Programación Competitiva

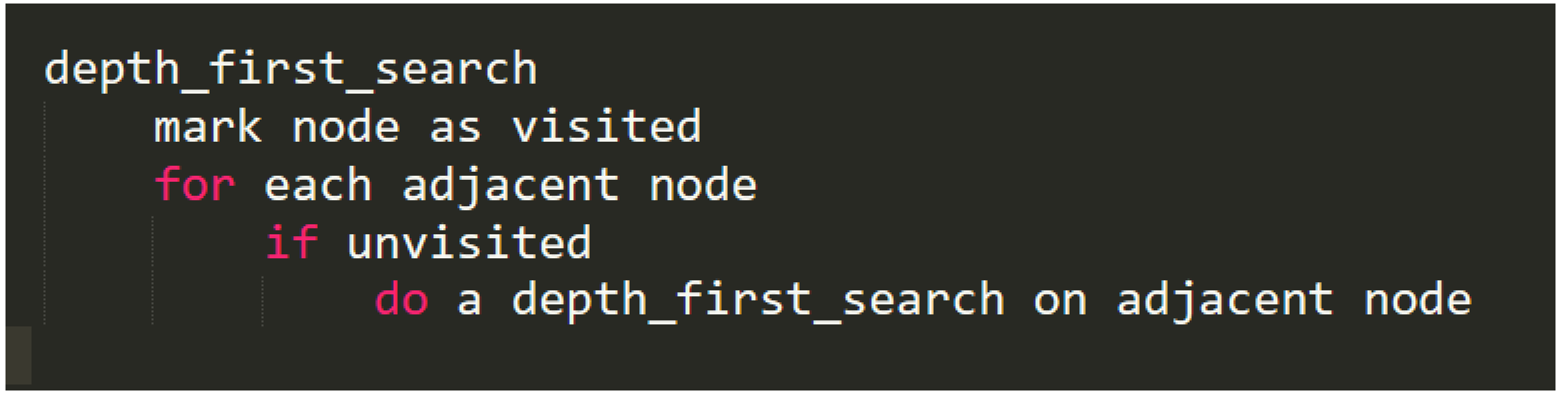
Alex Josué Flórez Farfán

Magister en Ciencias de la Computación

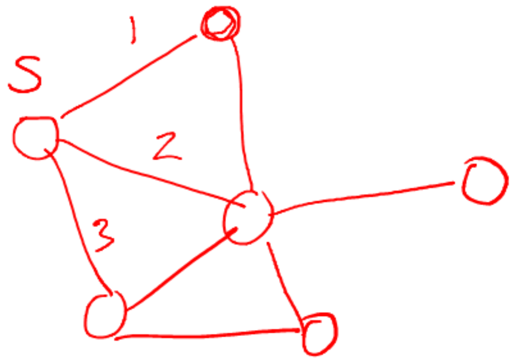
Ciencia de la Computación – UNSA
Segundo semestre 2021

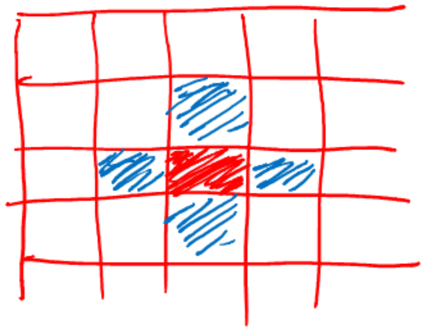
Depth First Search

Depth First Search

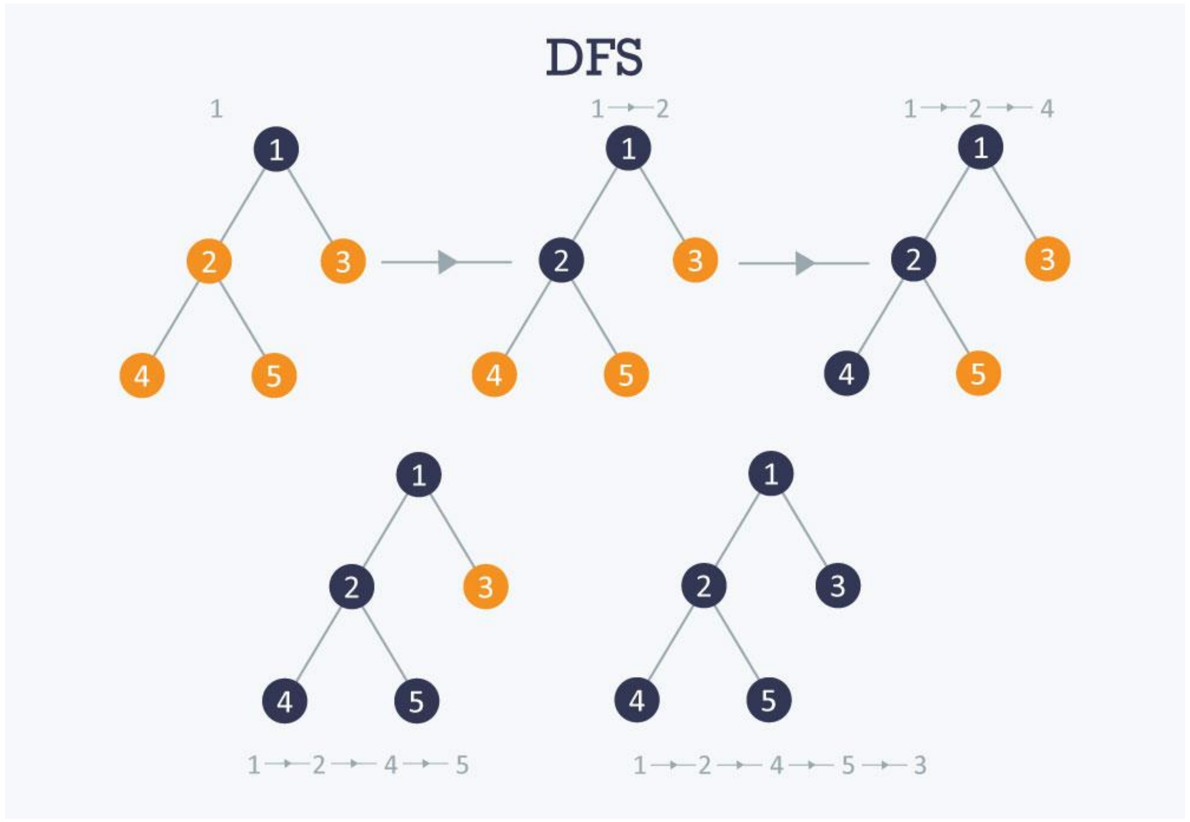


```
depth_first_search
    mark node as visited
    for each adjacent node
        if unvisited
            do a depth_first_search on adjacent node
```

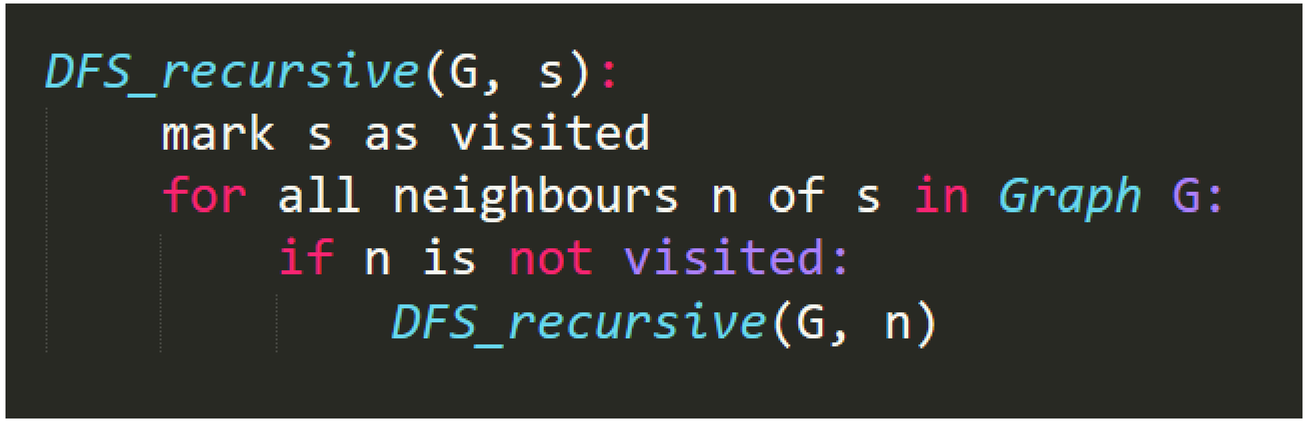




Depth First Search

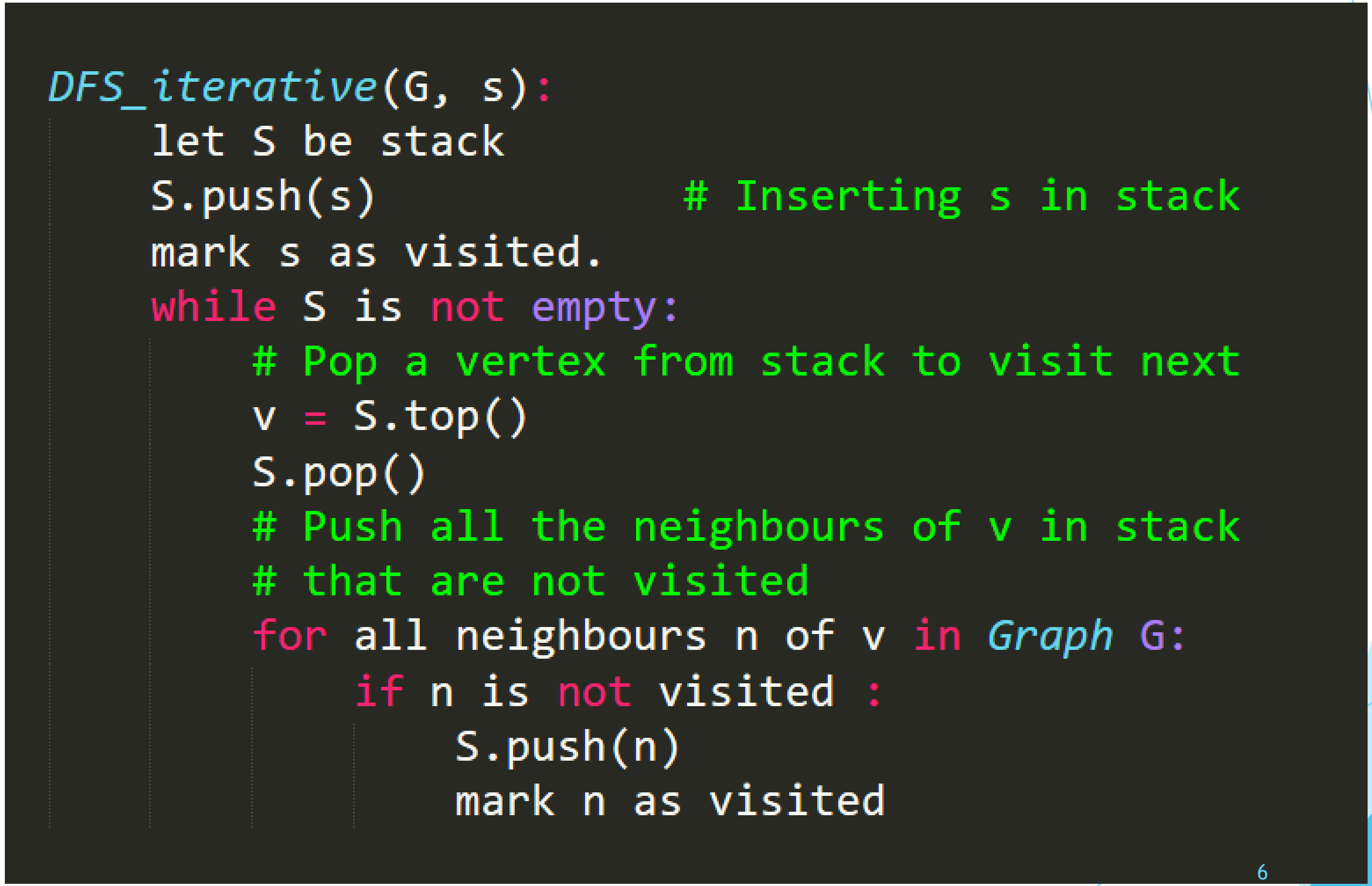


Depth First Search



```
DFS_recursive(G, s):
    mark s as visited
    for all neighbours n of s in Graph G:
        if n is not visited:
            DFS_recursive(G, n)
```

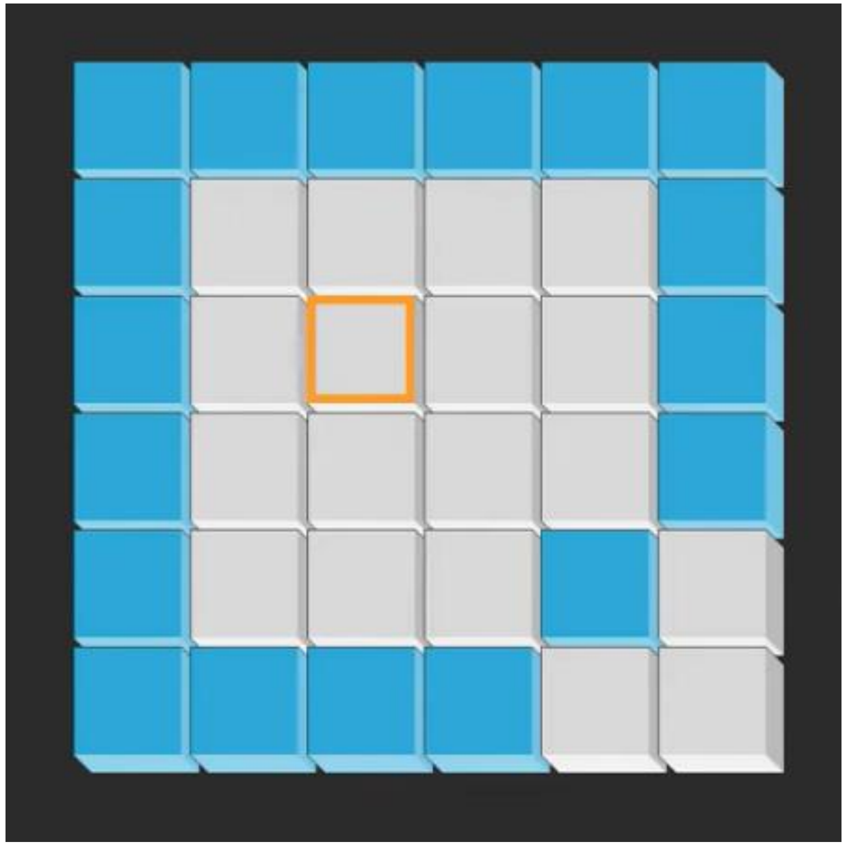
Depth First Search

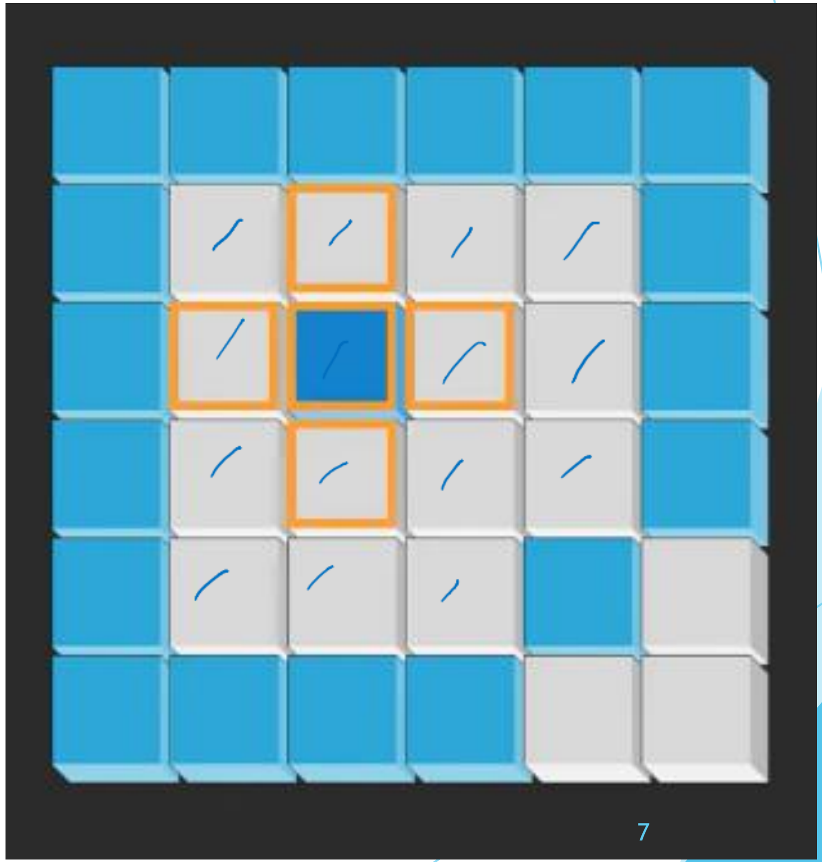


```
DFS_iterative(G, s):
    let S be stack
    S.push(s)            # Inserting s in stack
    mark s as visited.
    while S is not empty:
        # Pop a vertex from stack to visit next
        v = S.top()
        S.pop()
        # Push all the neighbours of v in stack
        # that are not visited
        for all neighbours n of v in Graph G:
            if n is not visited :
                S.push(n)
                mark n as visited
```

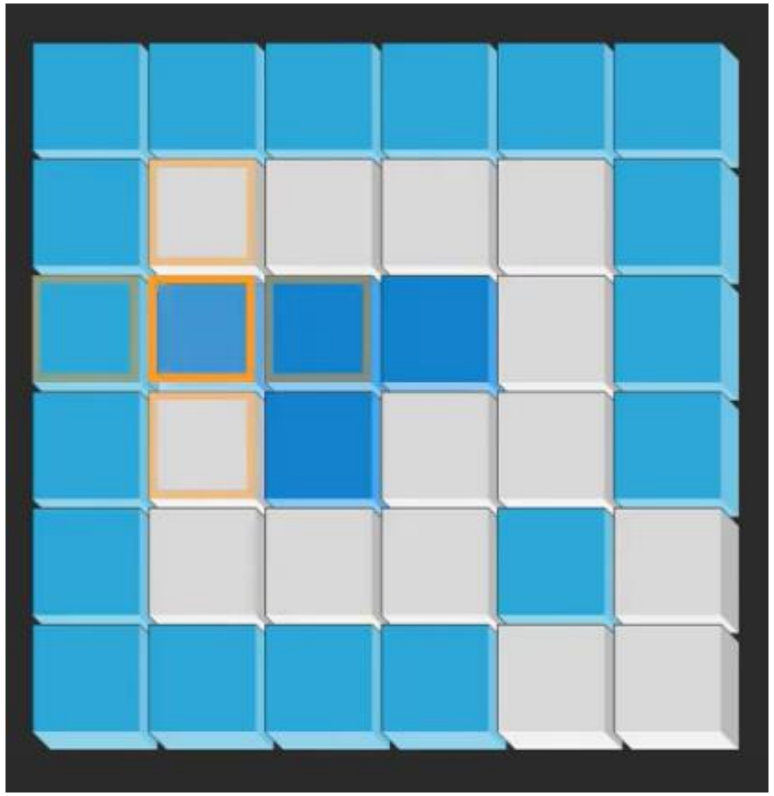
Flood Fill

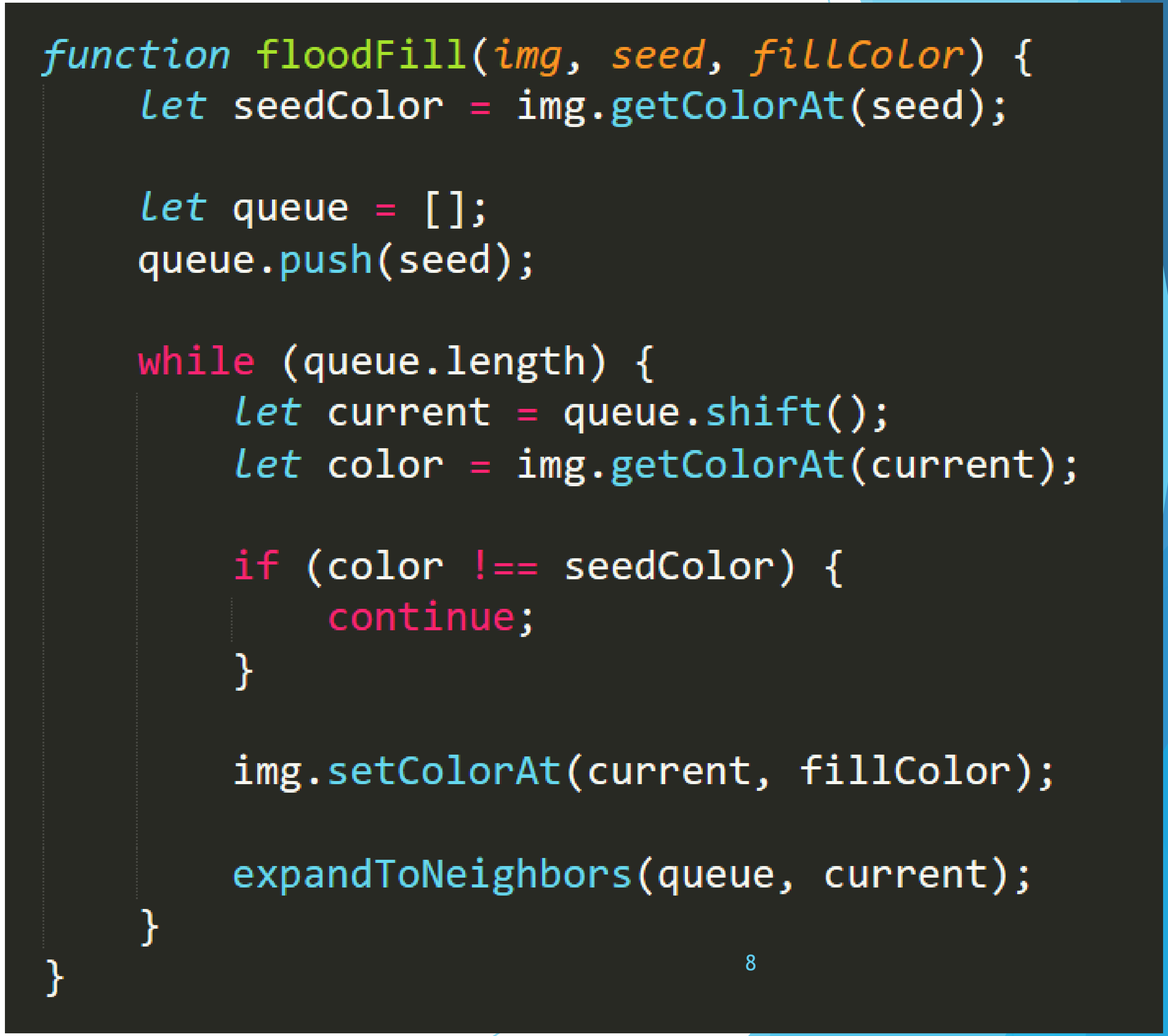
vector<vector<int>> grid





Flood Fill





```
function floodFill(img, seed, fillColor) {
    let seedColor = img.getColorAt(seed);

    let queue = [];
    queue.push(seed);

    while (queue.length) {
        let current = queue.shift();
        let color = img.getColorAt(current);

        if (color !== seedColor) {
            continue;
        }

        img.setColorAt(current, fillColor);

        expandToNeighbors(queue, current);
    }
}
```

1. Max Area of Island

Given a non-empty 2D array grid of 0's and 1's, an island is a group of 1's (representing land) connected 4-directionally (horizontal or vertical.)

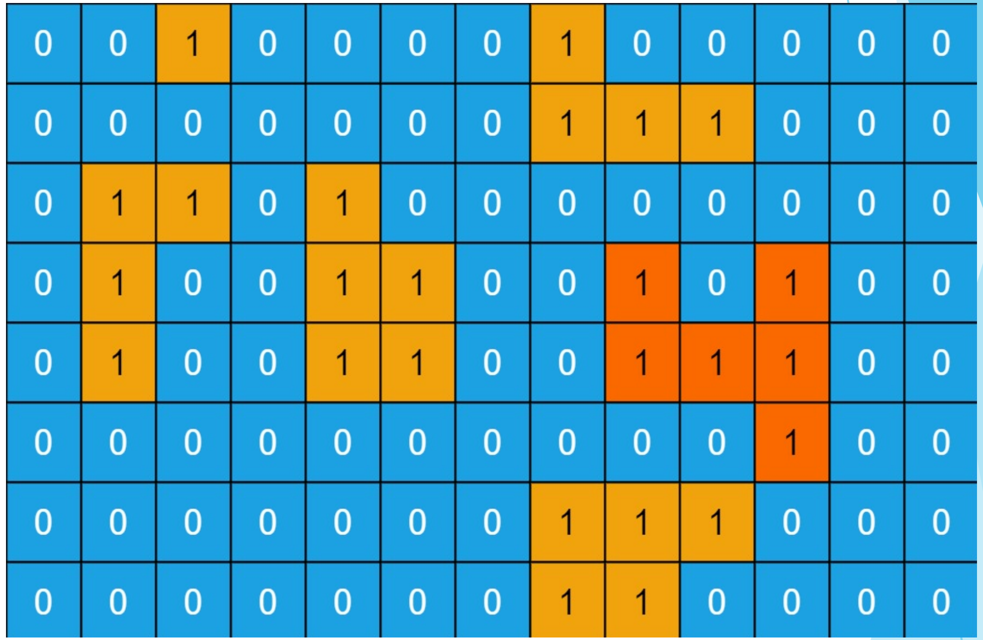
You may assume all four edges of the grid are surrounded by water.

Find the maximum area of an island in the given 2D array.

The length of each dimension in the given grid does not exceed 50.

Max Area of Island

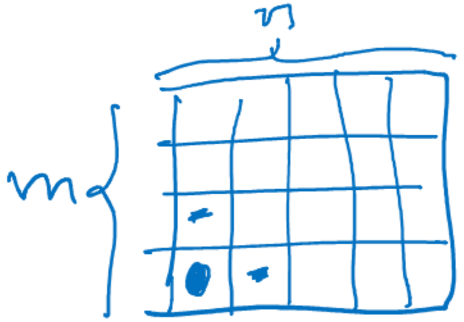
```
[[0,0,1,0,0,0,0,1,0,0,0,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,1,1,0,1,0,0,0,0,0,0,0,0],
 [0,1,0,0,1,1,0,0,1,0,1,0,0],
 [0,1,0,0,1,1,0,0,1,1,1,0,0],
 [0,0,0,0,0,0,0,0,0,0,1,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,0,0,0,0,0,0,1,1,0,0,0,0]]
```



0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	1	0	1	0	0
0	1	0	0	1	1	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0

Looking at each cell, if it is 1 we find out how big is the island, by recursively visiting all its neighbors.

Max Area of Island



```
int maxAreaOfIsland(vector<vector<int>>& grid) {
    m = grid.size();
    n = grid[0].size();
    int ans = 0;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++) {
            if (grid[i][j] == 1)
                ans = max(ans, dfs(grid, i, j));
    }
    return ans;
}
```

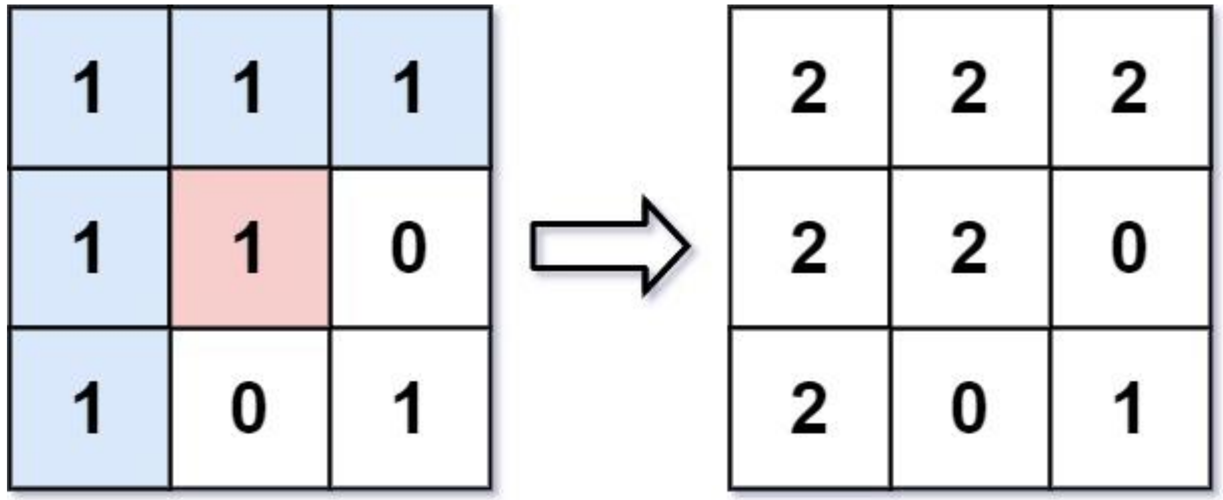
Max Area of Island

- https://leetcode.com/problems/max-area-of-island/

2. Flood Fill

- We have a grid of cells, each with some color.
- We want to change the color of the region containing a given cell (i, j).

Flood Fill



Flood Fill

- We can fill the cell (i, j), then recursively flood fill on the neighbors.
- We should only continue to fill neighbors if they have the same color as (i, j).
- We don’t want to visit the same cell more than once.
- To avoid infinite calls, we can change the color, and only then visit neighbors.
- We need to remember the original color of (i, j) in order to check neighbors, so we use a variable called tmp.

```
floodfill(i, j, c):
    tmp = color[i][j]
    color[i][j] = c
    for (x, y) neighbor of (i, j):
        if color[x][y] == tmp:
            floodfill(x, y, c)
```

Flood Fill

- https://leetcode.com/problems/flood-fill/

3. Making a Large Island

In a 2D grid of 0s and 1s, what is the size of the largest island if one is allowed to change at most one 0 to 1?

An island is a 4-directionally connected group of 1s.

1 <= grid.length = grid[0].length <= 50.

0 <= grid[i][j] <= 1.

Making a Large Island: Flood Fill

- First use DFS to find all islands.
- Store areas of all islands.
- Mark each island differently
- Also, for each new island, store its area.
- Then loops through all 0's, for each 0, computes the sum of the area of all neighboring islands.

Making a Large Island: Flood Fill

```
[[0,0,2,0,0,0,0,3,0,0,0,0,0],
 [0,0,0,0,0,0,0,3,3,3,0,0,0],
 [0,4,4,0,5,0,0,0,0,0,0,0,0],
 [0,4,0,0,5,5,0,0,6,0,6,0,0],
 [0,4,0,0,5,5,0,0,6,6,6,0,0],
 [0,0,0,0,0,0,0,0,0,0,6,0,0],
 [0,0,0,0,0,0,0,7,7,7,0,0,0],
 [0,0,0,0,0,0,0,7,7,0,0,0,0]]
```

Areas of the islands

1	4	4	5	6	5

Making a Large Island: Flood Fill

- Input: grid = [[1,0],
  [0,1]]
- Output: 3

- Input: grid = [[1,1],
  [1,0]]
- Output: 4

Making a Large Island

- https://leetcode.com/problems/making-a-large-island/

4. Island

Given a image, which is a grid where each cell is either land (denoted as ‘L’), water (denoted as ‘W’), or covered by clouds (denoted as ‘C’). Clouds mean that the square could either be land or water. Determine the minimum number of islands that is consistent with the given image.

```
Input:  Output              Input:      Output:
3 2     1                   3 4         0
LW                          CCCC
CC                          CCCC
WL                          CCCC
```

Island

- Start DFS when encountering a land cell
- During DFS, each encountered cloud cell is treated as a land cell

Island

- https://open.kattis.com/problems/islands3