codes\Practical2nd.cpp

```cpp
#include <iostream>
#include <vector>
#include <queue>

using namespace std;
#define vi vector<int>
#define vvi vector<vi>
#define pii pair<int, int>
#define vii vector<pii>
#define rep(i, a, b) for (int i = a; (i < b); i++)
#define ff first
#define ss second
#define setbits(x) builtin_popcount(x)
const int N = 1e5 + 2, MOD = 1e9 + 7;
// class graph{
// public:
// vector<vector<int>> adjm(n+1,vector<int>ex(n+1,0));
// };
void DFS(int node, vector<vector<int>>& adjm, vector<bool>& visited) {
    visited[node] = true;
    cout << node << " ";

    for (int i = 1; i < adjm.size(); i++) {
        if (adjm[node][i] == 1 && !visited[i]) {
            DFS(i, adjm, visited);
        }
    }
}

void DFT(vector<vector<int>>& adjm) {

    int startNode;
    cout << "Enter the starting node for DFS: ";
    cin >> startNode;

    vector<bool> visited(adjm.size(), false);
    DFS(startNode, adjm, visited);
}

void BFT(vector<vector<int>>& adjm) {

    int startNode;
    cout << "Enter the starting node for BFS: ";
    cin >> startNode;

    vector<bool> visited(adjm.size(), false);
    queue<int> q;
    visited[startNode] = true;
    q.push(startNode);

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (int i = 1; i < adjm.size(); i++) {
            if (adjm[node][i] == 1 && !visited[i]) {
                visited[i] = true;
                q.push(i);
            }
        }
    }
}

signed main()
{
    int n, m, z; // here {n} is number of nodes and {m} is nuuumber of edges
    cout << "Enter number of Nodes and Edges respectively" << endl;
    cin >> n >> m;
    // Code of adjecency matrix start
    vvi adjm(n + 1, vi(n + 1, 0));
    // cout<<"Now enter the Edges"<<endl;
    cout << "Enter your type" << endl
         << "1.For directed graph [1]" << endl
         << "2.For undirected graph [0]"<<endl;
    cin >> z;
    if (z == 1)
    {
        rep(i, 0, m)
        {
            int x, y;
            cout << "Enter Edge" << endl;
            cin >> x >> y;
            cout << endl;
            adjm[x][y] = 1;
        }
    }
    else if (z == 0)
    {
        rep(i, 0, m)
        {
            int x, y;
            cout << "Enter Edge" << endl;
            cin >> x >> y;
            cout << endl;
            adjm[x][y] = 1;
            adjm[y][x] = 1;
        }
    }
    else
    {
        cout << "Invalid entry";
    }
    cout << "Your adjecency Matrix is given as:" << endl;
    if (z == 1)
    {

        cout << "Your directed graph is:";
        rep(i, 0, n + 1)
        {
            rep(j, 1, n + 1)
            {
                cout << adjm[i][j] << " ";
            }
            cout << endl;
        }
    }
    else
    {
        cout << "Your Undirected graph is:";
        rep(i, 0, n + 1)
        {
            rep(j, 1, n + 1)
            {
                cout << adjm[i][j] << " ";
            }
            cout << endl;
        }
    }

    cout << "Choose the traversal method:" << endl;
    cout << "1. Depth First Traversal (DFS)" << endl;
    cout << "2. Breadth First Traversal (BFS)" << endl;
    int choice;
    cin >> choice;

    if (choice == 1) {
    DFT(adjm);

    } else if (choice == 2) {
    BFT(adjm);

    } else {
        cout << "Invalid choice!" << endl;
    }

    // Code of adjecency list start

    // vi adjl(n+1);
    // cout << "Enter your type" << endl
    //      << "1.For directed graph" << endl
    //      << "2.For undirected graph";
    // cin >> z;
    // if (z == 1)
    // {
    //     rep(i, 0, m)
    //     {
    //         int x, y;
    //         cout << "Enter Edge" << endl;
    //         cin >> x >> y;
    //         cout << endl;
    //         adjm[x].push_back(y);
    //     }
    // }
    // else if (z == 0)
    // {
    //     rep(i, 0, m)
    //     {
    //         int x, y;
    //         cout << "Enter Edge" << endl;
    //         cin >> x >> y;
    //         cout << endl;
    //         adjm[x].push_back(y);
    //         adjm[y].push_back(x);
    //     }
    // }
    // else
    // {
    //     cout << "Invalid entry";
    // }
    // cout << "Your adjecency Matrix is given as:" << endl;
    // if (z == 1)
    // {

    //     cout << "Your directed graph is:";
    //     rep(i, 0, n + 1)
    //     {
    //         cout << i << "->";
    //         for (int x : adjl[i].std::begin())
    //         {
    //             cout << x << " ";
    //         }
    //         cout << endl;
    //     }
    // }
    // else
    // {
    //     cout << "Your Undirected graph is:";
```

```cpp
    //     rep(i, 0, n + 1)
    //     {
    //         cout << i << "->";
    //         for (int x : adjl[i])
    //         {
    //             cout << x << " ";
    //         }
    //         cout << endl;
    //     }
    // }

    // Code of adjecency list end

    return 0;
}

```