

codes\Practical2nd.cpp

```
1  #include <iostream>
2  #include <vector>
3  #include <queue>
4
5  using namespace std;
6  #define vi vector<int>
7  #define vvi vector<vi>
8  #define pii pair<int, int>
9  #define vii vector<pii>
10 #define rep(i, a, b) for (int i = a; (i < b); i++)
11 #define ff first
12 #define ss second
13 #define setbits(x) builtin_popcount(x)
14 const int N = 1e5 + 2, MOD = 1e9 + 7;
15 // class graph{
16 // public:
17 // vector<vector<int>> adjm(n+1,vector<int>ex(n+1,0));
18 // };
19 void DFS(int node, vector<vector<int>>& adjm, vector<bool>& visited) {
20     visited[node] = true;
21     cout << node << " ";
22
23     for (int i = 1; i < adjm.size(); i++) {
24         if (adjm[node][i] == 1 && !visited[i]) {
25             DFS(i, adjm, visited);
26         }
27     }
28 }
29
30 void DFT(vector<vector<int>>& adjm) {
31
32     int startNode;
33     cout << "Enter the starting node for DFS: ";
34     cin >> startNode;
35
36     vector<bool> visited(adjm.size(), false);
37     DFS(startNode, adjm, visited);
38 }
39
40 void BFT(vector<vector<int>>& adjm) {
41
42     int startNode;
43     cout << "Enter the starting node for BFS: ";
44     cin >> startNode;
45
46     vector<bool> visited(adjm.size(), false);
47     queue<int> q;
48     visited[startNode] = true;
```

```

49     q.push(startNode);
50
51     while (!q.empty()) {
52         int node = q.front();
53         q.pop();
54         cout << node << " ";
55
56         for (int i = 1; i < adjm.size(); i++) {
57             if (adjm[node][i] == 1 && !visited[i]) {
58                 visited[i] = true;
59                 q.push(i);
60             }
61         }
62     }
63 }
64
65 signed main()
66 {
67     int n, m, z; // here {n} is number of nodes and {m} is nuuumber of edges
68     cout << "Enter number of Nodes and Edges respectively" << endl;
69     cin >> n >> m;
70     // Code of adjecency matrix start
71     vvi adjm(n + 1, vi(n + 1, 0));
72     // cout<<"Now enter the Edges"<<endl;
73     cout << "Enter your type" << endl
74         << "1.For directed graph [1]" << endl
75         << "2.For undirected graph [0]"<<endl;
76     cin >> z;
77     if (z == 1)
78     {
79         rep(i, 0, m)
80         {
81             int x, y;
82             cout << "Enter Edge" << endl;
83             cin >> x >> y;
84             cout << endl;
85             adjm[x][y] = 1;
86         }
87     }
88     else if (z == 0)
89     {
90         rep(i, 0, m)
91         {
92             int x, y;
93             cout << "Enter Edge" << endl;
94             cin >> x >> y;
95             cout << endl;
96             adjm[x][y] = 1;
97             adjm[y][x] = 1;
98         }

```

```

99     }
100    else
101    {
102        cout << "Invalid entry";
103    }
104    cout << "Your adjacency Matrix is given as:" << endl;
105    if (z == 1)
106    {
107
108        cout << "Your directed graph is:";
109        rep(i, 0, n + 1)
110        {
111            rep(j, 1, n + 1)
112            {
113                cout << adjm[i][j] << " ";
114            }
115            cout << endl;
116        }
117    }
118    else
119    {
120        cout << "Your Undirected graph is:";
121        rep(i, 0, n + 1)
122        {
123            rep(j, 1, n + 1)
124            {
125                cout << adjm[i][j] << " ";
126            }
127            cout << endl;
128        }
129    }
130
131    cout << "Choose the traversal method:" << endl;
132    cout << "1. Depth First Traversal (DFS)" << endl;
133    cout << "2. Breadth First Traversal (BFS)" << endl;
134    int choice;
135    cin >> choice;
136
137    if (choice == 1) {
138        DFT(adjm);
139
140    } else if (choice == 2) {
141        BFT(adjm);
142
143    } else {
144        cout << "Invalid choice!" << endl;
145    }
146
147    // Code of adjacency list start
148

```

```

149 // vi adjl(n+1);
150 // cout << "Enter your type" << endl
151 //     << "1.For directed graph" << endl
152 //     << "2.For undirected graph";
153 // cin >> z;
154 // if (z == 1)
155 // {
156 //     rep(i, 0, m)
157 //     {
158 //         int x, y;
159 //         cout << "Enter Edge" << endl;
160 //         cin >> x >> y;
161 //         cout << endl;
162 //         adjm[x].push_back(y);
163 //     }
164 // }
165 // else if (z == 0)
166 // {
167 //     rep(i, 0, m)
168 //     {
169 //         int x, y;
170 //         cout << "Enter Edge" << endl;
171 //         cin >> x >> y;
172 //         cout << endl;
173 //         adjm[x].push_back(y);
174 //         adjm[y].push_back(x);
175 //     }
176 // }
177 // else
178 // {
179 //     cout << "Invalid entry";
180 // }
181 // cout << "Your adjacency Matrix is given as:" << endl;
182 // if (z == 1)
183 // {
184
185 //     cout << "Your directed graph is:";
186 //     rep(i, 0, n + 1)
187 //     {
188 //         cout << i << "->";
189 //         for (int x : adjl[i].std::begin())
190 //         {
191 //             cout << x << " ";
192 //         }
193 //         cout << endl;
194 //     }
195 // }
196 // else
197 // {
198 //     cout << "Your Undirected graph is:";

```

```
199     //     rep(i, 0, n + 1)
200     //     {
201     //         cout << i << "->";
202     //         for (int x : adjl[i])
203     //             {
204     //                 cout << x << " ";
205     //             }
206     //         cout << endl;
207     //     }
208     // }
209
210     // Code of adjecency list end
211
212     return 0;
213 }
214
```