Hands-on 2: Segment tree

Eduardo Venturini

Problem 01: Min and Max

The solution to this problem uses a Segment Tree with lazy propagation.

Every node of the tree contains two values: `d`, the maximum value in the subtree before applying the propagation, and `e`, the value of an update query that needs to be propagated in every node of the subtree. In particular, the `e` - value of a node with no query waiting to be propagated is $+\infty$.

When an update query is called, the `e` - values of the nodes in the corresponding range are updated, then also the `d` - values of the nodes whose range is not completely inside nor outside the query range are updated. So, in this operation, $O(\log(n))$ nodes are updated.

When a max query is called, the `e` - values of the upper nodes are propagated until reaching the nodes in the range of the query, resulting in $O(\log(n))$ operations. Then, again with $O(\log(n))$ operations, we get the max value of the nodes defining the range.

Since building the tree needs $O(n)$ operations, the total time complexity is $O(n + m\log(n))$, where m is the number of queries.

Regarding the space complexity, the segment tree needs $\Theta(n)$ nodes, so the space complexity is $\Theta(n)$.

Problem 02: Queries and Operations

The solution to this problem doesn't use a segment tree, but can easily be modified to use one for a more complex code and a worse performance.

First of all, we notice that we can perform the queries and the operations of each query in every order. So the first step is to count how many times we need to perform each operation. Considering a vector containing the differences of this number for consecutive operations, for each query we need to update just 2 numbers. At the end of this process, we obtain the vector with the number of times each operation needs to be performed with a cumulative sum. The time complexity of these steps is $\Theta(m + k)$.

Next, we notice that performing an operation with value t for k times is the same as performing that operation with value kt once, so we need to perform only m operations with the new values. To achieve this last step, we can proceed as in the previous step: we consider the vector of differences of the initial array, then apply all the operations and finally recover the new array with a cumulative sum. As before, the time complexity is $\Theta(n + m)$.

The total time complexity is $\Theta(n + m + k)$.

For the space complexity, we need $\Theta(n + m)$ additional space for the difference vectors (although it's possible to build the second difference vector in-place).

Regarding the use of a segment tree, it's possible to substitute both the difference vectors with segment trees with lazy updates, whose implementation is very similar to the ones in the first problem. In this case, the space complexity remains the same, while the time complexity is

$$\begin{aligned}
&= \Theta(m + k\log(m) + n + m\log(n)) \\
&= \Theta(k\log(m) + m\log(n) + n) \\
&\leq \Theta((k + m)\log(n) + n)
\end{aligned}$$