WEDNESDAY, NOVEMBER 27, 2013

Various Usage of B.I.T.

I've always imagined what BIT (I will omit the dots, but don't confuse it with binary bits) can do and can't. I have seen numerous times, experienced solvers say "Oh it's easily solvable using BIT" leaving me wondering, "BIT on this problem! How on earth!" Still I haven't found quite a good answer for this question. Looks like BIT can do as much as you can imagine it to do (and obviously design it to do so).

However, I am not going to explain the structure of BIT or how it works etc. As there are lots of well documented articles available on the net. I can't help putting one specific link here, which is a TopCoder algorithm tutorial for BIT. So, if you are not familiar with BIT yet, first go ahead, and read that post.

In this post, we will discuss some of the most common application of BIT for solving data structure related problems, more specifically, where range sum updates and queries are involved.

Some methods used in this post:
Update(x, v): Adds V at the position x in BIT.
Query(x): Returns the cumulative value on position x in BIT.
Read the above tutorial to know how these methods are implemented.
We will be using 1-based indexing throughout the post.

(I) Point Update, Range Query:

This is the most common form of BIT, and just about 99% of online tutorials you will find on the net will describe this application. So here we don't have much to say. If your updates are like "Add v in the x position of the array", and queries are like "What is the sum of elements in index range [a, b] in the array?", then all you do for update is call Update(x, v) and answer queries by Query(b) - Query(a-1). Simple cumulative sum formula, nothing really astonishing. This problem is can be solved using BIT: SPOJ - MSE06H.

(II) Range Update, Point Query:

Although not something you see every now and then, but it's readily doable. Now you are required to update v over the range [a, b] and the query is performed on a single index x. So now you call update twice, first add v at index a, then remove v for indices larger than b, which is: Update(a, v); Update(b+1, -v); And for query, you simply call Query(x). Now to see why this works, the following reasoning can be made:

Lets consider, initially you have everything 0. So Query(x) will return 0, no matter what index you call it with. Suppose you have called Update(a, v) and Update(b+1, -v). Now what will happen if you call Query(x)? Three cases can arise:
1. if $x < a$, the query is not affected by the updates. So you get correct result.
2. if $x > b$, x will be affected by the update(a, v) since $x \ge a$, and update(b+1, -v) since $x \ge b+1$, therefore, v-v=0 so everything cancels out. Again, you get the correct result.
3. $a \le x \le b$. x is only affected by update(a, v), but not update(b+1, -v), therefore, Query(x)'s value is increased by v, and it will return the correct result.

As we can see, in this fashion, we can increment or decrement for a range [a, b], and get a single index effect for some index x. This problem is an example: SPOJ - UPDATEIT

(III) Range Update, Range Query:

I didn't even know it exists until I read some post in TopCoder forums (the post was made by: AnilKishore).

I will just re-state his explanation as it was quite clear itself. All we want here is to support range updates, and do range queries. As from the previous type, if we try to store range updates, the BIT structure effectively captures updates for single positions, instead of range/cumulative sums. However, we can do some tweaking to work around this problem.

Let's just consider only one update: Add v to [a, b] while the rest elements of the array is 0.
Now, consider sum(0, x) for all possible x, again three situation can arise:
1. $0 \le x < a$: which results in 0
2. $a \le x \le b$: we get v * (x - (a-1))
3. $b < x < n$: we get v * (b - (a-1))

This suggests that, if we can find v*x for any index x, then we can get the sum(0, x) by subtracting T from it, where:
1. $0 \le x <$: Sum should be 0, thus, T = 0
2. $a \le x \le$: Sum should be v*x-v*(a-1), thus, T = v*(a-1)
3. $b < x < n$: Sum should be 0, thus, T = -v*b + v*(a-1)

As, we can see, knowing T solves our problem, we can use another BIT to store this additive amount from which we can get:
0 for $x < a$, v*(a-1) for x in [a..b], -v*b+v(a-1) for $x > b$.

Now we have two BITs.
To add v in range [a, b]: Update(a, v), Update(b+1, -v) in the first BIT and Update(a, v*(a-1)) and Update(b+1, -v*b) on the second BIT.
To get sum in range [0, x]: you simply do Query_BIT1(x)*x - Query_BIT2(x);
Now you know how to find range sum for [a, b]. Just find sum(b) - sum(a-1) using the formula stated above.

Pretty impressive this one! SPOJ - HORRIBLE can be solved in this approach. And thanks to Iqram Mahmud for this nice problem.

(IV) 2D BIT

How to write Update and Query methods for 2D BIT is well described in the TopCoder tutorial I've mentioned above. The only thing to notice here is how the queries and updates work. 2D BIT is basically a BIT where each element is another BIT. Adding v on (x, y) means it's effect will be found throughout the rectangle [(x, y), (W, H)], and query for (x, y) gives you the result of the rectangle [(0, 0), (x, y)], assuming the total rectangle is [(0, 0), (W, H)]. SO when you query and update on this BIT, you have to be careful about how many times you are subtracting a rectangle and adding it. Simple set union formula works here. So if you want to get the result of a specific rectangle [(x1, y1), (x2, y2)], the following steps are necessary to do so:
V = Query(x2, y2)
V -= Query(x2, y1-1)
V -= Query(x1-1, y2)
V += Query(x1-1, y1-1)

This problem is an example: SPOJ - MATSUM

This page is likely to be updated if I find and manage to solve new types of BIT problems. Please let me know if there are mistakes. This post is inspired by some random posts lying around in TopCoder forums, I just wanted to put them in one place for future reference.