-------------------------------------------

Python Code Analysis and Viva Questions

-------------------------------------------

Practical Code: Breadth First Search (BFS) Recursive Traversal with Target Search

-------------------------------------------

Line by Line Explanation

-------------------------------------------

1. from collections import defaultdict, deque

- Imports defaultdict (for adjacency list) and deque (double-ended queue for BFS).

2. def bfs\_recursive(graph, queue, visited, target, found):

- Defines the recursive BFS function.

- Parameters:

graph: the graph as adjacency list.

queue: deque of (vertex, path) pairs.

visited: set of visited nodes.

target: the node to search for.

found: a mutable list flag indicating if the target is found.

3. if not queue or found[0]:

return

- Base condition: if queue is empty or target found, stop recursion.

4. vertex, path = queue.popleft()

- Remove the first element (FIFO) from queue.

5. print("Current path:", " -> ".join(path))

- Display the current traversal path.

6. if vertex == target:

print("BFS traversal:", " -> ".join(path))

print(f"Target node '{target}' found!")

found[0] = True

return

- If current vertex is the target, print the path and mark found.

7. visited.add(vertex)

- Mark the vertex as visited.

8. for neighbor in graph[vertex]:

if neighbor not in visited:

queue.append((neighbor, path + [neighbor]))

- For each unvisited neighbor, add it to the queue with the updated path.

9. bfs\_recursive(graph, queue, visited, target, found)

- Continue recursive BFS on the updated queue.

10. graph = defaultdict(set)

- Initialize the graph as an adjacency list.

11. Print student details.

12. num\_vertices = int(input("Enter the number of vertices: "))

- Input the number of vertices.

13. for \_ in range(num\_vertices):

vertex = input("Enter a vertex: ")

neighbors = input(f"Enter neighbors of {vertex} (separated by spaces): ").split()

graph[vertex].update(neighbors)

- Input vertex names and their neighbors.

14. Print the constructed graph.

15. start\_vertex = input("Enter the starting vertex: ")

target\_vertex = input("Enter the target vertex: ")

- Input start and target nodes.

16. Check if start or target exists in the graph.

17. Initialize visited set, queue (with start), and found flag.

18. Call bfs\_recursive().

19. If target not found, display message.

-------------------------------------------

Possible Viva / External Questions and Answers

-------------------------------------------

Q1: What is BFS?

A: Breadth First Search is a graph traversal algorithm that explores all neighbors of a node before moving to the next level.

Q2: What data structures are used in BFS?

A: A queue for managing nodes, a visited set or array, and optionally a parent/path list.

Q3: What is the time complexity of BFS?

A: O(V + E), where V = number of vertices and E = number of edges.

Q4: What is the space complexity of BFS?

A: O(V) for the visited set and the queue.

Q5: Why use deque for BFS?

A: Deque allows efficient appending and popping from both ends; here we use it as a FIFO queue.

Q6: Why use a mutable 'found' flag?

A: To share state across recursive calls, since booleans are immutable in Python.

Q7: What does the queue store here?

A: Tuples of (current vertex, path taken so far).

Q8: What are the applications of BFS?

A: Shortest path in unweighted graphs, connected components, cycle detection, level-order traversal.

Q9: What happens if the graph has cycles?

A: Without a visited set, BFS would revisit nodes indefinitely. The visited set prevents this.

Q10: How can we modify this code to find all shortest paths?

A: Track multiple paths and avoid stopping after finding the first target.

Q11: Difference between BFS and DFS?

A: BFS explores level by level using a queue; DFS explores depth-first using a stack or recursion.

Q12: What type of graph is this?

A: Directed or undirected, represented as an adjacency list.

Q13: Why is recursion less common in BFS?

A: BFS is naturally iterative due to the queue; recursion usually fits DFS better.

Q14: What are common issues in recursive BFS?

A: Stack overflow on deep queues, inefficient space usage if graph is large.

Q15: How does the program handle incorrect vertices?

A: It checks if the start or target exists before running BFS.

-------------------------------------------

Additional Syllabus-Based Questions (TE Computer Engineering, SPPU LP2)

-------------------------------------------

Q16: What is an adjacency list?

A: A data structure where each vertex maps to a list or set of its neighbors.

Q17: Explain BFS vs Dijkstra’s algorithm.

A: BFS finds the shortest path in unweighted graphs; Dijkstra’s finds the shortest path in weighted graphs.

Q18: What is the difference between tree traversal and graph traversal?

A: Tree traversal assumes no cycles and connectedness; graph traversal must handle cycles and disconnected components.

Q19: What is the role of LP2 in curriculum?

A: To provide hands-on implementation of algorithms, data structures, and problem-solving strategies.

Q20: What are common pitfalls in implementing BFS?

A: Not marking visited nodes, mishandling the queue, inefficient memory management.

-------------------------------------------

This document explains the BFS practical code, prepares you for external viva, and covers related syllabus concepts.