Segment Trees Cont’d

Recap of seg trees

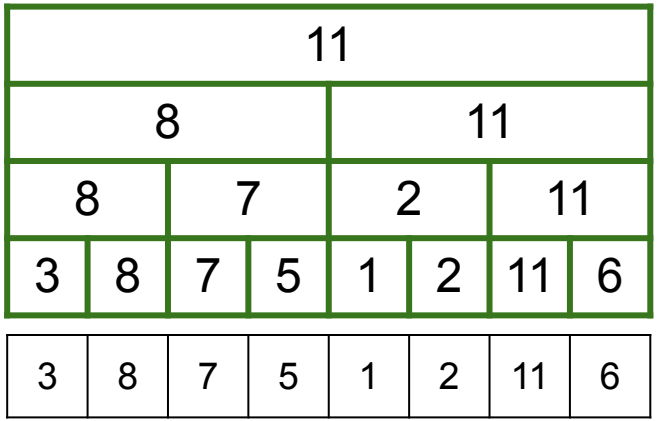
Can solve problems that look like:

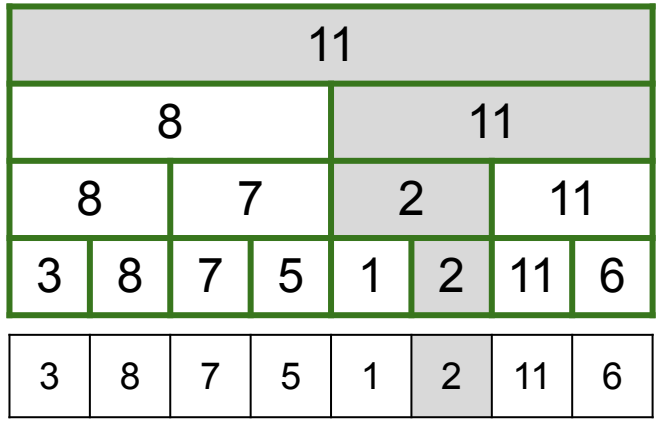
- You have an array of length N
- Maintain a data structure that supports the following operations in O(logN) time
- Update a value in the array
- Compute the result of some associative function over a subarray (the function is given beforehand, usually directly in the statement) (associative: (a * b) * c = a * (b * c))

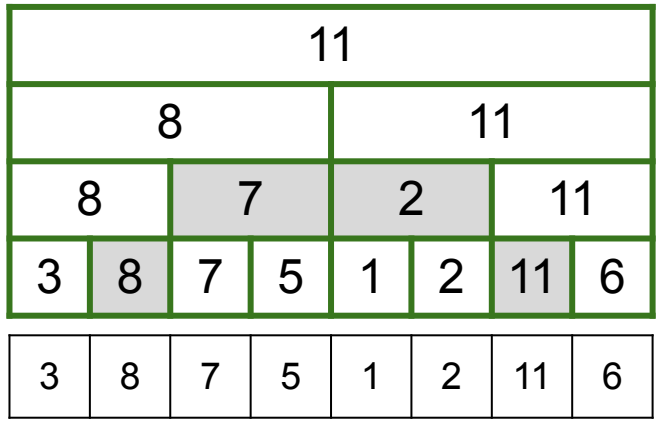
Data structure:

- Each node in the tree corresponds to the value of a function over a subarray of a length that's a power of 2
- Each node has 2 children and 1 parent (a perfect binary tree)
- The value of a non-leaf node can be calculated solely based on the values of its children
- Updates affect $\log_2 N$ nodes
- Queries can be calculated by combining at most $2\log_2 N$ nodes
- For implementation, the root is at index 1, node i has children i*2 and i*2+1, node i has parent i/2, the i-th element in the array is at TSZ+i (where TSZ I call the size of the tree, a power of two >= N)

Recap of seg trees

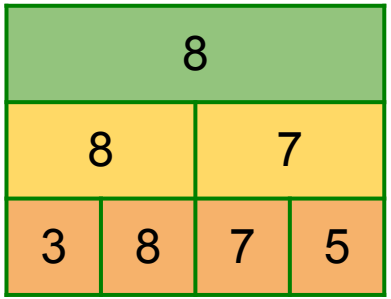






Array Representation

Segment tree



Root

Layer 1

Layer 2

Original array

3	8	7	5

Segment tree

0	8	8	7	3	8	7	5

The root is at index 1.

The children of the segment with index i are the segments with indices i*2 and i*2 + 1

Range Updates

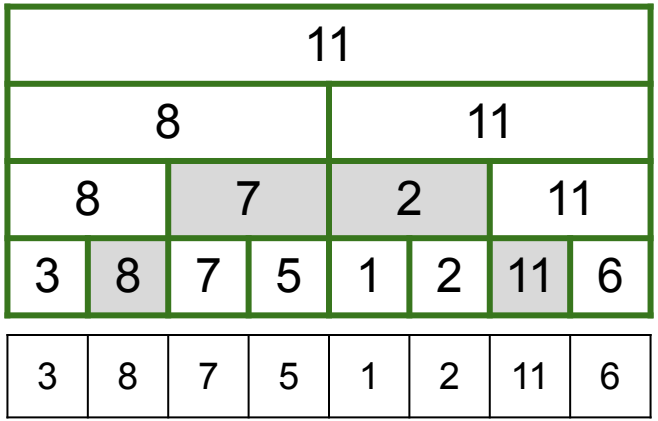
Range updates

Instead of updating a single value you have to update an entire range.

For example, let’s focus on supporting the following operations:

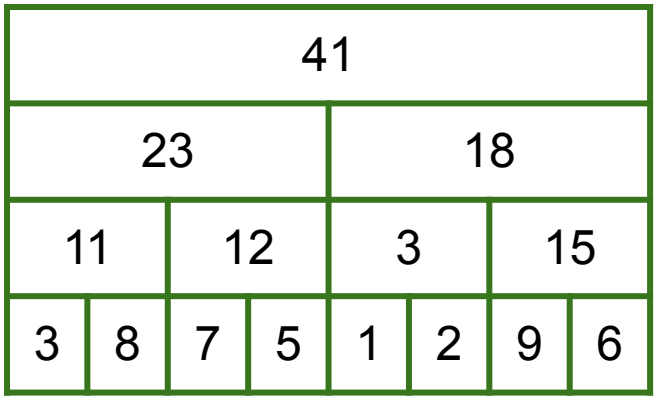
- Increase every value in the range [l, r) by x
- Find the sum of values in the range [l, r)

Range updates



Lazy Propagated Segment Trees

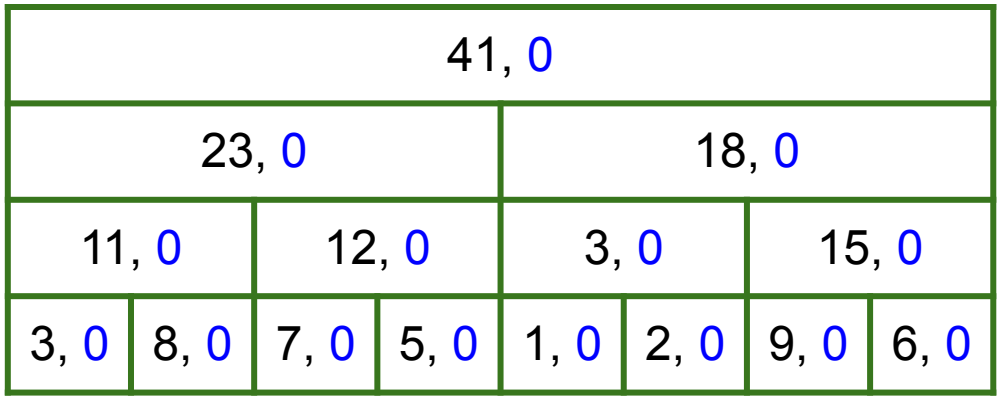
In a lazy propagated segment tree, every segment stores an additional value, called its lazy value.



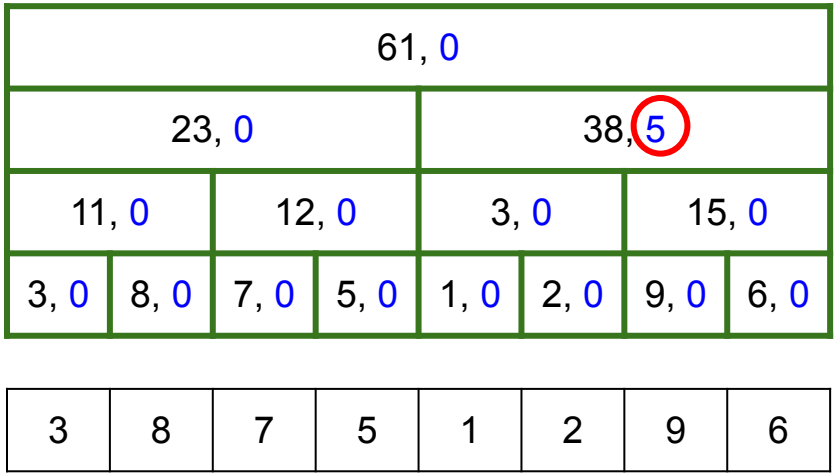
Lazy Propagated Segment Trees

In a lazy propagated segment tree, every segment stores an additional value, called its lazy value.

If a segment has a non-zero lazy value of *x*, then that means that every element in that segment was increased by *x*, but the values of all of all segments below this segment have not yet updated to reflect this change.

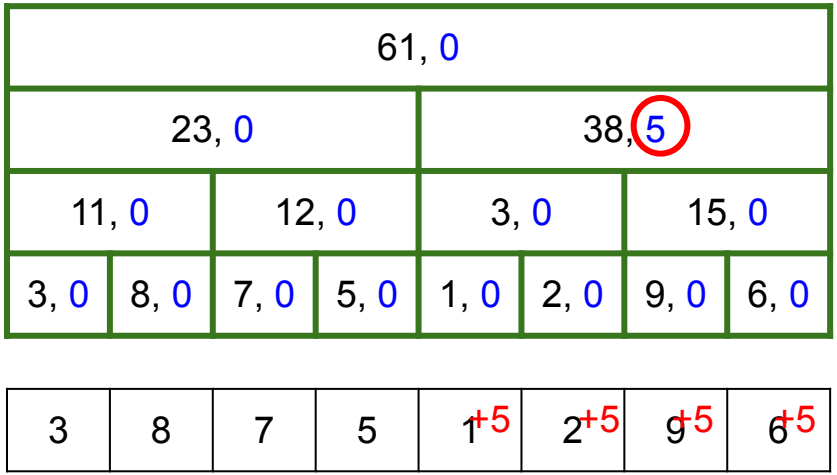


Lazy Propagated Segment Trees

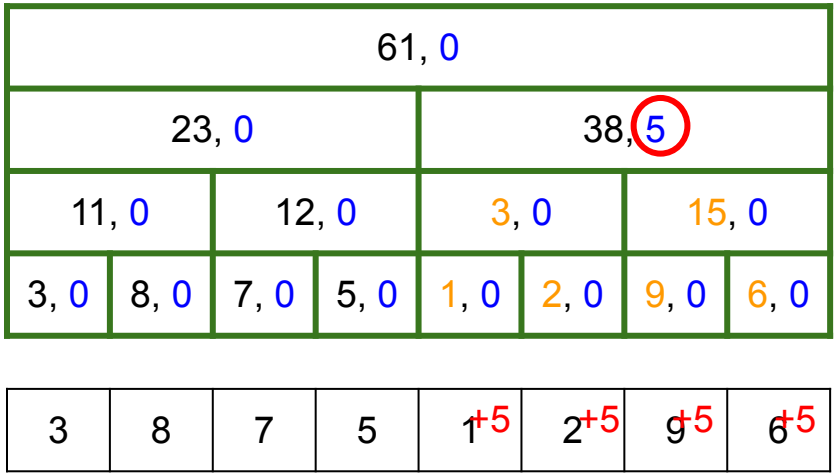


3	8	7	5	1	2	9	6

Lazy Propagated Segment Trees

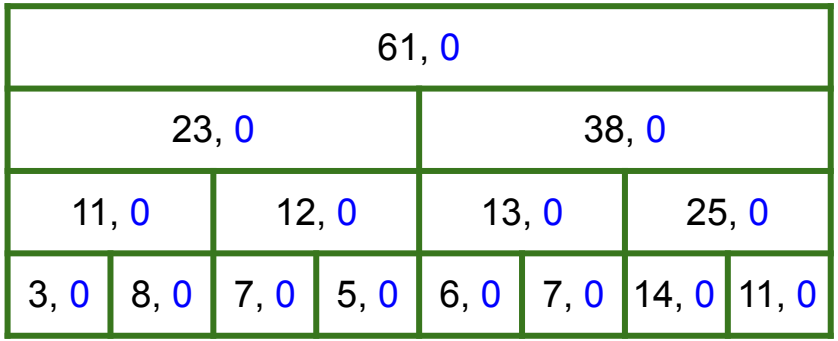


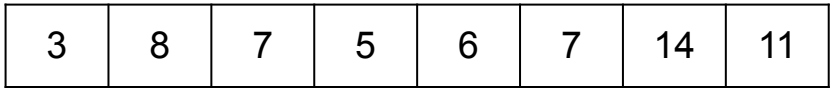
Lazy Propagated Segment Trees



=

After fully updating everything, the segment tree would look like this:





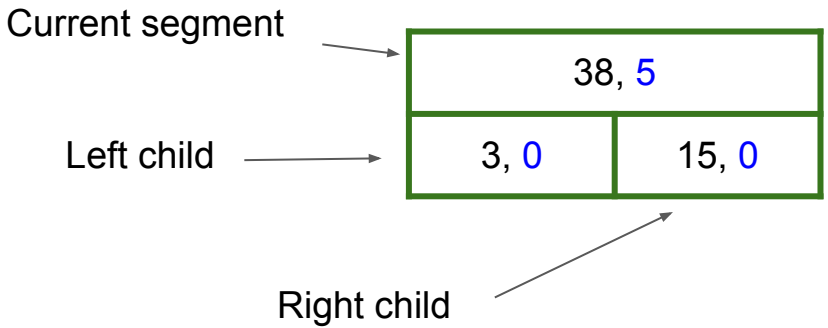
Update orange values later, when we actually need to.

How does this work?

Simple, before doing **any** operation with a segment, we check whether it has a lazy value. If it does, we update it (which takes **O(1)** time):

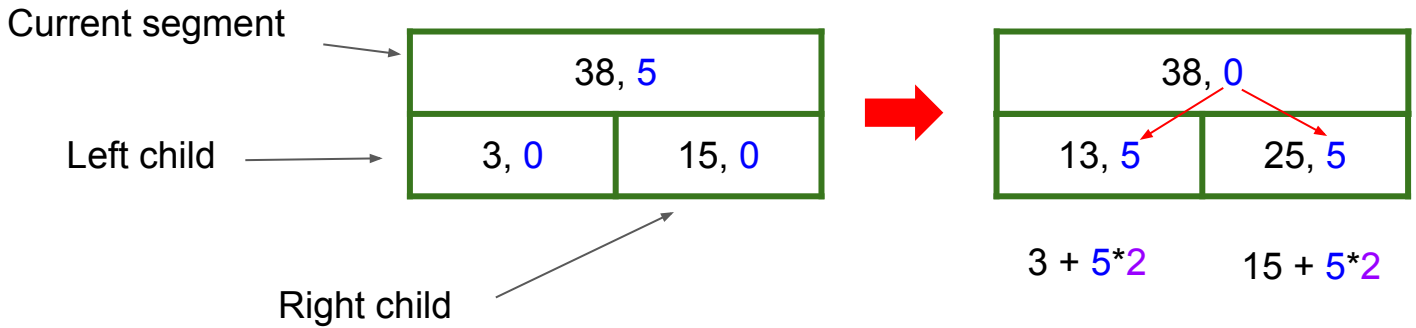
How does this work?

Simple, before doing **any** operation with a segment, we check whether it has a lazy value. If it does, we update it (which takes **O(1)** time):



How does this work?

Simple, before doing **any** operation with a segment, we check whether it has a lazy value. If it does, we propagate it (which takes **O(1)** time):



Note: this assumes you are doing the top-down recursive implementation.

Implementation

```
// queries: get sum of element in range
// updates: add v to each element in a range

const int TSZ = 1 << 18;
long long tree[TSZ*2], lazy[TSZ*2];

void build(){
    for(int i = TSZ-1; i > 0; i--){
        tree[i] = tree[i*2] + tree[i*2+1];
    }
}

void pushDown(int i, int segL, int segR){
    int mid = (segL+segR)/2;
    tree[i*2] += lazy[i]*(mid-segL);
    tree[i*2+1] += lazy[i]*(segR-mid);
    lazy[i*2] += lazy[i];
    lazy[i*2+1] += lazy[i];
    lazy[i] = 0;
}
```

```
int query(int i, int segL, int segR, int l, int r){
    if(r <= segL || l >= segR) return 0;
    if(l <= segL && r >= segR) return tree[i];
    int mid = (segL+segR)/2;
    pushDown(i, segL, segR);
    return query(i*2, segL, mid, l, r) + query(i*2+1, mid, segR, l, r);
}

int upd(int i, int segL, int segR, int l, int r, int v){
    if(r <= segL || l >= segR) return tree[i];
    if(l <= segL && r >= segR){
        lazy[i] += v;
        return tree[i] += (ll) v*(segR-segL);
    }
    int mid = (segL+segR)/2;
    pushDown(i, segL, segR);
    return tree[i] = upd(i*2, segL, mid, l, r, v) + upd(i*2+1, mid, segR, l, r, v);
}
```

Resources

Part 1 (up to point update seg trees):
https://docs.google.com/presentation/d/1zOvt1DMVvrXOuVWiBzEMTRCSI17lkSXT3p6EEuhm22U/edit?usp=sharing
2020 segment tree slides:
https://docs.google.com/presentation/d/1nUskDR6TRUoQcZSFLXHYoYsfURs1md9vajr2c9LvLMo/edit?usp=share_link

Lazy seg tree problems:
https://dmoj.ca/problem/lazy
https://dmoj.ca/problem/dmopc15c1p6
https://dmoj.ca/problem/acc3p4 (more interesting updates)