Binary Indexed Trees (Part 2)

Sreenath Are, Ryan Jian

October 11, 2013

1 Review – Augmented Binary Trees

Last week, we discussed a data structure that supports the following operations on a list x of length N, given associative binary operator $\oplus$:

1. Query(a, b): Compute the sum $x_a \oplus x_{a+1} \oplus x_{a+2} \oplus ... \oplus x_b$.
2. Update(i, v): Set x_i to v.

We achieved this by storing the $N/2$ sums of consecutive elements in x, and the $N/4$ sums of consecutive elements in this new list, and so on. These lists form a binary tree, where each element has at most two children that its value depends on. Changing one element in x involves one update on each level of the tree, which is $h = O(\log N)$. Any query can be evaluated by splitting the requested range into ranges covered by a total of at most two nodes per level. We can store this binary tree as an implicit structure in a regular array: let X_1 be the topmost node, and X_{2k} and X_{2k+1} be the two children of node X_k.

2 Range Updates and Range Queries

Now that we have the basics of BITs down we want to be able to efficiently support range updates and range queries, that is we want to be able to add val to $[a..b]$ and find the cumulative sum of $[a..b]$. To support range updates, notice that $Update(a, v)$ only affects indices greater than or equal to a and that if we run $Update(b + 1, -v)$ then we can cancel out part of the first update we performed. However, if we call $Query(a, b)$, the value stays the same when it should return $v * (b - (a - 1))$. Thus, we can no longer perform range queries, only point queries of the form $Query(i)$ that return the sum over $[0..i]$.

To get around the above problem we need to take a slightly different approach. For now, suppose we have some method of performing an update that adds val to $[a..b]$ (for the sake of simplicity we'll assume the remaining values are all 0). After a single update, the sum over the range $[0..p]$ is:

1. $0; 0 \leq p < a$
2. $val * (p - (a - 1)); a \leq p \leq b$
3. $val * (b - (a - 1)); b < p < N$

Notice that for an index p we can get the above sum by subtracting some factor X from $v * p$ for some value v:

1. $X = 0, v = 0; 0 \leq p < a$
2. $X = val, *(a - 1), v = val; a \leq p \leq b$
3. $X = val * (a - 1) - val * b, v = 0; b < p < N$

The form of these equations suggests that we should maintain another BIT, which we'll denote $B2$, to keep track of the X values in addition to our original BIT, which we'll call $B1$ that keeps track of v.

More specifically, when we add val to $[a..b]$ in $B1$, we need to call $Update_{B2}(a, val * (a-1))$ and $Update_{B2}(b+1, -val * b)$. The cumulative sum of $[0..p]$ then can be found by $Query_{B1}(p) * p - Query_{B2}(p)$.

Intuitively, what we have just done is to notice that the cumulative sum of a constant valued range update is in the form of the equation of a line. Computing a prefix sum is the discrete analogue of integration and so our observation makes sense because the integral of a non-zero constant function is a linear function. Consequently, given the index we're calculating the sum to, the sum can be computed using only TWO parameters as opposed to an arbitrary number like before. These two parameters correspond to the slope and intercept of the line, and are also the reason for the number of BITs we need.

3 Problems

1. (Brian Dean, 2012) FJ has set up a cow race with N $(1 \leq N \leq 100,000)$ cows running L laps around a circular track of length C $(1 \leq L, C \leq 25,000)$. The cows all start at the same point on the track and run at different speeds, with the race ending when the fastest cow has run the total distance of $L * C$. FJ notices several occurrences of one cow overtaking another. Count the total number of crossing events during the entire race.

2. (Brian Dean, 2011) Farmer John has lined up his N $(1 \leq N \leq 100,000)$ cows each with height H_i $(1 \leq H_i \leq 1,000,000,000)$ to take a picture of a contiguous subsequence of the cows, such that the median height is at least a certain threshold X $(1 \leq X \leq 1,000,000,000)$. Count the number of possible subsequences.

3. (SPOJ BRCKTS) Given a bracket expression of length N $(1 \leq N \leq 30,000)$ process M operations. There are two types of operations, a replacement, which changes the ith bracket into its opposite, and a check, which determines whether a bracket expression is correct.

4. (Codeforces ABBYY Cup 3.0 B2) The Smart Beaver has many beavers, numbered from 1 to N. He wants to shave the beavers using a special machine, the Beavershave 5000. The Beavershave 5000 will shave any subsequence of beavers whose numbers are strictly increasing from left to right in a single pass. Write a program to support the following operations: 1. Determine the minimum number of Beavershave 5000 passes to shave all the beavers with ids between x and y. 2. Swap beavers x and y.

5. (Codeforces Round #197 Div. 2 D) Xenia has a sequence a, consisting of $2N$ non-negative integers for which he wants to calculate some value v. First, Xenia writes down the OR of adjacent elements of sequence a. Next, Xenia writes the XOR of adjacent elements of the sequence obtained after the first iteration. At the third iteration Xenia writes the OR of the adjacent elements of the sequence obtained after the second iteration. And so on; the operations of OR and XOR alternate until Xenia is left with a single number v. You are given Xenia's initial sequence and m queries. Each query is a pair of integers p, b that means that you need to perform the assignment $a_p = b$. After each query, you need to print the new value v for the new sequence a.

6. Support range updates where the values form an arithmetic progression. (Hint: what is the integral of a linear function?)