Maximum size rectangle binary sub-matrix with all 1s

Given a binary matrix, find the maximum size rectangle binary-sub-matrix with all 1's.

```
Input :   0 1 1 0
          1 1 1 1
          1 1 1 1
          1 1 0 0

Output :  1 1 1 1
          1 1 1 1
```

We strongly recommend you to minimize your browser and try this yourself first.

We have discussed a dynamic programming based solution for finding largest square with 1s.

In this post an interesting method is discussed that uses largest rectangle under histogram as a subroutine. Below are steps. The idea is to update each column of a given row with corresponding column of previous row and find largest histogram area for for that row.

```
Step 1: Find maximum area for row[0]
Step 2:
    for each row in 1 to N - 1
        for each column in that row
            if A[row][column] == 1
              update A[row][column] with
                A[row][column] += A[row - 1][column]
    find area for that row
    and update maximum area so far
```

Illustration :

```
step 1:    0 1 1 0  maximum area  = 2
step 2:
    row 1  1 2 2 1  area = 4, maximum area becomes 4
    row 2  2 3 3 2  area = 8, maximum area becomes 8
    row 3  3 4 0 0  area = 6, maximum area remains 8
```

Below is C++ implementation. It is strongly recommended to refer this post first as most of the code taken from there.

```
// C++ program to find largest rectangle with all 1s
// in a binary matrix
#include<bits/stdc++.h>
using namespace std;

// Rows and columns in input matrix
#define R 4
#define C 4

// Finds the maximum area under the histogram represented
// by histogram.  See below article for details.
// http://www.geeksforgeeks.org/largest-rectangle-under-histogram/
int maxHist(int row[])
{
    // Create an empty stack. The stack holds indexes of
    // hist[] array/ The bars stored in stack are always
    // in increasing order of their heights.
    stack<int> result;

    int top_val;  // Top of stack

    int max_area = 0; // Initialize max area in current
                      // row (or histogram)

    int area = 0;     // Initialize area with current top
```

```
    int area = 0;    // Initialize area with current top

    // Run through all bars of given histogram (or row)
    int i = 0;
    while (i < C)
    {
        // If this bar is higher than the bar on top stack,
        // push it to stack
        if (result.empty() || row[result.top()] <= row[i])
            result.push(i++);

        else
        {
            // If this bar is lower than top of stack, then
            // calculate area of rectangle with stack top as
            // the smallest (or minimum height) bar. 'i' is
            // 'right index' for the top and element before
            // top in stack is 'left index'
            top_val = row[result.top()];
            result.pop();
            area = top_val * i;

            if (!result.empty())
                area = top_val * (i - result.top() - 1 );
            max_area = max(area, max_area);
        }
    }

    // Now pop the remaining bars from stack and calculate area
    // with every popped bar as the smallest bar
    while (!result.empty())
    {
        top_val = row[result.top()];
        result.pop();
        area = top_val * i;
        if (!result.empty())
            area = top_val * (i - result.top() - 1 );

        max_area = max(area, max_area);
    }
    return max_area;
}

// Returns area of the largest rectangle with all 1s in A[][]
int maxRectangle(int A[][C])
{
    // Calculate area for first row and initialize it as
    // result
    int result = maxHist(A[0]);

    // iterate over row to find maximum rectangular area
    // considering each row as histogram
    for (int i = 1; i < R; i++)
    {

        for (int j = 0; j < C; j++)

            // if A[i][j] is 1 then add A[i -1][j]
            if (A[i][j]) A[i][j] += A[i - 1][j];


        // Update result if area with current row (as last row)
        // of rectangle) is more
        result = max(result, maxHist(A[i]));
    }

    return result;
}

// Driver code
int main()
{
    int A[][C] = { {0, 1, 1, 0},
```

```
                  {1, 1, 1, 1},
                  {1, 1, 1, 1},
                  {1, 1, 0, 0},
                };

    cout << "Area of maximum rectangle is "
         << maxRectangle(A);

    return 0;
}
```

Output :

```
Area of maximum rectangle is 8
```

Time Complexity : O(R x X)