Segment Tree ( Problem 3 )

**Problem**

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

* Add value **X** over a range from **L ... R**
* Find sum of all elements in a range from **L ... R**

**Structure**

It is simple to understand that we need to store only sum in segment tree. But since there is a range update therefore we need **lazy** also which will be of same size of segment tree. If lazy of **ith** index is not zero than it means we need to add **lazy[i]** to all the elements present in that range.Hence following will be the structure :

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

int lazy[4\*N]; // size of lazy is same as segment tree

**Build Tree**

* Initially lazy of each node will be 0.
* If current node is leaf node(only one element is present in that range) then sum will be equal to that single element.
* Else sum of current node = sum of left child + sum of right child.

Following is code snippet to build segment tree :

// 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)

{

lazy[index] = 0;

if (rangeLeft == rangeRight) {

segTree[index] = A[rangeLeft];

return;

}

int mid = (rangeLeft + rangeRight) / 2;

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

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

segTree[index] = segTree[2\*index] + segTree[2\*index+1];

}

**Update Tree**

Start from root node and follow steps given below.

* If current node's range is completely outside update query's range then return.
* If current node's lazy is not 0 then update current nodes value, update current node's children's lazy and mark current node's lazy as 0.
* If current node's range completely lies inside update query's range then update current node's value and update its children's lazy value.
* Else update both of it's children and then update current node's value as sum of both of it's children's value.

Following is code snippet to update segment tree :

// updateLeft will be left most index of update range

// updateRight will be right most index of update range

// newValue will be new value which is needed to be added to updateRange

// 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 updateLeft, int updateRight, int newValue, int rangeLeft, int rangeRight, int index)

{

if (updateRight < rangeLeft || rangeRight < updateLeft)

return;

if (lazy[index] != 0) {

segTree[index] += lazy[index]\*(rangeRight - rangeLeft + 1);

lazy[2\*index] += lazy[index];

lazy[2\*index+1] += lazy[index];

lazy[index] = 0;

}

if (updateLeft <= rangeLeft && rangeRight <= updateRight) {

segTree[index] += newValue\*(rangeRight - rangeLeft + 1);

lazy[2\*index] += newValue;

lazy[2\*index+1] += newValue;

return;

}

int mid = (rangeLeft + rangeRight) / 2;

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

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

segTree[index] = segTree[2\*index] + segTree[2\*index+1];

}

**Query Tree**

Start from root node and follow steps given below.

* If current node's range is completely outside query's range then return 0.
* If current node's lazy is not 0 then update current nodes value, update current node's children's lazy and mark current node's lazy as 0.
* If current node's range completely lies inside query's range then return current node's value.
* Else query on both children and return sum of their answers.

Following is code snippet to query segment tree :

// queryLeft will be left most index of query range

// queryRight will be right most index of query 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

int query\_tree (int queryLeft, int queryRight, int rangeLeft, int rangeRight, int index)

{

if (queryRight < rangeLeft || rangeRight < queryLeft)

return 0;

if (lazy[index] != 0) {

segTree[index] += lazy[index]\*(rangeRight - rangeLeft + 1);

lazy[2\*index] += lazy[index];

lazy[2\*index+1] += lazy[index];

lazy[index] = 0;

}

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

return segTree[index];

int mid = (rangeLeft + rangeRight) / 2;

return query\_tree(queryLeft, queryRight, rangeLeft, mid, 2\*index) + query\_tree(queryLeft, queryRight, mid+1, rangeRight, 2\*index+1);

}