
🔵 AI / DS / DBMS Practical – BFS & DFS Traversal

🟣 1. THEORY (Explain this to External Examiner)

✅ What is a Graph?

A **graph** is a data structure used to represent **connections** between objects.

A graph has:

- **Nodes / Vertices** → points
- **Edges** → links between points

Example:
5 → 3 and 5 → 7 means 5 is connected to 3 and 7.

✔️ Where graphs are used?

- Google Maps
- Facebook / Instagram friend network
- Web search engine crawlers
- AI search algorithms
- Computer networks
- Recommendation systems

🟢 2. BFS (Breadth First Search)

✅ Definition

BFS is a graph traversal algorithm that:

- Visits nodes **level by level**
- Starts from a node and explores all nearby nodes first
- Uses a **Queue (FIFO)**

🎯 Example

Start from 5 → visit 3 and 7 first → then neighbors of 3 → then neighbors of 7.

🔵 3. DFS (Depth First Search)

✅ Definition

DFS is a graph traversal algorithm that:

- Goes **deep into a path first**
- When no further nodes, it goes back (backtracking)
- Uses **Recursion (stack)**

🎯 Example

Start from 5 → go to 3 → go to 2 → go to 4 ... etc.

🔥 4. BFS vs DFS — Differences

BFS	DFS
Uses **Queue**	Uses **Stack / Recursion**
Level-wise traversal	Depth-wise traversal
Finds **shortest path**	Does NOT give shortest path
Uses more memory	Uses less memory
Good for wide graphs	Good for deep graphs

🟡 5. FULL PYTHON CODE (BFS + DFS)

📌 Write this exactly in exam

```
# ---------------- BFS CODE ----------------


graph = {
  '5': ['3', '7'],
  '3': ['2', '4'],
  '7': ['8'],
  '2': [],
  '4': ['8'],
```

```
    '8': []
}

visited = []
queue = []

def BFS(graph, visited, node):
    visited.append(node)
    queue.append(node)

    while queue:
        m = queue.pop(0)
        print(m, end=" ")

        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

print("Following is BFS traversal:")
BFS(graph, visited, '5')



# ---------------- DFS CODE ----------------

graph = {
    '5': ['3', '7'],
    '3': ['2', '4'],
```

```
    '7': ['8'],
    '2': [],
    '4': [],
    '8': []
}

visited = set()

def DFS(visited, graph, node):
    if node not in visited:
        print(node)
        visited.add(node)
        for neighbour in graph[node]:
            DFS(visited, graph, neighbour)

print("\nFollowing is DFS traversal:")
DFS(visited, graph, '5')
```

🔵 6. LINE-BY-LINE EXPLANATION (VERY EASY LANGUAGE)

🟩 BFS Code Explanation

1. Creating a graph

graph = { '5': ['3', '7'], ... }

- Graph stored as a **dictionary**
- Key = node
- Value = list of its neighbours

---

2. Creating empty visited list

visited = []

Stores nodes we already visited, so we don't visit again.

---

3. Queue for BFS

queue = []

- BFS uses queue
- First in → first out

---

4. BFS function

def BFS(graph, visited, node):

We create a function for BFS.

---

5. Add starting node to visited & queue

visited.append(node)

queue.append(node)

- First node added to visited
- Then added to queue

---

6. While queue is not empty

while queue:

Loop continues until queue becomes empty.

---

7. Remove first element from queue

m = queue.pop(0)

- Removes and returns first element
- This is BFS behavior (FIFO)

---

8. Print current node

print(m, end=" ")

Output BFS traversal.

---

9. Visit neighbours

for neighbour in graph[m]:

Get all neighbour nodes of current node.

---

10. Check if neighbour is not visited

if neighbour not in visited:

Avoids re-visiting and infinite loop.

---

11. Add neighbour to visit

visited.append(neighbour)

queue.append(neighbour)

This ensures BFS continues level-wise.

🟥 **DFS Code Explanation**

---

1. Visited set

visited = set()

Set used because it avoids duplicates automatically.

---

2. DFS function

def DFS(visited, graph, node):

Defines DFS.

---

3. Check if node already visited

if node not in visited:

Prevent re-visiting.

---

4. Print node

print(node)

Displays traversal.

---

5. Add node to visited set

visited.add(node)

---

6. Recursively visit neighbours

for neighbour in graph[node]:

DFS(visited, graph, neighbour)

- This is where DFS goes **deep first**
- Recursion automatically uses stack

🌟 7. VIVA QUESTIONS WITH FULL MARK ANSWERS

(Use these exact answers to impress external)

🔵 BASIC QUESTIONS

1. What is BFS?

Answer:
Breadth First Search is a graph traversal technique where:

- Nodes are visited **level by level**
- It uses **Queue**
- It checks all nearest neighbours first

Points:

1. Level-wise
2. Uses queue

3. Finds shortest path
4. Used in Maps and Networking

---

2. What is DFS?

Answer:
Depth First Search is a graph traversal algorithm where:

- We go **deep into one branch** before going to another
- Uses **Recursion/Stack**
- Good for exploring deep structures

Points:

1. Depth-wise
2. Uses recursion
3. Less memory
4. Used in maze solving

---

3. Why do we need visited[]?

Answer:
Because:

1. To avoid visiting same node again
2. To prevent infinite loops
3. To make traversal efficient
4. To ensure correct output

---

4. Why BFS uses Queue?

Answer:
Because BFS follows **FIFO (first in first out)**, and queue naturally represents this behaviour.

---

5. Why DFS uses Recursion?

Answer:
Because recursion naturally behaves like **stack**, and DFS needs stack to remember previous nodes.

---

6. Applications of BFS

1. Shortest path
2. Web crawling
3. Social network friend suggestion
4. GPS navigation
5. Broadcasting in Networks

---

7. Applications of DFS

1. Detecting cycles in graph
2. Solving mazes
3. Used in AI backtracking
4. Topological sorting
5. Path finding

---

8. Difference between BFS and DFS

BFS:

1. Uses queue
2. Level-wise
3. More memory
4. Good for shortest path
5. Wide graphs

DFS:

1. Uses recursion
2. Depth-wise
3. Less memory

4. No shortest path guarantee
5. Deep graphs

---

9. What is a graph?

A graph is a structure of **nodes** and **edges** used to show relationships.

---

10. What is traversal?

Traversal means visiting all nodes of a graph exactly once in a particular order.
