Breadth First Search:

```
class Solution {
  public:
    // Function to return Breadth First Traversal of given graph.
    vector<int> bfsOfGraph(int V, vector<int> adj[]) {
    queue<int>q;
    q.push(0);
    vector<int>ans;
    vector<int>visi(V,0);
    visi[0]=1;
    while(!q.empty()){
        int node=q.front();
        q.pop();
        visi[node]=1;
        ans.push_back(node);
      for(int x:adj[node]){
          if(!visi[x]){
              q.push(x);
              visi[x]=1;
          }
      }
    }
    return ans;
    }
};
```

No. of Provinces:

```
class Solution {
public:
    void bfs(int node, vector<int> adjList[],vector<int>&visi) {
    queue<int>q;
    q.push(node);
    visi[node]=1;
    while(!q.empty()){
        int node=q.front();
        q.pop();
      for(int x:adjList[node]){
          if(!visi[x]){
              q.push(x);
              visi[x]=1;
          }
      }
    }
    int findCircleNum(vector<vector<int>>& isConnected) {
```

```
        int n=isConnected.size();
        vector<int>adjList[n+1];
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(isConnected[i][j]==1){
                    adjList[i+1].push_back(j+1);
                    // adjList[j].push_back(i);
                }
            }
        }

        vector<int>visi(n+1,0);
        int count=0;
        for(int node=1;node<=n;node++){
            if(visi[node]==0){/
                count++;
                cout<<count;
                bfs(node,adjList,visi);
            }
        }
        return count;
    }
};
```

No. of Islands:

```
class Solution {
public:
    void bfs(int i,int
j,vector<vector<char>>&grid,vector<vector<int>>&visi,int n,int m){
        queue<pair<int,int>>q;
        q.push({i,j});
        visi[i][j]=1;
        while(!q.empty()){
            pair<int,int>p=q.front();
            q.pop();
            int x=p.first;
            int y=p.second;

            int dx[4]={0,0,-1,1};
            int dy[4]={1,-1,0,0};
            for(int t=0;t<4;t++){
                int newX=x+dx[t];
                int newY=y+dy[t];
```

```
if(newX>=0&&newX<n&&newY>=0&&newY<m&&visi[newX][newY]==0&&grid[newX][newY]=='
1'){
                    q.push({newX,newY});
                    visi[newX][newY]=1;
                }
            }
        }
    }
    int numIslands(vector<vector<char>>& grid) {
        int n=grid.size();
        int m=grid[0].size();
        vector<vector<int>>visi(n,vector<int>(m,0));

        int count=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(visi[i][j]==0&&grid[i][j]=='1'){
                    count++;
                    bfs(i,j,grid,visi,n,m);
                }
            }
        }
        return count;
    }
};
```

Max Area of island:

```
class Solution {
public:
    int bfs(int i,int j,vector<vector<int>>&grid,vector<vector<int>>&visi,int
n,int m){
        queue<pair<int,int>>q;
        q.push({i,j});
        visi[i][j]=1;
        int count=0;
        while(!q.empty()){
            pair<int,int>p=q.front();
            q.pop();
            int x=p.first;
            int y=p.second;
            count++;
            int dx[4]={0,0,-1,1};
            int dy[4]={1,-1,0,0};
            for(int t=0;t<4;t++){
```

```
                int newX=x+dx[t];
                int newY=y+dy[t];

if(newX>=0&&newX<n&&newY>=0&&newY<m&&visi[newX][newY]==0&&grid[newX][newY]==1
){
                    q.push({newX,newY});
                    visi[newX][newY]=1;
                }
            }
        }
        return count;
    }
    int maxAreaOfIsland(vector<vector<int>>& grid) {
         int n=grid.size();
        int m=grid[0].size();
        vector<vector<int>>visi(n,vector<int>(m,0));

        int maxCount=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(visi[i][j]==0&&grid[i][j]==1){
                    int count=bfs(i,j,grid,visi,n,m);
                    maxCount=max(maxCount,count);
                }
            }
        }
        return maxCount;
    }
};
```