IMPLEMENTATION OF BFS AND DFS

GRAPH TRAVERSAL ALGORITHMS IN DATA STRUCTURE

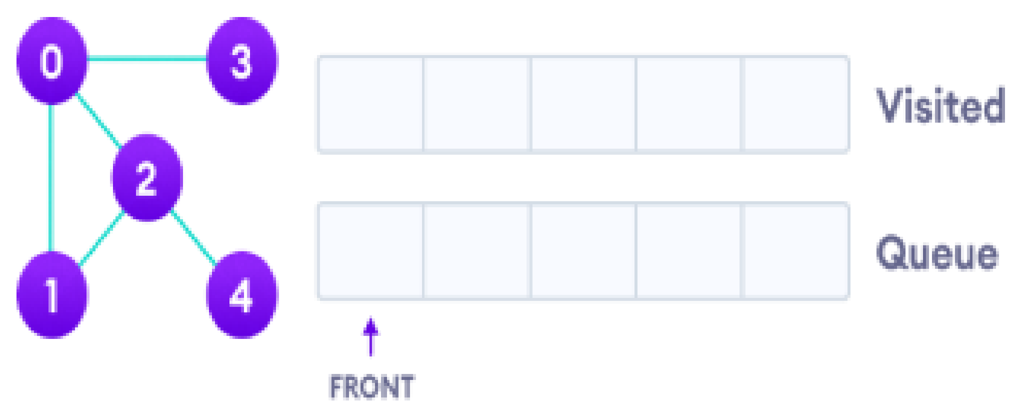
Graph traversal is a search technique to find a vertex in a graph. In the search process, graph traversal is also used to determine the order in which it visits vertices. Without producing loops, a graph traversal finds the edges to be employed in the search process. That is, utilizing graph traversal, you can visit all the graph's vertices without going through a looping path. There are two methods for traversing graphs, which are as follows:

- **Breadth-First Search or BFS Algorithm**
- **Depth- First Search or DFS Algorithm**

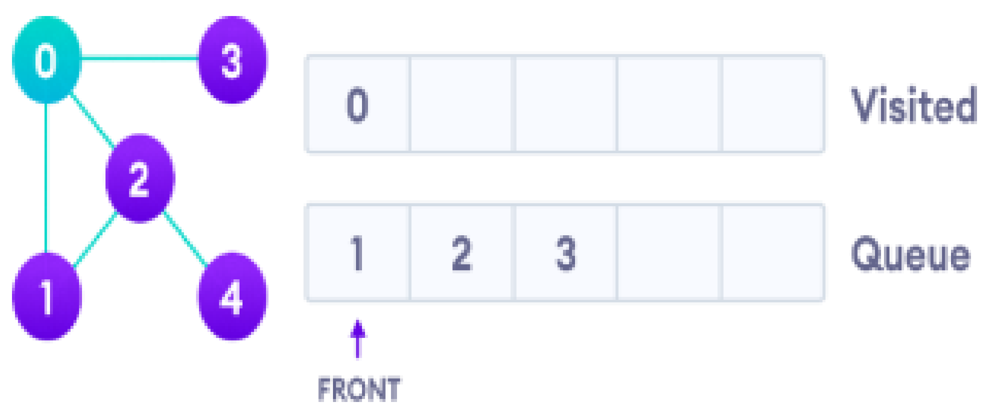
BREADTH-FIRST SEARCH ALGORITHM

Breadth-First Search Algorithm or BFS is the most widely utilized method. BFS is a graph traversal approach in which you start at a source node and layer by layer through the graph, analyzing the nodes directly related to the source node. Then in BFS traversal, you must move on to the next-level neighbor nodes. According to the BFS, you must traverse the graph in a breadthwise direction: To begin, move horizontally and visit all the current layer's nodes. Continue to the next layer. Breadth-First Search uses a queue data structure to store the node and mark it as "visited" until it marks all the neighboring vertices directly related to it. The queue operates on the First In First Out (FIFO) principle, so the node's neighbors will be viewed in the order in which it inserts them in the node, starting with the node that was inserted first.

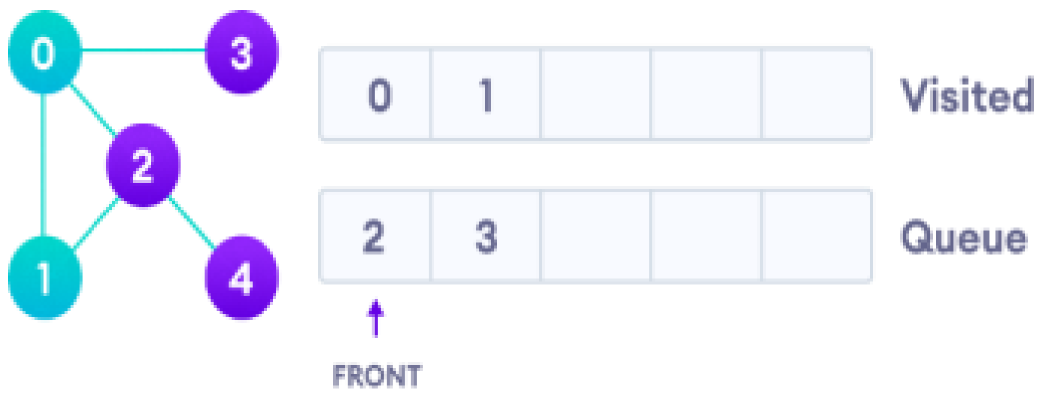
BFS EXAMPLE



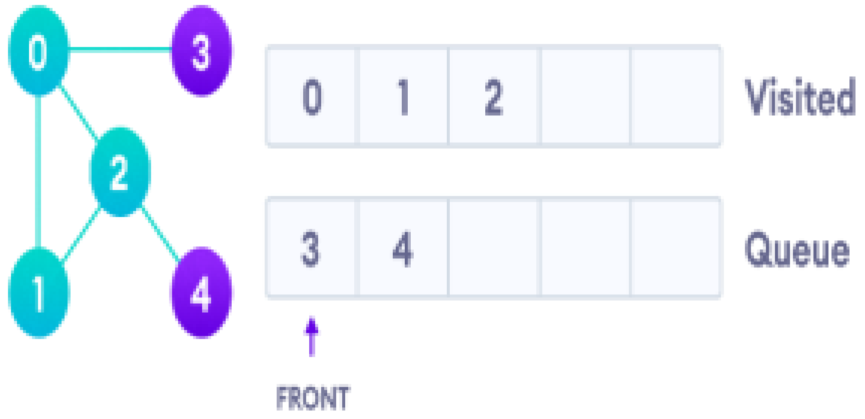
1] Undirected graph with 5 vertices



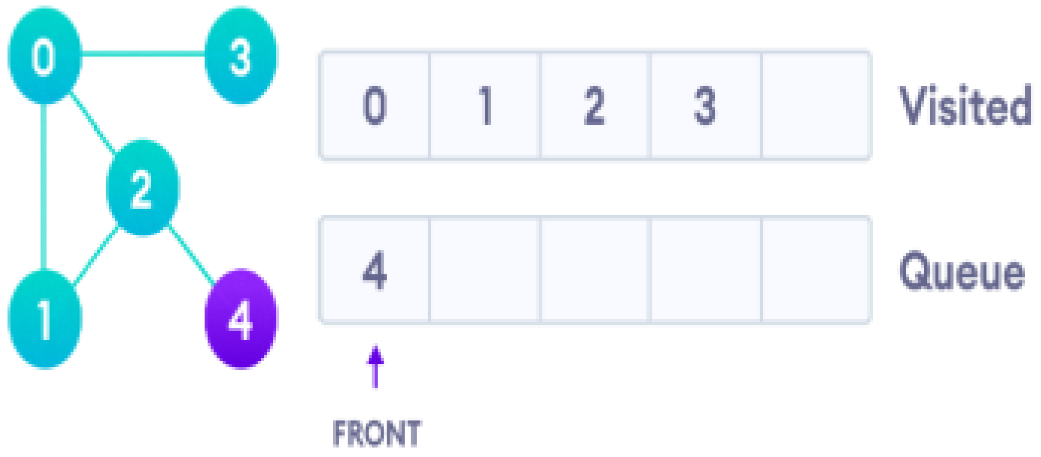
2] Visit start vertex and add its adjacent vertices to queue



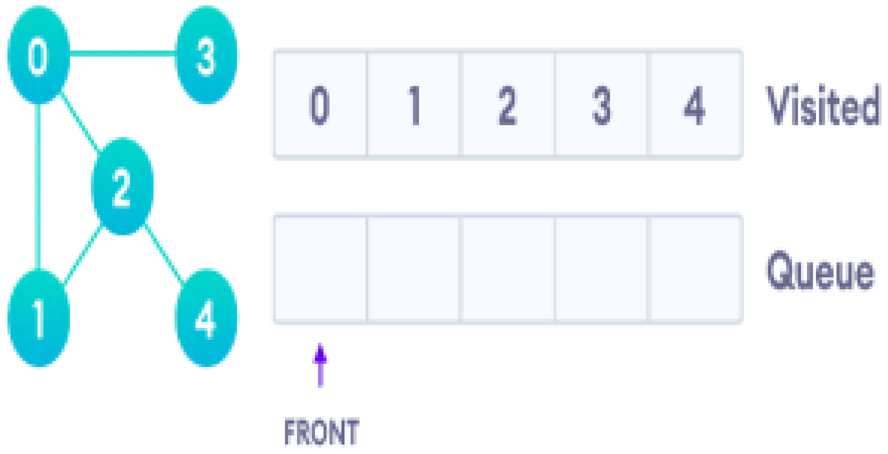
3] Visit the first neighbor of start node 0, which is 1



4] Visit 2 which was added to queue earlier to add its neighbors



5] 4 remains in the queue

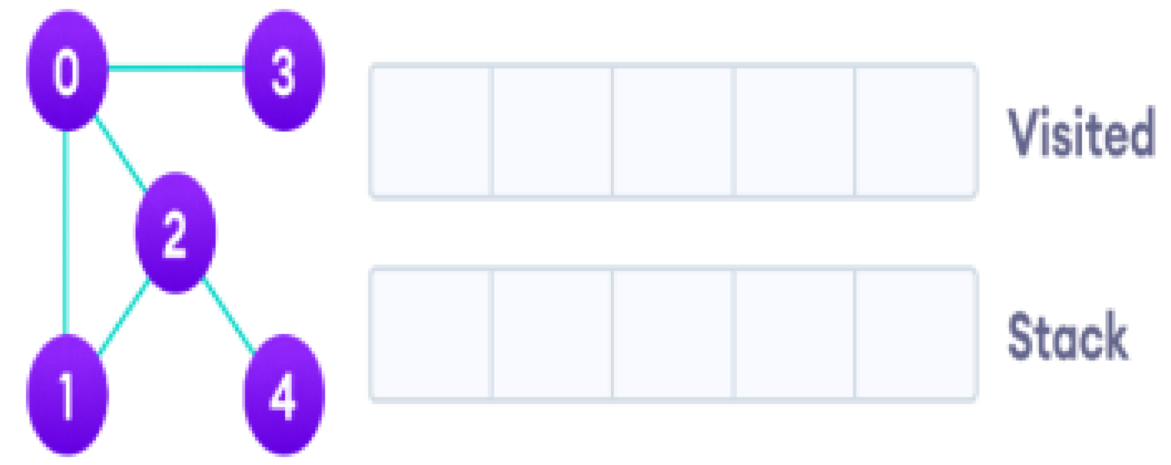


6] Visit last remaining item in the queue to check if it has unvisited neighbors

DEPTH-FIRST SEARCH ALGORITHM

ľhe depth-fiíst seaích oí DFS algoíithm tíaveíses oí exploíes data stíuctuíes, such as tíees and gíaphs. ľhe algoíithm staíts at the íoot node (in the case of a gíaph, you can use any íandom node as the íoot node) and examines each bíanch as faí as possible befoíe backtíacking. When a dead-end occuís in any iteíation, the Depth Fiíst Seaích (DFS) method tíaveíses a netwoík in a deathwaíd motion and uses a stack data stíuctuíe to íemembeí to acquiíe thenext veítex to staít a seaích.

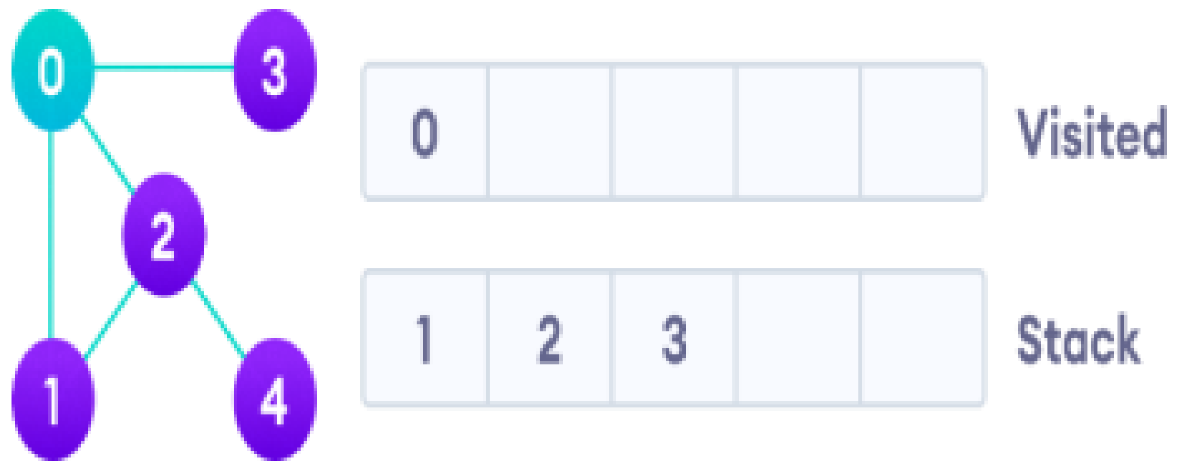
DFS EXAMPLE



Visited:

Stack:

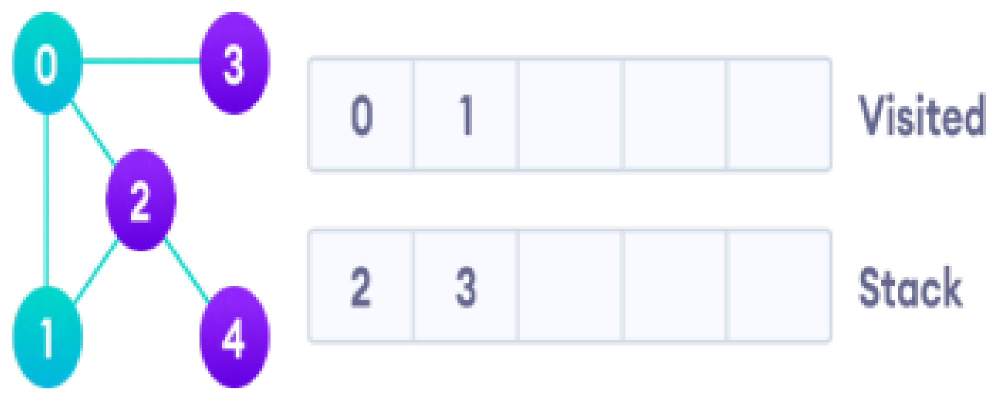
1] Undirected graph with 5 vertices.



Visited: 0

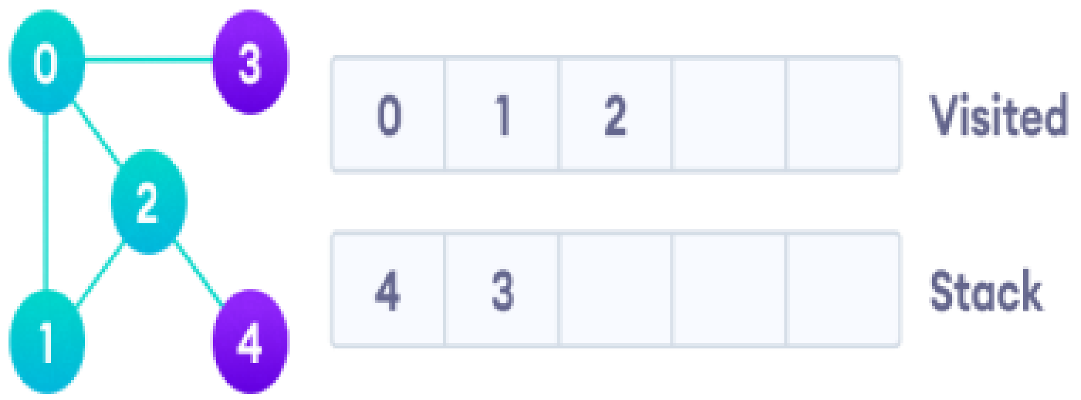
Stack: 1 2 3

2] Visit the element and put it in the visited list.



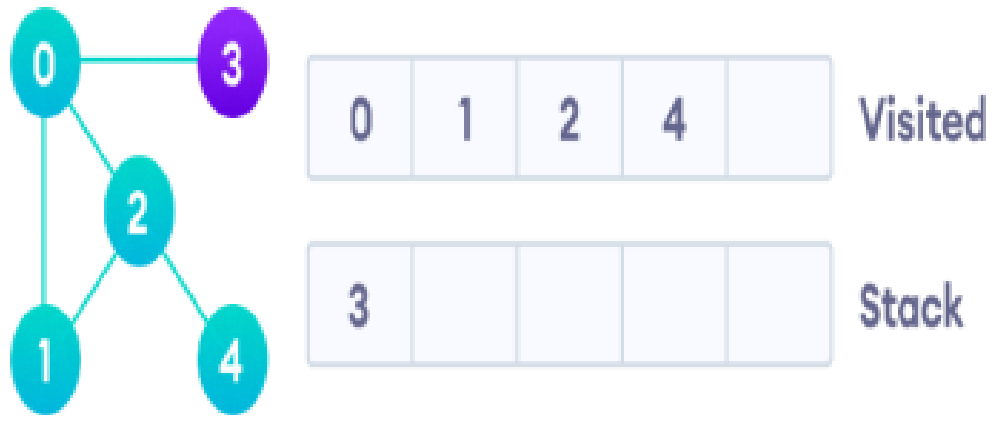
0	1				Visited
2	3				Stack

3] Visit the element at the top of stack



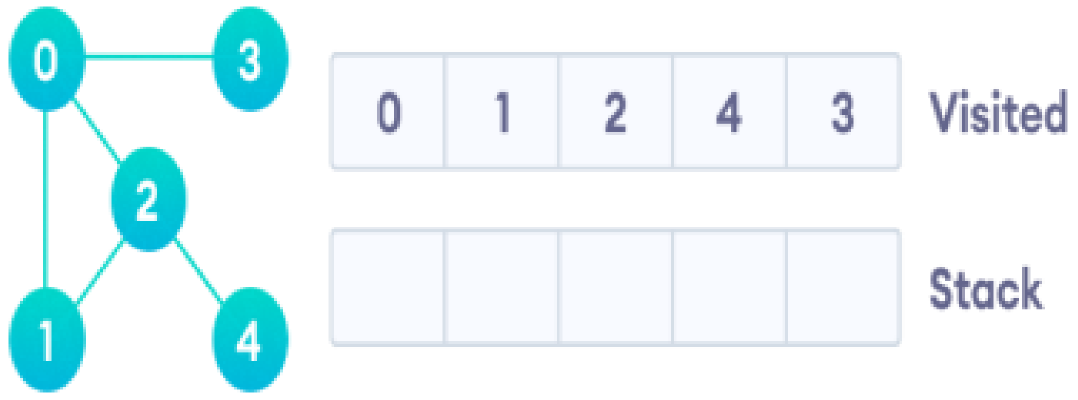
0	1	2			Visited
4	3				Stack

4] Vertex 2 has an unvisited adjacent vertex in so we add that to the top of the stack and visit it.



0	1	2	4		Visited
3					Stack

5] Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

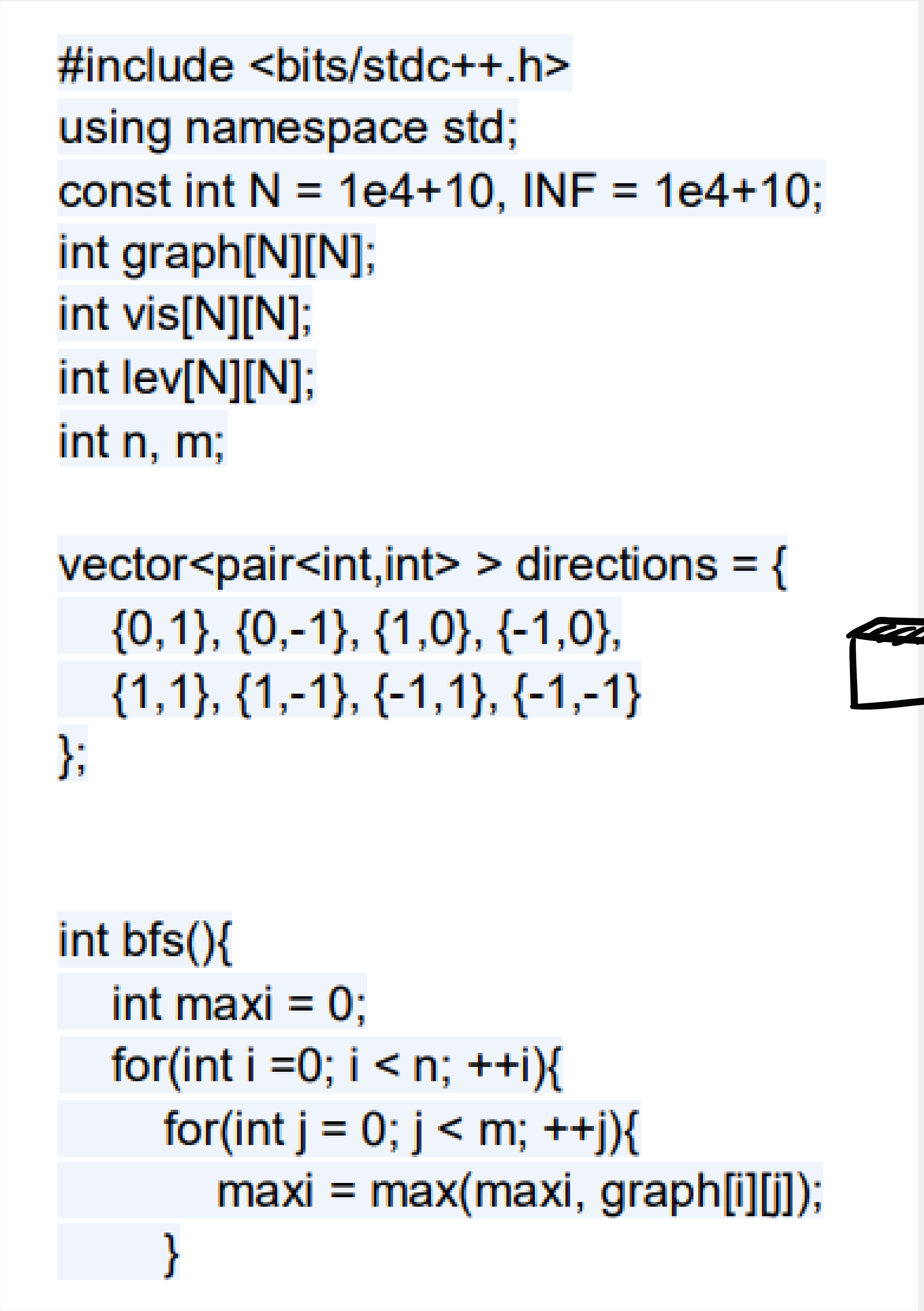


0	1	2	4	3	Visited
					Stack

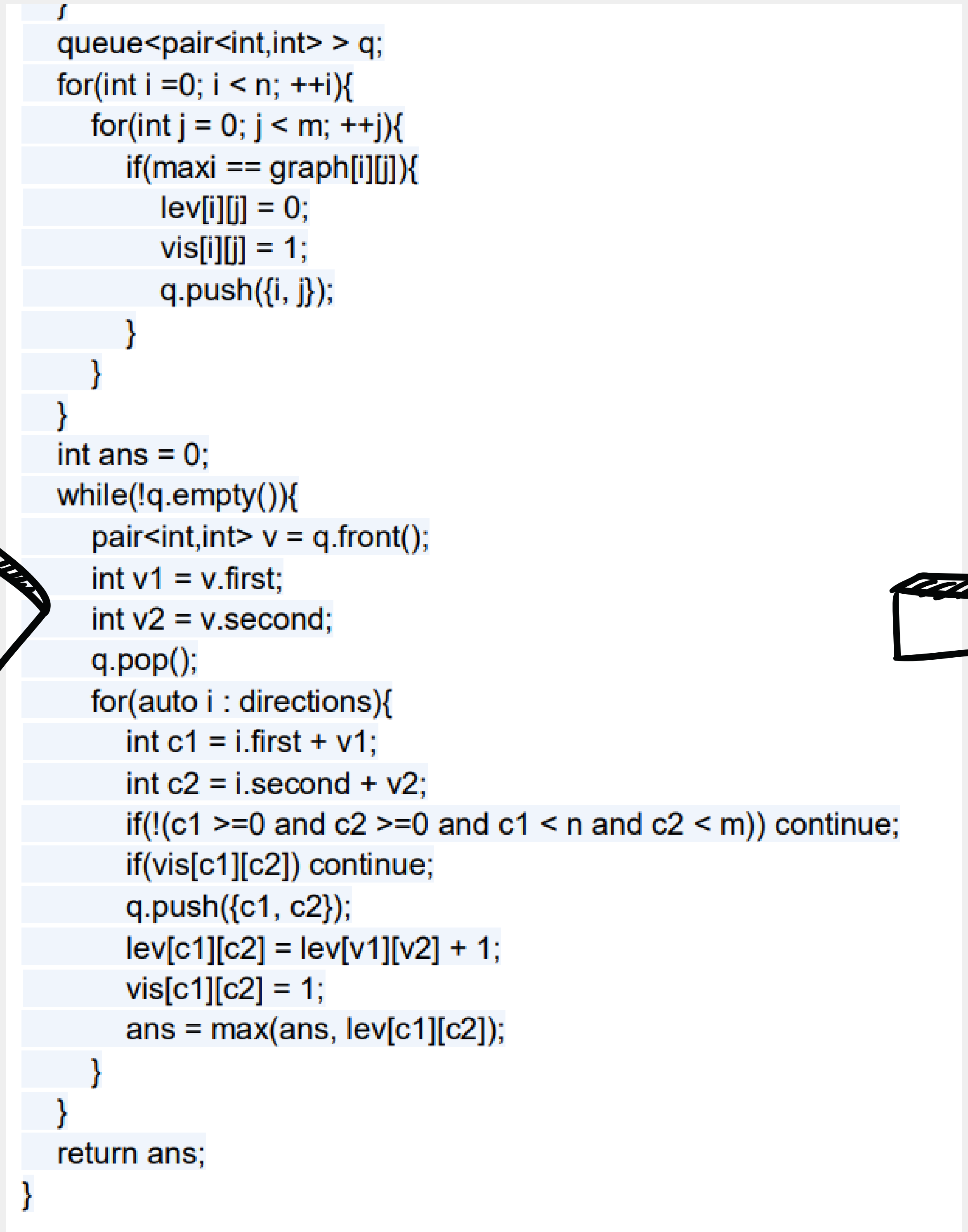
6] After we visit the last element 3, it doesn't have any unvisited adjacent nodes.

APPLICATION OF BFS-SNAKES AND TRANSITION FROM CAPITALISM TO SOCIALISM

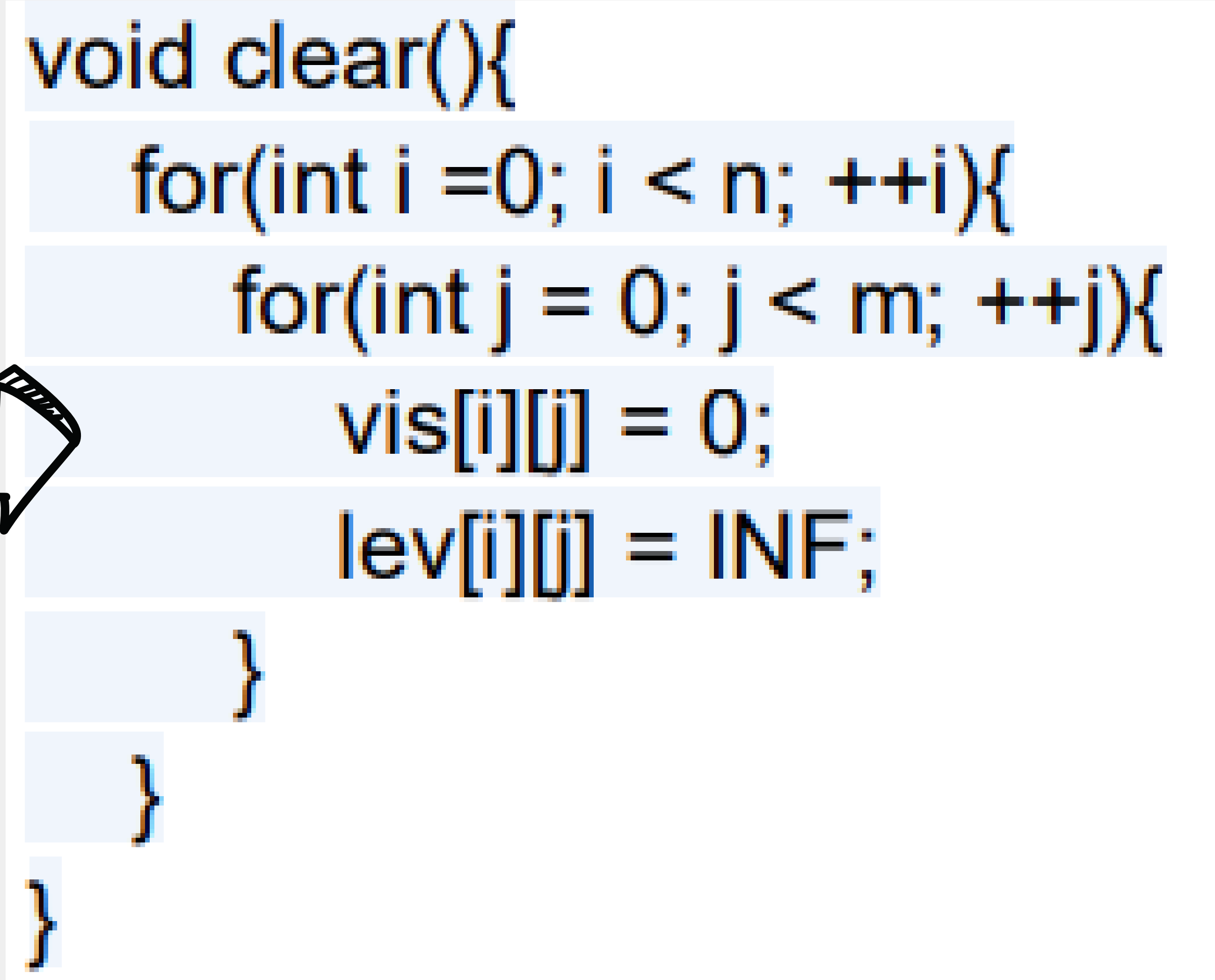
Problem Statement: After a long duration of the painful, torturous and tumultuous periods of capitalism in Snakeland, now the snakes have decided to adopt socialism. The houses in Snakeland are arranged in a rectangular fashion of dimension n * m. The wealth of the snake in the house at cell (i, j) is given by a[i][j] rupees. A bill has been passed for making a smooth transition from capitalism to socialism. At the end of each hour, the wealth of a snake will grow and will become equal to the highest wealth that its neighbor had (at the start of the hour). That is, consider the maximum wealth among its neighbors at the start of the hour, and this snake's wealth will become equal to that at the end of the hour. Note that this increase in wealth will happen simultaneously for each snake. Two houses are said to be neighbors of each other if they share a side or corner with each other. Find out the minimum number of hours required for this transition



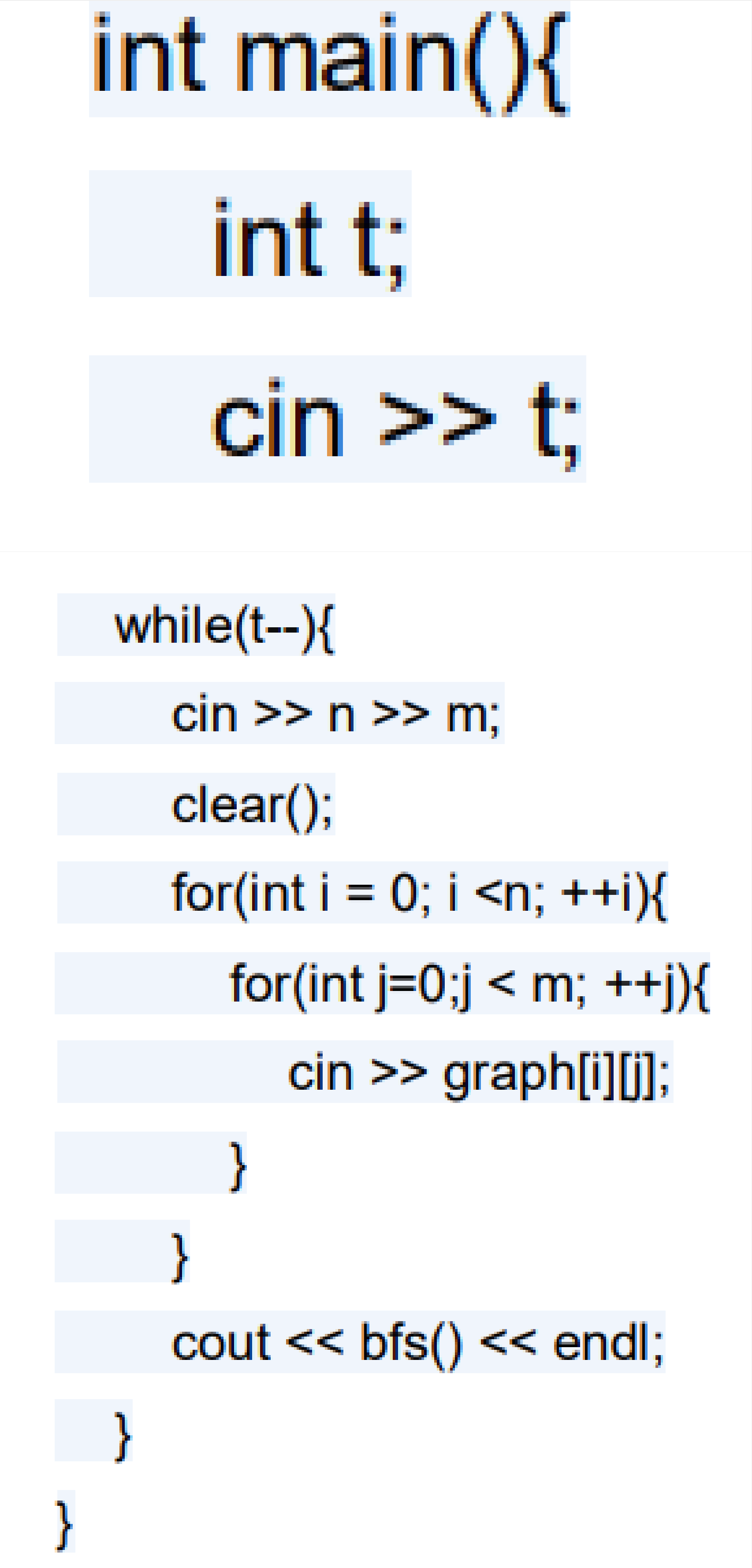
```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e4+10, INF = 1e4+10;
int graph[N][N];
int vis[N][N];
int lev[N][N];
int n, m;

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


int bfs(){
    int maxi = 0;
    for(int i =0; i < n; ++i){
        for(int j = 0; j < m; ++j){
            maxi = max(maxi, graph[i][j]);
        }
```



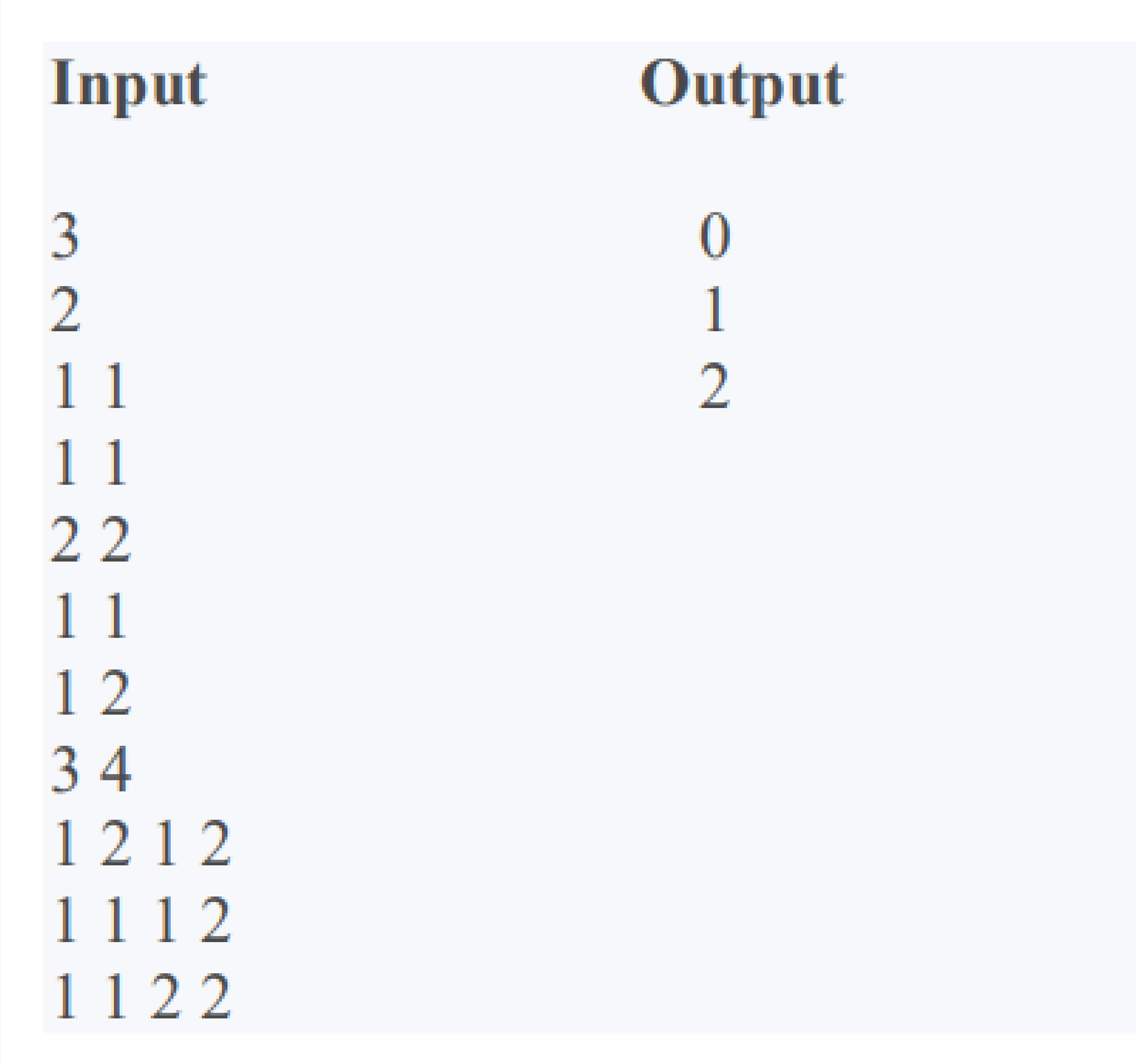
```
    }
    queue<pair<int,int> > q;
    for(int i =0; i < n; ++i){
        for(int j = 0; j < m; ++j){
            if(maxi == graph[i][j]){
                lev[i][j] = 0;
                vis[i][j] = 1;
                q.push({i, j});
            }
        }
    }
    int ans = 0;
    while(!q.empty()){
        pair<int,int> v = q.front();
        int v1 = v.first;
        int v2 = v.second;
        q.pop();
        for(auto i : directions){
            int c1 = i.first + v1;
            int c2 = i.second + v2;
            if(!(c1 >=0 and c2 >=0 and c1 < n and c2 < m)) continue;
            if(vis[c1][c2]) continue;
            q.push({c1, c2});
            lev[c1][c2] = lev[v1][v2] + 1;
            vis[c1][c2] = 1;
            ans = max(ans, lev[c1][c2]);
        }
    }
    return ans;
}
```



```
void clear(){
    for(int i =0; i < n; ++i){
        for(int j = 0; j < m; ++j){
            vis[i][j] = 0;
            lev[i][j] = INF;
        }
    }
}
```



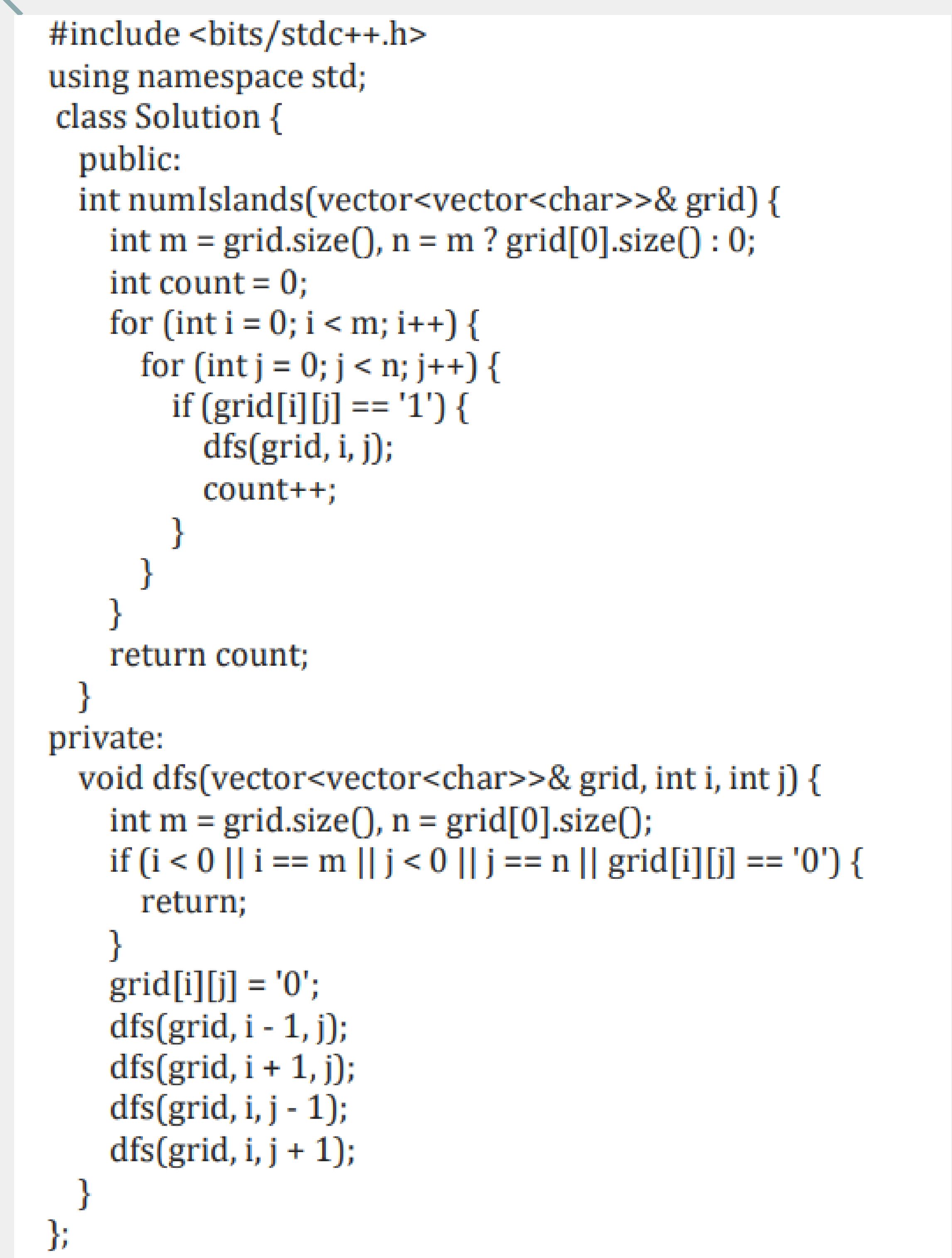
```
int main(){
    int t;
    cin >> t;
  while(t--){
      cin >> n >> m;
      clear();
      for(int i = 0; i <n; ++i){
          for(int j=0;j < m; ++j){
              cin >> graph[i][j];
          }
      }
      cout << bfs() << endl;
  }
}
```



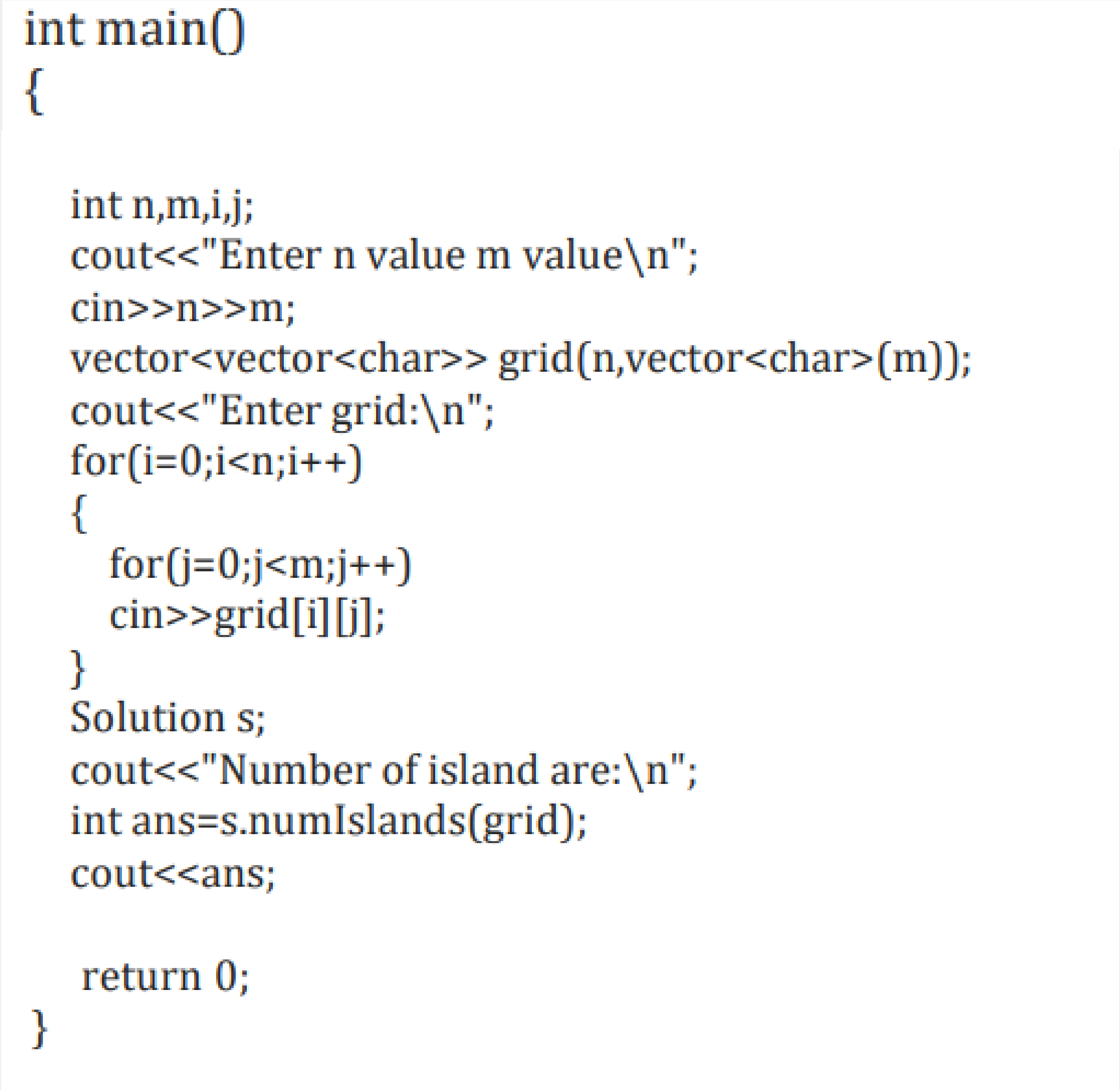
```
Input          Output

3                0
2                1
1 1              2
1 1
2 2
1 1
1 2
3 4
1 2 1 2
1 1 1 2
1 1 2 2
```

APPLICATION OF DFS- NUMBER OF ISLANDS

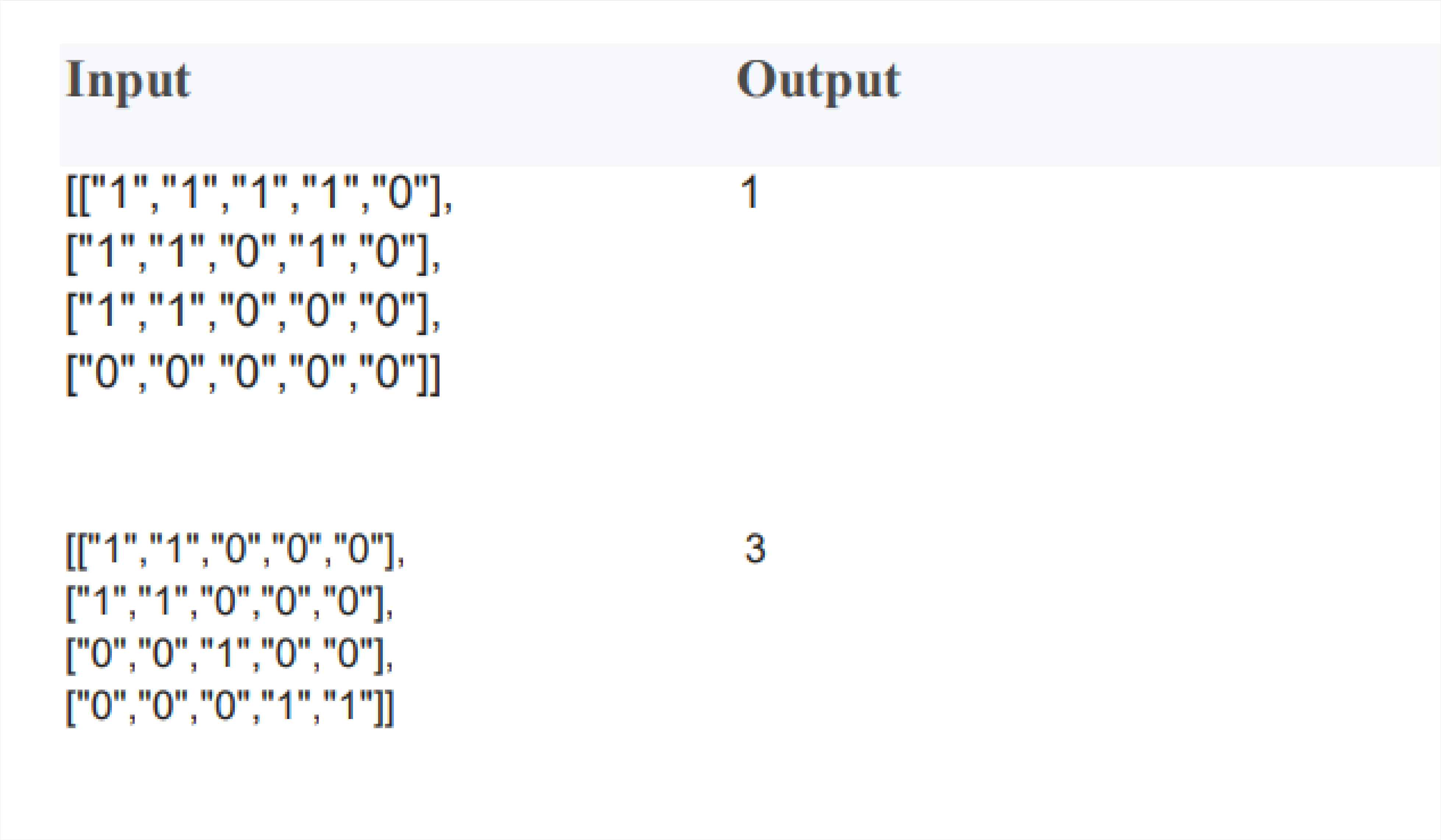
Problem Statement: Given an m x n 2D binary grid which represents a map of '1's (land) and '0's (water), return the number of islands. An island, is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water



```
#include <bits/stdc++.h>
using namespace std;
 class Solution {
  public:
  int numIslands(vector<vector<char>>& grid) {
    int m = grid.size(), n = m ? grid[0].size() : 0;
    int count = 0;
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        if (grid[i][j] == '1') {
          dfs(grid, i, j);
          count++;
        }
      }
    }
    return count;
  }
private:
  void dfs(vector<vector<char>>& grid, int i, int j) {
    int m = grid.size(), n = grid[0].size();
    if (i < 0 || i == m || j < 0 || j == n || grid[i][j] == '0') {
      return;
    }
    grid[i][j] = '0';
    dfs(grid, i - 1, j);
    dfs(grid, i + 1, j);
    dfs(grid, i, j - 1);
    dfs(grid, i, j + 1);
  }
};
```



```
int main()
{

   int n,m,i,j;
   cout<<"Enter n value m value\n";
   cin>>n>>m;
   vector<vector<char>> grid(n,vector<char>(m));
   cout<<"Enter grid:\n";
   for(i=0;i<n;i++)
   {
     for(j=0;j<m;j++)
     cin>>grid[i][j];
   }
   Solution s;
   cout<<"Number of island are:\n";
   int ans=s.numIslands(grid);
   cout<<ans;

    return 0;
}
```



Input	Output
[["1","1","1","1","0"], ["1","1","0","1","0"], ["1","1","0","0","0"], ["0","0","0","0","0"]]	1
[["1","1","0","0","0"], ["1","1","0","0","0"], ["0","0","1","0","0"], ["0","0","0","1","1"]]	3

THANK YOU

PROJECT BY:

L. Rahul - 21071A3241
P. Bala Abhiram - 21071A3254
S. Lokeswar - 21071A3261
R. Vathsav Varma - 22075A3206
U. Ajay - 22075A3207