Nó đang giả sử một mảng toàn số 0 (để dễ hình dung) và hai câu lệnh mình đã thực hiện update sau đó **tìm giá trị của index x**. Lưu ý quan trọng là chỉ tìm index x mà thôi.

Theo bài thì:

TH1: x < a (left) thì cái queries này sẽ trả về 0 với trường hợp này (Nghĩa là nó sẽ trả về giá trị phần tử tại đó mà không thay đổi.)

TH2: x > b (right) thì cái x này vẫn sẽ trả về đúng giá trị đó vì lúc này x>= a thì ảnh hưởng với update(a,v) sau đó ảnh hưởng bởi update(b+1,-v), khi này thì v-v = 0 nên chả có gì xảy ra.

TH3: a <= x <= b, khi này chỉ ảnh hưởng bởi update(a,v) mà kh ảnh hưởng bởi update(b+1,-v) nên giá trị tăng đúng bằng v.



Code bên trên thực hiện update trên một mảng 0 gồm 5 phần tử. Thực hiện dựng BIT thông qua hàm updateBIT.

Mảng [0,0,0,0,0] giờ sẽ được update từ 2 đến 4 (mảng này là từ 0 đến 4).

Giờ đến đoạn update:

Thực hiện update từ 2 đến 4 thông qua hàm update => Hàm thực hiện 2 thao tác đơn gồm updateBIT ở vị trí l (trái) lên 2 và r+1 (phải +1). Tại sao?

Tại như đã giải thích ở trên, khi xác định vị trí của x sẽ phải đi qua 3 TH đã nêu nên là trên lý thuyết chỉ cần update 2 đầu là có thể tìm ra được.

Ví dụ: Sau khi update từ 2 đến 4 thì mảng thành [0,0,2,2,2]. BIT là [0,0,2,0,2]

Tìm vị trí thứ 4 thì sẽ là vị trí thứ 5 trên BIT. Thực hiện: sum+= BIT[5] (=2) rồi dùng công thức bit sẽ ra lệnh sum += BIT[4] (=2) rồi thực hiện tiếp công thức bit ra index = 0, kết thúc. Thế là ra kq là 2. Tương tự với những index khác.

Khi này cuối cùng mình sẽ dùng một mảng ban đầu + với giá trị trên thì sẽ ra giá trị tại vị trí đó.

Để hỗ trợ range update và sau đó **thực hiện range queries.** Lưu ý range queries là thứ chính.

BIT thực hiện update từng điểm rất tốt nên mình thay đổi một xíu kĩ thuật để nó thực hiện được cho cả đoạn (range queries).

Lại thực hiện trên một mảng 0 để dễ hình dung với việc thêm v cho đoạn [a,b].

Giờ công việc mình làm là tính tổng từ 0 đến index x (tất nhiên là sau khi đã thêm v cho đoạn [a,b]).

TH1: 0 <= x < a: ra kết quả là 0 vì không có đoạn v nào được cập nhật tại đây.

TH2: a <= x < b: ra kết quả là v \* (x – (a-1)) vì có nghĩa là v được cập nhật đúng x – (a – 1) lần.

TH3: b <= x < n: ra kết quả là v \* (b – (a-1)) vì có nghĩa là v được cập nhất đúng từ a-1 đến b lần.

Đến đây chúng ta sẽ nghĩ đến một vấn đề, nếu có phải có sẵn v và x thì có thể tính dễ dàng v\*x, khi này chúng ta sẽ lấy nó rồi trừ T. Với T là:

Thì 3 trường hợp trên sẽ biến thành:

TH1: 0 <= x < a: ra kết quả là 0 nên T = 0

TH2: a <= x < b: ra kết quả là v \* (x – (a-1)) nên T = v\*(a-1)

TH3: b <= x < n: ra kết quả là 0 nên T = -v\*b + v\*(a-1)

Từ giờ chỉ cần thêm một BIT để lưu trữ T. Như vậy ta có 2 cây đúng theo công thức này:

Update(a, v), Update(b+1, -v) in the first BIT.

Update(a, v\*(a-1)) and Update(b+1, -v\*b) on the second BIT.

Từ khi này tính tổng sum(0,x) đơn giản là Query\_BIT1(x)\*x - Query\_BIT2(x).

Cuối cùng là tính tổng sum(a,b) thì là sum(0,b) – sum(0,a-1)

     updateBit(BITTree1, n, r + 1, -val)

    updateBit(BITTree2, n, l, val \* (l - 1))

    updateBit(BITTree2, n, r + 1, -val \* r)

def rangeSum(l: int, r: int, BITTree1: list, BITTree2: list) -> int:

    return summation(r, BITTree1, BITTree2) - summation(l - 1, BITTree1, BITTree2)

if \_\_name\_\_ == "\_\_main\_\_":

    n = 5

    BITTree1 = [0] \* (n + 1)

    BITTree2 = [0] \* (n + 1)

    l = 2

    r = 4

    val = 5

    updateRange(BITTree1, BITTree2, n, val, l, r)

    l = 1

    r = 4

    print("Sum of elements from [%d,%d] is %d" %(l, r, rangeSum(l, r, BITTree1, BITTree2)))

Code bên trên thực hiện tính range sum từ một mảng 0 gồm 5 phần tử. Thực hiện update đoạn từ index 2 đến 4 lên 5 đơn vị (mảng này từ 0).

Sau đó tính tổng từ 1 đến 4 thông qua hàm rangeSum ( lấy sum(0,r) – sum(0,l-1))

Kết quả đạt được là 15.

Thông qua đoạn giải thích bên trên, chúng ta hiểu là mình phải lưu đoạn chênh (giống bài trước) vào một BIT (2) riêng biệt đồng thời là một BIT (1) để lưu hòng truy xuất từng điểm.

Ví dụ thực hiện update:

Hàm updaterange thực hiện update cho BIT1 như bài trước, đồng thời update BIT2 cho các đoạn chênh đã giải thích.

Khi này mảng ban đầu là: [0,0,5,5,5]. BIT1 là [0,0,0,5,5], BIT2 là [0,0,0,5,5].

Giải thích một chút: BIT1 đã biết thì không giải thích lại. BIT2 là update một giá trị là val \*(l-1) = 5\*(2-1) = 5 nên là sẽ update ở vị trí 3 và 4 trong mảng ( nghĩa là vị trí 4 và 5 trên BIT). BIT2 đồng thời cộng một giá trị -val\*r vào đoạn r+1 (ở đây nó quá tầm nên không có gì xảy ra cả, nếu có gì thì sẽ thành số âm thường là vậy).

Khi này thực hiện hàm tính tổng bằng hàm rangeSum.

Hàm rangeSum thực hiện summation từ cận phải – hàm này thực hiện hàm getSum (thực ra là truy xuất giá trị của x) rồi lấy giá trị của x nhân cho giá trị của index cận trái đem trừ cho hàm getSum lấy giá trị của x cho BIT2.

Thực tế hàm này sẽ tính như sau: trả về giá trị 5 \* 4 – 5 = 15.

Hàm rangeSum tiếp tục thực hiện summation từ cận trái và thực hiện như trên.

Thực tế hàm này sẽ tính như sau: trả về giá trị 0 \* 2 – 0 = 0.

Cuối cùng rangeSum sẽ trừ hai giá trị còn lại: 15 – 0 = 15. Đây chính là giá trị đúng.

Đối với một bài cụ thể với mảng khác 0. Mình có thể gọi cây BIT thứ 3 để tính rangeSum như bình thường sau đó cộng nó với giá trị đã ra trong bài trên.