Gfg editorial of max rectangle in matrix

Saturday, October 30, 2021 7:45 PM

Algorithm:

1. Run a loop to traverse through the rows.
2. Now If the current row is not the first row then update the row as follows, if matrix[i][j] is not zero then matrix[i][j] = matrix[i-1][j] + matrix[i][j].
3. Find the maximum rectangular area under the histogram, consider the ith row as heights of bars of a histogram. This can be calculated as given in this article Largest Rectangular Area in a Histogram
4. Do the previous two steps for all rows and print the maximum area of all the rows.
   Note: It is strongly recommended to refer this post first as most of the code taken from there.

Implementation

- C++
- Java
- Python3
- C#
- Javascript

```
// Java program to find largest rectangle with all 1s
// in a binary matrix
import java.io.*;
import java.util.*;

class GFG {
    // Finds the maximum area under the histogram
    // represented by histogram.  See below article
for

    static int maxHist(int R, int C, 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<Integer> result = new Stack<Integer>();

        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

        // 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.peek()] <= 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.peek()];
                result.pop();
```

```
                area = top_val * i;

                if (!result.empty())
                    area
                        = top_val * (i -
result.peek() - 1);
                max_area = Math.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.peek()];
            result.pop();
            area = top_val * i;
            if (!result.empty())
                area = top_val * (i - result.peek() -
1);

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

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

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

        return result;
    }

    // Driver code
    public static void main(String[] args)
    {
        int R = 4;
        int C = 4;

        int A[][] = {
            { 0, 1, 1, 0 },
            { 1, 1, 1, 1 },
            { 1, 1, 1, 1 },
            { 1, 1, 0, 0 },
        };
        System.out.print("Area of maximum rectangle
is "
                         + maxRectangle(R, C, A));
    }
}
```

```
// Contributed by Prakriti Gupta
```

Output

From <https://www.geeksforgeeks.org/maximum-size-rectangle-binary-sub-matrix-1s/>

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

Complexity Analysis:

Time Complexity: O(R x C).
Only one traversal of the matrix is required, so the time complexity is O(R X C)
Space Complexity: O(C).
Stack is required to store the columns, so so space complexity is O(C)

From <https://www.geeksforgeeks.org/maximum-size-rectangle-binary-sub-matrix-1s/>