MATHEMATICAL EXPLANATIONS

SIDDHANT CHAUDHARY

Abstract. In this document, we will mathematically explore all of the concepts being used.

Contents

1. Non-Linear Data Structures

1.1. Binary Indexed Trees.

Fenwick Trees are used to solve the problem of dynamic range sum queries efficiently. They are useful because they are much simpler to code and much faster than segment trees. Let's begin with a lemma.

Lemma 1.1. *For any integer type* `i`*,* `i&(-i)` *produces the least significant bit in* `i`*.*

Proof. Note that negative integers are stored using the two's complement. So, if we have an N-bit positive integer x, $-x$ will have the binary representation of $2^N - x$ (equivalently, one flips the bits of x, and adds a 1 to obtain the representation of $-x$). Now, suppose the least significant bit of x is l steps from the right. This means that all positions $0, 1, ..., l-1$ steps from the right are 0s in the binary representation of x. So, if we flip the bits of x, all these positions will be 1. Adding a 1, we see that the bit at position l is set to 1, and positions $0, ..., l-1$ from the right are all 0s. Taking the and, we see that the result has only a 1 at position l, proving the claim. ■

Moving on, we now describe the Fenwick Tree. Suppose we have an array $A[1, ..., n]$ (we use one-based indexing). Let $FT[1, ..., n]$ be another array; $FT[i]$, for any i, will store the sum of all elements in the range $[i - LS(i) + 1, i]$, where $LS(i)$ is the *least-significant bit* of i. For example, if i is a power of 2, then $LS(i) = i$, and hence $FT[i]$ will be the sum

$$FT[i] = \sum_{k=1}^{i} A[k]$$

Here is the nice trick: given an index b, we want to compute

$$\sum_{k=1}^{b} A[k]$$

To do this, we compute the values $\{FT[b_0], FT[b_1], ..., FT[b_k]\}$. Here, $b_0 = b$ and $b_i = b_{i-1} - LS(b_{i-1})$ for each suitable i. In simple words, starting with b, we strip off the least significant bit one by one. We only need to do this for $O(\log b)$ many steps. Using this, given the function FT is *good enough* (for example, here FT is

Date: December 2019.

the sum function, which has many nice properties), we can compute range queries in logarithmic time.

Let's see how this structure handles update queries. Suppose we want to update the element at index k. So, we want to change $A[k]$ to some new value. So, we will have to update $FT[i]$ for all those indices i which include the index k in their summation. In other words, we want to update all i such that

$$i - LS(i) + 1 \leq k \leq i$$

Then, by doing casework on $LS(i)$, we see that all such i are obtained by the sequence $\{c_0, ..., c_k\}$, where $c_0 = k$ and $c_{i+1} = c_i + LS(k)$. In other words, starting from k, we add least significant digits (this is one of the beautiful connections between these operations). So again, we only need to update $O(\log n)$ many values.