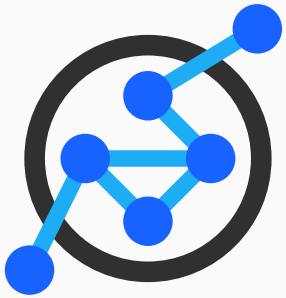


INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

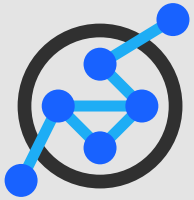
Segment Trees

Johannes Kapfhammer

12 February 2020

Idea

Warm up: divide and conquer



1. Split array in half.
2. Solve each half recursively.
3. Combine the results from each half.

Warm up: divide and conquer

1. Split array in half.
2. Solve each half recursively.
3. Combine the results from each half.

Example: Maximum of

3 1 7 4 9 2 5 6

Warm up: divide and conquer

1. Split array in half.
2. Solve each half recursively.
3. Combine the results from each half.

Example: Maximum of

3 1 7 4 9 2 5 6

3 1 7 4 | 9 2 5 6

3 1 | 7 4 | 9 2 | 5 6

3 | 1 | 7 | 4 | 9 | 2 | 5 | 6

Warm up: divide and conquer

1. Split array in half.
2. Solve each half recursively.
3. Combine the results from each half.

Example: Maximum of

3 1 7 4 9 2 5 6

3 1 7 4 | 9 2 5 6

3 1 | 7 4 | 9 2 | 5 6

3 | 1 | 7 | 4 | 9 | 2 | 5 | 6

3 | 7 | 9 | 6

Warm up: divide and conquer

1. Split array in half.
2. Solve each half recursively.
3. Combine the results from each half.

Example: Maximum of

3 1 7 4 9 2 5 6

3 1 7 4 | 9 2 5 6

3 1 | 7 4 | 9 2 | 5 6

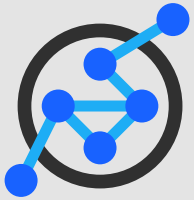
3 | 1 | 7 | 4 | 9 | 2 | 5 | 6

3 | 7 | 9 | 6

7 | 9

9

Single element update

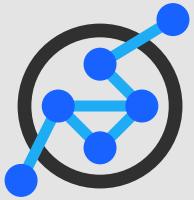


INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

3	1	7	4	9	2	5	6
3	1	0	4	9	2	5	6

What changes?

Single element update



INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

3 1 7 4 9 2 5 6

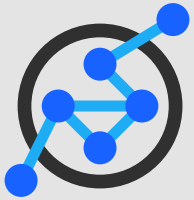
3 1 0 4 9 2 5 6

What changes?

3 | 1 | 7 | 4 | 9 | 2 | 5 | 6

3 | 1 | 0 | 4 | 9 | 2 | 5 | 6

Single element update



INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

3 1 7 4 9 2 5 6

3 1 0 4 9 2 5 6

What changes?

3 | 1 | 7 | 4 | 9 | 2 | 5 | 6

3 | 1 | 0 | 4 | 9 | 2 | 5 | 6

3 | 4 | 9 | 6

Single element update

3 1 7 4 9 2 5 6

3 1 0 4 9 2 5 6

What changes?

3 | 1 | 7 | 4 | 9 | 2 | 5 | 6

3 | 1 | 0 | 4 | 9 | 2 | 5 | 6

3 | 4 | 9 | 6

4 | 9

9

Single element update

3 1 7 4 9 2 5 6

3 1 0 4 9 2 5 6

What changes?

3 | 1 | 7 | 4 | 9 | 2 | 5 | 6

3 | 1 | 0 | 4 | 9 | 2 | 5 | 6

3 | 4 | 9 | 6

4 | 9

9

- At most $\log n$ elements change.
- If the conquer step runs in $\mathcal{O}(1)$, we can recompute them quickly.

Range queries

Can we compute the maximum of other ranges?

3 1 7 4 6 2 5 9

Range queries

Can we compute the maximum of other ranges?

3 1 7 4 6 2 5 9

3 | 1 | 7 | 4 | 6 | 2 | 5 | 9

3 | 7 | 6 | 9

7 | 9

9

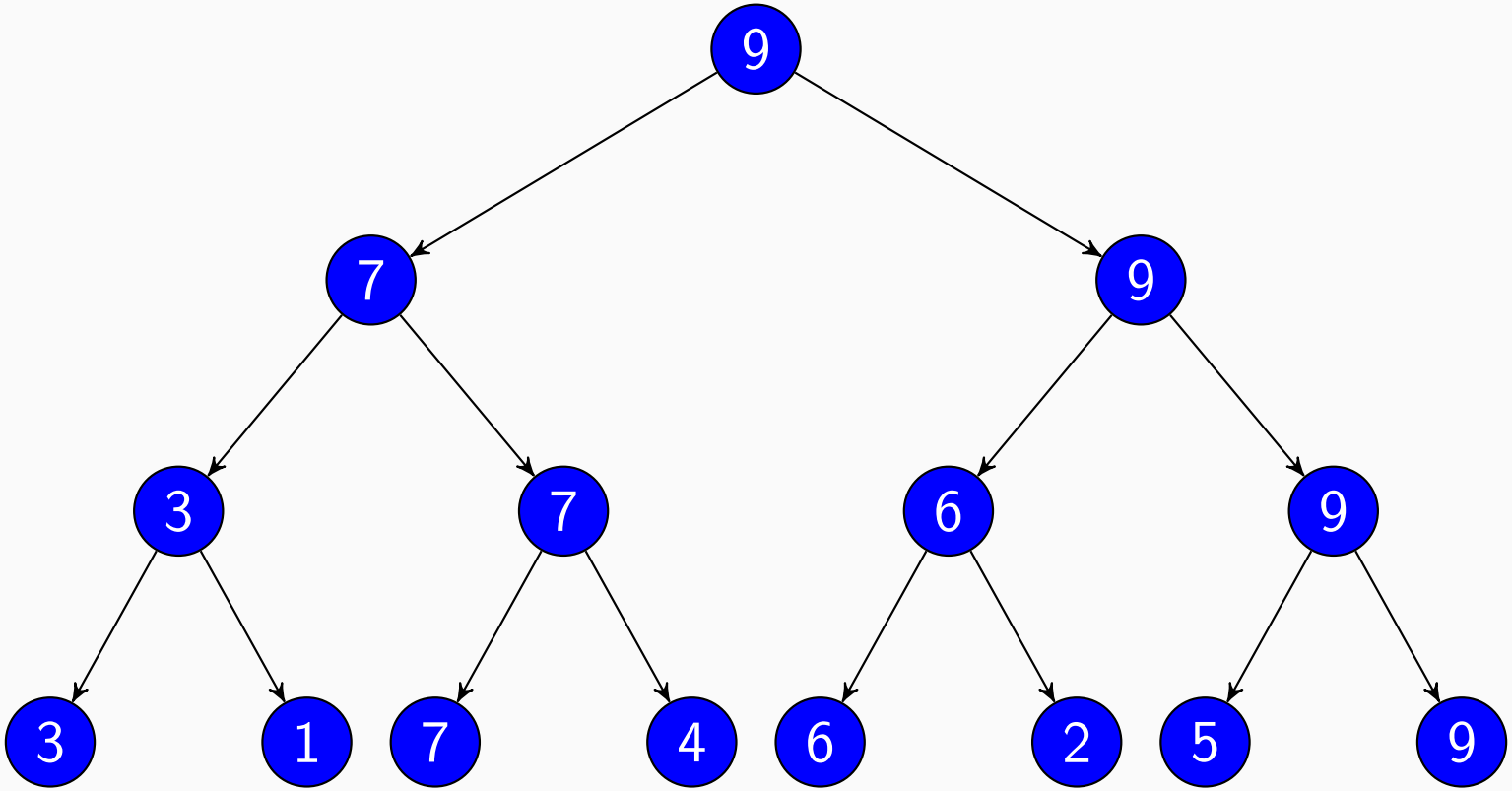
Range queries

Can we compute the maximum of other ranges?

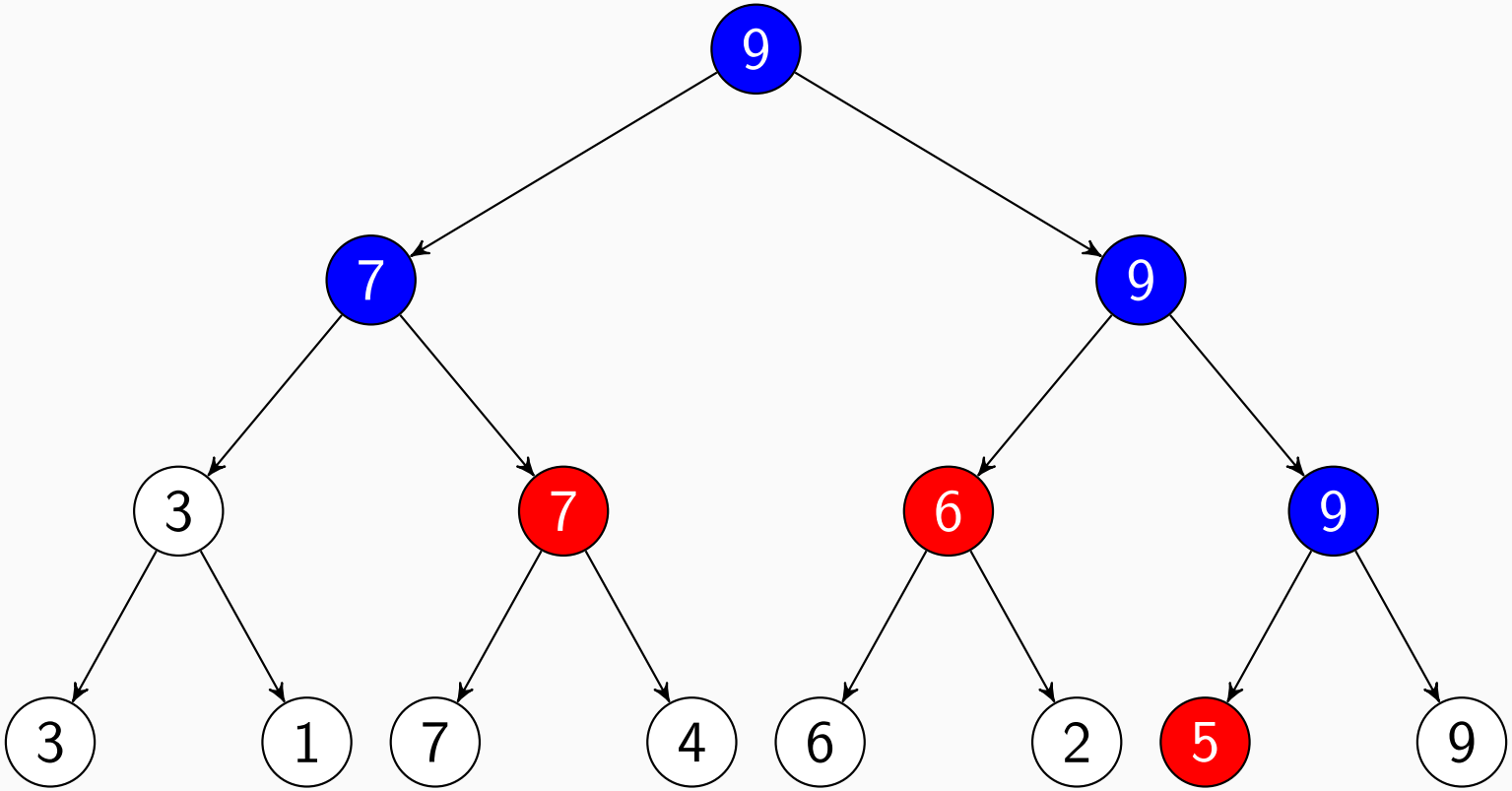
```
3   1   7   4   6   2   5   9
3 | 1 | 7 | 4 | 6 | 2 | 5 | 9
  3   |   7   |   6   |   9
      7       |       9
              9
```

- If the conquer step runs in $\mathcal{O}(1)$, we can answer queries fast.

Store recursion tree



Range queries



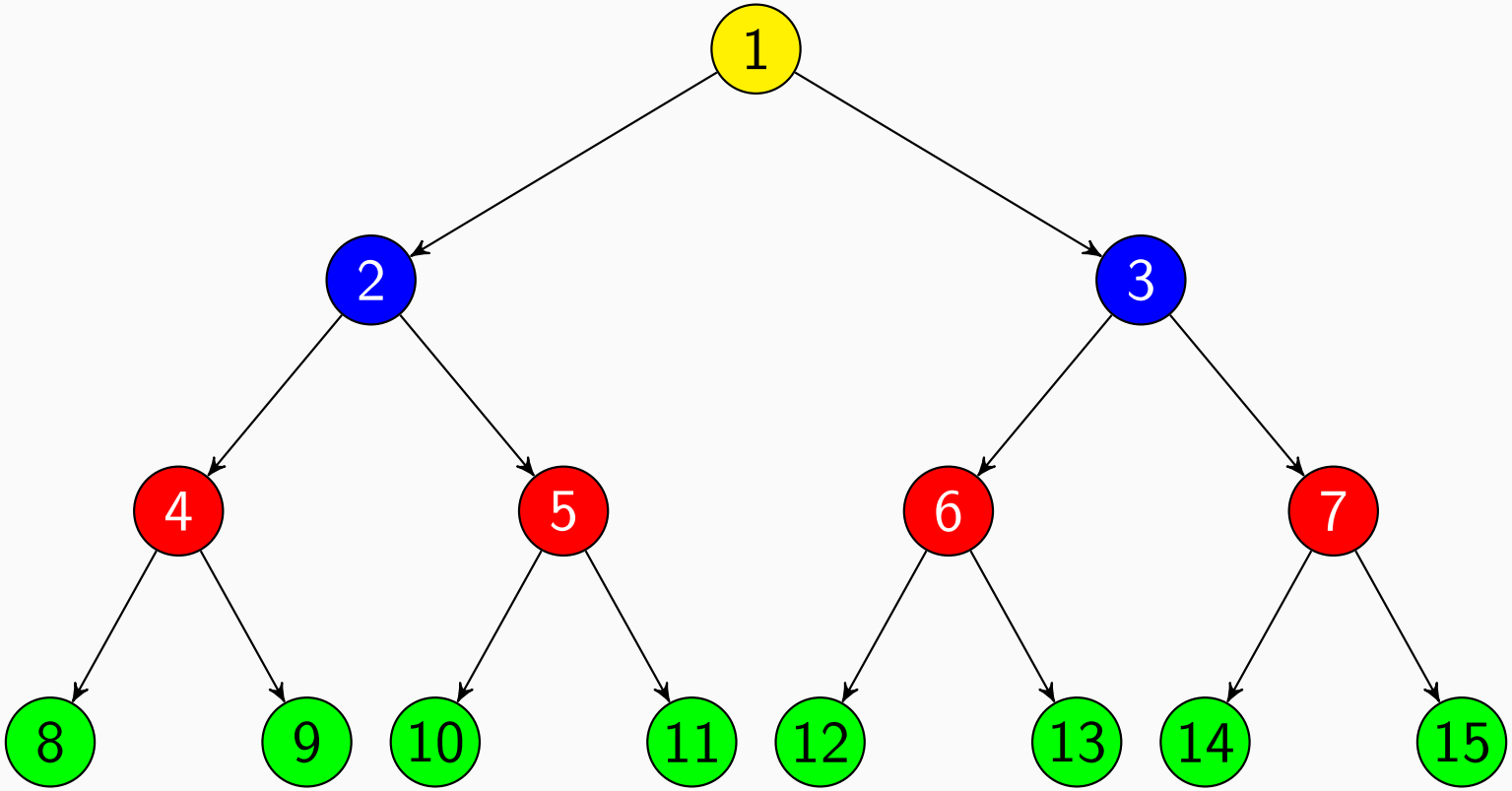
Implementation

Different styles

- Pointers: Slow, most memory, persistency.
- Heap: Fast, versatile, close to divide and conquer.
- Iterative: Fastest, less memory.

I'll focus on the *Heap* style.

Node indices



For node i:

- children $2 \cdot i$ and $2 \cdot i + 1$
- parent $\lfloor i/2 \rfloor$

Groundwork


```
1  int next_power_of_two(unsigned x) {
2    return 1<<(__lg(x-1)+1);
3  }
4
5  struct SegmentTree {
6    int n;
7    vector<int> tree;
8
9    // build segtree of size at least min_n
10   SegmentTree(int min_n)
11     : n(next_power_of_two(min_n)),
12       tree(2*n) {}
13 };
```

Build


```
1  SegmentTree(vector<Value> const& base)
2    : n(next_power_of_two(base.size())),
3      tree(2*n) {
4    for (int i=0; i<(int)base.size(); ++i)
5      tree[n+i] = base[i];
6    build(1, 0, n);
7  }
8  void build(int pos, int l, int r){
9    if (r - l == 1) return;
10   int m = (l+r)/2;
11   build(2*pos, l, m);
12   build(2*pos+1, m, r);
13   tree[pos] = max(tree[2*pos], tree[2*pos+1]);
14 }
```

Update


```
1  void update(int i, int val) {
2    assert(0 <= i && i < n);
3    return update_(i, val, 1, 0, n);
4  }
5
6  void update_(int i, int val, int pos, int l, int r) {
7    if (l == i && i+1 == r) { tree[pos] = val; return; }
8    if (i < l || i >= r) { return; }
9    int m = (l+r)/2;
10   update_(i, val, 2*pos, l, m);
11   update_(i, val, 2*pos+1, m, r);
12   tree[pos] = max(tree[2*pos], tree[2*pos+1]);
13 }
```

Query


```
1  int query(int l, int r) {
2    assert(0 <= l && l < r && r <= n);
3    return query_(l, r, 1, 0, n);
4  }
5
6  int query_(int ql, int qr, int pos, int l, int r) {
7    if (ql <= l && r <= qr) { return tree[pos]; }
8    if (qr <= l || r <= ql) { return -INF; }
9    int m = (l+r)/2;
10   int ans_l = query_(ql, qr, 2*pos, l, m);
11   int ans_r = query_(ql, qr, 2*pos+1, m, r);
12   return max(ans_l, ans_r);
13 }
```

Possible query types

Which operations out of those can we use?

$+$, $-$, $\cdot$, $/$, pow, OR, XOR, NOR, AND, NAND gcd, $==$

Other possible queries

Anything that can be conquered in $\mathcal{O}(1)$.
The operations have to be associative $a \circ (b \circ c) = (a \circ b) \circ c$.

Examples

$$+, \cdot, \max, \min, \gcd, \text{AND}, \text{OR}, \text{XOR}, \ldots$$

Things that don't work

$$-, /, \text{pow}, \text{NAND}, \text{NOR}, ==, \ldots$$

Runtime: $\mathcal{O}(N)$ for build, $\mathcal{O}(\log N)$ per update/query.

Example

Example

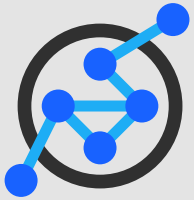
Given an array of N numbers a_i, execute the following queries.

- `up x b`: Set a_x to b.
- `q l r`: Return the sum of the maximum sum subarray of $a_l, a_{l+1}, \ldots, a_{r-1}$.

How can we apply divide and conquer?

Which values to Store?

Divide and conquer approach



INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

Which values to Store?

- Sum of max sum subarray.
- Maximum prefix.
- Maximum suffix.
- Sum of all elements.

Divide and conquer approach

Which values to Store?

- Sum of max sum subarray.
- Maximum prefix.
- Maximum suffix.
- Sum of all elements.

How to conquer?

Divide and conquer approach

Which values to Store?

- Sum of max sum subarray.
- Maximum prefix.
- Maximum suffix.
- Sum of all elements.

How to conquer?

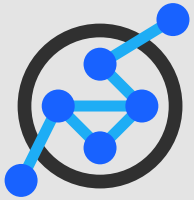
- Max sum subarray can be in one half, or it crosses the boundary.
- Maximum prefix/suffix can be in either half.
- Sum is sum of both sums

This runs in $\mathcal{O}(1)$, so we can use it in a segment tree.

Implementation – conquer


```
1  struct Value {
2      int max, pre, suf, sum;
3  };
4  Value combine(Value const& a, Value const& b){
5      return Value {
6          max({a.max, b.max, a.suf + b.pre}),
7          max(a.pre, a.sum + b.pre),
8          max(a.suf + b.sum, b.suf),
9          a.sum + b.sum
10     };
11 }
```

Implementation – build



INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

```
1  void build(int pos, int l, int r){
2    if (r - l == 1) return;
3    int m = (l+r)/2;
4    build(2*pos, l, m);
5    build(2*pos+1, m, r);
6    tree[pos] = combine(tree[2*pos], tree[2*pos+1]);
7  }
```

Implementation – update


```
1  void update(int i, int val) {
2    assert(0 <= i && i < n);
3    return update_(i, Value{val, val, val, val}, 1, 0, n);
4  }
5
6  void update_(int i, Value val, int pos, int l, int r) {
7    if (l == i && i+1 == r) { tree[pos] = val; return; }
8    if (i < l || i >= r) { return; }
9    int m = (l+r)/2;
10   update_(i, val, 2*pos, l, m);
11   update_(i, val, 2*pos+1, m, r);
12   tree[pos] = combine(tree[2*pos], tree[2*pos+1]);
13 }
```

Implementation – query


```
1  const Value neutral_element = {-INF, -INF, -INF, 0};
2
3  Value query_(int ql, int qr, int pos, int l, int r) {
4    if (ql <= l && r <= qr) { return tree[pos]; }
5    if (qr <= l || r <= ql) { return neutral_element; }
6    int m = (l+r)/2;
7    Value ans_l = query_(ql, qr, 2*pos, l, m);
8    Value ans_r = query_(ql, qr, 2*pos+1, m, r);
9    return combine(ans_l, ans_r);
10 }
```

Checklist

- The operations have to be associative:
  $a \circ (b \circ c) = (a \circ b) \circ c$.
- There needs to be a neutral element e:
  $e \circ a = a \circ e = a$ for all a.

```
1  struct Value { ... };
2  Value combine(Value a, Value b) { ... }
3  const Value neutral_element = ...;
```