Maximal Rectangles

Problem Statement: Given a m x n binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

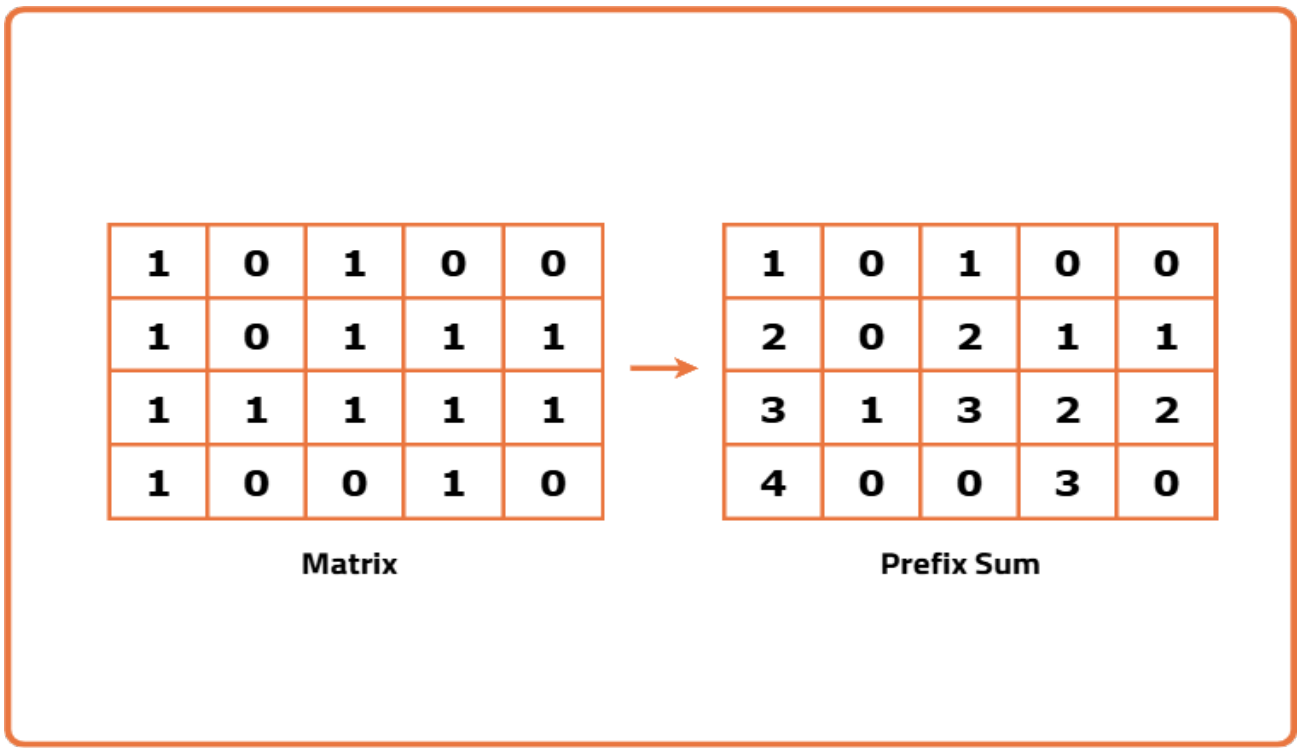
Examples

```
Example 1:
Input: matrix = [[1, 0, 1, 0, 0],
                 [1, 0, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 0, 0, 1, 0]]
Output: 6
Explanation: The largest rectangle of only 1's has area 6, formed by the 2×3
block of 1's in rows 1 and 2, columns 2 to 4.

Example 2:
Input: matrix = [[1]]
Output: 1
Explanation: In this case, there is only one rectangle with area 1.
```

Approach
Algorithm

- Convert the binary matrix into a matrix of histogram heights using the prefix sum technique
- For each cell in the histogram matrix, calculate the height of consecutive 1's ending at that cell
- If a cell contains a 0, reset the height to 0
- For each row in the histogram matrix, treat it as a histogram and compute the largest rectangle area
- Use a stack-based approach to find the largest rectangle in the histogram for each row
- Track the maximum rectangle area found across all rows
- Return the maximum rectangle area as the result



Level 0 :

1	0	1	0	0
2	0	2	1	1
2	1	3	2	2
4	0	0	3	0

maxArea = 1

Level 1 :

1	0	1	0	0
2	0	2	1	1
3	1	3	2	2
4	0	0	3	0

maxArea = 3

Level 2 :

1	0	1	0	0
2	0	2	1	1
2	1	3	2	2
4	0	0	3	0

maxArea = 6

Level 3 :

1	0	1	0	0
2	0	2	1	1
3	1	3	2	2
4	0	0	3	0

maxArea = 6

Code

```
import java.util.*;

class Solution {

    // Function to find the largest rectangle area in a histogram
    private int largestRectangleArea(int[] heights) {
        int n = heights.length; // Size of the array
        Stack<Integer> st = new Stack<>(); // Stack to store indices
        int largestArea = 0; // Variable to store largest area
        int area; // Variable to store current area
        int nse, pse; // Next smaller element index, Previous smaller element
index

        // Traverse through the array
        for (int i = 0; i < n; i++) {
            // Pop elements from the stack until we find a smaller element
            while (!st.isEmpty() && heights[st.peek()] >= heights[i]) {
                int ind = st.pop(); // Get the index of the top element of the
stack
                pse = st.isEmpty() ? -1 : st.peek(); // Previous smaller element
index
                nse = i; // Next smaller element index is current index
                area = heights[ind] * (nse - pse - 1); // Calculate area for the
popped element
                largestArea = Math.max(largestArea, area); // Update largest
area
            }
            st.push(i); // Push the current index onto the stack
        }

        // For elements that are not popped from the stack
        while (!st.isEmpty()) {
            nse = n; // NSE for such elements is size of the array
            int ind = st.pop(); // Get the index of the top element of the stack
            pse = st.isEmpty() ? -1 : st.peek(); // Previous smaller element
index
            area = heights[ind] * (nse - pse - 1); // Calculate the area
            largestArea = Math.max(largestArea, area); // Update largest area
        }

        return largestArea; // Return the largest area found
    }

    // Function to find the largest rectangle area containing all 1s in a matrix
    public int maximalAreaOfSubMatrixOfAll1(int[][] matrix) {
        int n = matrix.length; // Number of rows
        int m = matrix[0].length; // Number of columns
        int[][] prefixSum = new int[n][m]; // Prefix sum matrix to store heights

        // Fill up the prefix sum matrix column wise
        for (int j = 0; j < m; j++) {
            int sum = 0;
            for (int i = 0; i < n; i++) {
                sum += matrix[i][j]; // Update sum for current column
                if (matrix[i][j] == 0) {
                    prefixSum[i][j] = 0; // No base for height if matrix[i][j]
is 0
                    sum = 0; // Reset sum for next row
                } else {
                    prefixSum[i][j] = sum; // Store the height for 1s
                }
```

```
            }
        }

        int maxArea = 0; // Variable to store maximum area

        // Traverse through each row and calculate the area
        for (int i = 0; i < n; i++) {
            int area = largestRectangleArea(prefixSum[i]); // Get the largest
area for current row
            maxArea = Math.max(area, maxArea); // Update maximum area
        }

        return maxArea; // Return the maximum area
    }

}

public class Main {
    public static void main(String[] args) {
        // Input matrix representing binary matrix
        int[][] matrix = {
            {1, 0, 1, 0, 0},
            {1, 0, 1, 1, 1},
            {1, 1, 1, 1, 1},
            {1, 0, 0, 1, 0}
        };

        // Create an instance of Solution class
        Solution sol = new Solution();

        // Call the function to find the largest rectangle area containing all
1s
        int ans = sol.maximalAreaOfSubMatrixOfAll1(matrix);

        // Print the result
        System.out.println("The largest rectangle area containing all 1s is: " +
ans);
    }
}
```

Complexity Analysis

Time Complexity: O(N*M), since filling the prefix sum matrix takes O(N*M) time, and every row (of length M) is treated as a histogram for which the largest histogram is found in linear(O(2*M)). Thus, time taking overall is O(N*M).

Space Complexity: O(N*M), since the prefix sum array takes up O(N*M) space, and finding the largest rectangle in each histogram (of length M) takes O(M) space due to stack.