Handson 2

Sofia Pisani

November 25, 2023

1 Segment Tree

We implemented our own segment tree which we used as a base for both of the given problems. Our segment tree is contained in an array of size 2k-1, where k is the power of 2 successive to n. As such we only work on complete binary trees, accepting that some nodes will be left uninitialized for the sake of simplicity. This only adds a linear component to the space complexity as the resulting tree will be of at most 4n nodes.

2 Min and Max

We used a segment tree containing the local maximums of the array. Each node also contains a minimum field initialized to $+\infty$, and each maximum query returns the minimum between its current result and its minimum. In this way we can handle the update query lazily by inserting the minimum at the highest node covering the given range, making sure to update all parents maximums afterwards. The space complexity is linear, as the tree contains at most 4n nodes, and each node uses constant space. The time complexity of the solution is $O((m)logn + n)$ as our initialization step is linear, touching each node only once, and each query is $O(logn)$, as segment tree queries.

3 Is There

We used a segment tree built over a base array going from 0 to n-1. The base array contains the number of overlapping segments in each point. Each node of the segment tree stores an HashSet, which contains all covered ks. In this way we can answer an IsThere query by checking if the given k is contained in the covering nodes, which is done in constant time thanks to the HashSet. The base array is built in linear time from the given ranges.

The running time is $O(logn)$ per query, as each query just necessitates visiting the correct covering nodes and checking in constant time if the node we are looking for is present in the range. The construction of the segment tree is also linear, touching each node only once.

The space complexity is slightly higher, at $O(nlogn)$. While it may seem that each node requires n additional space to store a full HashSet, we can note that a node covering x base squares can contain at most x different values. As such, a node covering half of the base array will only require n/2 space and so on. In this way we can see that at each level of the segment tree, adding together all nodes of the same height, we have at most a full copy of the underlying array, getting us to our $O(nlogn)$ space complexity.