Two Dimensional Segment Tree | Sub-Matrix Sum

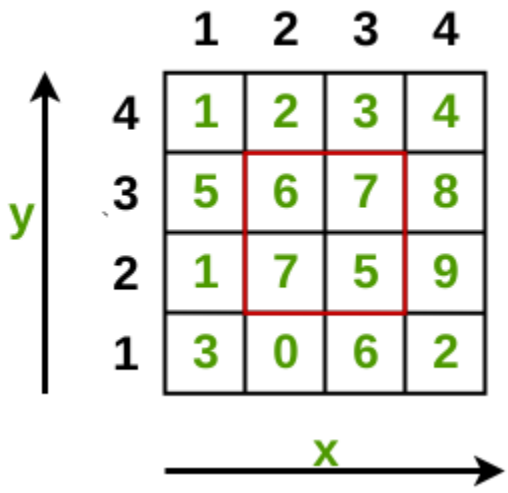
Given a rectangular matrix **M[0…n-1][0…m-1]**, and queries are asked to find the sum / minimum / maximum on some sub-rectangles **M[a…b][e…f]**, as well as queries for modification of individual matrix elements (i.e **M[x] [y] = p**).

We can also answer sub-matrix queries using Two Dimensional Binary Indexed Tree.

In this article, We will focus on solving sub-matrix queries using two dimensional segment tree.Two dimensional segment tree is nothing but segment tree of segment trees.

Prerequisite : Segment Tree – Sum of given range

Algorithm :
We will build a two-dimensional tree of segments by the following principle:
1 . In First step, We will construct an ordinary one-dimensional segment tree, working only with the first coordinate say 'x' and 'y' as constant. Here, we will not write number in inside the node as in the one-dimensional segment tree, but an entire tree of segments.
2. The second step is to combine the values of segmented trees. Assume that in second step instead of combining the elements we are combining the segment trees obtained from the step first.
Consider the below example. Suppose we have to find the sum of all numbers inside the highlighted red area



Step 1 : We will first create the segment tree of each strip of y- axis.We repesent the segment tree here as an array where child node is **2n** and **2n+1** where n > 0.

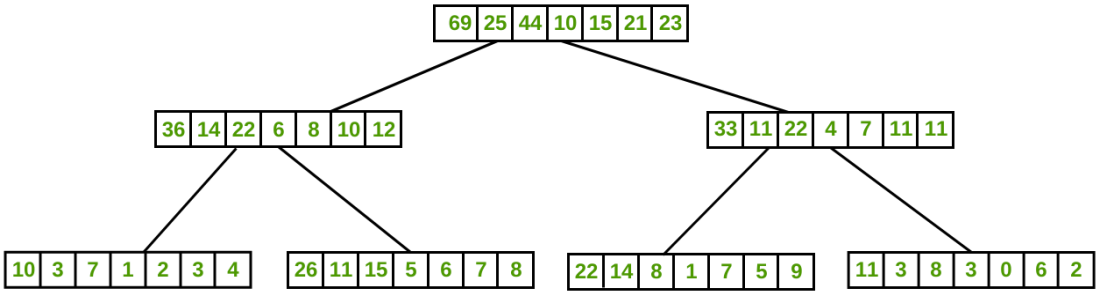
Segment Tree for strip **y=1**

Segment Tree for Strip **y = 2**

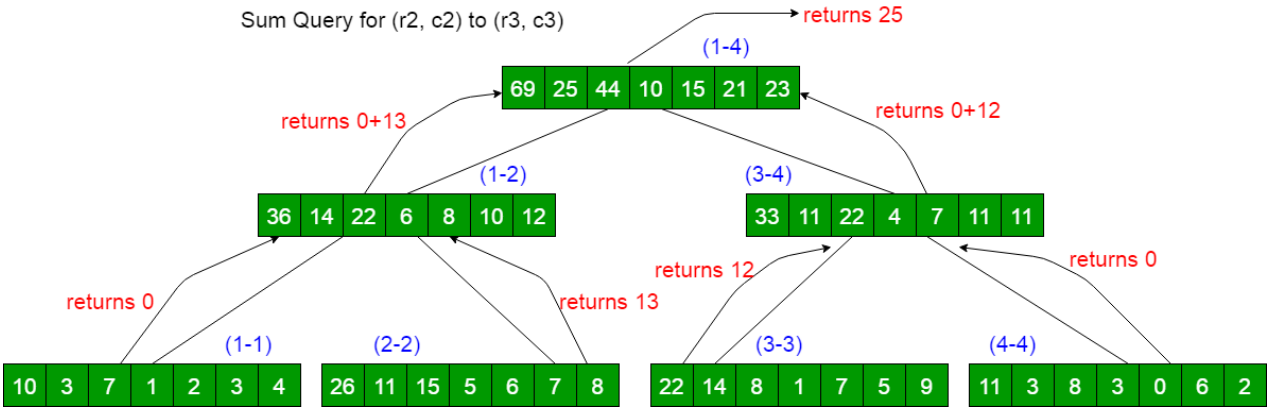
Segment Tree for Strip **y = 3**

Segment Tree for Strip **y = 4**

Step 2: In this step, we create the segment tree for the rectangular matrix where the base node are the strips of y-axis given above.The task is to merge above segment trees.



Sum Query :



Thanks to **Sahil Bansal** for contributing this image.

Processing Query :

We will respond to the two-dimensional query by the following principle: first to break the query on the first coordinate, and then, when we reached some vertex of the tree of segments with the first coordinate and then we call the corresponding tree of segments on the second coordinate.

This function works in time **O(logn*log m)**, because it first descends the tree in the first coordinate, and for each traversed vertex of that tree, it makes a query from the usual tree of segments along the second coordinate.

Modification Query :

We want to learn how to modify the tree of segments in accordance with the change in the value of an element **M[x] [y] = p** .It is clear that the changes will occur only in those vertices of the first tree of segments that cover the coordinate x, and for the trees of the segments corresponding to them, the changes will only occur in those vertices that cover the coordinate y. Therefore, the implementation of the modification request will not be very different from the one-dimensional case, only now we first descend the first coordinate, and then the second.

Output for the highlighted area will be 25.

Below is the implementation of above approach :

```
// C++ program for implementation

// of 2D segment tree.

#include <bits/stdc++.h>

using namespace std;


// Base node of segment tree.

int ini_seg[1000][1000] = { 0 };


// final 2d-segment tree.

int fin_seg[1000][1000] = { 0 };


// Rectangular matrix.

int rect[4][4] = {

   { 1, 2, 3, 4 },

   { 5, 6, 7, 8 },

   { 1, 7, 5, 9 },

   { 3, 0, 6, 2 },
```

```
};

// size of x coordinate.
int size = 4;

/*
 * A recursive function that constructs
 * Inital Segment Tree for array rect[][] = { }.
 * 'pos' is index of current node in segment
 * tree seg[]. 'strip' is the enumeration
 * for the y-axis.
*/

int segment(int low, int high,
          int pos, int strip)
{
   if (high == low) {
      ini_seg[strip][pos] = rect[strip][low];
   }
   else {
      int mid = (low + high) / 2;
      segment(low, mid, 2 * pos, strip);
      segment(mid + 1, high, 2 * pos + 1, strip);
      ini_seg[strip][pos] = ini_seg[strip][2 * pos] +
                        ini_seg[strip][2 * pos + 1];
   }
}
```

```
/*
 * A recursive function that constructs
 * Final Segment Tree for array ini_seg[][] = { }.
*/
int finalSegment(int low, int high, int pos)
{
   if (high == low) {

      for (int i = 1; i < 2 * size; i++)
         fin_seg[pos][i] = ini_seg[low][i];
   }
   else {
      int mid = (low + high) / 2;
      finalSegment(low, mid, 2 * pos);
      finalSegment(mid + 1, high, 2 * pos + 1);

      for (int i = 1; i < 2 * size; i++)
         fin_seg[pos][i] = fin_seg[2 * pos][i] +
                     fin_seg[2 * pos + 1][i];
   }
}

/*
* Return sum of elements in range from index
* x1 to x2 . It uses the final_seg[][] array
* created using finalsegment() function.
* 'pos' is index of current node in
* segment tree fin_seg[][].
```

```
*/
int finalQuery(int pos, int start, int end,
            int x1, int x2, int node)
{
   if (x2 < start || end < x1) {
      return 0;
   }

   if (x1 <= start && end <= x2) {
      return fin_seg[node][pos];
   }

   int mid = (start + end) / 2;
   int p1 = finalQuery(2 * pos, start, mid,
                  x1, x2, node);

   int p2 = finalQuery(2 * pos + 1, mid + 1,
                  end, x1, x2, node);

   return (p1 + p2);
}

/*
* This fuction calls the finalQuery fuction
* for elements in range from index x1 to x2 .
* This fuction queries the yth coordinate.
*/
int query(int pos, int start, int end,
```

```
        int y1, int y2, int x1, int x2)
{
   if (y2 < start || end < y1) {
      return 0;
   }

   if (y1 <= start && end <= y2) {
      return (finalQuery(1, 1, 4, x1, x2, pos));
   }

   int mid = (start + end) / 2;
   int p1 = query(2 * pos, start,
                  mid, y1, y2, x1, x2);
   int p2 = query(2 * pos + 1, mid + 1,
                  end, y1, y2, x1, x2);

   return (p1 + p2);
}

/* A recursive function to update the nodes
   which for the given index. The following
   are parameters : pos --> index of current
   node in segment tree fin_seg[][]. x ->
   index of the element to be updated. val -->
   Value to be change at node idx
*/
int finalUpdate(int pos, int low, int high,
                int x, int val, int node)
```

```
{
   if (low == high) {
      fin_seg[node][pos] = val;
   }
   else {
      int mid = (low + high) / 2;

      if (low <= x && x <= mid) {
         finalUpdate(2 * pos, low, mid, x, val, node);
      }
      else {
         finalUpdate(2 * pos + 1, mid + 1, high,
                              x, val, node);
      }

      fin_seg[node][pos] = fin_seg[node][2 * pos] +
                        fin_seg[node][2 * pos + 1];
   }
}

/*
 This funtion call the final update function after
 visiting the yth coordinate in the segment tree fin_seg[][].
*/
int update(int pos, int low, int high, int x, int y, int val)
{
   if (low == high) {
      finalUpdate(1, 1, 4, x, val, pos);
```

```
    }
    else {
        int mid = (low + high) / 2;

        if (low <= y && y <= mid) {
            update(2 * pos, low, mid, x, y, val);
        }
        else {
            update(2 * pos + 1, mid + 1, high, x, y, val);
        }

        for (int i = 1; i < size; i++)
            fin_seg[pos][i] = fin_seg[2 * pos][i] +
                              fin_seg[2 * pos + 1][i];
    }
}

// Driver program to test above functions
int main()
{
    int pos = 1;
    int low = 0;
    int high = 3;

    // Call the ini_segment() to create the
    // inital segment tree on x- coordinate
    for (int strip = 0; strip < 4; strip++)
        segment(low, high, 1, strip);
```

```
    // Call the final function to built the 2d segment tree.
    finalSegment(low, high, 1);

    /*
  Query:
  * To request the query for sub-rectangle y1, y2=(2, 3) x1, x2=(2, 3)
  * update the value of index (3, 3)=100;
  * To request the query for sub-rectangle y1, y2=(2, 3) x1, x2=(2, 3)
 */
    cout << "The sum of the submatrix (y1, y2)->(2, 3), "
         << " (x1, x2)->(2, 3) is "
         << query(1, 1, 4, 2, 3, 2, 3) << endl;

    // Function to update the value
    update(1, 1, 4, 2, 3, 100);

    cout << "The sum of the submatrix (y1, y2)->(2, 3), "
         << "(x1, x2)->(2, 3) is "
         << query(1, 1, 4, 2, 3, 2, 3) << endl;

    return 0;
}
```

Copy CodeRun on IDE

Output:

The sum of the submatrix (y1, y2)->(2, 3), (x1, x2)->(2, 3) is 25

The sum of the submatrix (y1, y2)->(2, 3), (x1, x2)->(2, 3) is 118

Time complexity :
Processing Query : O(logn*logm)

Modification Query: O(2*n*logn*logm)
Space Complexity : O(4*m*n)