Segment Trees

Bjarki Ágúst Guðmundsson

School of Computer Science
Reykjavík University

viRUs Training

Motivation

The Segment Tree

Types of Queries/Updates

- Sum
- Min
- Max
- GCD
- And a lot more

- Set
- Add
- And a lot more

Representation

- Pointers
- Binary heap
- Hybrid

Representation

```
struct segment {
    segment *left, *right;
    int l, r;
    int val;
};
```

Construction

```
segment* build(int l, int r) {
    if (l > r) return NULL;
    segment *res = new segment;
    res->left = NULL;
    res->right = NULL;
    res->l = l;
    res->r = r;
    if (l < r) {
        int m = (l+r) / 2;
        if (l <= m) {
            res->left = build(l, m);
        }
        if (m+1 <= r) {
            res->right = build(m+1, r);
        }
        res->val = 0;
        if (res->left) res->val += res->left->val;
        if (res->right) res->val += res->right->val;
    } else {
        res->val = ARR[l];
    }
    return res;
}
```

Querying

```
int query(segment *st, int l, int r) {
    if (!st || r < st->l || st->r < l) return 0;
    if (l <= st->l && st->r <= r) {
        return st->val;
    }
    return query(st->left, l, r) + query(st->right, l, r);
}
```

Updating

```
void update(segment *st, int idx, int val) {
    if (!st || idx < st->l || st->r < idx) return;
    if (idx <= st->l && st->r <= idx) {
        st->val += val;
        return;
    }
    update(st->left, idx, val);
    update(st->right, idx, val);
    st->val = 0;
    if (st->left) st->val += st->left->val;
    if (st->right) st->val += st->right->val;
    // or just st->val += val;
}
```

Range Updating

- Problem: Add constant to range

Lazy Propagation

- Lazy values
- Propagation
- Combining lazy values

Lazy Propagation

```
struct segment {
    segment *left, *right;
    int l, r;
    int val;
    int lazy; // remember to initialize to INF
};
```

Lazy Propagation

```
void propagate(segment *st) {
    if (!st || st->lazy == INF) return;

    st->val += (st->r - st->l + 1) * st->lazy;

    if (st->left) {
        if (st->left->lazy == INF) st->left->lazy = st->lazy;
        else st->left->lazy += st->lazy;
    }
    if (st->right) {
        if (st->right->lazy == INF) st->right->lazy = st->lazy;
        else st->right->lazy += st->lazy;
    }

    st->lazy = INF;
}
```

Lazy Propagation

```
void range_update(segment *st, int l, int r, int val) {
    if (!st || r < st->l || st->r < l) return;

    propagate(st);

    if (l <= st->l && st->r <= r) {
        st->lazy = val;
        return;
    }

    range_update(st->left, l, r, val);
    range_update(st->right, l, r, val);

    st->val = 0;
    if (st->left) {
        propagate(st->left);
        st->val += st->left->val;
    }
    if (st->right) {
        propagate(st->right);
        st->val += st->right->val;
    }
}
```

Lazy Propagation

```
int query(segment *st, int l, int r) {
    if (!st || r < st->l || st->r < l) return 0;
    propagate(st);
    if (l <= st->l && st->r <= r) {
        return st->val;
    }
    return query(st->left, l, r) + query(st->right, l, r);
}
```

- Similar in `update`

Persistent Segment Trees

- Explain persistence
- Use cases

- Problem: Given array of **small** integers, count number of integers less than k in the range $[l, r]$

Persistent Segment Trees

```
segment* update(segment *st, int idx, int val) {
    if (!st || idx < st->l || idx > st->r) return st;
    if (idx <= st->l && st->r <= idx) {
        segment *res = new segment;
        res->left = res->right = NULL;
        res->l = res->r = idx;
        res->val = st->val + val;
        return res;
    }
    segment *res = new segment;
    res->l = st->l;
    res->r = st->r;
    res->left = update(st->left, idx, val);
    res->right = update(st->right, idx, val);
    res->val = 0;
    if (res->left) res->val += res->left->val;
    if (res->right) res->val += res->right->val;
    return res;
}
```

Implicit Segment Tree

- Coordinate compression
- Problem: Given array of **large** integers, count number of integers less than k in the range $[l, r]$

- Lazy version
- Problem: Given a huge array (initially consisting of zeros), update element or query sum of elements in the range $[l, r]$

Segment Tree of X

- We can afford more work in the building phase

Segment Tree of Arrays

- Problem: Given array of **large** integers, count number of integers less than k in the range $[l, r]$

Segment Tree of Sets

- Problem: Given array of integers, check if x appears in the range $[l, r]$

Segment Tree of Tries

- Problem: Given array of integers, determine the maximum value of $a_i \oplus x$ where $l \leq i \leq r$

Segment Tree of Hashes

- Problem: Given a string, want to know if certain substrings are equal

Segment Tree of Segment Trees

- Memory: $O(n^k)$
- Time: $O(\log(n)^k)$

- Problem: Given array of integers, count how many integers in the range $[l, r]$ are between a and b
- Problem: Given array of arrays, …

Segment Tree of X

- Union-Find
- Fenwick Trees
- Convex Hulls

Segment Tree on Euler Tour

Dynamic Segment Tree

- Adding/removing from back
- Only deleting
- Only Adding
- Adding and deleting

Application: Dynamic Connectivity

References

- http://codeforces.com/blog/entry/15890
- http://codeforces.com/blog/entry/3327
- http://blog.anudeep2011.com/persistent-segment-trees-explained-with-spoj-problems/
- http://codeforces.com/blog/entry/15296