# Answer of 1

1 a) The graph is not connected. The connected components are {D,E,I} and {A,B,C,F,G,H}.

1 b) Spanning Tree/Forest of graph G is as follows:

A drawing of a network

Description automatically generated with medium confidence

1 c) No, G does not meet the criteria for being a Hamiltonian graph as there are two connected components. So, a cycle that goes through each vertices exactly once cannot be constructed due to disconnects.

1 d) No, a vertex cover of size 5 or less doesn’t exist. As at least one vertex will not be covered if we try with size 5 or less. We’ll need at least size 6.

# Answer of 2

A drawing of a spider web

Description automatically generated

# Answer of 3

# Answer of 4

class Solution {

public int numIslands(char[][] grid) {

boolean[][] visited = new boolean[grid.length][grid[0].length];

int numberOfIslands = 0;

for (int i = 0; i < grid.length; i++) {

for (int j = 0; j < grid[i].length; j++) {

if (grid[i][j] == '1' && !visited[i][j]) {

numberOfIslands++;

dfs(grid, visited, i, j);

}

}

}

return numberOfIslands;

}

private void dfs(char[][] grid, boolean[][] visited, int i, int j) {

if (i < 0 || i >= grid.length || j < 0 || j >= grid[i].length || grid[i][j] == '0' || visited[i][j]) {

return;

}

visited[i][j] = true;

dfs(grid, visited, i - 1, j); // up

dfs(grid, visited, i + 1, j); // down

dfs(grid, visited, i, j - 1); // left

dfs(grid, visited, i, j + 1); // right

}

}

# Answer of 5

class Solution {

public int maxAreaOfIsland(int[][] grid) {

boolean[][] visited = new boolean[grid.length][grid[0].length];

int maxArea = 0;

for (int i = 0; i < grid.length; i++) {

for (int j = 0; j < grid[i].length; j++) {

if (grid[i][j] == 1 && !visited[i][j]) {

maxArea = Math.max(maxArea, dfs(grid, visited, i, j));

}

}

}

return maxArea;

}

private int dfs(int[][] grid, boolean[][] visited, int i, int j) {

if (i < 0 || i >= grid.length || j < 0 || j >= grid[i].length || grid[i][j] == 0 || visited[i][j]) {

return 0;

}

visited[i][j] = true;

return 1 + dfs(grid, visited, i - 1, j) +

dfs(grid, visited, i + 1, j) +

dfs(grid, visited, i, j - 1) +

dfs(grid, visited, i, j + 1);

}

}

# Answer of 6

class Solution {

public boolean exist(char[][] board, String word) {

boolean[][] visited = new boolean[board.length][board[0].length];

for (int i = 0; i < board.length; i++) {

for (int j = 0; j < board[i].length; j++) {

if (board[i][j] == word.charAt(0) && dfs(board, visited, word, 0, i, j)) {

return true;

}

}

}

return false;

}

private boolean dfs(char[][] board, boolean[][] visited, String word, int wordIndex, int i, int j) {

if (i < 0 || i >= board.length || j < 0 || j >= board[i].length ||

board[i][j] != word.charAt(wordIndex) || visited[i][j]) {

return false;

}

if (wordIndex == word.length() - 1) {

return true;

}

visited[i][j] = true;

boolean result = dfs(board, visited, word, wordIndex + 1, i - 1, j) ||

dfs(board, visited, word, wordIndex + 1, i + 1, j) ||

dfs(board, visited, word, wordIndex + 1, i, j - 1) ||

dfs(board, visited, word, wordIndex + 1, i, j + 1);

visited[i][j] = false;

return result;

}

}