Hands On 2

Competitive Programming and Contests

Leonardo Crociani (Mat. 615392)

Contents

1 Hands On 2 **2**

1.1 Introduction 2

1.2 Min Max Problem 2

1.3 IsThere Problem 3

1.4 Source code and tests 4

1.5 References 4

1 Hands On 2

1.1 Introduction

The solutions for both problems have been implemented using Segment Trees with lazy propagation. The data structures implemented are designed specifically for `i32` numbers.

1.2 Min Max Problem

Generic trees are naturally recursive data structures. For this reason, to solve this problem, I implemented a `SegmentTree` struct that could be defined recursively.
A `SegmentTree` struct has the following fields:

- `key` (The value in the node)
- `lazy` (The lazy value for that node. Initially, `lazy = key`)
- `range` (The range of applicability of that node)
- `left` (The optional left child)
- `right` (The optional right child)

Once the tree is instantiated, each query is executed as one would normally do on a segment tree, with the only consideration that the `lazy` field in each node's children contains the minimum between the update value and the previous value. The construction of the tree requires

$$O(n) \tag{1}$$

Each query (`Update()` or `Max()`), due to the properties of the tree, requires

$$O(\log(n)) \tag{2}$$

Therefore, the overall complexity is:

$$O(n + m \cdot \log(n)) \tag{3}$$

where `n` is the size of the initial array and `m` is the number of queries.

1.3 IsThere Problem

For this problem, I found that an array-based implementation could have been more suitable. The idea is to start with a 0-initialized Segment Tree and consider each segment as an update of 1 in its range. At the end of the updates, every node will have the maximum number of overlapping segments in its range. Every node is also a struct with two fields:

- `key`
- `lazy` (Initialized with 0)

Each `Update()` or `IsThere()` query takes

$$O(\log(n)) \tag{4}$$

So, given `n` segments and `m` `IsThere()` queries, the overall complexity is:

$$O((n + m) \cdot \log(n)) \tag{5}$$

1.4 Source code and tests

In the .zip I've provided, you can find `main.rs` (used for testing) and `lib.rs` (the actual implementation of the data structures and algorithms) in `/hands-on/2/`. There is also another file `no_lazy.rs` in which I implemented the exercises also in a non-lazy approach. To run the tests, open a terminal in the code folder and execute `cargo run`. Please note that you will need to initialize the code folder by running `cargo new <any_name>` first. The code expects to find a folder named `tests` with two subfolders inside: `1` and `2`. Each of these subfolders should contain the pairs of .txt test files.

1.5 References

To quickly review the structure of Lazy Segment Trees studied in class, I have consulted Tushar Roy's explanation on YouTube, which can be found at this link.