**Experiment 9**

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C013 | Name: Ashmit Jain |
| Class: B | Batch: B1 |
| Date of Experiment: | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task1:**

#include <iostream>

#include <queue>

#include <stack>

#define MAX\_VERTICES 100 // Define the maximum number of vertices

using namespace std;

class Graph {

int V; // Number of vertices

int adj[MAX\_VERTICES][MAX\_VERTICES]; // Adjacency matrix representation

bool visited[MAX\_VERTICES]; // Visited array for traversals

public:

Graph(int V); // Constructor

void addEdge(int v, int w); // Function to add an edge to the graph

void BFS(int start); // Function to perform BFS

void DFS(int start); // Function to perform DFS

void DFSUtil(int v); // Utility function for DFS

};

// Constructor

Graph::Graph(int V) {

this->V = V;

// Initialize the adjacency matrix and visited array

for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

adj[i][j] = 0; // No edges initially

}

visited[i] = false; // Not visited initially

}

}

// Function to add an edge to the graph

void Graph::addEdge(int v, int w) {

adj[v][w] = 1; // Add edge v -> w

adj[w][v] = 1; // Add edge w -> v (undirected graph)

}

// Function to perform BFS

void Graph::BFS(int start) {

queue<int> q;

fill(begin(visited), begin(visited) + V, false); // Reset visited array

visited[start] = true;

q.push(start);

cout << "BFS Traversal: ";

while (!q.empty()) {

int v = q.front();

cout << v << " ";

q.pop();

for (int i = 0; i < V; i++) {

if (adj[v][i] == 1 && !visited[i]) { // If there's an edge and not visited

visited[i] = true;

q.push(i);

}

}

}

cout << endl;

}

// Utility function for DFS

void Graph::DFSUtil(int v) {

visited[v] = true;

cout << v << " ";

for (int i = 0; i < V; i++) {

if (adj[v][i] == 1 && !visited[i]) {

DFSUtil(i);

}

}

}

// Function to perform DFS

void Graph::DFS(int start) {

fill(begin(visited), begin(visited) + V, false); // Reset visited array

cout << "DFS Traversal: ";

DFSUtil(start);

cout << endl;

}

int main() {

Graph g(5); // Create a graph with 5 vertices

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(1, 4);

g.addEdge(2, 4);

g.BFS(0); // Perform BFS starting from vertex 0

g.DFS(0); // Perform DFS starting from vertex 0

return 0;

}

**B.2 Input and Output:**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

***Input;***

g.addEdge(0, 1);

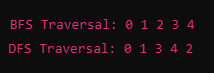
g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(1, 4);

g.addEdge(2, 4);

***Ouput:***

******

**B.3 Observations and learning [w.r.t. all tasks]:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

 **Traversal Order**:

* **BFS** explores layer by layer, while **DFS** goes deep along a branch before backtracking.

 **Output Differences**:

* The sequence of visited nodes varies between BFS and DFS due to their distinct exploration strategies.

 **Pathfinding**:

* BFS guarantees the shortest path in unweighted graphs, whereas DFS does not.

 **Memory Usage**:

* BFS generally uses more memory (queue) than DFS (stack or recursion), especially in dense graphs.

 **Use Cases**:

* BFS is suited for shortest path and level-order tasks; DFS is better for complete explorations, topological sorting, and cycle detection.

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

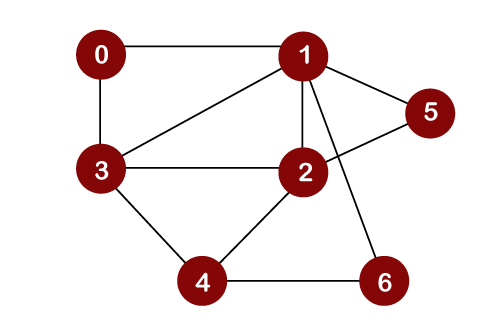
*BFS and DFS are crucial for graph traversal, each with unique strengths. BFS is ideal for shortest path problems, while DFS is useful for deep explorations. Choosing between them depends on the specific needs of the problem, such as path length requirements or memory efficiency.*

4o mini

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

Give BFS and DFS traversals for the following graph



### Breadth-First Search (BFS)

BFS uses a queue and explores each level of neighbors before going deeper. The BFS traversal of the graph starting from Node 0 would be:

1. Start from Node 0.
2. Visit Nodes connected to 0: [1, 3].
3. Visit Nodes connected to 1 and 3: [2, 4].
4. Visit Nodes connected to 2 and 4: [5, 6].

**BFS Order:** 0 -> 1 -> 3 -> 2 -> 4 -> 5 -> 6

### Depth-First Search (DFS)

DFS uses a stack (or recursion) and goes as deep as possible along each branch before backtracking. The DFS traversal of the graph starting from Node 0 would be:

1. Start from Node 0.
2. Go to Node 1.
3. Go to Node 2.
4. Go to Node 5.
5. Backtrack to Node 2, then go to Node 3.
6. Go to Node 4.
7. Go to Node 6.

**DFS Order:** 0 -> 1 -> 2 -> 5 -> 3 -> 4 -> 6