BFS + TopoLogical Sort\bfs.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;

#define N (int)2e6 + 5  // Define the maximum number of nodes

vector<int> vec[N];  // Adjacency list to store the graph

bool vis[N];  // Visited array to track if a node is visited
int lev[N];   // Array to store the level (distance) of each node from the source

// BFS function to perform breadth-first search
void bfs(int startNode) {
    // Queue to store the nodes for BFS
    queue<int> que;

    // Start BFS from the source node
    que.push(startNode);
    lev[startNode] = 0;   // Level of the source node is 0
    vis[startNode] = 1;    // Mark the source node as visited

    // While the queue is not empty, process each node
    while (!que.empty()) {
        int currentNode = que.front();  // Get the front node from the queue
        cout << currentNode << endl;    // Print the current node (for debugging purposes)
        que.pop();  // Remove the node from the queue

        // Traverse all the neighbors of the current node
        for (int i = 0; i < vec[currentNode].size(); i++) {
            int neighbor = vec[currentNode][i];

            // If the neighbor is not visited, visit it
            if (vis[neighbor] == 0) {
                que.push(neighbor);  // Add the neighbor to the queue
                vis[neighbor] = 1;   // Mark the neighbor as visited
                lev[neighbor] = lev[currentNode] + 1;  // Set the level of the neighbor
            }
        }
    }
}

int main() {
    int n, m;

    // Input number of nodes and edges
    cin >> n >> m;

    // Initialize the visited array to 0 (unvisited)
    for (int i = 1; i <= n; i++) {
        vis[i] = 0;
    }

    // Input the edges and construct the adjacency list
    for (int i = 1; i <= m; i++) {
        int a, b;
        cin >> a >> b;

        // Add edges in both directions (since the graph is undirected)
        vec[a].push_back(b);
        vec[b].push_back(a);
    }

    // Perform BFS starting from node 1
    bfs(1);

    return 0;
}

```