Problem 302: Smallest Rectangle Enclosing Black Pixels

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an

m x n

binary matrix

image

where

0

represents a white pixel and

1

represents a black pixel.

The black pixels are connected (i.e., there is only one black region). Pixels are connected horizontally and vertically.

Given two integers

x

and

y

that represents the location of one of the black pixels, return

the area of the smallest (axis-aligned) rectangle that encloses all black pixels

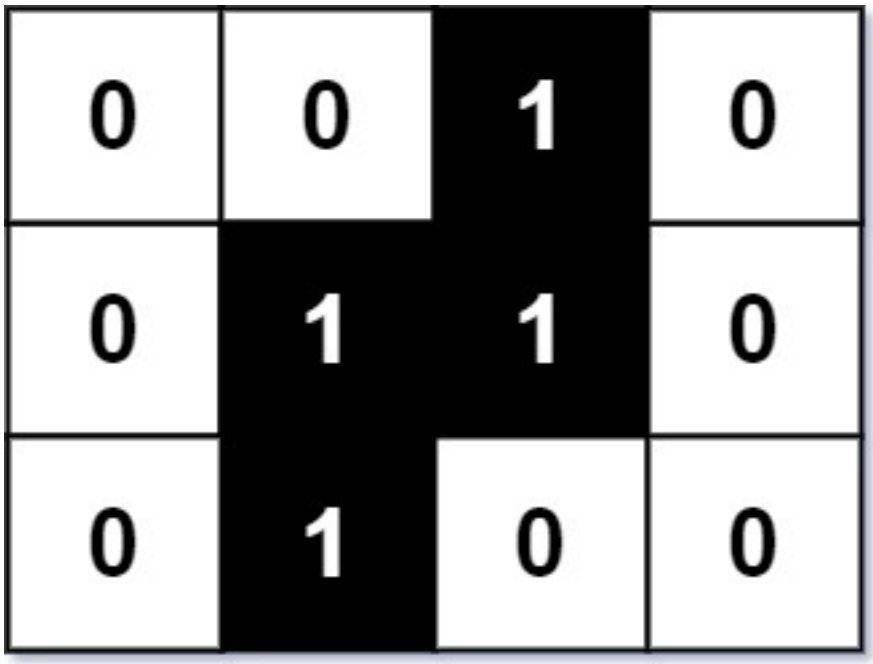
.

You must write an algorithm with less than

O(mn)

runtime complexity

Example 1:



Input:

image = [["0","0","1","0"],["0","1","1","0"],["0","1","0","0"]], x = 0, y = 2

Output:

6

Example 2:

Input:

image = [["1"]], x = 0, y = 0

Output:

1

Constraints:

m == image.length

n == image[i].length

1 <= m, n <= 100

image[i][j]

is either

'0'

or

'1'

.

0 <= x < m

0 <= y < n

image[x][y] == '1'.

The black pixels in the

image

only form

one component

.

Code Snippets

C++:

```
class Solution {
public:
int minArea(vector<vector<char>>& image, int x, int y) {

}
};
```

Java:

```
class Solution {
public int minArea(char[][] image, int x, int y) {

}
}
```

Python3:

```
class Solution:
def minArea(self, image: List[List[str]], x: int, y: int) -> int:
```

Python:

```
class Solution(object):
def minArea(self, image, x, y):
"""
:type image: List[List[str]]
:type x: int
:type y: int
```

```
:rtype: int
"""
```

JavaScript:

```
/**
* @param {character[][]} image
* @param {number} x
* @param {number} y
* @return {number}
*/
var minArea = function(image, x, y) {

};
```

TypeScript:

```
function minArea(image: string[][], x: number, y: number): number {

};
```

C#:

```
public class Solution {
public int MinArea(char[][] image, int x, int y) {

}
}
```

C:

```
int minArea(char** image, int imageSize, int* imageColSize, int x, int y) {

}
```

Go:

```
func minArea(image [][]byte, x int, y int) int {

}
```

Kotlin:

```
class Solution {
fun minArea(image: Array<CharArray>, x: Int, y: Int): Int {

}
}
```

Swift:

```
class Solution {
func minArea(_ image: [[Character]], _ x: Int, _ y: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_area(image: Vec<Vec<char>>, x: i32, y: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Character[][]} image
# @param {Integer} x
# @param {Integer} y
# @return {Integer}
def min_area(image, x, y)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $image
* @param Integer $x
* @param Integer $y
* @return Integer
*/
function minArea($image, $x, $y) {
```

```
}
}
```

Dart:

```
class Solution {
int minArea(List<List<String>> image, int x, int y) {

}
}
```

Scala:

```
object Solution {
def minArea(image: Array[Array[Char]], x: Int, y: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_area(image :: [[char]], x :: integer, y :: integer) :: integer
def min_area(image, x, y) do

end
end
```

Erlang:

```
-spec min_area(Image :: [[char()]], X :: integer(), Y :: integer()) ->
integer().
min_area(Image, X, Y) ->
.
```

Racket:

```
(define/contract (min-area image x y)
(-> (listof (listof char?)) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Rectangle Enclosing Black Pixels
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int minArea(vector<vector<char>>& image, int x, int y) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Rectangle Enclosing Black Pixels
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int minArea(char[][] image, int x, int y) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Rectangle Enclosing Black Pixels
Difficulty: Hard
Tags: array, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def minArea(self, image: List[List[str]], x: int, y: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minArea(self, image, x, y):
"""
:type image: List[List[str]]
:type x: int
:type y: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Smallest Rectangle Enclosing Black Pixels
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {character[][]} image
* @param {number} x
* @param {number} y
```

```
* @return {number}
*/
var minArea = function(image, x, y) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Rectangle Enclosing Black Pixels
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function minArea(image: string[][], x: number, y: number): number {

};
```

C# Solution:

```
/*
* Problem: Smallest Rectangle Enclosing Black Pixels
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MinArea(char[][] image, int x, int y) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Rectangle Enclosing Black Pixels
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int minArea(char** image, int imageSize, int* imageColSize, int x, int y) {

}
```

Go Solution:

```
// Problem: Smallest Rectangle Enclosing Black Pixels
// Difficulty: Hard
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func minArea(image [][]byte, x int, y int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minArea(image: Array<CharArray>, x: Int, y: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minArea(_ image: [[Character]], _ x: Int, _ y: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Smallest Rectangle Enclosing Black Pixels
// Difficulty: Hard
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn min_area(image: Vec<Vec<char>>, x: i32, y: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Character[][]} image
# @param {Integer} x
# @param {Integer} y
# @return {Integer}
def min_area(image, x, y)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $image
* @param Integer $x
* @param Integer $y
* @return Integer
*/
function minArea($image, $x, $y) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int minArea(List<List<String>> image, int x, int y) {

}
}
```

Scala Solution:

```
object Solution {
def minArea(image: Array[Array[Char]], x: Int, y: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_area(image :: [[char]], x :: integer, y :: integer) :: integer
def min_area(image, x, y) do

end
end
```

Erlang Solution:

```
-spec min_area(Image :: [[char()]], X :: integer(), Y :: integer()) ->
integer().
min_area(Image, X, Y) ->
.
```

Racket Solution:

```
(define/contract (min-area image x y)
(-> (listof (listof char?)) exact-integer? exact-integer? exact-integer?)
)
```