G. H. RAISONI COLLEGE OF ENGG., NAGPUR
(An Autonomous Institute under UGC Act 1956)
Department of Computer Science & Engg.

Date: 20/08/2020

Practical Subject: Design and Analysis of Algorithms
Session: 2020-21

Student Details:

Roll Number	58
Name	Shivam Tawari
Semester	3
Section	A
Branch	Artificial Intelligence

Practical Details: Practical Number- 9

Practical Aim	To implement and analyze time complexity of Graph Traversing Algorithm.
Theory & Algorithm	**Theory:** ***Graph Traversal Algorithms:*** Graph traversal means visiting every vertex and edge exactly once in a well-defined order. While using certain graph algorithms, you must ensure that each vertex of the graph is visited exactly once. The order in which the vertices are visited are important and may depend upon the algorithm or question that you are solving. During a traversal, it is important that you track which vertices have been visited. The most common way of tracking vertices is to mark them.

	Breadth First search: BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes. ***Algorithm:*** Step 1: START Step 2: Take an Empty Queue. Step 3: Select a starting node (visiting a node) and insert it into the Queue. Step 4: Provided that the Queue is not empty, extract the node from the Queue and insert its child nodes (exploring a node) into the Queue. Step 5: Print the extracted node. Step 6: STOP ***Depth First Search:*** The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking. Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected. ***Algorithm:*** Step 1: START

	Step 2: Pick a starting node and push all its adjacent nodes into a stack. Step 3: Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack. Step 4: Repeat this process until the stack is empty. Step 5: STOP
Complexity	***Breadth First Search:*** Time complexity: O(V + E), where V is the number of vertices and E is the number of edges in the graph. ***Depth First search:*** Time complexity: O(V + E), where V is the number of vertices and E is the number of edges in the graph.

Program	**Breadth First Search:**
	main.c
	`#include<stdio.h>`
	`int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;`
	
	`void bfs(int v) {`
	`for(i = 1; i <= n; i++)`
	
	`if(a[v][i] && !visited[i])`
	`q[++r] = i;`
	`if(f <= r) {`
	`visited[q[f]] = 1;`
	`bfs(q[f++]);`
	`}`
	`}`
	
	`void main() {`
	`int v;`
	`printf("\n Name: Shivam Tawari");`
	`printf("\n Roll no: A-58 \n Semester: 3");`
	`printf("\n Enter the number of vertices:");`
	`scanf("%d", &n);`
	
	`for(i=1; i <= n; i++) {`
	`q[i] = 0;`
	`visited[i] = 0;`
	`}`
	
	`printf("\n Enter graph data in matrix form:\n");`
	`for(i=1; i<=n; i++) {`
	`for(j=1;j<=n;j++) {`

```
            scanf("%d", &a[i][j]);
        }
    }

    printf("\n Enter the starting vertex:");
    scanf("%d", &v);
    bfs(v);
    printf("\n The node which are reachable are:\n");

    for(i=1; i <= n; i++) {
        if(visited[i])
            printf("%d\t", i);
        else {
            printf("\n Bfs is not possible. Not all nodes are reachable");
            break;
        }
    }
}
```

Depth First Search:

main.cpp

```
#include "iostream"
#include "vector"
using namespace std;

vector<vector<int>> adj;
int n;
vector<bool> visited;
vector<int> solution;

void dfs(int v) {
    visited[v] = true;
    for (int u=0; u<n; u++) {
        if (adj[v][u] == 1 && (!visited[u]))    {
            solution.push_back(u);
            dfs(u);
        }
    }
}

int main(int argc, char const *argv[]) {

    vector<int> vec;
    int temp;

    cout << "\n Name: Shivam Tawari";
    cout << "\n Roll no: A-58 Section: A";

    cout << "\n Enter Total Nodes: ";
    cin >> n;
```

```
    cout << " Enter Adjacency Matrix: \n";
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            cin >> temp;
            vec.push_back(temp);
        }
        adj.push_back(vec);
        vec.clear();
        visited.push_back(false);
    }

    cout << " Enter Source Node: ";
    cin >> temp;
    solution.push_back(temp);
    dfs(temp);

    cout << " DFS Solution: ";
    for (auto x: solution)  {
        cout << x << " ";
    }

    return 0;
}
```

Output

Breadth First Search:

```
Name: Shivam Tawari
Roll no: A-58
Semester: 3
Enter the number of vertices:2

Enter graph data in matrix form:
1
2
1
1


Enter the starting vertex:2

The node which are reachable are:
1       2
```

Depth First Search:

```
Name: Shivam Tawari
Roll no: A-58 Section: A
Enter Total Nodes: 3
Enter Adjacency Matrix:
1
0
1
1
1
0
1
0
0
Enter Source Node: 1
DFS Solution: 1 0 2
```