Segment Trees

Swiss Olympiad in Informatics

February 15, 2018

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

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

What changes?

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

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

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

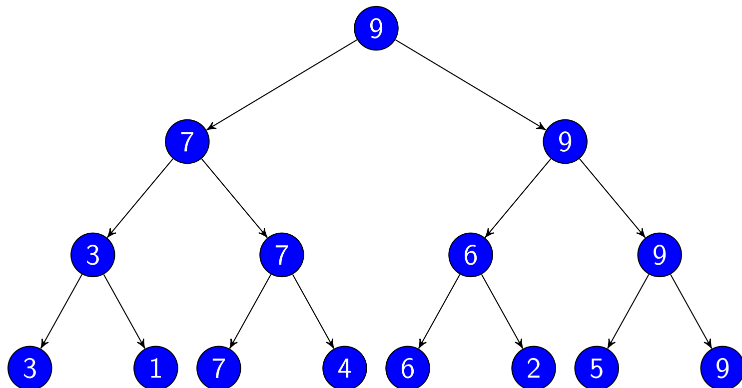
Can we compute the maximum of other ranges?

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

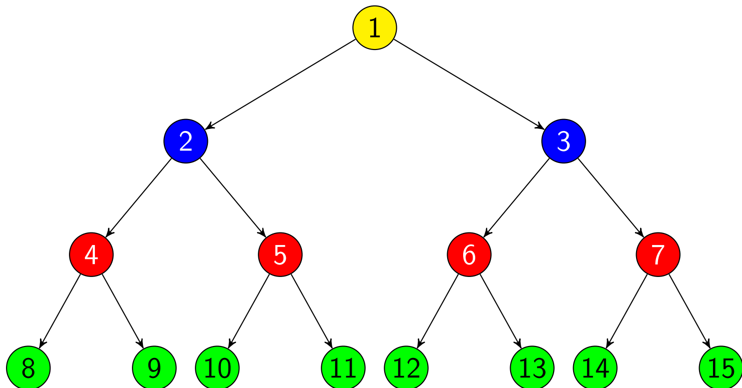


Different styles

- 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 $\lfloor i/2 \rfloor$

Build

```
vector<int> tree(4*n); // 2*n is too small!!
vector<int> base(n);
void build(int n, int a, int b){
    if(a==b){
        tree[n] = base[a];
    } else {
        build(2*n, a, (a+b)/2);
        build(2*n+1, (a+b)/2+1, b);
        tree[n] = max(tree[2*n], tree[2*n+1]);
    }
}
...
build(1, 0, n-1);
```

Update

```
1  vector<int> tree(4*n);
2  ...
3  // set base[pos] to val
4  void update(int n, int a, int b, int const&pos, int const&val){
5      if(pos<a || pos>b) return;
6      if(a==b){
7          tree[n] = val;
8      } else {
9          update(2*n, a, (a+b)/2, pos, val);
10         update(2*n+1, (a+b)/2+1, b, pos, val);
11         tree[n] = max(tree[2*n], tree[2*n+1]);
12     }
13 }
14 ...
15 update(1, 0, n-1, pos, val);
```

Query

```
1  vector<int> tree(4*n);
2  ...
3  // [a, b] is tree range, [l, r] query range
4  int query(int n, int a, int b, int const&l, int const&r){
5      if(r<a || b<l) return -inf;
6      if(l<=a && b<=r) return tree[n];
7      return max(
8          query(2*n, a, (a+b)/2, l, r),
9          query(2*n+1, (a+b)/2+1, b, l, r));
10 }
11 ...
12 query(1, 0, n-1, l, r);
```

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

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 $a_l, a_{l+1}, \ldots, a_r$.

How can we apply divide and conquer?

Divide and conquer approach

Which values to Store?

Divide and conquer approach

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 Node{
2       int max, pre, suf, sum;
3  };
4  Node conquer(Node const&l, Node const&r){
5       return Node{
6            max(l.max, max(r.max, l.suf+r.pre)),
7            max(l.pre, l.sum+r.pre),
8            max(l.suf+r.sum, r.suf),
9            l.sum+r.sum};
10 }
11 ...
```

Implementation – build

```
1  ...
2  vector<Node> tree;
3  vector<int> base;
4  void build(int n, int a, int b){
5      if(a==b){
6          tree[n] = Node{base[a], base[a], base[a], base[a]};
7      } else {
8          build(2*n, a, (a+b)/2);
9          build(2*n+1, (a+b)/2+1, b);
10         tree[n] = conquer(tree[2*n], tree[2*n+1]);
11     }
12 }
13 ...
14 build(1, 0, n-1);
```

Implementation – update

```
1   ...
2   void update(int n, int a, int b, int const&pos, int const&val){
3       if(pos<a || pos>b) return;
4       if(a==b){
5           tree[n] = Node{val, val, val, val};
6       } else {
7           update(2*n, a, (a+b)/2, pos, val);
8           update(2*n+1, (a+b)/2+1, b, pos, val);
9           tree[n] = conquer(tree[2*n], tree[2*n+1]);
10      }
11  }
12  ...
13  update(1, 0, n-1, pos, val);
```

Implementation – query

```
1   ...
2   Node query(int n, int a, int b, int const&l, int const&r){
3       if(r<a || b<l) return Node{-inf, -inf, -inf, 0};
4       if(l<=a && b<=r) return tree[n];
5       return conquer(
6           query(2*n, a, (a+b)/2, l, r),
7           query(2*n+1, (a+b)/2+1, b, l, r));
8   }
9   ...
10  query(1, 0, n-1, l, r);
```