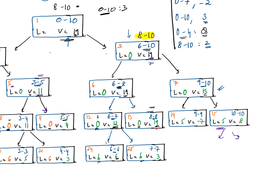
Segment Trees 2
Wednesday, 3 August 2022 12:05

arr: 0 1 2 3 4 5 6 7 8 9 10
4 4 5 4 8 4 10 3 14 -2 13

0-10 → 0-10 :3

Q1:
3-8, +5
0-7, -2
0-10, 3
8-10 : 2



Problem:
update

1) For that node if lazy exists
→ update lazy value & root value
→ lower lazy value pass to children

2) Check overlapping
→ No overlapping : {return}
→ Full overlapping :
→ update current node value
→ Pass lazy value to children
→ Partial overlapping :
→ Lazy update with left & right
→ Update Tree[node] based on its children value
Tree[2*node], Tree[2*node+1]

// Leaves & Node N } Base & same
Tree[node], lazy[node]

Query

1) For that node if lazy exists
→ update lazy value & root value
→ lower lazy value pass to children

2) Check overlapping
→ No overlapping : {return concerned}
→ Full overlapping :
→ return Tree[node]
→ Partial overlapping :
→ left = Query (left part)
→ Right = Query (right)
→ return on base based left & right

void update(int tree[], int lazy[], int l, int r, int l, int r, int ql, int qr, int val) {
segment range, query range

if(lazy[i] != 0) {
//recursive
tree[i] = tree[i] + lazy[i]
// pass to children
if(l != r) {
lazy[2*i] += lazy[i]
lazy[2*i+1] += lazy[i]
}
lazy[i] = 0

Step 1

// Unique overlapping

// Not overlapping
if(r < ql or ql < l) {return}

// Complete overlapping
if(ql <= l && r <= qr) {
// update current node
Tree[i] += val;
// pass val as lazy to children
if(l != r) {
lazy[2*i] += val
lazy[2*i+1] += val
}
return;

// Partial overlap
int m = (l+r)/2
update(tree, lazy, 2*i, l, m, ql, qr, val)
update(tree, lazy, 2*i+1, m+1, r, ql, qr, val)
// update current node value
Tree[i] = max(Tree[2*i], Tree[2*i+1])
}

int Query (int tree[], int lazy[], int l, int r, int l, int r, int ql, int qr)
segment range, query range

if(lazy[i] != 0) {
//recursive
tree[i] = tree[i] + lazy[i]
// pass to children
if(l != r) {
lazy[2*i] += lazy[i]
lazy[2*i+1] += lazy[i]
}
lazy[i] = 0

// Unique overlapping

// Not overlapping
if(r < ql or ql < l) {return INT_MIN}

// Complete overlapping
if(ql <= l && r <= qr) {
return Tree[i]
}

// Partial overlap
int m = (l+r)/2
int left = Query (tree, lazy, 2*i, l, m, ql, qr, val)
int right = Query (tree, lazy, 2*i+1, m+1, r, ql, qr, val)
return max(left, right)
}

2b) Lazy update, (Range Update Sum)
→ N → 1 l—r val
2 l—r sum

void update(int tree[], int lazy[], int l, int r, int l, int r, int ql, int qr, int val) {
segment range, query range

if(lazy[i] != 0) {
//recursive
tree[i] = tree[i] + lazy[i]*[r-l+1] ①
// pass to children
if(l != r) {
lazy[2*i] += lazy[i] ②
lazy[2*i+1] += lazy[i]
}
lazy[i] = 0

→ Range sum
Same

// Unique overlapping

// Not overlapping
if(r < ql or ql < l) {return}

→ return 0

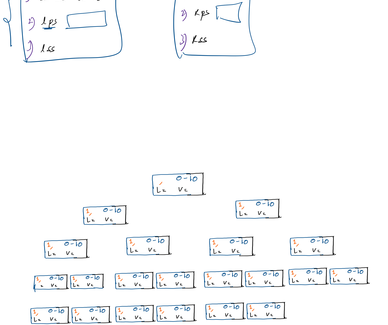
// Complete overlapping
if(ql <= l && r <= qr) {
// update current node
Tree[i] += val * (r-l+1) ③
// pass val as lazy to children
if(l != r) {
lazy[2*i] += val
lazy[2*i+1] += val
}
return;

return Tree[i]

// Partial overlap
int m = (l+r)/2
update(tree, lazy, 2*i, l, m, ql, qr, val)
update(tree, lazy, 2*i+1, m+1, r, ql, qr, val)
// update current node value
Tree[i] = Tree[2*i] + Tree[2*i+1] ④

return l + r

Segment + Lazy Propagation
1) Figure what to store at tree[i]
2) Figure out what to at lazy[i]
3) How to update lazy[i] & current node
4) How to pass lazy[i] to child nodes
5) How to get Tree[i] using its children
6) How to combine your left query & right query to get overall ans.

TC: (log n) → update
→ query

2c) Given l — r, Is this range sum min subarray sum.
0 10