**Maximal Rectangle**

**Hard**

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:**

Calendar

Description automatically generated

**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 = [["0"]]

**Output:** 0

**Example 3:**

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

**Output:** 1

**Constraints:**

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

class Solution {

public:

    int LargestArea(vector<int> arr) {

        int n=arr.size();

        vector<int> pre(n), suff(n);

        stack<int> st;

        for (int i=n-1; i>=0; i--) {

            if (st.empty()) suff[i]=n;

            else {

                while (!st.empty() and arr[st.top()]>=arr[i]) {

                    st.pop();

                }

                if (st.empty()) suff[i]=n;

                else suff[i]=st.top();

            }

            st.push(i);

        }

        while (!st.empty()) st.pop();

        for (int i=0; i<n; i++) {

            if (st.empty()) pre[i]=-1;

            else {

                while (!st.empty() and arr[st.top()]>=arr[i]) {

                    st.pop();

                }

                if (st.empty()) pre[i]=-1;

                else pre[i]=st.top();

            }

            st.push(i);

        }

        int ans=0;

        for (int i=0; i<n; i++) {

            int val=(suff[i]-pre[i]-1)\*arr[i];

            ans=max(ans, val);

        }

        return ans;

    }

    int maximalRectangle(vector<vector<char>>& matrix) {

        int m=matrix.size(), n=matrix[0].size();

        vector<int> arr(n, 0);

        int area=0;

        for (int r=0; r<m; r++) {

            for (int c=0; c<n; c++) {

                if (matrix[r][c]=='0') arr[c]=0;

                else arr[c]+=1;

            }

            area=max(area, LargestArea(arr));

        }

        return area;

    }

};