Segment Trees

An introduction to segment trees

Joël Mathys

2018-02-15

Swiss Olympiad in Informatics

Introduction

Overview

Array

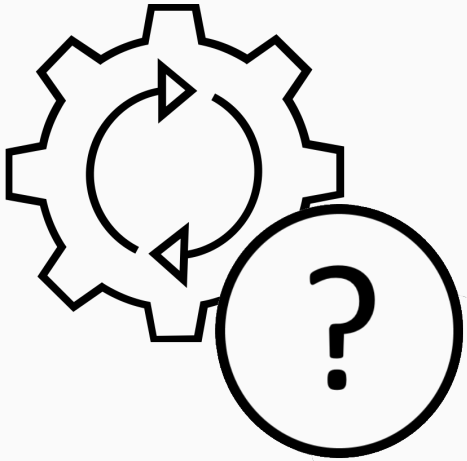
Prefix Sum

Segment Tree

Repetition

- Given Array
- Calculate sum
- Operations:
  - range queries
  - updates

Prefix Sum

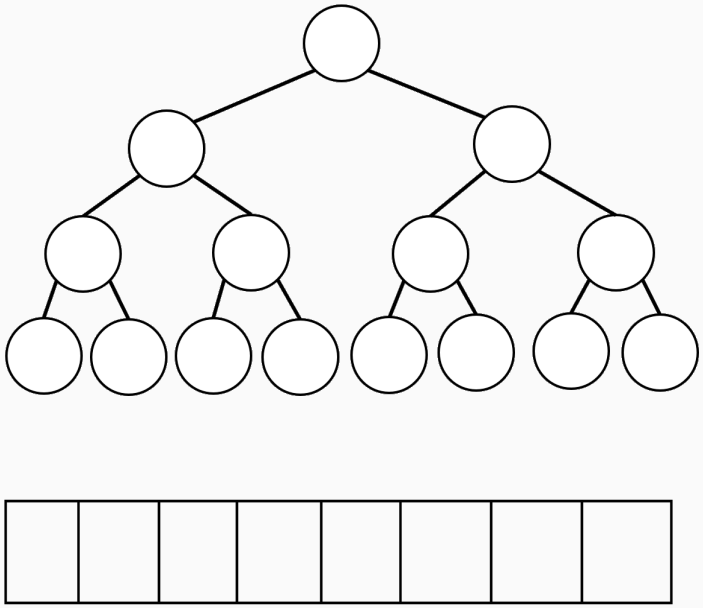


Segment Tree

Segment Trees

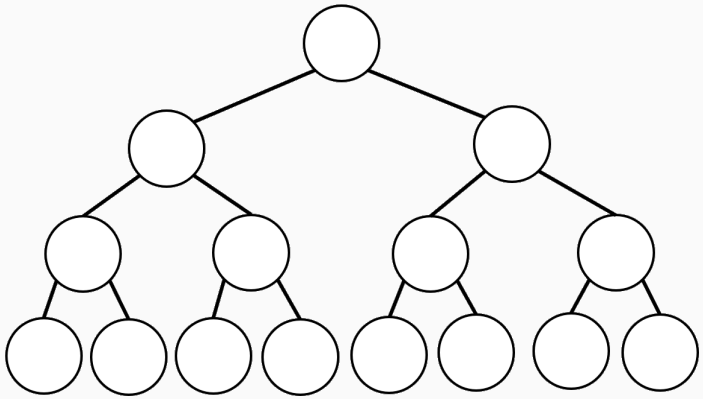
Segment Trees

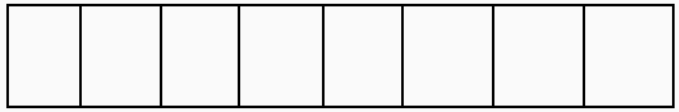
- Operations:
  - range queries
  - single updates



Segment Trees

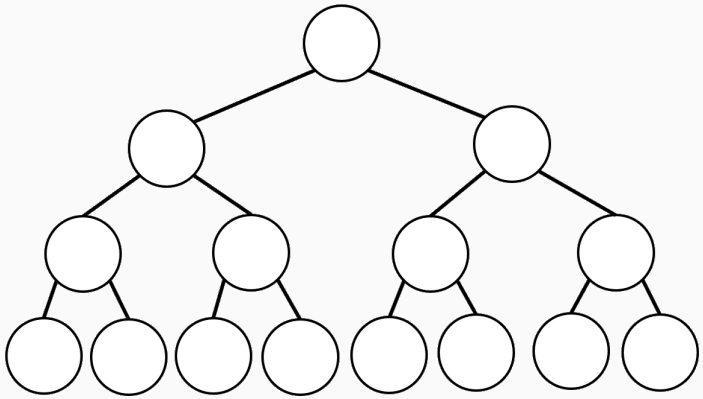
- Operations:
  - range queries
  - single updates
- Idea
  - not from left to right
  - but left AND right

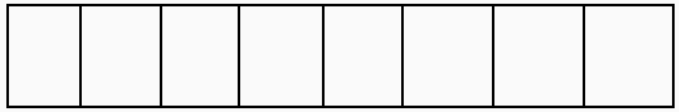




Segment Trees

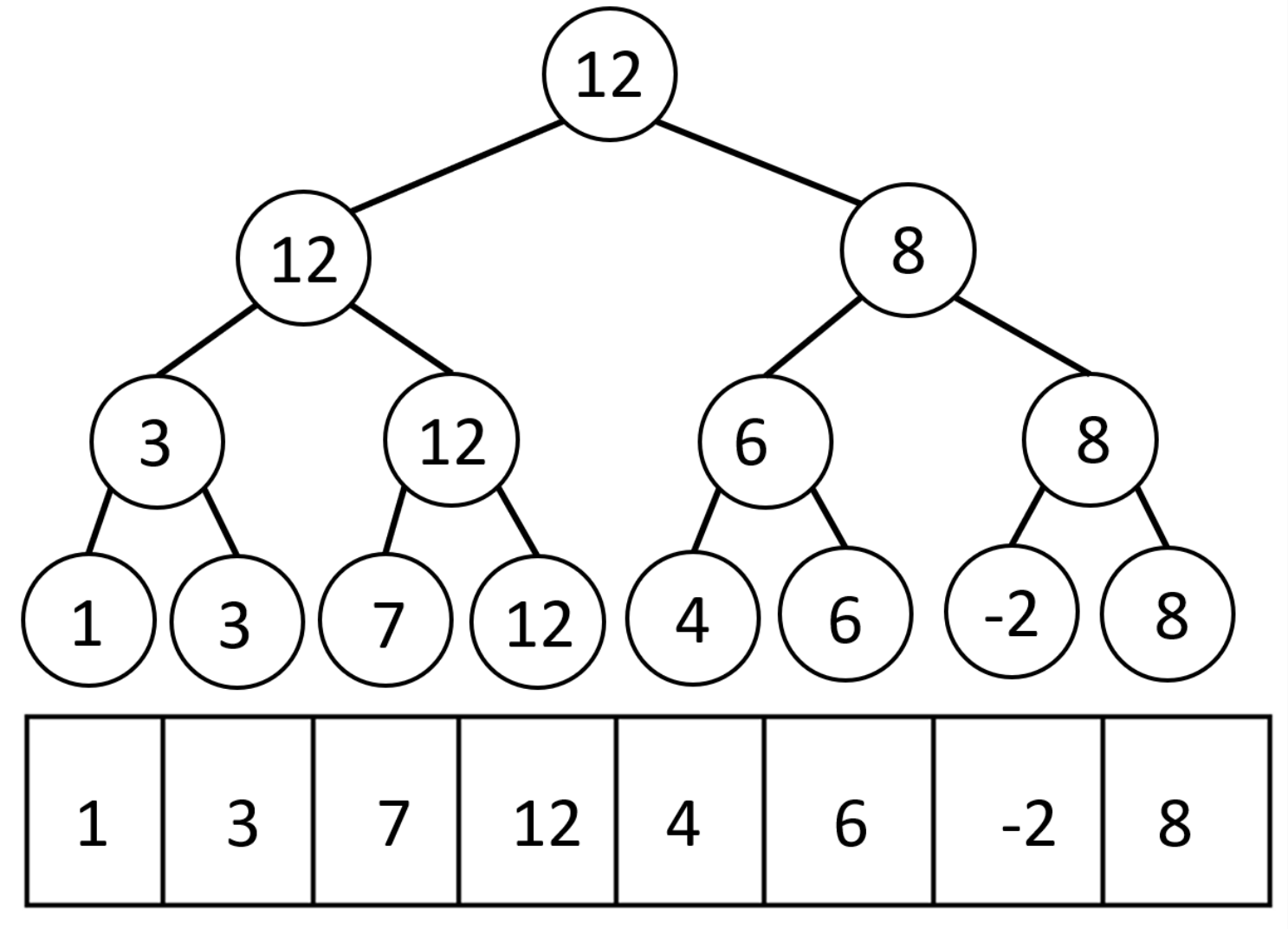
- Operations:
  - range queries
  - single updates
- Idea
  - not from left to right
  - but left AND right
- Build a binary Tree
- Lowest Level is Array
  - upper levels
    combine lower two children





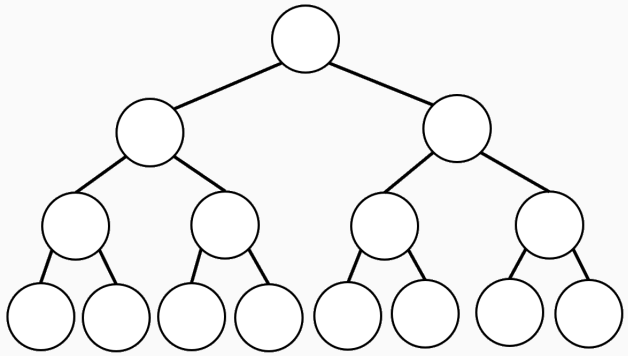
$\Rightarrow$ Get down to $O(log(n))$

Segtree Example



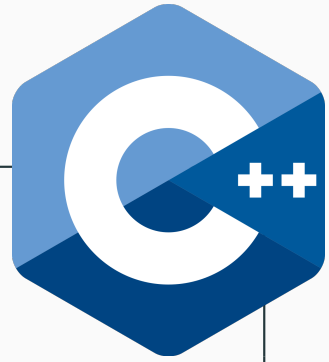
Building a Segtree

- store in an array
  - level by level
  - root at position 1
  - left child 2*pos
  - right child 2*pos+1
- extend to nearest power of 2
  $\rightarrow$ need $4 * n$ memory
- parent node range $[a, b]$:
  left child: $[a, (a + b)/2]$
  right child $[(a + b)/2 + 1, b]$



C++ Implementation

- could implement iteratively
- we do recursively
- functions:
  - build
  - update
  - query

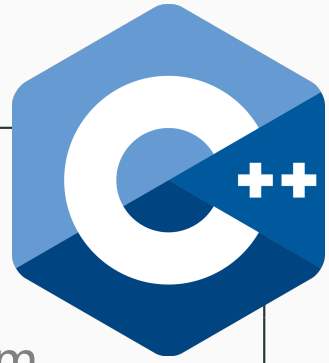


```
int n; // array size
vector<int> tree(4*n); // segment tree
vector<int> base(n); // values

void build(int x, int a, int b) // initialize tree
void update(int x, int a, int b, int pos, int val) // update
int query(int x, int a, int b, int l, int r) // answer queries
```

C++ Example

- build(int x, int a, int b)
- Remember
  - left/right child 2n, 2n+1
  - root at position 1, full range
  - leafs only cover one element



```
void build(int x, int a, int b){ // initialize tree
if(a == b){tree[x] = base[a];} // at a leaf
  else{ // build left, right children, then take max of them
        build(2*x, a, (a+b)/2);
        build(2*x+1,(a+b)/2+1, b);
        tree[x] = max(tree[2*x], tree[2*x+1]);
  }
}
build(1, 0, n-1);
```

C++ Example

- update(int x, int a, int b,int pos, int val)
- Remember
  - go down the tree, update when back
  - set $base[pos] = val$

```
void update(int x ,int a, int b, int pos, int val){
        if(pos <a || pos > b){return;} // out of range
        if(a == b){ tree[x] = val;} // at leaf
        else{ // push update down to children
                update(2*x, a, (a+b)/2, pos, val);
                update(2*x+1, (a+b)/2+1, b, pos, val);
                tree[x] = max(tree[2*x], tree[2*x+1]);
        }
}
update(1, 0, n-1, pos, val);
```

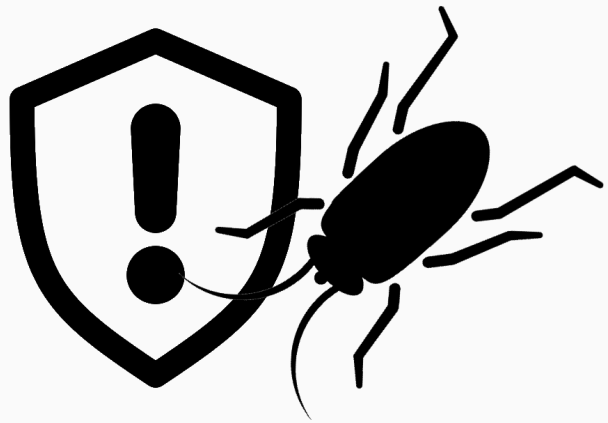
C++ Example

- query(int x, int a, int b,int l, int r)
- Remember
  - query range $[l, r]$
  - node range $[a, b]$

```
void query(int x ,int a, int b, int l, int r){
        if(b < l || r < a){return -inf;} // out of range
        // covered by noderange
        if(l <= a && b <= r){ return tree[x];}
        // push query down to children
        int ql = query(2*x, a, (a+b)/2, l,r);
        int qr = query(2*x+1, (a+b)/2+1, b, l,r);
        return max(ql, qr);
}
query(1, 0, n-1, l,r);
```

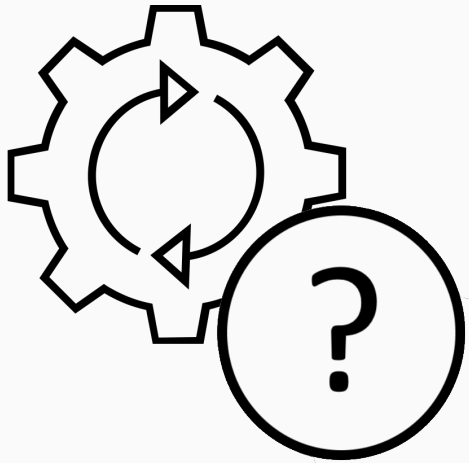
Common Mistakes

- Out of bounds
  - check if at a leaf
  - visit 2n, 2n+1
  - allocate enough memory
- Wrong results
  - always update parents after change!
  - return a neutral result when out of bounds (-inf, 0)
  - calculate right ranges for nodes! (a+b)/2
- Other bugs
  - always start with node 1 and range 0 to n-1
  - ranges are always inclusive!

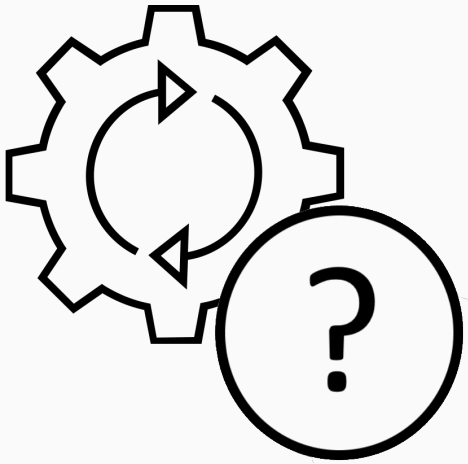


Possible Extensions

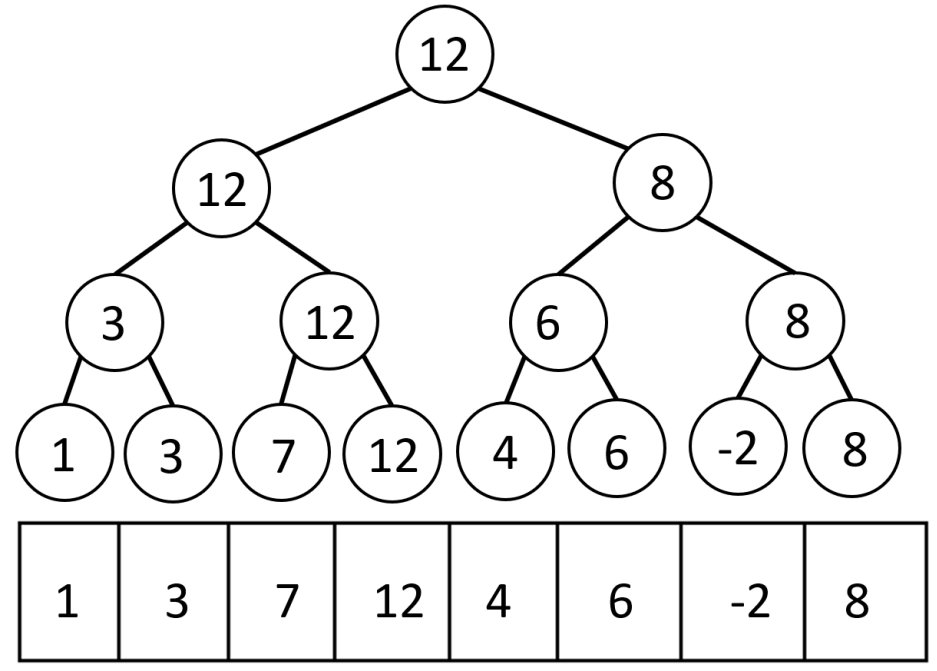
- apply other operations
  need to be associative
  $(a * b) * c = a * (b * c)$
- common: max, min, sum
- do different queries:
  - range updates needs lazy



Summary

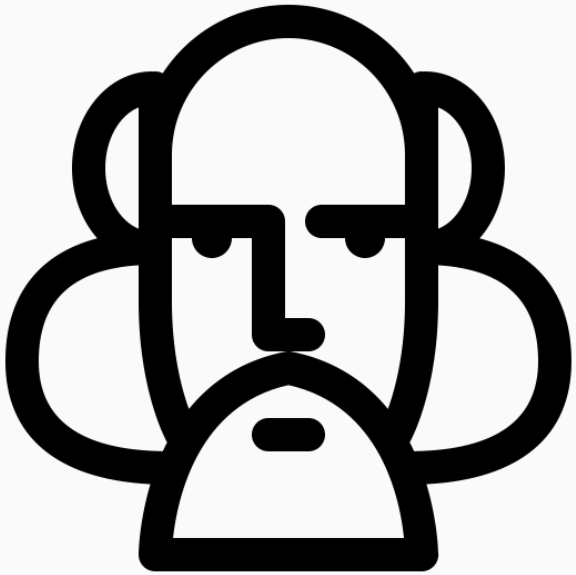


Queries
ranges and updates



Segment Trees
build, update, query

Questions



I code - therefore I am