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

Python Code Analysis and Viva Questions

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

Practical Code: Depth First Search (DFS) Recursive Traversal with Target Search

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

Line by Line Explanation

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

1. from collections import defaultdict

- Imports defaultdict, a special dictionary type that creates default values for missing keys.

2. def dfs\_recursive(graph, vertex, target, visited, path, found):

- Defines the recursive DFS function.

- Parameters:

graph: the graph represented as adjacency list.

vertex: the current node being visited.

target: the node we are searching for.

visited: a set to keep track of visited nodes.

path: the current path being followed.

found: a mutable list with a flag (True/False) indicating if the target is found.

3. if found[0]:

return

- If the target has already been found, exit early.

4. visited.add(vertex)

- Mark the current vertex as visited.

5. path.append(vertex)

- Add the current vertex to the traversal path.

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

- Display the current path during traversal.

7. if vertex == target:

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

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

found[0] = True

return

- If we reach the target, print the traversal path and set the flag.

8. for neighbor in graph[vertex]:

if neighbor not in visited:

dfs\_recursive(graph, neighbor, target, visited, path, found)

- For each unvisited neighbor, call DFS recursively.

9. path.pop()

- Backtrack by removing the last vertex when done exploring.

10. graph = defaultdict(set)

- Create the graph as an adjacency list using defaultdict.

11. Print student details.

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

- Input 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 vertices and their neighbors dynamically.

14. Print the constructed graph.

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

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

- Input the start and target vertices.

16. Check if start or target exists in graph.

17. Initialize visited, path, and found.

18. Call dfs\_recursive().

19. If target not found, display message.

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

Possible Viva / External Questions and Answers

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

Q1: What is DFS?

A: Depth First Search is a graph traversal algorithm that explores as far as possible along each branch before backtracking.

Q2: What data structures are used in DFS?

A: Typically a stack (or function call stack in recursion), a visited set or array, and optionally a parent or path list.

Q3: What is the time complexity of DFS?

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

Q4: What is the space complexity of DFS?

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

Q5: What is defaultdict?

A: A dictionary from the collections module that provides a default value for missing keys automatically.

Q6: Why use a mutable 'found' list?

A: In Python, integers and booleans are immutable. To update a shared flag across recursive calls, we use a list that can be mutated.

Q7: What does 'backtrack' mean in DFS?

A: It means undoing the last step (popping from the path) when returning from a recursive call, to explore other paths.

Q8: What are the applications of DFS?

A: Pathfinding, cycle detection, topological sorting, connected components, solving puzzles/mazes.

Q9: What happens if the graph has cycles?

A: Without a visited set, DFS would enter an infinite loop. The visited set prevents revisiting nodes.

Q10: How can we modify this code to print all paths from start to target?

A: Remove the 'found' flag and allow DFS to continue even after finding one target, collecting all valid paths.

Q11: Difference between DFS and BFS?

A: DFS explores deep paths first; BFS explores all neighbors layer by layer using a queue.

Q12: What type of graph is being used here?

A: An undirected or directed graph represented as an adjacency list.

Q13: What is the role of recursion in this code?

A: It replaces the explicit stack, allowing the function call stack to handle exploration order.

Q14: What are common issues in recursive DFS?

A: Stack overflow on deep recursion, forgetting to mark nodes visited, infinite loops on cyclic graphs.

Q15: How does the program handle incorrect start or target vertices?

A: It checks if the vertices exist in the graph before starting DFS.

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

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

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

Q16: What are graph representations?

A: Adjacency list, adjacency matrix, and edge list.

Q17: Explain other graph traversal algorithms.

A: BFS (Breadth First Search), Dijkstra's algorithm, Bellman-Ford, A\* for weighted graphs.

Q18: What is the difference between tree and graph?

A: A tree is an acyclic connected graph; a graph may have cycles and may be disconnected.

Q19: What is the importance of LP2 in curriculum?

A: It gives hands-on practice in implementing and understanding algorithms and data structures, preparing students for real-world software engineering.

Q20: What are common pitfalls in implementing DFS?

A: Not tracking visited nodes, mishandling backtracking, exceeding recursion limits.

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

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