Program Name- Two Dimensional Binary Indexed Tree a.k.a Fenwick Tree

Project Category- Advanced Data Structure

Pre-requisites- http://www.geeksforgeeks.org/binary-indexed-tree-or-fenwick-tree-2/

Explanation-

We know that to answer range sum queries on a 1-D array efficiently, binary indexed tree (or Fenwick Tree) is the best choice (even better than segment tree due to less memory requirements and a little faster than segment tree).

But can we answer sub-matrix sum queries efficiently using Binary Indexed Tree ?

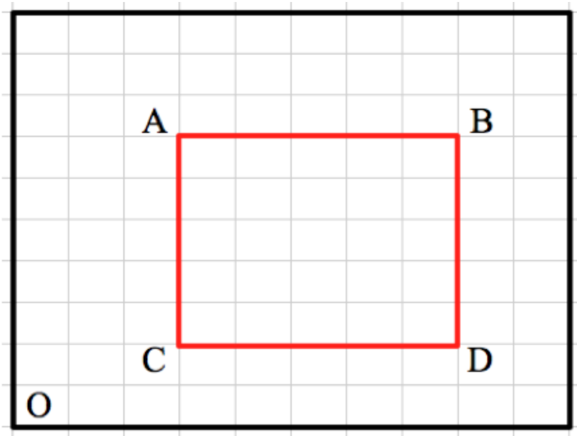
The answer is yes. This is possible using a 2D BIT.

What is a 2D BIT ?

A 2D BIT is nothing but an array of 1D BIT. (For more on 1D BIT see- http://www.geeksforgeeks.org/binary-indexed-tree-or-fenwick-tree-2/)

Algorithm-

We consider the below example. Suppose we have to find the sum of all numbers inside the highlighted area-

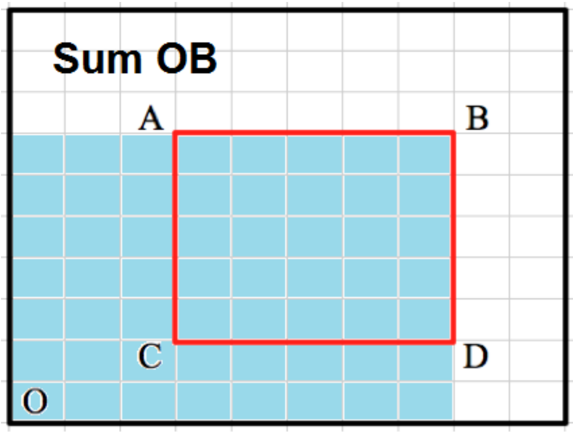


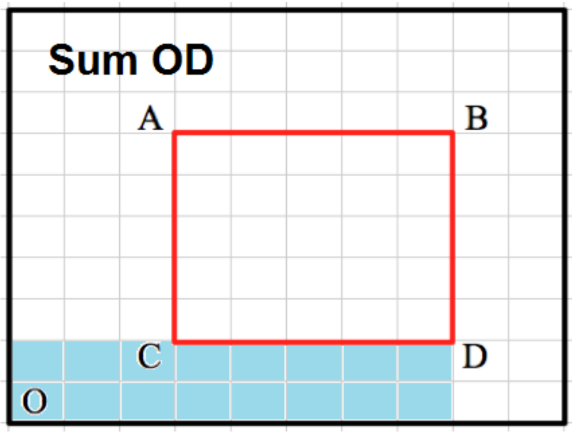
70	37	23	57	27	22	90	99	22	59
47	63	33	1	42	46	6	70	98	93
36	62	50	21	92	27	60	29	15	34
53	3	88	45	57	39	83	81	79	56
28	63	89	20	47	15	84	18	82	33
26	87	11	76	79	5	94	55	73	51
17	82	86	10	96	5	42	43	51	6
44	76	51	4	15	99	52	11	70	89
66	36	92	85	50	21	72	27	52	65
60	0	67	37	59	14	33	13	36	36

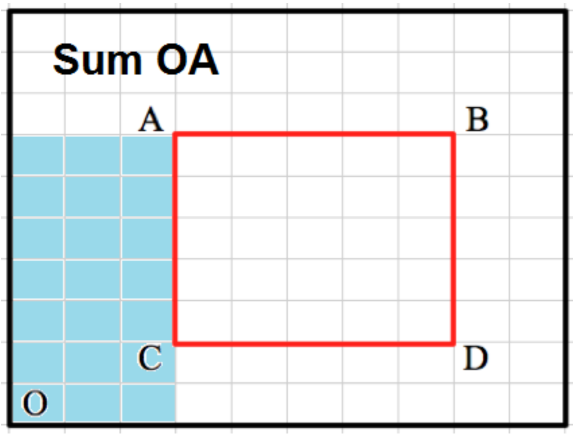
We assume the origin of the matrix at the bottom - O

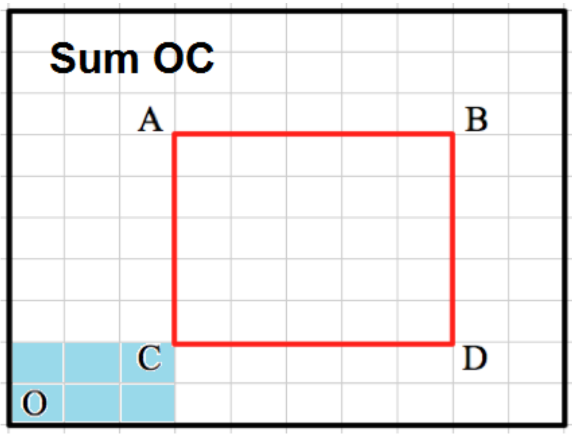
Then a 2D BIT exploits the fact that-

Sum under the marked area = Sum(OB) - Sum(OD) – Sum(OA) + Sum(OC)









In our program, we use the getSum(x, y) function which finds the sum of the matrix from (0, 0) to (x, y).

Hence the below formula

→Sum under the marked area = Sum(OB) - Sum(OD) – Sum(OA) + Sum(OC) gets reduced to

→ Query(x1,y1,x2,y2) = getSum(x2, y2)-getSum(x2, y1-1)-getSum(x1-1, y2)+getSum(x1-1, y1-1)

where,

x1 = x-coordinate of C

y1 = y-coordinate of C

x2 = x-coordinate of B

y3 = y-coordinate of B

The updateBIT(x, y, val) function updates all the elements under the region – (x, y) to (N, M)

where,

N = maximum X co-ordinate of the whole matrix.

M = maximum Y co-ordinate of the whole matrix.

The rest procedure is quite similar to that of 1D Binary Indexed Tree.

Time Complexity-

Both updateBIT(x, y, val) function and getSum(x, y) function takes O(log(NM)) time.

Building the 2D BIT takes O(NM log(NM))

Since in each of the queries we are calling getSum(x, y) function so answering all the Q queries takes O(Q.log(NM)) time.

Hence the overall time complexity of the program is O((NM+Q).log(NM)) where,

N = maximum X co-ordinate of the whole matrix.

M = maximum Y co-ordinate of the whole matrix.

Q = Number of queries.

Auxiliary Space -

O(NM) to store the BIT and the auxiliary array

References-

https://www.topcoder.com/community/data-science/data-science-tutorials/binary-indexed-trees/