Segment Trees (part 2)

Swiss Olympiad in Informatics

February 15, 2018

More queries

Earlier we saw that a segment tree can support

- Computing the maximum over a range.
- Changing a single element.

So how about

- `F m`: Find the minimal j such that $\max_{k=0}^{j} a_k > m$.

Slow approach

Query

Find the minimal j such that $\max_{k=0}^{j} a_k > m$.

If j is given, we can compute $\max_{k=0}^{j} a_k$ in $\mathcal{O}(\log n)$ time.

Slow approach

Query

Find the minimal j such that $\max_{k=0}^{j} a_k > m$.

If j is given, we can compute $\max_{k=0}^{j} a_k$ in $\mathcal{O}(\log n)$ time.
$\Rightarrow$ we can use a binary search for j to get $\mathcal{O}((\log n)^2)$ runtime.
Can we do faster?

Faster approach

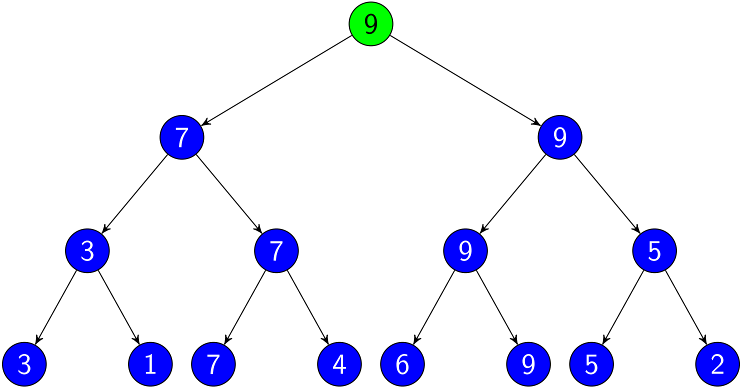
Idea: Start at the root and walk down the tree.
Look at the value of the left son to figure out where to go.

- If the value is $> m$, go left.
- if the value is $\leq m$, go right.

This runs in $\mathcal{O}(\log n)$.

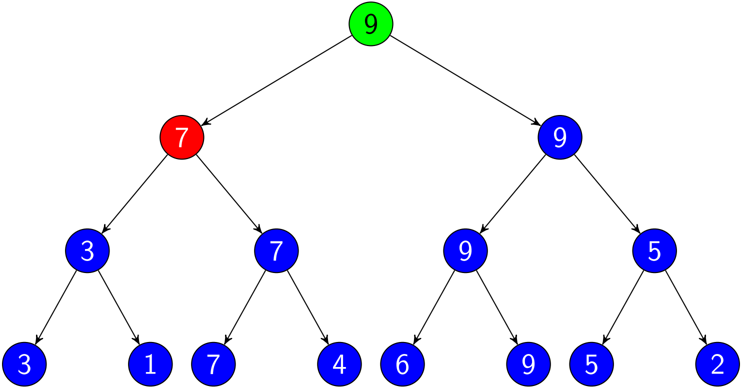
Faster approach – example

$m = 6$



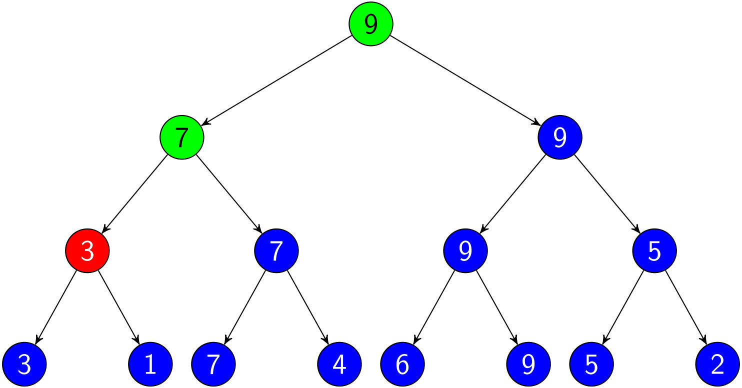
Faster approach – example

$m = 6$



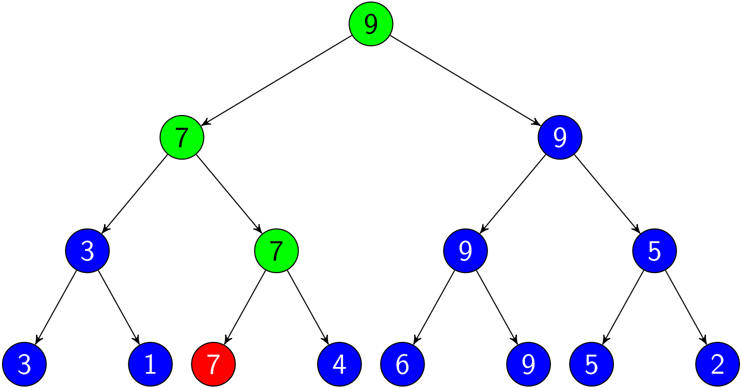
Faster approach – example

$m = 6$



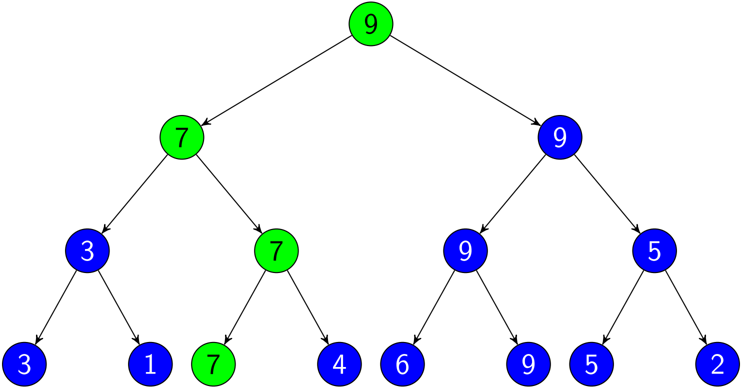
Faster approach – example

$m = 6$



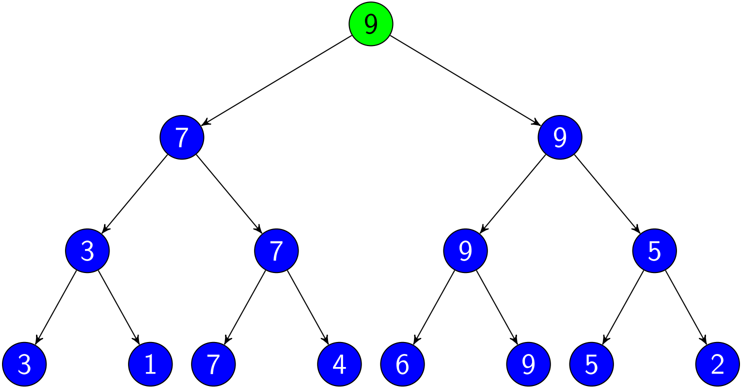
Faster approach – example

$m = 6$



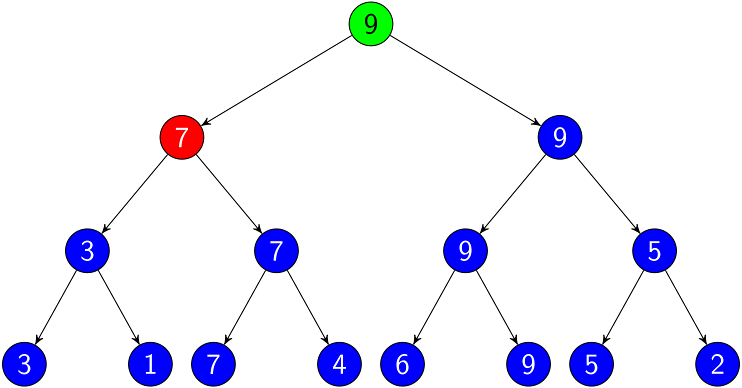
Faster approach – example

$m = 8$



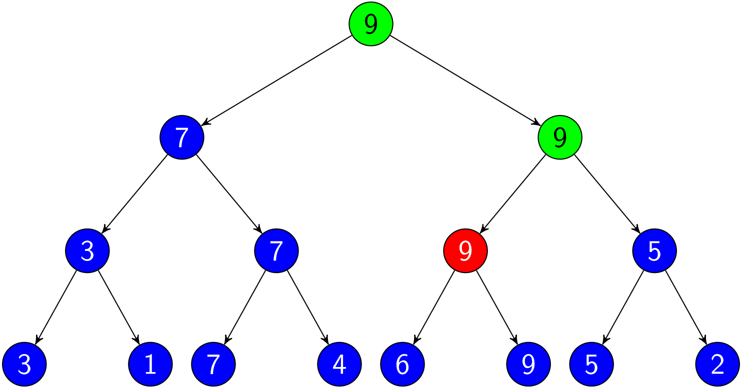
Faster approach – example

$m = 8$



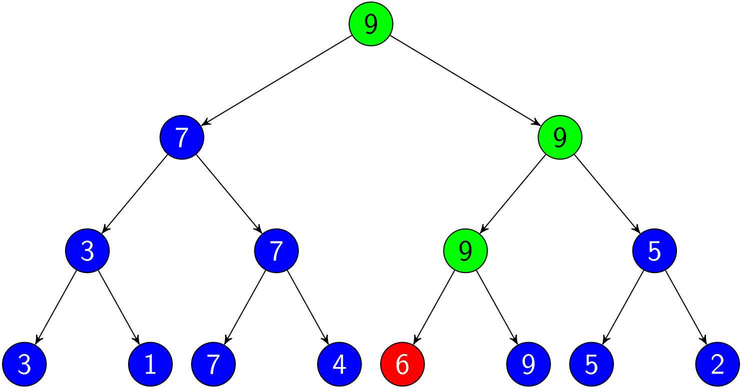
Faster approach – example

$m = 8$



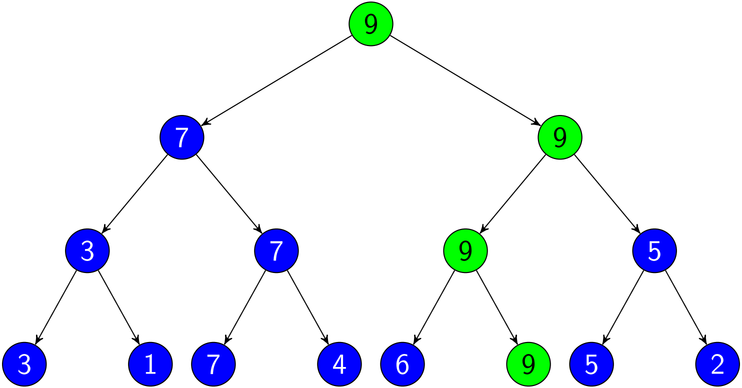
Faster approach – example

$m = 8$



Faster approach – example

$m = 8$



Faster approach – implementation

```
vector<int> tree(4*n); // segtree storing max
...
int search(int n, int a, int b, int const&m){
    if(tree[n]<=m) return -1;
    if(a==b) return a;
    if(tree[2*n] > m) return search(2*n, a, (a+b)/2, m);
    else return search(2*n+1, (a+b)/2+1, b, m);
}
...
search(1, 0, n-1, m);
```

A different example

Given n non-negative integers a_i perform the following types of operations.

- `U x b`: change a_x to b. $(b \geq 0)$
- `Q m`: Find the minimal j such that $\sum_{k=0}^{j} a_j > m$.

A different example

Given n non-negative integers a_i perform the following types of operations.

- `U x b`: change a_x to b. $(b \geq 0)$
- `Q m`: Find the minimal j such that $\sum_{k=0}^{j} a_j > m$.

Start from the root. Check the value L of the left son.

A different example

Given n non-negative integers a_i perform the following types of operations.

- `U x b`: change a_x to b. $(b \geq 0)$
- `Q m`: Find the minimal j such that $\sum_{k=0}^{j} a_j > m$.

Start from the root. Check the value L of the left son.

- If $L > m$ go left.

A different example

Given n non-negative integers a_i perform the following types of operations.

- `U x b`: change a_x to b. $(b \geq 0)$
- `Q m`: Find the minimal j such that $\sum_{k=0}^{j} a_j > m$.

Start from the root. Check the value L of the left son.

- If $L > m$ go left.
- If $L \leq m$ go right and set $m' = m - L$.

A different example – implementation

```
vector<int> tree(4*n); // segtree storing sum
...
int search(int n, int a, int b, int const&m){
    if(tree[n]<=m) return -1;
    if(a==b) return a;
    if(tree[2*n] > m) return search(2*n, a, (a+b)/2, m);
    else return search(2*n+1, (a+b)/2+1, b, m-tree[2*n]);
}
...
search(1, 0, n-1, m);
```

DP with segment tree

Segment trees can be used to speed up certain DP computations.

Task

Given n distinct integers a_i from 1 to n, compute the number of increasing subsequences of a modulo $10^9 + 7$.

Increasing subsequences – DP

Let $\mathrm{DP}[i]$ be the number of increasing subsequences ending at index i. The answer is $1 + \sum_{i=0}^{n-1} DP[i]$.

$$\mathrm{DP}[i] = 1 + \sum_{\substack{j=0 \\ a_j < a_i}}^{i-1} DP[j]$$

Increasing subsequences – DP

Let $\mathrm{DP}[i]$ be the number of increasing subsequences ending at index i. The answer is $1 + \sum_{i=0}^{n-1} DP[i]$.

$$\mathrm{DP}[i] = 1 + \sum_{\substack{j=0 \\ a_j < a_i}}^{i-1} DP[j]$$

A naive computation takes $\Theta(n^2)$ time.
The sum over a_j with $a_j < a_i$ look like a segment tree query.

Increasing subsequences – speedup

$$\mathrm{DP}[i] = 1 + \sum_{\substack{j=0 \\ a_j < a_i}}^{i-1} DP[j]$$

- To compute the sum, do a sum-query on $[1, a_i - 1]$.
- Then update by $DP[i]$ at position a_i.

This speeds up the solution to $\Theta(n \log n)$.

Task

Queries on 2D $n \times n$ grid:

- `U x y b`: Set $a_{x,y}$ to b.
- `Q` $\mathtt{x}_1$ $\mathtt{x}_2$ $\mathtt{y}_1$ $\mathtt{y}_2$: Compute $\sum_{\substack{x_1 \leq x \leq x_2 \\ y_1 \leq y \leq y_2}} a_{x,y}$.

Task

Queries on 2D $n \times n$ grid:

- `U x y b`: Set $a_{x,y}$ to b.
- `Q x1 x2 y1 y2`: Compute $\sum_{\substack{x_1 \leq x \leq x_2 \\ y_1 \leq y \leq y_2}} a_{x,y}$.

Quad tree: Split both x and y in a step. ($\Theta(n)$ worst case)

Task

Queries on 2D $n \times n$ grid:

- `U x y b`: Set $a_{x,y}$ to b.
- `Q` $\texttt{x}_1$ $\texttt{x}_2$ $\texttt{y}_1$ $\texttt{y}_2$: Compute $\sum_{\substack{x_1 \le x \le x_2 \\ y_1 \le y \le y_2}} a_{x,y}$.

Quad tree: Split both x and y in a step. ($\Theta(n)$ worst case)

Segment tree of segment trees: $\mathcal{O}((\log n)^2)$ worst case.

2D segment tree – query

```
vector<vector<int> > tree(4*N, vector<int>(4*M));
...
int qy(int n, int m, int a, int b, int l, int r){
    if(r<a || b<l) return 0;
    if(l<=a && b<=r) return tree[n][m];
    return qy(n, 2*m, a, (a+b)/2, l, r) +
           qy(n, 2*m+1, (a+b)/2+1, b, l, r);
}
int qx(int n, int a, int b, int x1, int x2, int y1, int y2){
    if(x1<a || b<x2) return 0;
    if(x1<=a && b<=x2) return qy(n, 1, 0, M-1, y1, y2);
    return qx(n, 2*n, a, (a+b)/2, x1, x2, y1, y2) +
           qx(n, 2*n+1, (a+b)/2+1, b, x1, x2, y1, y2);
}
...
qx(1, 0, N-1, x1, x2, y1, y2);
```

2D segment tree – update

```
void uy(int n, int m, int a, int b, int y, int val, bool x_leaf){
    if(y<a || b<y) return;
    if(a==b){
        if(x_leaf) tree[n][m] = val;
        else tree[n][m] = tree[2*n][m] + tree[2*n+1][m];
    } else {
        uy(n, 2*m, a, (a+b)/2, y, val, x_leaf);
        uy(n, 2*m+1, (a+b)/2+1, b, y, val, x_leaf);
        tree[n][m] = tree[n][2*m] + tree[n][2*m+1];
}   }
void ux(int n, int a, int b, int x, int y, int val){
    if(x<a || b<x) return;
    if(a==b){
        uy(n, 1, 0, M-1, y, val, true);
    } else {
        ux(n, 2*n, a, (a+b)/2, x, y, val)
        ux(n, 2*n+1, (a+b)/2+1, x, y, val);
        uy(n, 1, 0, M-1, y, val, false);
}   }
```

Summary

A Segment tree can in $\mathcal{O}(\log n)$

- Answer range queries.
- Update single elements.
- Search for a leaf.

Can be extended to higher dimensions.

Note: Code is prone to bugs, practice helps a lot!