# Segment Trees

Topics to be covered -

- Build, point-update, range-query. Basic min, max, sum segment tree.
- Problem: https://www.spoj.com/problems/GSS3/
- Introduction to merge sort-tree.

Segment tree is a powerful data structure that allows the following:

- Building the tree in O(N)
- Range based queries on any associative function in O(logN)
- Updating a specific node in O(logN)
- Updating a range of values

  Note - All the above complexities assume that merging two children nodes takes constant time.

## Implementation

Segment tree is a binary tree. We can use an array to represent it.

Each node stores the value of the query function for a specific range [l, r].

The left child stores that value for the lower half [l, (l + r)/2].

The right child stores that value for the upper half [((l + r) / 2) + 1, r].

In array representation, the index for the left child for node `i` will be `2*i+1` and that for the right child will be `2*i+2`, when using 0-based indexing.

## Building the tree:

We recursivley build the tree, starting from the leaves to the root.

pseudocode:

```
s: start index of range
e: end index of range
c: current index
arr: the array containg the values
f: Any merge function (sum, max etc.)


build(s, e, c):
    //  Reached leaf
    if s == e:
        segtree[c] = arr[s]
    else:
        //  build left subtree
        build(s, mid, 2*c+1)
        //  builf right subtree
        build(mid+1, e, 2*c+2)

        segtree[c] = f(segtree[2*c+1], segtree[2*c+2])
```

## Query on the tree:

For querying each and every value, 3 cases arise:

- node’s range is completely within query range
- node’s range is completely outside query range
- node’s range is partially inside and partially outside the query range

If the range is completely outside, we need a specific value [call it anti] such that0 `f(anti, v) == v`, for every `v`.
If the range is partial, we recurse for the query in both left and right halves.

pseudocode:

```
s, e, c: same as above
[l, r]: query range
query(s, e, l, r, c):
    //  Completely outside
    if r < s or l > e:
        return anti

    //  Completely within
    if l <= s and r >= e:
        return segtree[c]

    //  Partial overlap
    //  Left half
    q1 = query(s, mid, l, r, 2*c+1)
    //  Right half
    q2 = query(mid+1, e, l, r, 2*c+2)

    return f(q1, q2)
```

## Updating a specific value:

To update an element, we look if it falls in the left or right half and then update it’s parents with the new value
Note that every node that represents a range that contains a single value [i, i] ***must*** be a leaf

pseudocode:

```
s, e, c, arr: same as above
i: index which we want to update
v: the new value of that index i
update(s, e, i, v, c):
    //  Reached that node!
    if s == e:
        arr[i] = v
        segtree[c] = v
    else:
        //  Value in lower range [left child]
        if i <= mid:
            update(s, mid, i, v, 2*c+1)
        //  Value in higher range [right child]
        else:
            update(mid+1, e, i, v, 2*c+2)

        //  Values of children are updated, update current
node's value now
        segtree[c] = f(segtree[2*c+1], segtree[2*c+2])
```

## Practice [SPOJ GSS3]

### Problem:

You are given a sequence A of N (N <= 50000) integers between -10000 and 10000. On this sequence you have to apply M (M <= 50000) operations:

- modify the i-th element in the sequence or for given x y
- print max{Ai + Ai+1 + … + Aj | x<=i<=j<=y }.

### Input:

The first line of input contains an integer N. The following line contains N integers, representing the sequence A1…AN.

The third line contains an integer M. The next M lines contain the operations in following form:

- 0 x y: modify Ax into y (|y|<=10000).
- 1 x y: print max{Ai + Ai+1 + ... + Aj | x<=i<=j<=y }.

## Output:

For each query, print an integer as the problem required.

## Idea:

For every node in the segment tree, we have to keep track of 4 values in such a way that each of them can be calculated for the parent from it's children without any extra calculations

The values we choose to store are:

- sum
- maximum prefix sum
- maximum suffix sum
- maximum subarray sum

Dividing into sections:

1. Build segment tree
2. query on segment tree
3. point update a node

### Build:

Using the child’s values, we can directly calculate the values for the parent

How?

- p.sum = l.sum + r.sum
- p.maxL = max(l.maxL, l.sum + r.maxL)
- p.maxR = max(r.maxR, l.maxR + r.sum)
- p.maxS = max(r.maxS, l.maxS, l.maxR + r.maxL)

Thus, segment tree building part is over.

## Query:

This part is a bit tricky.
Instead of returning a single value at every step, we return a struct [or whichever way the values were stored] which contains the information [sum, maxL, maxR, maxS] for that specific search space.

Terminology used:

- query range: The space for which we have to calculate the answer
- search space: The space which we are looking at currently [current node’s]

How?

- If the search space and range coincide, return current node

- If the query range completely lies in the lower half of current space, recur for left child
- If the query range completely lies in the higher half of current space, recur for right child
- Else if query range is divided, recur in both halves and get a node that has the said properties.
  How?
    - ret.sum = l.sum + r.sum
    - ret.maxL = max(l.maxL, l.sum + r.maxL)
    - ret.maxR = max(r.maxR, l.r + r.sum)
    - ret.maxS = max(ret,maxL, ret.maxL, l.maxS, r.maxS, l.maxR + r.maxL)

This ends the query part

## Updation:

Here, the same concepts are used.

While updating, everything is same except calculation of values for the parent once we have the values for the child.
This calculation is done in the same way as we did in the query part.

With this, we come to the end of updation.

Credits - Yoogottam Khandelwal