Segment Tree ( Problem 2 )

**Problem**

Given an array **A** of **N** elements. There are two type of queries :

* Update value at index **i** by **X**
* Given a range **L ... R** find **max(A[i]+A[i+1]+...+A[j]) L ≤ i ≤ j ≤ R**. In other words find the subarray in range **L ... R** whose sum is maximum.

**Approach**

Consider we are given a range **L ... R** for which we need to answer 2nd query. Suppose we break this range into two parts **L ... M** and **M+1 ... R** where **M = (L+R)/2**. Suppose we have a range **i ... j** (such that **L ≤ i ≤ j ≤ R**) whose sum is maximum. One of the following case is possible :

* **L ≤ i ≤ j ≤ M**
* **M+1 ≤ i ≤ j ≤ R**
* **L ≤ i ≤ M** and **M+1 ≤ j ≤ R**

Hence answer for given range will be maximum of the following values:

* Answer for range **L ... M**.
* Answer for range **M+1 ... R**.
* Maximum suffix sum of range **L ... M** + Maximum prefix sum of range **M+1 ... R**

**Structure**

From above discussion we know that for each range we need to calculate required answer, maximum prefix sum, maximum suffix sum and total sum of that range (will be needed to calculate maximum prefix and suffix sum).Hence following will be the structure :

struct node

{

int answer, prefixSum, suffixSum, totalSum;

} segTree[4\*N]; // size of segment tree is 4 times size of given input

**Build Tree**

Calculating answer for current node is discussed above.   
Maximum prefix sum = max (Maximum prefix sum of left child, Total sum of left child + Maximum prefix sum of right child)   
Maximum suffix sum = max (Maximum suffix sum of right child, Total sum of right child + Maximum suffix sum of left child)   
Total sum = Total sum of left child + Total sum of right child  
Following is code snippet to build segment tree :

node merge(node X, node Y)

{

node Z;

Z.answer = max(max(X.answer, Y.answer), X.suffixSum + Y.prefixSum);

Z.prefixSum = max(X.prefixSum, X.totalSum + Y.prefixSum);

Z.suffixSum = max(Y.suffixSum, Y.totalSum + X.suffixSum);

Z.totalSum = X.totalSum + Y.totalSum

return Z;

}

// rangeLeft will be left most index of current range

// rangeRight will be right most index of current range

// index will be location of current node in array of segTree

void build\_tree (int rangeLeft, int rangeRight, int index)

{

if (rangeLeft == rangeRight) {

segTree[index].answer = A[rangeLeft];

segTree[index].prefixSum = A[rangeLeft];

segTree[index].suffixSum = A[rangeLeft];

segTree[index].totalSum = A[rangeLeft];

return;

}

int mid = (rangeLeft + rangeRight) / 2;

build\_tree(rangeLeft, mid, 2\*index);

build\_tree(mid+1, rangeRight, 2\*index+1);

segTree[index] = merge(segTree[2\*index], segTree[2\*index+1]);

}

**Update Tree**

We will start from root node and try to reach the leaf node of the index at which we need to update the value. After reaching leaf node we will update value of all the nodes which were visited during traversal from root to leaf.  
Following is code snippet to update segment tree :

// updateIndex will be the index whose value is needed to be updated in original array.

// newValue will be new value which is needed to be updated at updateIndex

// rangeLeft will be left most index of current range

// rangeRight will be right most index of current range

// index will be location of current node in array of segTree

void update\_tree (int updateIndex, int newValue, int rangeLeft, int rangeRight, int index)

{

if (rangeLeft <= updateIndex && updateIndex <= rangeRight) {

if (rangeLeft == rangeRight) {

segTree[index].answer = newValue;

segTree[index].prefixSum = newValue;

segTree[index].suffixSum = newValue;

segTree[index].totalSum = newValue;

return;

}

int mid = (rangeLeft + rangeRight) / 2;

update\_tree(rangeLeft, mid, 2\*index);

update\_tree(mid+1, rangeRight, 2\*index+1);

segTree[index] = merge(segTree[2\*index], segTree[2\*index+1]);

}

}

**Query Tree**

* Start from root and check if current range completely lies within query range than return current node's required value.
* If current node completely lies outside the query range than return node with answer, prefixSum, sufixSum, totalSum as -∞.
* Else query on current node's children, merge their answers and return.

Following is code snippet to query segment tree :

// queryLeft will be left most index of current query's range

// queryRight will be right most index of current query's range

// rangeLeft will be left most index of current range

// rangeRight will be right most index of current range

// index will be location of current node in array of segTree

node get\_answer(int queryLeft, int queryRight, int rangeLeft, int rangeRight, int index)

{

if (queryRight < rangeLeft || rangeRight < queryLeft || queryRight < queryLeft) {

node tmp;

tmp.answer = tmp.prefixSum = tmp.suffixSum = tmp.totalSum = INT\_MIN;

return tmp;

}

if (queryLeft <= rangeLeft && rangeRight <= queryRight)

return segTree[index];

int mid = (rangeLeft + rangeRight) / 2;

return merge(get\_answer(queryLeft, queryRight, rangeLeft, mid, 2\*index), get\_answer(queryLeft, queryRight, mid+1, rangeRight, 2\*index+1));

}