

Question:

You are given a two-dimensional array (matrix) of potentially unequal height and width containing only 0s and 1s. Each 0 represents land, and each 1 represents part of a river. A river consists of any number of 1s that are either horizontally or vertically adjacent (but not diagonally adjacent). The number of adjacent 1s forming a river determine its size. Write a function that returns an array of the sizes of all rivers represented in the input matrix. Note that these sizes do not need to be in any particular order.

Sample input:

Input:	Your Solution	Our Solution
--------	---------------	--------------

Run Code

```
vector<int> riverSizes(vector<vector<int>>> matrix) {
    vector<int> sizes = {};
    vector<vector<int>>> visited(matrix.size(),
                                vector<int>(matrix[0].size(), false));
    for (int i = 0; i < matrix.size(); i++) {
        for (int j = 0; j < matrix[i].size(); j++) {
            if (visited[i][j]) {
                continue;
            }
            traverseNode(i, j, matrix, &visited, &sizes);
        }
    }
    return sizes;
}
```

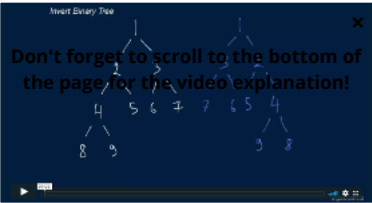
Output: Custom Output Raw Output

Run your code when you feel ready.

Help: Hide Show

There are many ways to store the edges of a matrix, which consists of horizontal and vertical edges adjacent to it in the input matrix, you need to store the horizontal edges of a group of neighboring 1s as one row in the matrix. Treating the matrix as a graph, where each element in the matrix is a node in the graph with up to 4 neighboring nodes above, below, to the left, and to the right, and traversing it using recursive graph traversal algorithms like Depth First Search or Breadth First Search.

Tests: [Our Tests](#) [Your Tests](#) [Hide](#) [Show](#)



Video Explanation

[Go to Code Walkthrough](#)

```

1  def traverseNode(i, j, matrix, visited, sizes):
2      return sizes
3
4  def traverseNode(i, j, matrix, visited, sizes):
5      currentSize = 0
6      nodesToExplore = [(i, j)]
7      while len(nodesToExplore) > 0:
8          currentNode = nodesToExplore.pop()
9          i = currentNode[0]
10         j = currentNode[1]
11         if visited[i][j]:
12             continue
13         visited[i][j] = True
14         if matrix[i][j] == 0:
15             continue
16         currentSize += 1
17         unvisitedNeighbors = getUnvisitedNeighbors(i, j, matrix, visited)
18         for neighbor in unvisitedNeighbors:
19             nodesToExplore.append(neighbor)
20
21     return currentSize

```

[Questions List \(/questions\)](#)