**AI LAB 6 – Breadth First Search (BFS)**

**Task 1 – BFS without Queue and without Node**

**Objective**

To understand and implement the **Breadth First Search (BFS)** algorithm using only lists — without using a queue data structure or Node class.

**Concept**

* **Breadth First Search (BFS)** is a graph traversal algorithm that explores all neighbors of a node **before moving to the next level**.
* Normally, BFS uses a **queue**, but in this task, we will simulate that behavior using **lists** only.
* This approach still works level-by-level, visiting all nodes on the same level before moving to the next.

**Algorithm**

1. Start from a given node and store it in a **list named level**.
2. Create an empty list visited to keep track of visited nodes.
3. While level is not empty:
   * Create a new empty list next\_level.
   * For each node in the current level:
     + If it’s not visited, print it and mark it visited.
     + Add its neighbors to next\_level.
   * Replace level with next\_level.
4. Repeat until all nodes are visited.

**Python Code**

# BFS Implementation without Queue and without Node

graph = {

'A': ['B', 'C'],

'B': ['D', 'E'],

'C': ['F'],

'D': [],

'E': ['G'],

'F': [],

'G': []

}

def bfs\_without\_queue(start):

visited = []

level = [start]

while level:

next\_level = []

for node in level:

if node not in visited:

print(node, end=" ")

visited.append(node)

for neighbor in graph[node]:

next\_level.append(neighbor)

level = next\_level

print("BFS Traversal (Without Queue and Node):")

bfs\_without\_queue('A')

**Output**

BFS Traversal (Without Queue and Node):

A B C D E F G

**Explanation**

* The algorithm starts at **A**.
* It first visits all neighbors of **A** → B, C.
* Then it moves to the next level → neighbors of B and C → D, E, F.
* Finally, it visits the next level → neighbor of E → G.

Thus, BFS visits nodes **level by level**.

**Task 2 – BFS using Queue and Node**

**Objective**

To implement **Breadth First Search (BFS)** using a **Queue** and **Node class**, showing how BFS works in a more realistic object-oriented approach.

**Concept**

* **BFS** uses a **queue (FIFO – First In First Out)** to visit nodes in the order they were discovered.
* A **Node** class stores the value and the list of its **neighbors**.
* Each node’s neighbors are enqueued and processed in order.

**Algorithm**

1. Create a class **Node** with two attributes:
   * value: stores data
   * neighbors: stores a list of connected nodes
2. Create a **queue** and add the starting node.
3. While the queue is not empty:
   * Remove the first node (front of the queue).
   * If it hasn’t been visited, print it and mark it as visited.
   * Add all its neighbors to the queue.
4. Continue until all nodes are visited.

**Python Code**

# BFS Implementation using Queue and Node

class Node:

def \_\_init\_\_(self, value):

self.value = value

self.neighbors = []

def bfs\_with\_queue(start\_node):

visited = []

queue = [start\_node]

while queue:

current = queue.pop(0)

if current not in visited:

print(current.value, end=" ")

visited.append(current)

for neighbor in current.neighbors:

queue.append(neighbor)

# Create nodes

A = Node('A')

B = Node('B')

C = Node('C')

D = Node('D')

E = Node('E')

F = Node('F')

G = Node('G')

# Define connections (edges)

A.neighbors = [B, C]

B.neighbors = [D, E]

C.neighbors = [F]

E.neighbors = [G]

# Run BFS

print("BFS Traversal (With Queue and Node):")

bfs\_with\_queue(A)

**Output**

BFS Traversal (With Queue and Node):

A B C D E F G

**Explanation**

* The BFS starts from node **A**.
* It enqueues all its connected neighbors.
* The queue ensures that nodes are visited **in the same order they are discovered**.
* As each node is dequeued, its neighbors are added to the queue.
* The traversal continues until no nodes remain in the queue.

.