rohithv63.medium.com

85 : Maximal Rectangle Leetcode Hard - Rohith Vazhathody - Medium

Rohith Vazhathody

5-7 minutes

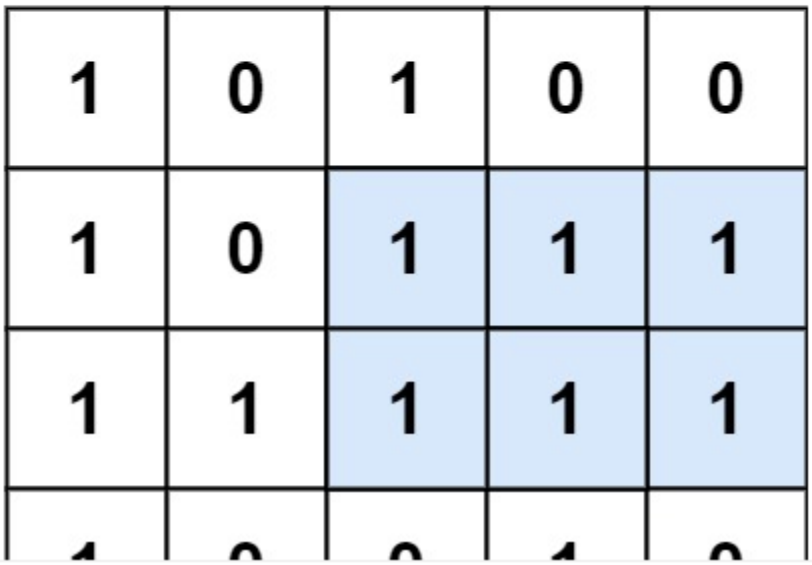
---

85. Maximal Rectangle

Hard 4081 86 Add to List Share

Given a `rows x cols` binary `matrix` filled with `0`'s and `1`'s, find the largest rectangle containing only `1`'s and return *its area*.

Example 1:



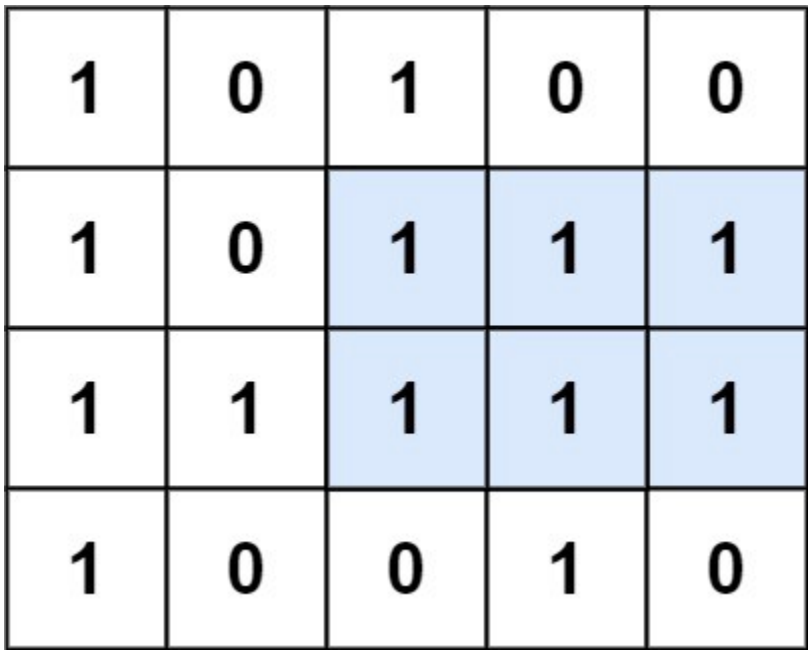
Leetcode 85 : Maximal Rectangle

Problem Statement :

Given a `rows x cols` binary `matrix` filled with `0`'s and `1`'s, find the largest rectangle containing only `1`'s and return *its area*.

Sample Test Cases :

Example 1:



Sample Visualisation for 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 maximal rectangle is shown in the above picture.

Example 2:

Input: matrix = []
Output: 0

Example 3:

Input: matrix = [["0"]]
Output: 0

Example 4:

Input: matrix = [["1"]]
Output: 1

Example 5:

Input: matrix = [["0","0"]]
Output: 0

Constraints :

- `rows == matrix.length`
- `cols == matrix[i].length`
- `0 <= row, cols <= 200`
- `matrix[i][j]` is `'0'` or `'1'`.

Approach :

After reading the statement, we may assume this is similar to another problem that is maximal square (Leetcode : 221 Maximal Square). But the way we need to do this problem is somewhat different from Maximal Square. Yes, we need to use dynamic programming but we also need to use the exact concept as that of another problem Largest Rectangle in Histogram.

Here we only need to consider the 1's and need to find the largest rectangle possible.

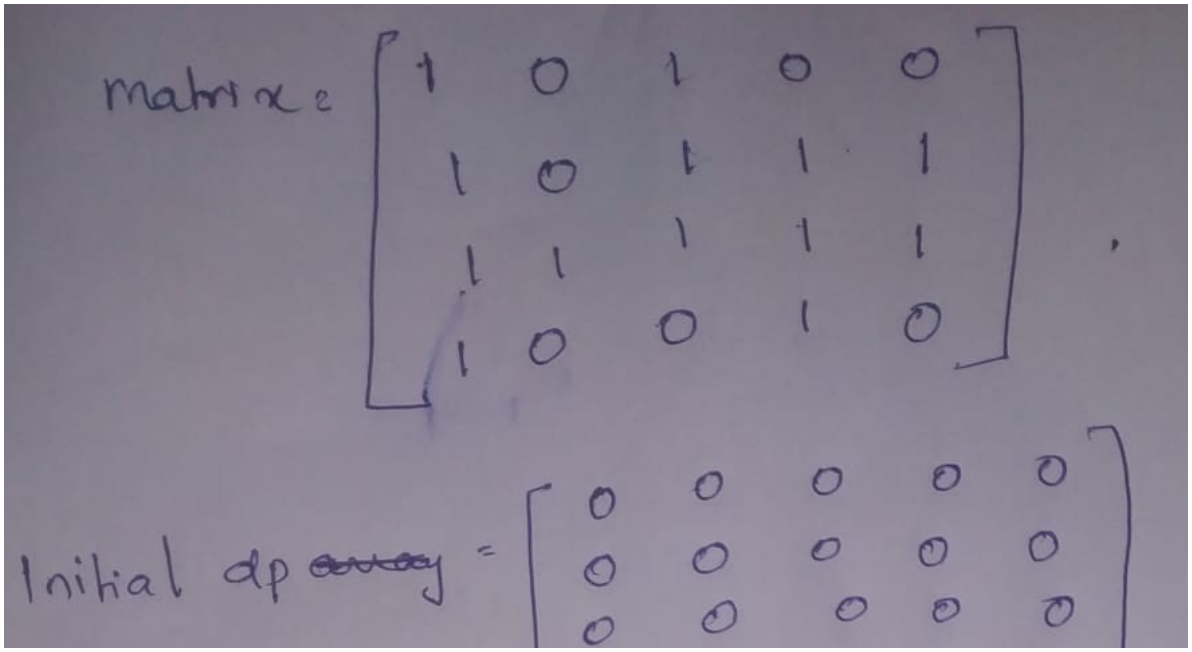
The first thing to do is create a dp matrix of size row and col which is exactly same as the input matrix.

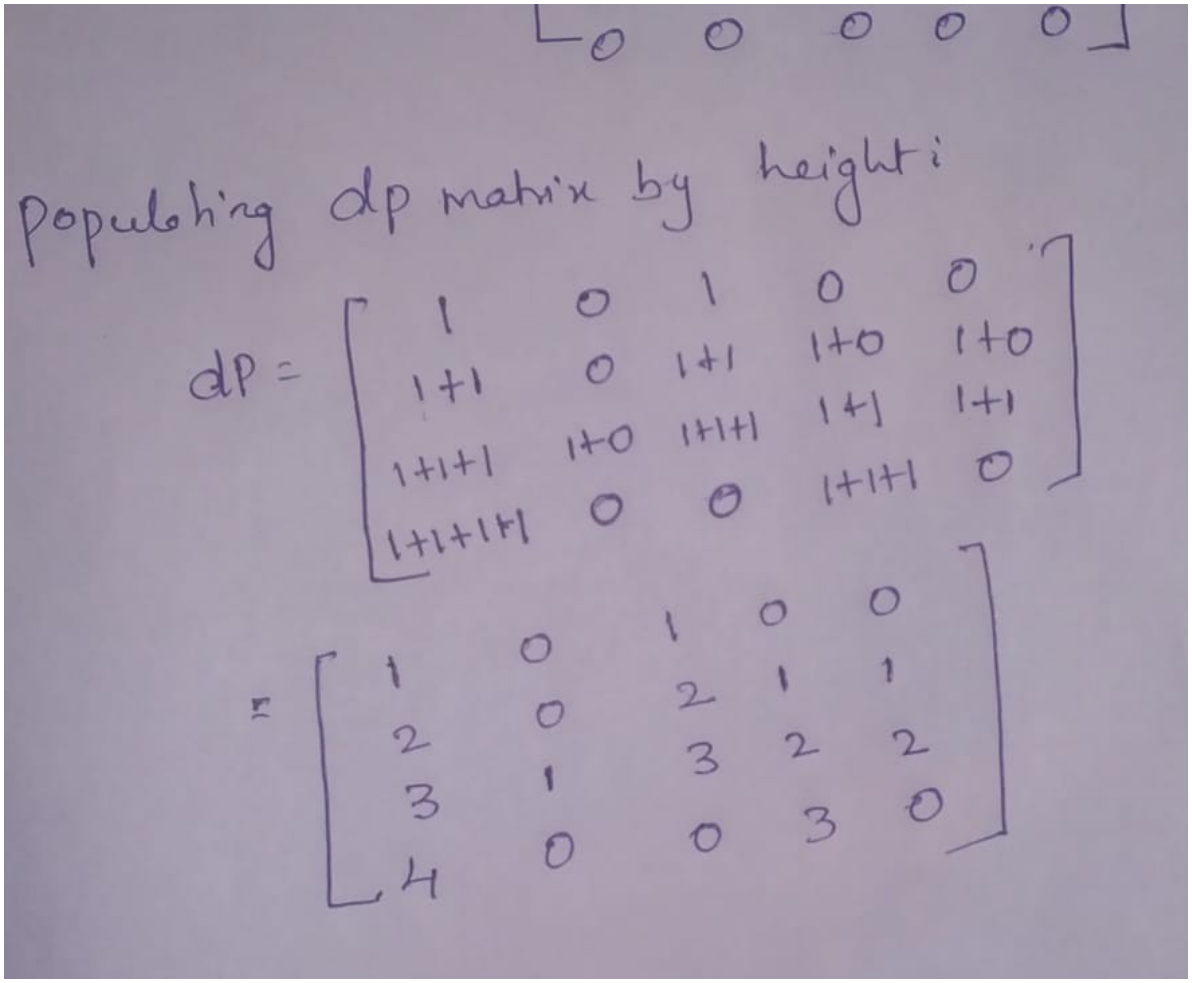
Now we need to populate the dp matrix with the heights. How we are populating with the height?

Say we are now currently on a cell say (i, j) which is 1. We look on the top of the cell ie. (i-1, j) and check whether it is one too. If it is 1, then we update the value of cell(i, j) as valueOf(i-1, j) + inputMatrixCellValueOf(i, j).

Same way, if we are at the cell (i, j) which is 0. Then no matter what is the value on the top, the value we are populating on the dp matrix will be 0 as the height cannot be updated if we have a value 0.

```
for (int i=0; i<row; i++) {
        for (int j=0; j<col; j++) {
            if (i == 0) {
                dp[i][j] = matrix[i][j] - '0';
            }
            else {
                if (matrix[i][j] == '1')
                    dp[i][j] = dp[i - 1][j] + matrix[i][j] - '0';
            }
        }
    }
```



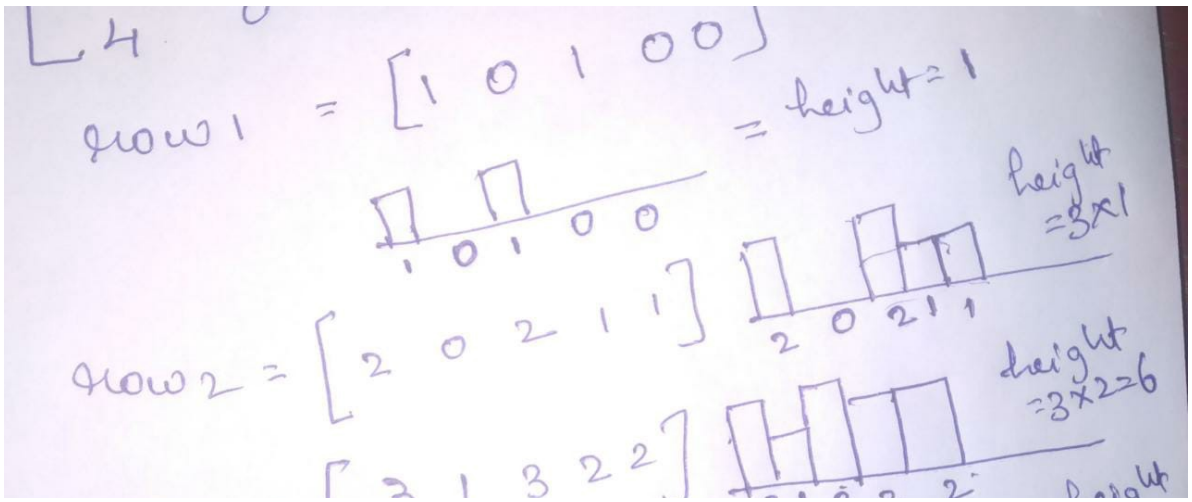


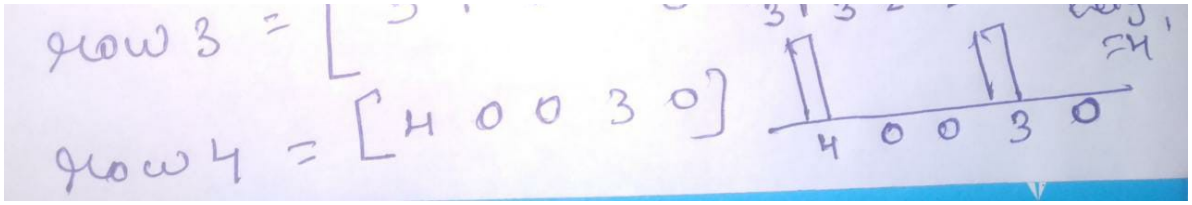
Populating dp matrix with height

Like this, the height is calculated and now we have to apply the concept of Largest rectangle in a histogram. We consider each of the row from the dp matrix and find the largest rectangle from each individual row. After processing each row, then we have the maximal rectangle.

```
// same code for largest rectangle in a histogram
    public int largestRectangleHistogram(int [] h) {
        if (h == null || h.length == 0)
            return 0;
        int length = h.length;
        int [] left = new int [length];
        int [] right = new int [length];
```

```
        int max = 0;
        left[0] = -1;
        right[length - 1] = length;
        for (int i=1; i<length; i++) {
            int currentIndex = i - 1;
            while (currentIndex >= 0 && h[currentIndex] >= h[i]) {
                currentIndex = left[currentIndex];
            }
            left[i] = currentIndex;
        }
        for (int i=length - 2; i>=0; i--) {
            int currentIndex = i + 1;
            while (currentIndex < length && h[currentIndex] >= h[i]) {
                currentIndex = right[currentIndex];
            }
            right[i] = currentIndex;
        }
        for (int i=0; i<length; i++) {
            max = Math.max(max, h[i] * (right[i] - left[i] - 1));
        }
        return max;
    }
```





Finding the maximum height from each row.

Thus we find the maximum height from each row and the maximum of those is the answer we are looking.

Complete Code :

```
class Solution {
    public int maximalRectangle(char[][] matrix) {
        int row = matrix.length;
        if (row == 0)
            return 0;
        int col = matrix[0].length;
        int max = 0;
        int [][] dp = new int [row][col];
        // calculate the height
        for (int i=0; i<row; i++) {
            for (int j=0; j<col; j++) {
                if (i == 0) {
                    dp[i][j] = matrix[i][j] - '0';
                }
                else {
                    if (matrix[i][j] == '1')
                        dp[i][j] = dp[i - 1][j] + matrix[i][j] - '0';
                }
            }
        }
```

```
        for (int [] eachRow : dp) {
            max = Math.max(max,
largestRectangleHistogram(eachRow));
        }
        return max;
    }
        // same code for largest rectangle in a histogram
    public int largestRectangleHistogram(int [] h) {
        if (h == null || h.length == 0)
            return 0;
        int length = h.length;
        int [] left = new int [length];
        int [] right = new int [length];
        int max = 0;
        left[0] = -1;
        right[length - 1] = length;
        for (int i=1; i<length; i++) {
            int currentIndex = i - 1;
            while (currentIndex >= 0 && h[currentIndex] >= h[i]) {
                currentIndex = left[currentIndex];
            }
            left[i] = currentIndex;
        }
        for (int i=length - 2; i>=0; i--) {
            int currentIndex = i + 1;
            while (currentIndex < length && h[currentIndex] >= h[i]) {
                currentIndex = right[currentIndex];
            }
            right[i] = currentIndex;
        }
```

```
        for (int i=0; i<length; i++) {
            max = Math.max(max, h[i] * (right[i] - left[i] - 1));
        }
        return max;
    }
}
```

Time Complexity and Space Complexity:

Time : O(row * col)

Space : O(row * col) new dp array

Github Repo :