Assignment Solutions : Graphs - 2 Problems on DFS and BFS

Q1 Flood Fill **Leetcode:-733.**

Solution :

Code :

```
class Solution {
public:
    vector<vector<int>> floodFill(vector<vector<int>>&
image, int sr, int sc, int newColor) {
        int rows = image.size();
        int cols = image[0].size();
        int originalColor = image[sr][sc];

        if (originalColor == newColor) {
            return image;
        }

        dfs(image, sr, sc, originalColor, newColor);
        return image;
    }

    void dfs(vector<vector<int>>& image, int i, int j, int
originalColor, int newColor) {
        if (i < 0 || i >= image.size() || j < 0 || j >=
image[0].size() || image[i][j] != originalColor) {
            return;
        }

        image[i][j] = newColor;

        dfs(image, i + 1, j, originalColor, newColor);
        dfs(image, i - 1, j, originalColor, newColor);
        dfs(image, i, j + 1, originalColor, newColor);
        dfs(image, i, j - 1, originalColor, newColor);
    }
};
```

Q2 Max Area of Island Leetcode:-695.

Solution :

Code :

```
class Solution {
public:
    int maxAreaOfIsland(vector<vector<int>>& grid) {
        int maxArea = 0;
        for (int i = 0; i < grid.size(); ++i) {
            for (int j = 0; j < grid[0].size(); ++j) {
                if (grid[i][j] == 1) {
                    maxArea = max(maxArea, dfs(grid, i, j));
                }
            }
        }
        return maxArea;
    }

    int dfs(vector<vector<int>>& grid, int i, int j) {
        if (i < 0 || i >= grid.size() || j < 0 || j >=
grid[0].size() || grid[i][j] == 0) {
            return 0;
        }

        grid[i][j] = 0;  // Mark as visited

        return 1 + dfs(grid, i + 1, j) + dfs(grid, i - 1, j)
+ dfs(grid, i, j + 1) + dfs(grid, i, j - 1);
    }
};
```

Q3 Keys and Rooms Leetcode:-841.

Solution :

Code :

```
class Solution {
public:
    bool canVisitAllRooms(vector<vector<int>>& rooms) {
        int n = rooms.size();
        vector<bool> visited(n, false);
        dfs(rooms, 0, visited);
        for (bool roomVisited : visited) {
            if (!roomVisited) {
                return false;
            }
        }

        return true;
    }

    void dfs(vector<vector<int>>& rooms, int room,
vector<bool>& visited) {
        visited[room] = true;

        for (int nextRoom : rooms[room]) {
            if (!visited[nextRoom]) {
                dfs(rooms, nextRoom, visited);
            }
        }
    }
};
```

Q4 Shortest Path in Binary Matrix **Leetcode:-1091.**

Solution :

Code :

```
class Solution {
public:
    int shortestPathBinaryMatrix(vector<vector<int>>& grid)
{
        int n = grid.size();
        if (grid[0][0] == 1 || grid[n - 1][n - 1] == 1) {
            return -1;
        }

        vector<vector<int>> directions = {{-1, 0}, {1, 0},
{0, -1}, {0, 1}, {-1, -1}, {-1, 1}, {1, -1}, {1, 1}};
        queue<pair<int, int>> q;
        q.push({0, 0});
        grid[0][0] = 1;

        while (!q.empty()) {
            auto curr = q.front();
            q.pop();
            int x = curr.first;
            int y = curr.second;
            int dist = grid[x][y];

            if (x == n - 1 && y == n - 1) {
                return dist;
            }

            for (const auto& dir : directions) {
                int nx = x + dir[0];
                int ny = y + dir[1];

                if (nx >= 0 && nx < n && ny >= 0 && ny < n
&& grid[nx][ny] == 0) {
                    q.push({nx, ny});
                    grid[nx][ny] = dist + 1;
                }
            }
        }

        return -1;
    }
};
```

Q5 As Far from Land as Possible **Leetcode:-1162.**

Solution :

Code :

```
class Solution {
public:
    int maxDistance(vector<vector<int>>& grid) {
        int n = grid.size();
        queue<pair<int, int>> q;

        // Enqueue all land cells
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (grid[i][j] == 1) {
                    q.push({i, j});
                }
            }
        }
        if (q.empty() || q.size() == n * n) {
            return -1;  // No water or land
        }

        vector<vector<int>> directions = {{-1, 0}, {1, 0},
{0, -1}, {0, 1}};
        int maxDistance = -1;

        while (!q.empty()) {
            int size = q.size();

            for (int i = 0; i < size; ++i) {
                auto curr = q.front();
                q.pop();

                for (const auto& dir : directions) {
                    int nx = curr.first + dir[0];
                    int ny = curr.second + dir[1];

                    if (nx >= 0 && nx < n && ny >= 0 && ny <
n && grid[nx][ny] == 0) {
                        q.push({nx, ny});
                        grid[nx][ny] = 1;  // Mark as
visited
                    }
                }
            }

            maxDistance++;
        }

        return maxDistance;
    }
};
```