SEGMENT TREE

Date: 17-06-2020

1. Segment tree is used in the cases where there are multiple range queries on an array and also modifications of the same array. The time complexities are as follows:
    a. Query : O(log N)
    b. Update : O(log N)

2. The reason for using segment tree is, there are approaches by which only either Update or Query functions can be made in O(1), but the complexity for other operation would be O(N). And for cases where N is very high, this may not be suitable.

3. This is where segment tree proves to be useful, It uses at most O(4N) space, and O(log N) time for both the functions and hence is preferred in many cases.

4. Queries include:
    a. Finding sum within a given range

    b. Finding max/min element within a given range etc.

5. So, with all that said, what exactly is a segment tree??

    a. Segment tree is a binary tree, where each node represents an interval.

    b. Consider an array A of size N, and a corresponding Segment Tree T:
        i. Root of T will represent whole array A[0: N-1]
        ii. Each leaf in the segment tree represents the array element
        iii. The internal nodes in T represents the union of the child nodes.

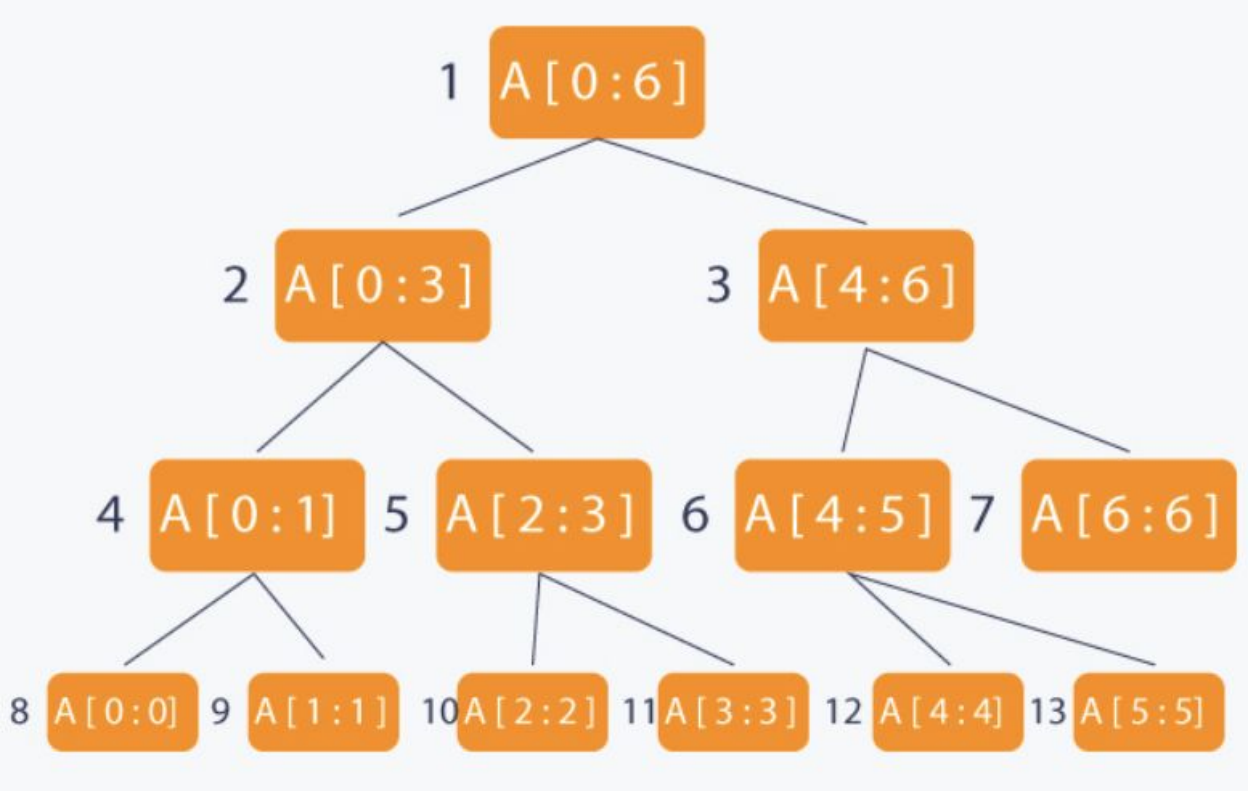
    c. Initially, root represents the interval A[0: N-1]. Then it's broken down into two half intervals or segments. The child nodes represent A[0: (0+(N-1))/2] and A[(0+(N-1))/2 +1: (N-1)]

    d. That is, the partition is calculated by mid = (low+high)/2. The child nodes then become [low,mid], and [mid+1, high]

    e. Since at each step, the segment is divided into half, the height of segment tree will be log2N.

    f. Total number of nodes: N leaf nodes representing the array elements + N-1 internal nodes = 2*N-1

g. Once the segment tree is built, it's structure cannot be changed, but the values can only be updated.



6. Implementation of a segment tree:
   Since a segment tree is a binary tree, a simple linear array is enough to represent the segment tree.

7. Implementation process:

   a. First, we start at the root node. We need to recursively travel till the leaf nodes, by dividing the interval into half at each step.
   b. Each of the leaf node represents the array elements.
   c. After reaching the leaf node, we backtrack. Now, while backtracking we carry the property that we want to store in the nodes. For example, if it's the maximum value in a range that we want to find, we recursively backtrack with the maximum value of the two child nodes.
   d. After reaching the root node again, we have a segment tree with us… Voila :)

8. Querying process: (Ex: max value within a given range) To query on a given range, we need to check three conditions:

    a. Range represented by the node is inside the query range: RETURN VALUE OF NODE

    b. Range represented by the node is completely outside the query range: RETURN INT_MIN (in case of max query)

    c. Range represented by the node is partially inside and partially outside the query range: RETURN MAX OF LEFT CHILD AND RIGHT CHILDE (TRAVERSE IN BOTH THE DIRECTIONS)

9. Updating process:

    a. Given the index of the node at which the element is to be updated, and the value to add.
    b. Start with the root node, look at the interval in which the element is present and recurse accordingly on the left or the right child.
    c. Finally when low=high=index given (when leaf node is reached). Stop and update the value at that index.
    d. When recursively traversing up, this updated value is propagated upwards to other range values.