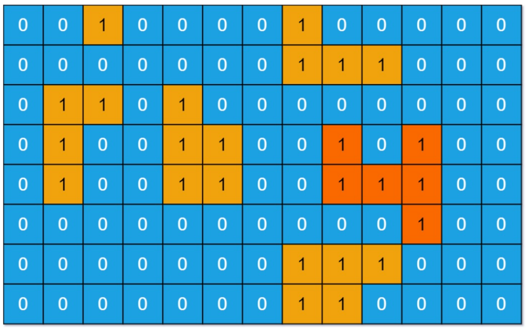
Quiz 5
Total - [25 points]

Name:
NetID:

[Graph DFS] You are given an m x n binary matrix grid. 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. The area of an island is the number of cells with a value 1 in the island. Return the maximum area of an island in grid. If there is no island, return 0. [**10 points**]

Example 1:

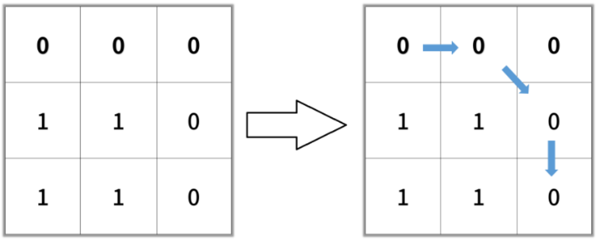


0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	1	0	1	0	0
0	1	0	0	1	1	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0

output: 6

```
class Solution {
    private int m, n;
    private int[][] grid;
    private boolean[][] seen;

    public int maxAreaOfIsland(int[][] grid) {
        this.grid = grid;
        this.m = grid.length;
        this.n = grid[0].length;
        this.seen = new boolean[m][n];
        int maxArea = 0;

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (!seen[i][j] && grid[i][j] == 1) {
                    int area = dfs(i, j);
                    maxArea = Math.max(maxArea, area);
                }
            }
        }
        return maxArea;
    }

    private int dfs(int r, int c) {
        if (r < 0 || r >= m || c < 0 || c >= n || seen[r][c] || grid[r][c] == 0) {
            return 0;
        }
          seen [r] [c] = True.
      int area = 1
          area = area + dfs [r-1][c]
          area = area + dfs [r+1][c]
          area = area + dfs [r][c+1]
          area = area + dfs [r][c-1]

        return area;
    }
}
```

[Graph BFS] Given an n x n binary matrix grid, return the length of the shortest clear path in the matrix. If there is no clear path, return -1. A clear path is a path from the top-left cell (0, 0) to th e bottom-right cell (n - 1, n - 1) such that all visited cells are 0. You may move 8-directionally (up, down, left, right, or diagonally). [**5points**]



Shortes → BFS

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

```
class State {
    int row;
    int col;
    int steps;
    State(int row, int col, int steps) {
        this.row = row;
        this.col = col;
        this.steps = steps;
    }
}

class Solution {
    int n;
    int[][] directions = new int[][]{{-1, -1}, {-1, 0}, {-1, 1}, {0, -1}, {0, 1}, {1, -1}, {1, 0}, {1,
1}};

    public int shortestPathBinaryMatrix(int[][] grid) {
        if (grid[0][0] == 1) {
            return -1;
        }

        n = grid.length;

        boolean[][] seen = new boolean[n][n];
        seen[0][0] = true;
        Queue<State> queue = new LinkedList<>();
        queue.add(new State(0, 0, 1)); // row, col, steps

        while (!queue.isEmpty()) {
            State state = queue.remove();
            int row = state.row, col = state.col, steps = state.steps;
            if (BLANK1) { (row == n-1 && col= n-1)
                return steps;
            }

going to neighbor
            for (int[] direction: directions) {
                int nextRow = row + direction[0], nextCol = col + direction[1];
                if (valid(nextRow, nextCol, grid) && BLANK2) {      !seen[nextRow][nextCol]
                    BLANK3; seen[nextRow][nextCol] = true;
                    BLANK4; queue.add(new State(nextRow, nextCol, step+1))
                }
            }
        }

        return -1;
    }
    public boolean valid(int row, int col, int[][] grid) {
        return 0 <= row && row < n && 0 <= col && col < n && BLANK5;
    }
}
```

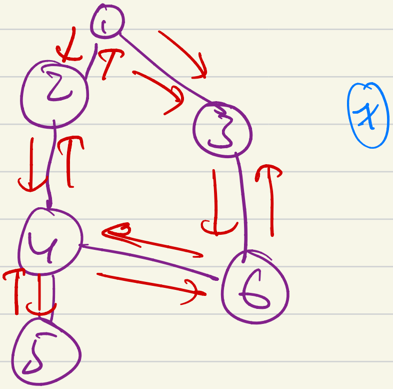
Programming Assn. 5

① IsConnected

→ Do a bfs

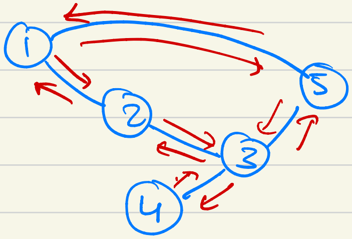
→ keep a count of all node

→ check if count == Num of Nodes

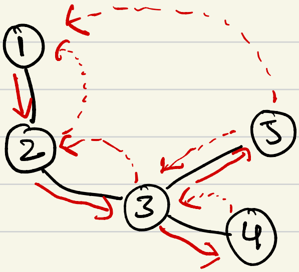


② hasCycle

Hypothetically, I do a DFS from node ①

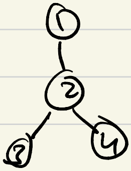


DFS:-

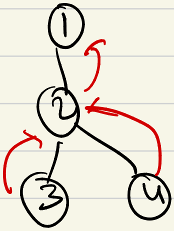


cyclic

eg a graph without cycle



DFS ⇒



Acyclic

observations:-

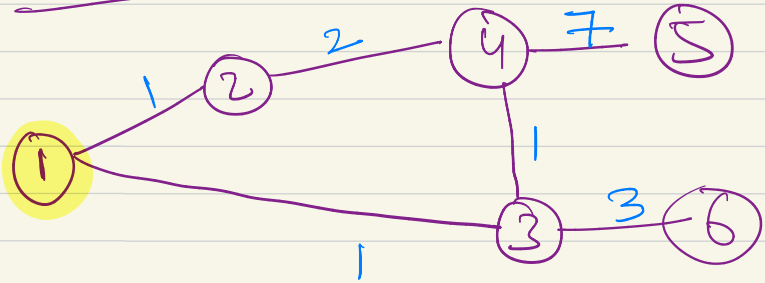
① In Cyclic ⇒ Atleast 1 node has > 1 backedge

② In Cyclic ⇒ there can be a non-parent back edge.

① Check when there is a back edge:-

- ↳ how? ⇒ when $\underbrace{\text{visited [neigh] = True}}_{\text{Existance of BE}}$
- ↳ check if this back edge is to its parent

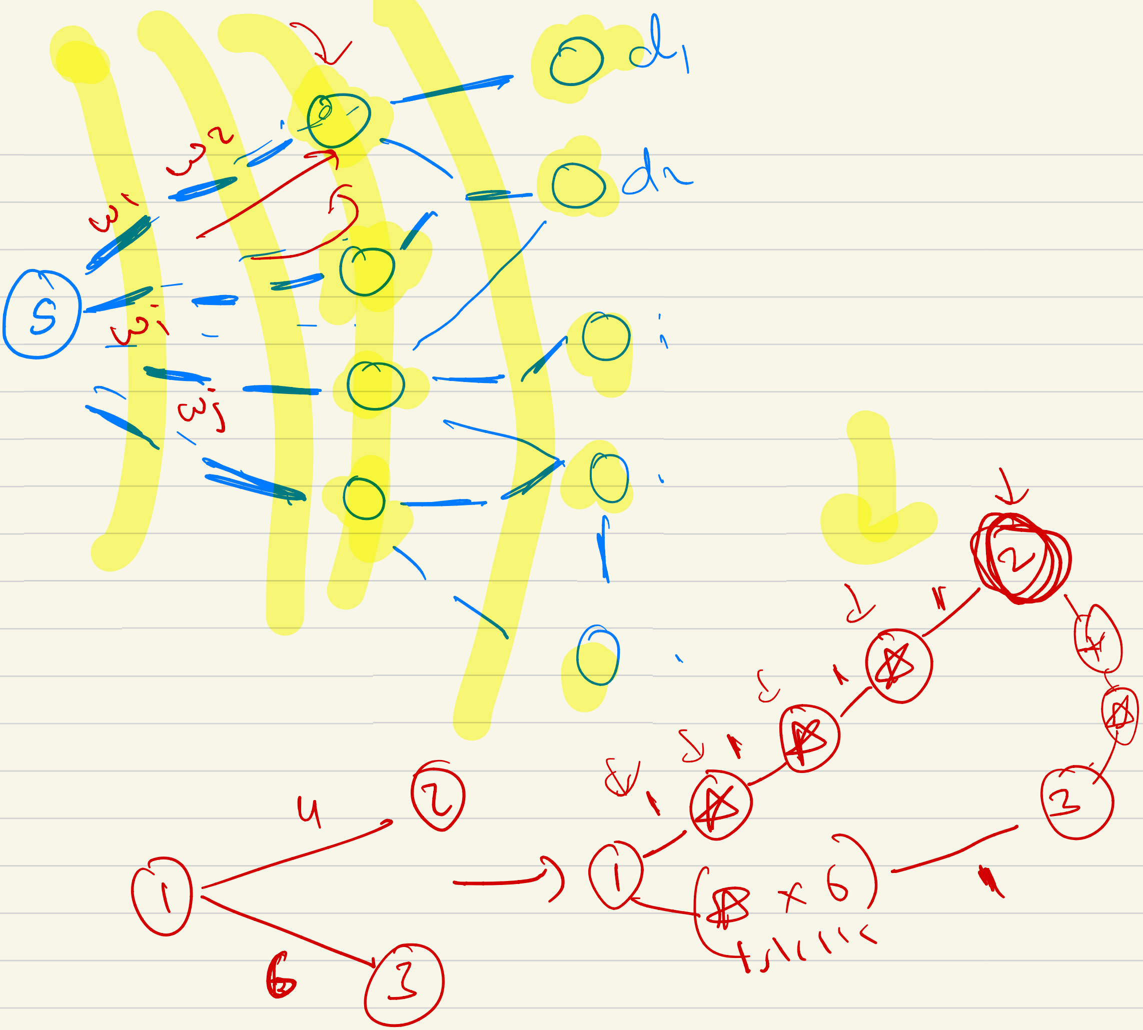
Intution behind Dijkstra



① → ④ } → 2 steps

① → ② } → 1 step

① → ⑥ } → 4 steps



s
w1
w2
wi
wj
d1
d2
1
2
3
4
6
x 6