Smallest Rectangle Enclosing Black Pixels

An image is represented by a binary matrix with `0` as a white pixel and `1` as a black pixel. The black pixels are connected, i.e., there is only one black region. Pixels are connected horizontally and vertically. Given the location `(x, y)` of one of the black pixels, return the area of the smallest (axis-aligned) rectangle that encloses all black pixels.

For example, given the following image:

```
[
  "0010",
  "0110",
  "0100"
]
```

and `x = 0`, `y = 2`,
Return `6`.

Solution 1

Suppose we have a 2D array

```
"000000111000000"
"000000101000000"
"000000101100000"
"000001100100000"
```

Imagine we project the 2D array to the bottom axis with the rule "if a column has any black pixel it's projection is black otherwise white". The projected 1D array is

```
"000001111100000"
```

> **Theorem**
>
> If there are only one black pixel region, then in a projected 1D array all the black pixels are connected.
>
> **Proof by contradiction**
>
> Assume to the contrary that there are disconnected black pixels at `i` and `j` where `i < j` in the 1D projection array. Thus there exists one column `k`, `k in (i, j)` and and the column `k` in the 2D array has no black pixel. Therefore in the 2D array there exists at least 2 black pixel regions separated by column `k` which contradicting the condition of "only one black pixel region".
>
> Therefore we conclude that all the black pixels in the 1D projection array is connected.

This means we can do a binary search in each half to find the boundaries, if we know one black pixel's position. And we do know that.

To find the left boundary, do the binary search in the `[0, y)` range and find the first column vector who has any black pixel.

To determine if a column vector has a black pixel is `O(m)` so the search in total is `O(m log n)`

We can do the same for the other boundaries. The area is then calculated by the boundaries. Thus the algorithm runs in `O(m log n + n log m)`

Java

```java
private char[][] image;
public int minArea(char[][] iImage, int x, int y) {
    image = iImage;
    int m = image.length, n = image[0].length;
    int left = searchColumns(0, y, 0, m, true);
    int right = searchColumns(y + 1, n, 0, m, false);
    int top = searchRows(0, x, left, right, true);
    int bottom = searchRows(x + 1, m, left, right, false);
    return (right - left) * (bottom - top);
}
private int searchColumns(int i, int j, int top, int bottom, boolean opt) {
    while (i != j) {
        int k = top, mid = (i + j) / 2;
        while (k < bottom && image[k][mid] == '0') ++k;
        if (k < bottom == opt)
            j = mid;
        else
            i = mid + 1;
    }
    return i;
}
private int searchRows(int i, int j, int left, int right, boolean opt) {
    while (i != j) {
        int k = left, mid = (i + j) / 2;
        while (k < right && image[mid][k] == '0') ++k;
        if (k < right == opt)
            j = mid;
        else
            i = mid + 1;
    }
    return i;
}
//  Runtime: 1 ms
```

C++

```cpp
vector<vector<char>> *image;
int minArea(vector<vector<char>> &iImage, int x, int y) {
    image = &iImage;
    int m = int(image->size()), n = int((*image)[0].size());
    int top = searchRows(0, x, 0, n, true);
    int bottom = searchRows(x + 1, m, 0, n, false);
    int left = searchColumns(0, y, top, bottom, true);
    int right = searchColumns(y + 1, n, top, bottom, false);
    return (right - left) * (bottom - top);
}
int searchRows(int i, int j, int low, int high, bool opt) {
    while (i != j) {
        int k = low, mid = (i + j) / 2;
        while (k < high && (*image)[mid][k] == '0') ++k;
        if (k < high == opt)
            j = mid;
        else
            i = mid + 1;
    }
    return i;
}
int searchColumns(int i, int j, int low, int high, bool opt) {
    while (i != j) {
        int k = low, mid = (i + j) / 2;
        while (k < high && (*image)[k][mid] == '0') ++k;
        if (k < high == opt)
            j = mid;
        else
            i = mid + 1;
    }
    return i;
}
// Runtime: 20 ms
```

Python

```
def minArea(self, image, x, y):
    top = self.searchRows(image, 0, x, True)
    bottom = self.searchRows(image, x + 1, len(image), False)
    left = self.searchColumns(image, 0, y, top, bottom, True)
    right = self.searchColumns(image, y + 1, len(image[0]), top, bottom, False)
    return (right - left) * (bottom - top)

def searchRows(self, image, i, j, opt):
    while i != j:
        m = (i + j) / 2
        if ('1' in image[m]) == opt:
            j = m
        else:
            i = m + 1
    return i

def searchColumns(self, image, i, j, top, bottom, opt):
    while i != j:
        m = (i + j) / 2
        if any(image[k][m] == '1' for k in xrange(top, bottom)) == opt:
            j = m
        else:
            i = m + 1
    return i
# Runtime: 56 ms
```

Java (DRY)

```
private char[][] image;
public int minArea(char[][] iImage, int x, int y) {
    image = iImage;
    int m = image.length, n = image[0].length;
    int top = search(0, x, 0, n, true, true);
    int bottom = search(x + 1, m, 0, n, false, true);
    int left = search(0, y, top, bottom, true, false);
    int right = search(y + 1, n, top, bottom, false, false);
    return (right - left) * (bottom - top);
}
private boolean isWhite(int mid, int k, boolean isRow) {
    return ((isRow) ? image[mid][k] : image[k][mid]) == '0';
}
private int search(int i, int j, int low, int high, boolean opt, boolean isRow) {
    while (i != j) {
        int k = low, mid = (i + j) / 2;
        while (k < high && isWhite(mid, k, isRow)) ++k;
        if (k < high == opt)
            j = mid;
        else
            i = mid + 1;
    }
    return i;
}
//  Runtime: 2 ms
```

C++ (DRY)

```
vector<vector<char>> *image;
int minArea(vector<vector<char>> &iImage, int x, int y) {
    image = &iImage;
    int m = int(image->size()), n = int((*image)[0].size());
    int top = search(0, x, 0, n, true, true);
    int bottom = search(x + 1, m, 0, n, false, true);
    int left = search(0, y, top, bottom, true, false);
    int right = search(y + 1, n, top, bottom, false, false);
    return (right - left) * (bottom - top);
}
bool isWhite(int mid, int k, bool isRow) {
    return ((isRow) ? (*image)[mid][k]:(*image)[k][mid]) == '0';
}
int search(int i, int j, int low, int high, bool opt, bool isRow) {
    while (i != j) {
        int k = low, mid = (i + j) / 2;
        while (k < high && isWhite(mid, k, isRow)) ++k;
        if (k < high == opt)
            j = mid;
        else
            i = mid + 1;
    }
    return i;
}
// Runtime: 24 ms
```

Python (DRY, from Stefan's cool solution)

```
def minArea(self, image, x, y):
    top = self.search(0, x, lambda mid: '1' in image[mid])
    bottom = self.search(x + 1, len(image), lambda mid: '1' not in image[mid])
    left = self.search(0, y, lambda mid: any(image[k][mid] == '1' for k in xrange
(top, bottom)))
    right = self.search(y + 1, len(image[0]), lambda mid: all(image[k][mid] == '0
' for k in xrange(top, bottom)))
    return (right - left) * (bottom - top)

def search(self, i, j, check):
    while i != j:
        mid = (i + j) / 2
        if check(mid):
            j = mid
        else:
            i = mid + 1
    return i
# Runtime: 56 ms
```

written by dietpepsi original link here

Solution 2

```
def minArea(self, image, x, y):
    def first(lo, hi, check):
        while lo < hi:
            mid = (lo + hi) / 2
            if check(mid):
                hi = mid
            else:
                lo = mid + 1
        return lo
    top    = first(0, x,             lambda x: '1' in image[x])
    bottom = first(x, len(image),    lambda x: '1' not in image[x])
    left   = first(0, y,             lambda y: any(row[y] == '1' for row in image
))
    right  = first(y, len(image[0]), lambda y: all(row[y] == '0' for row in image
))
    return (bottom - top) * (right - left)
```

written by StefanPochmann original link here

Solution 3

DFS or BFS is the intuitive solution for this problem while the problem is with a tag "binary search". So can anyone provide a binary search answer. DFS complexity is O(m * n) and if binary search it would be O(n * lgm + m * lgn)

```
public class Solution {
    private int minX = Integer.MAX_VALUE, minY = Integer.MAX_VALUE, maxX = 0, max
Y = 0;
    public int minArea(char[][] image, int x, int y) {
        if(image == null || image.length == 0 || image[0].length == 0) return 0;
        dfs(image, x, y);
        return(maxX - minX + 1) * (maxY - minY + 1);
    }
    private void dfs(char[][] image, int x, int y){
        int m = image.length, n = image[0].length;
        if(x < 0 || y < 0 || x >= m || y >= n || image[x][y] == '0') return;
        image[x][y] = '0';
        minX = Math.min(minX, x);
        maxX = Math.max(maxX, x);
        minY = Math.min(minY, y);
        maxY = Math.max(maxY, y);
        dfs(image, x + 1, y);
        dfs(image, x - 1, y);
        dfs(image, x, y - 1);
        dfs(image, x, y + 1);
    }

}
```

written by czonzhu original link here