19, 7]$) is now marked as absent and the value you replaced it with is marked as present, and then add back the contributions of these two elements if they're present. Doing this will successfully give you s_1.

$$\Delta_1 = \begin{bmatrix} 10, & \\ 1, & \text{Insert 4 in slot 5} \\ 1, & \text{Insert 17 in slot 2} \\ -1+1, & \text{Remove 17 from slot 2, insert 18 in slot 2} \\ 1, & \text{Insert 11 in slot 6} \\ 1, & \text{Insert 1 in slot 7} \\ 1, & \text{Insert 17 in slot 8} \\ -1+1, & \text{Remove 4 from slot 5, insert 5 in slot 5} \\ -1+1, & \text{Remove 5 from slot 5, insert 5 in slot 5} \\ -1+0 & \text{Remove 18 from slot 2, insert 2 in slot 2} \end{bmatrix}$$

Note that the real Δ_1 array is never found by our algorithm (although the sum of all elements of its prefix sum excluding the first is (s_i)), I've provided it here solely for reference (the algorithm wouldn't be fast enough if it actually found Δ_1).

1 Code

Submission