Largest Black SubRectangle

- Một hình chữ nhật kích thước *n* x *m* được chia thành các ô vuông con 1 x 1 với 2 màu đen hoặc trắng. Hình chữ nhật được biểu diễn bởi ma trận *A*(*n* x *m*) trong đó *A*(*i, j*) = 1 có nghĩa ô hàng *i*, cột *j* là ô đen và *A*(*i, j*) = 0 có nghĩa ô vuông hàng *i* cột *j* là ô trắng.
- Hãy xác định hình chữ nhật con của bảng đã cho bao gồm toàn ô đen và có diện tích lớn nhất.
- **Dữ liệu**
  - · Dòng 1: chứa số nguyên dương *n* và *m* (1 <= *n, m* <= 1000)
  - · Dòng i+1 (i = 1,…, *n*): chứa hàng thứ i của ma trận *A*
- **Kết quả**
  - · Ghi ra diện tích của hình chữ nhật lớn nhất tìm được

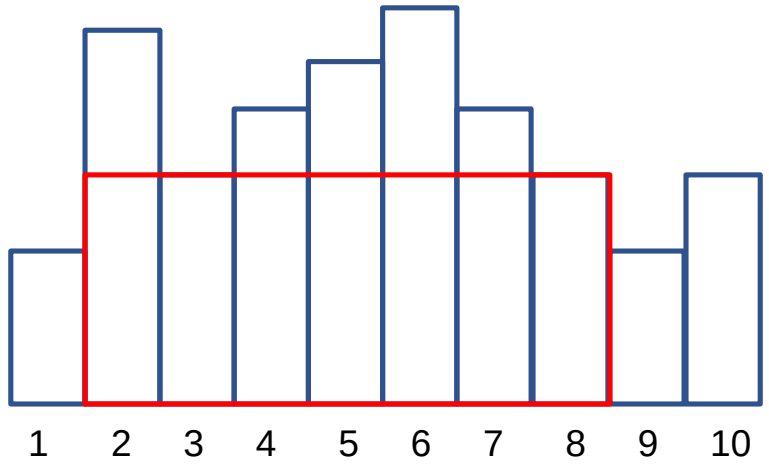
Largest Black SubRectangle

- Example

stdin	stdout
4 4 0 1 1 1 1 1 1 0 1 1 0 0 1 1 1 0	6

Hint

- Solve sub-problem: Column *i* have height *h*[*i*] (*i* = 1, 2, …, *n*). Find the way to cut out the largest-area rectangle from the given configure
- For each index *i* :
  - Move left and move right as far as possible to cut out the largest rectangle having height *h*[*i*]
  - *R*[*i*]: the index *j* such that *h*[*i*] > *h*[*j*] and *j* (*i* < *j*) is the nearest to *i*
  - *L*[*i*]: the index *j* such that *h*[*i*] > *h*[*j*] and *j* (*i* > *j*) is the nearest to *i*
  - The largest are built from column *i* is: (*R*[*i*] – *L*[*i*] - 1)***h*[*i*]
- Use a Stack S for storing indices

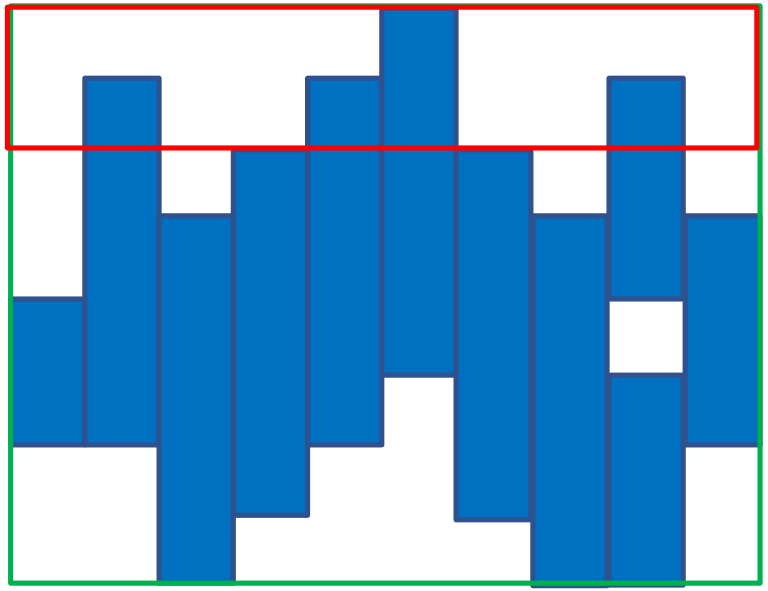



```
h[0] = -1, h[n+1] = -1
S ← empty stack
for i = 1 → n+1 do{
    while S not empty and h[i] < h[S.top] do{
        R[S.top] = i; S.pop();
    }
    S.push(i);
}
```

```
h[0] = -1, h[n+1] = -1
S ← empty stack
for i = n → 0 do{
    while S not empty and h[i] < h[S.top] do{
        L[S.top] = i; S.pop();
    }
    S.push(i);
}
```

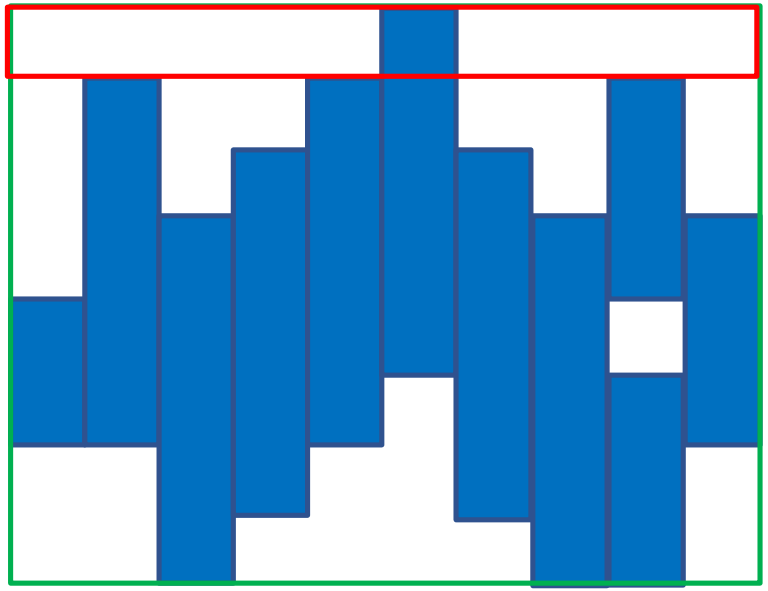
Hint

- Let a horizontal line moving down



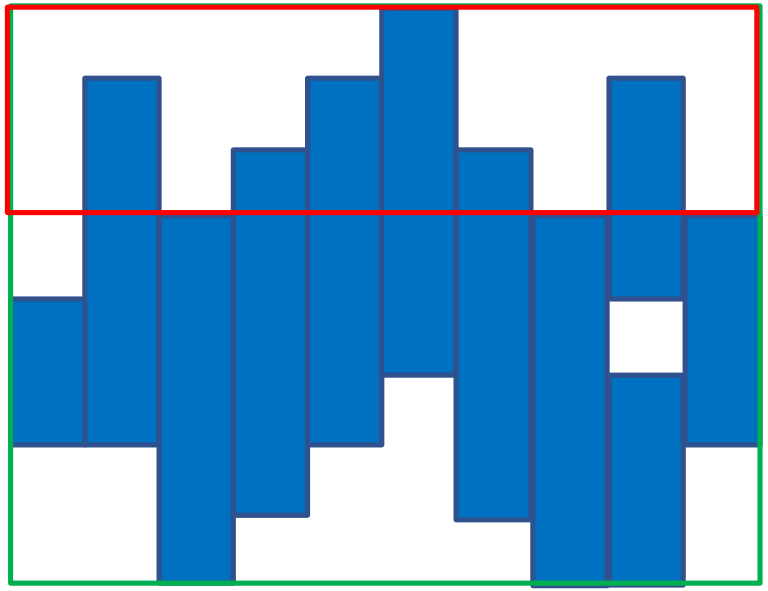
Hint

- Let a horizontal line moving down



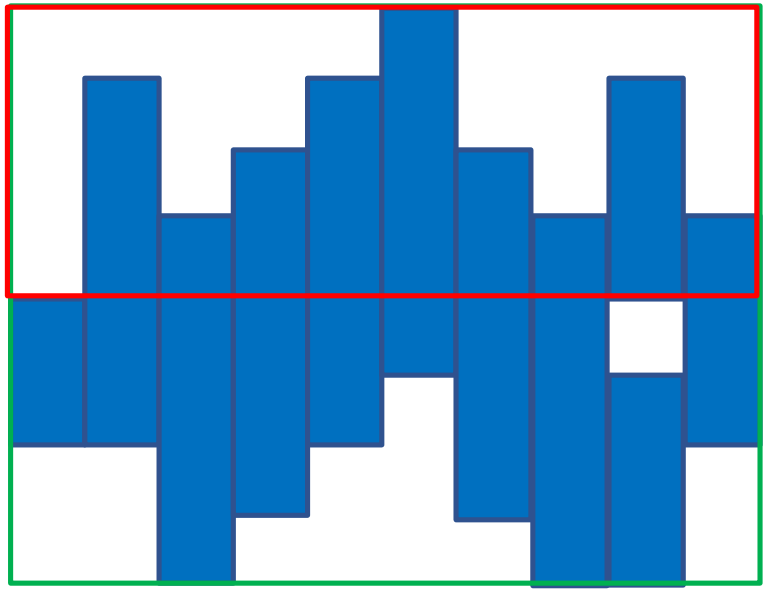
Hint

- Let a horizontal line moving down



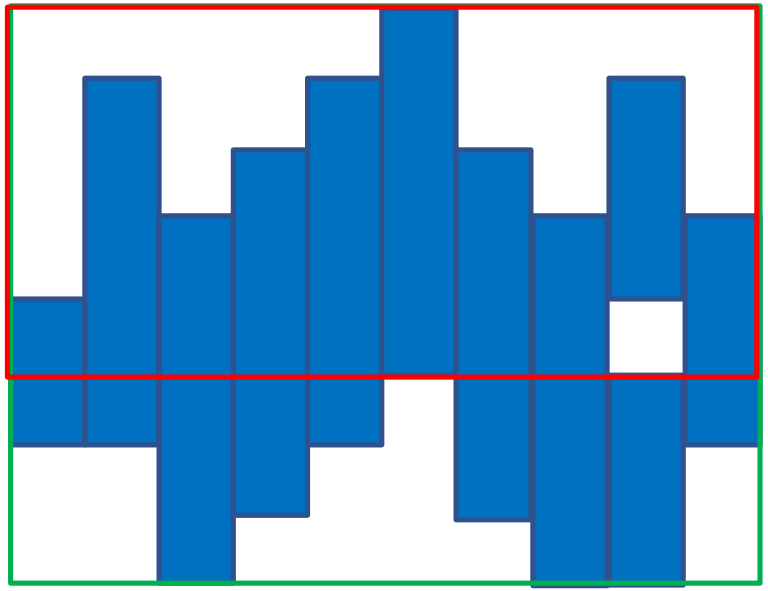
Hint

- Let a horizontal line moving down



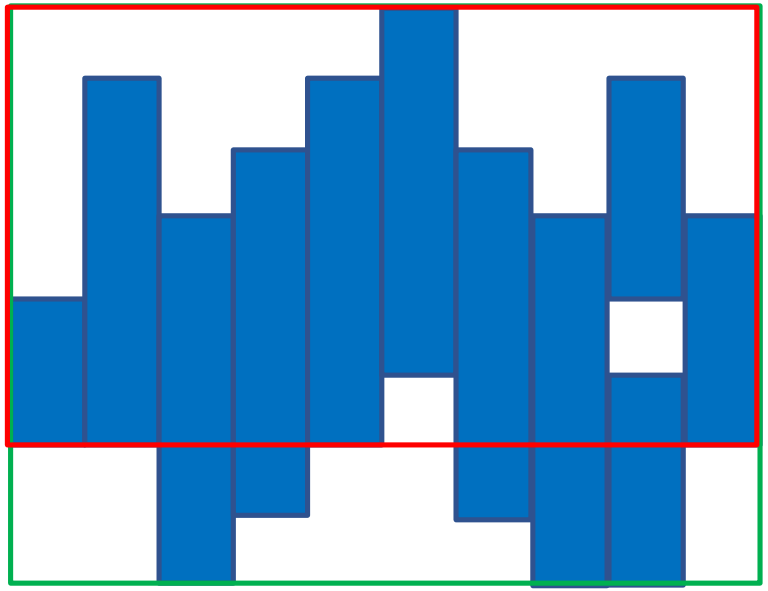
Hint

- Let a horizontal line moving down



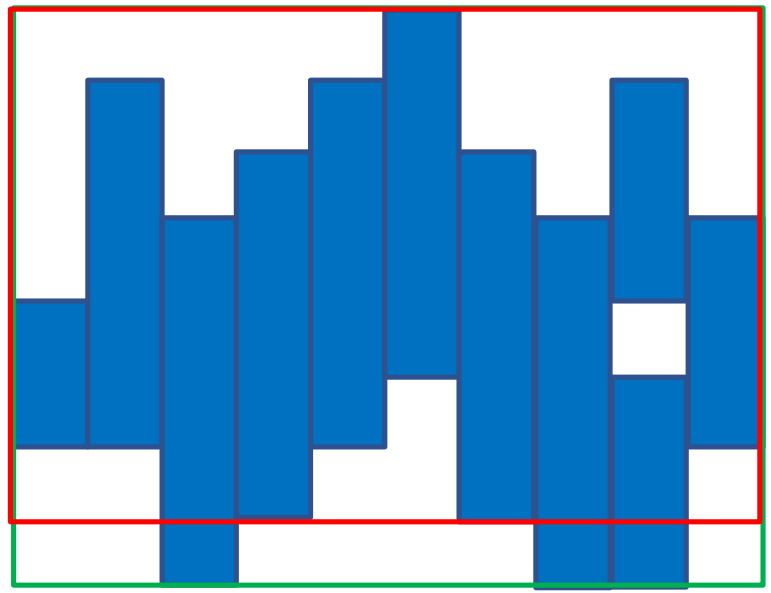
Hint

- Let a horizontal line moving down



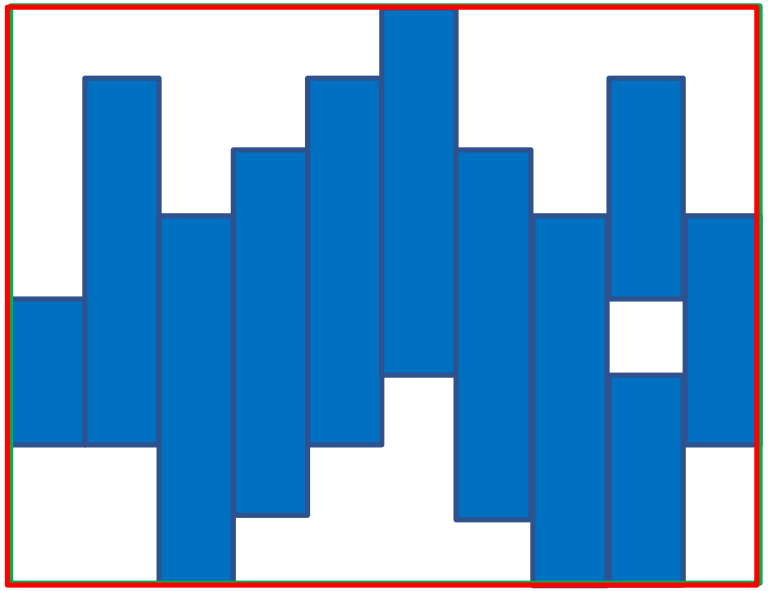
Hint

- Let a horizontal line moving down



Hint

- Let a horizontal line moving down



Implementation

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e3+1;
int a[N][N];
int n,m;
int ans;
long long h[N];
stack<long long> S;
vector<long long> V;
long long L[N],R[N] ;
void input(){
    cin >> n >> m;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++) cin >> a[i][j];
    }
}
```

Implementation

```
long long compute(){
            h[0] = -1; h[m+1] = -1;  V.clear();
            for(int i = 1; i <= m+1; i++){
                    while(!V.empty() && h[i] < h[V[V.size()-1]]){      R[V[V.size()-1]] = i; V.pop_back();     }
                    V.push_back(i);
            }
            for(int i = m; i >= 0; i--){
                while(!V.empty() && h[i] < h[V[V.size()-1]]){   L[V[V.size()-1]] = i; V.pop_back();   }
                V.push_back(i);
            }
            unsigned long long ans = 0;
            for(int i = 1; i <= m; i++){
                unsigned long long c = (R[i] - L[i] - 1)*h[i];     ans = ans < c ? c : ans;
            }
            return ans;
}
```

Implementation

```
void solve(){
    long long ans = 0;
    for(int i = 1; i <= m; i++) h[i] = 0;
    for(int i =1 ; i <= n; i++){
        for(int j = 1; j <= m; j++){  if(a[i][j] == 0)   h[j] = 0; else h[j] += 1;  }
        long long t = compute();
        if(t > ans) ans = t;
    }
    cout << ans;
}
int main(){
    input();
    solve();
    return 0;
}
```