Pragmatic Programming

Session 5 - Fenwick and Segment Trees

Max Nilsson

23rd November

A Puzzle

- Let $\texttt{n} = 10^6 + 1$. A rack of servers in a server hall can be modelled by a 1-indexed array `int servers[n]` where `server[i]` represents how warm server i is $(1 \leq i \leq n)$.
- You will be given `n` range sum queries `(a, b)`, and you should supply the total temperature of the servers between server `a` and `b`.
- Solution: Use a Prefix array.
- But suppose that you also should be able to handle the update query type `(a, x)` which represents that the temperature of server `a` has changed by x degrees.

The Fenwick Tree

- Draw a tree with root `0` and parent of node `i` is `p[i]` := `i-lsb(i)` where `lsb` is the least significant bit operator. Each node i should contain the range sum $\sum_{k=\texttt{p[i]}+1}^{i} \texttt{servers}[k]$.
- This tree will have n nodes, height $\mathcal{O}(\log n)$, and width (nodes per depth) $\mathcal{O}(n)$.
- The update and range queries can be done in $\mathcal{O}(\log(n))$

```
void updateFenwick(int a, ll x){
    while(a < numberOfServers) {
        fenwick[a] += x, a += (a & (-a));}
ll rangeSumFenwick(int a){
    ll sum = 0;
    while(a) sum += fenwick[a], a -= (a & (-a));
    return sum;}
```

The Segment Tree

- The segment tree is more versatile than a Fenwick tree. It can support more things and be generalized to a lazy and multidimensional structure. The downside is that it takes double amount of memory (but still $\mathcal{O}(n)$ space complexity) and that it is harder to code.
- It can in principle be implemented to answer point updates and range queries with respect to *any* associative function (in or case addition).
- The idea is to let the root node of the tree contain the total range sum $\sum_{k=1}^{n} \texttt{server}[k]$ and let its two children contain the partial sums $\sum_{k=1}^{\lceil n/2 \rceil} \texttt{server}[k]$, $\sum_{k=\lceil n/2 \rceil+1}^{n} \texttt{server}[k]$ respectively. The full (balanced) binary tree is defined in this way recursively.
- As with the segment tree, the tree will be implemented with an underlying implicit data structure (an array). This array will have size bounded above by

$$\sum_{k=0}^{\lceil \log_2 n \rceil} 2^k < 2^{\lceil \log_2 n \rceil + 1} \leq 2^{\log_2 n + 2} = 4n = \mathcal{O}(n)$$

  and have height $\mathcal{O}(\log n)$.

Queries on a Segment Tree (Updating)

- Updating the temperature of a server is pretty straight forward. We simply traverse the tree from root to leaf and update all nodes which hold information of server a. This can be implemented recursively and have time complexity $\mathcal{O}(\log n)$, since this is the height of the tree.

```
void updateSegment(int v, int l, int r, int a, ll x) {
    if (l == r) { segment[v] += x; return; }
    int mid = (l + r) / 2;
    if (a <= mid) updateSegment(v*2, l, mid, a, x);
    else updateSegment(v*2+1, mid+1, r, a, x);
    segment[v] = segment[v*2] + segment[v*2+1];
}
```

Queries on a Segment Tree (Range Sum)

- The idea here is pretty basic: recursively iterate through the tree and accumulate the values needed.

```
ll rangeSumSegment(int v, int l, int r, int a, int b) {
    if (a > b) return 0;
    if (a == l && b == r) return segment[v];
    int mid = (l + r) / 2;
    return rangeSumSegment(v*2, l, mid, a, min(b, mid))
    + rangeSumSegment(v*2+1, mid+1, r, max(a, mid+1), b);
}
```

- What is fantastic is that this algorithm is $\mathcal{O}(\log n)$. By induction, one can show that the number of nodes visited for each depth in the tree, is at most 4.

The Lazy Segment Tree

- Let us summarize what we have found so far

	Point Update	Range Update
Point Query	Array	-
Range Query	Fenwick/Segment	Lazy Segment

- It is possible to implement a lazy variant of the segment tree. This structure only updates the nodes when needed and still has complexities $\mathcal{O}(\log n)$ for both range updates and range queries.
- Of course, a lazy segment tree can also be used for the Range Update and Point Query combination.
- I won't go into the details here, since I don't really know them. You can look up some information here and see an implementation here.

The Multidimensional Segment Tree

- Suppose that we have a matrix `int mat[n][n]` for which we would like to perform point updates and range queries, where the range is a subrectangle.
- Then we can basically build a segment tree, such that for each node we build another segment tree. The first segment tree represents the `x`-coordinate and the other segment trees represent the `y`-coordinate.
- The space complexity would equal $\mathcal{O}(16n^2) = \mathcal{O}(n^2)$, i.e. still linear in the number of elements of `mat`.
- The update and queries would be worse, but not by much: $\mathcal{O}(\log^2 n)$.
- A similar structure could be made for the Fenwick tree. See here for an implementation.

This Week

- Skim read chapter 11.2 in Sannemo[1] and chapter 9 in Laaksonen[2].
- Solve half of this weeks 11 problems (truncated).

[1] Johan Sannemo. "Principles of Algorithmic Problem Solving". In: *Draft version* (2018).

[2] Antti Laaksonen. *Guide to competitive programming*. Springer, 2020.