CPSC 490 - PROBLEM SOLVING IN COMPUTER SCIENCE

Implementation of Segment Trees

Paul Liu, Kent Williams-King,
Nasa Rouf

LAST TIME - SEGMENT TREES

Last time we talked about

→ Query/update problems in $O(\log n)$

→ Segment trees

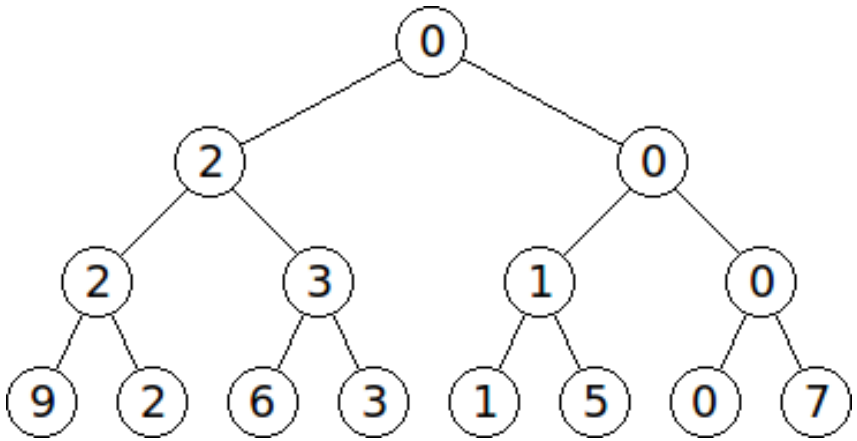
Today:

→ Implementation of Segment Trees

→ More practice with query problems

SEGMENT TREES

Important: We assume that the input array size is a power of 2.



This means the initialization of the tree looks like:

```
// Set some large power of 2.
const int MAXN = 1<<17;
// Our tree is just a flat array.
int T[2*MAXN];
```

SEGMENT TREES - IMPLEMENTATION

Tricks we will use:

- → We will store the tree in an array $A[0...2n]$.
- → The root will be in $A[1]$.
- → Node i's children are $A[2i]$ and $A[2i+1]$.
- → Node i's parent is $i/2$.

SEGMENT TREES - CONSTRUCTING A SEG. TREE

Easy way:

→ Create a segment tree with 0 value at every node.

→ Call insert once for each item in the input.

→ Complexity: $O(n \log n)$.

Hard way:

→ Use a for loop to go from the bottommost layer to the root.

→ Complexity: $O(n)$.

SEGMENT TREES - CONSTRUCTING A SEG. TREE

```
void build(int A[MAXN]) {
    // Take care of bottommost layer.
    for (int i = 0; i < MAXN; i++) {
        T[MAXN+i] = A[i];
    }

    // Take care of the rest of the layers
    // from bottom up.
    for (int j = MAXN/2; j > 0; j /= 2) {
        for (int i = j; i < 2*j; i++) {
            T[i] = min(T[2*i], T[2*i+1]);
        }
    }
}
```

SEGMENT TREES - PERFORMING UPDATES

For updates:

- $\rightarrow$ Identify leaf for which value is changed.
- $\rightarrow$ Change value at leaf.
- $\rightarrow$ Recursively ascend the tree from the affected leaf.
- $\rightarrow$ Recalculate values stored at all nodes on the path to root.

Update complexity: $O(\log n)$ since we traverse $\log_2(n)$ levels up to the root.

SEGMENT TREES - PERFORMING UPDATES

```
void update(int x, int val) {
    // Change the value at the leaf.
    int v = MAXN + x;
    T[v] = val;

    // Propagate the change all the way to the root
    for (int i = v/2 ; i > 0; i /= 2) {
        T[i] = min(T[2*i], T[2*i+1]);
    }
}
```

SEGMENT TREES - PERFORMING QUERIES

For queries:

→ Recursively descend the tree from the root.

→ Split query interval into two parts that the left and right child are responsible for respectively.

→ Recurse on these two parts.

Base cases:

→ If a query interval is empty, return 0.

→ If the interval a node is responsible for is within the query interval, return the answer right away.

SEGMENT TREES - PERFORMING QUERIES

```
int query(int x, int y, int i=1, int l=0, int r=MAXN-1) {
    // If [l,r] is not in [x, y] return INF
    if (x > r || y < l) return INF;
    // If [l, r] is completely in [x, y]
    // return the node's value
    if (x <= l && r <= y) return T[i];

    // Otherwise, recurse.
    return min( query(x, y, 2*i, l, (l+r)/2),
                query(x, y, 2*i+1, (l+r)/2+1, r) );
}
```

In Java: Overload two query functions to get default parameters.

SEGMENT TREES - EVERYTHING

```
int T[2*MAXN];
void update(int x, int val) {
    T[MAXN+x] = val;
    for (int i = (MAXN+x)/2 ; i > 0; i /= 2)
        T[i] = min(T[2*i], T[2*i+1]);
}

int query(int x, int y, int i=1, int l=0, int r=MAXN-1) {
    if (x > r || y < l) return INF;
    if (x <= l && r <= y) return T[i];
    return min( query(x, y, 2*i,l, (l+r)/2),
                query(x, y, 2*i+1, (l+r)/2+1, r));
}
```

MORE PRACTICE WITH SEGMENT TREES

Say we have an array $A[0...n-1]$. Given a subrange $A[a...b]$, I want the maximum sum subarray within this subrange.

How do we use a range tree to do query and update operations in $O(\log n)$ time?

MORE PRACTICE WITH SEGMENT TREES 2

Say we have an array $A[0...n-1]$. Given a subrange $A[a...b]$, I want the maximum sum subarray that *contains* this subrange.

How do we use one (or more) range trees to do query and update operations in $O(\log n)$ time?

MORE PRACTICE WITH SEGMENT TREES 2

Say we have a string of brackets: "()(()(..." (up to 1 million characters). Our update/queries are:

- → Update: We can change any character in the string from a '(' to a ')' and vice versa.
- → Query: We want to know if the string forms a correct bracket sequence.

How do we use a range tree to do query and update operations in $O(\log n)$ time?

NEXT TIME - MEMORY OPTIMAL QUERY STRUCTURES

Segment trees need roughly storage equal to 2x the input array. Can we do better?

→ We make a data structure that needs exactly n storage for an array of size n.

→ ...and it will be easier to code.