Dbfs pract 2 :-

Certainly! This code defines a simple graph class with methods to add edges, perform depth-first search (DFS), and breadth-first search (BFS). Let's go through each part of the code:

1. `class Graph:`: Defines a class named `Graph` to represent a graph data structure.

2. `def __init__(self):`: Initializes the `Graph` object with an empty adjacency list.

3. `self.adj_list = {}`: Initializes the adjacency list to an empty dictionary, where keys are vertices and values are lists of adjacent vertices.

4. `def add_edge(self, u, v):`: Adds an edge between vertices `u` and `v` by updating the adjacency lists of both vertices.

5. `if u not in self.adj_list:`: Checks if vertex `u` is not already in the adjacency list and adds it if not.

6. `if v not in self.adj_list:`: Checks if vertex `v` is not already in the adjacency list and adds it if not.

7. `self.adj_list[u].append(v)`: Adds vertex `v` to the adjacency list of vertex `u`.

8. `self.adj_list[v].append(u)`: Adds vertex `u` to the adjacency list of vertex `v` (since the graph is undirected).

9. `def dfs(self, start, visited=None):`: Performs a depth-first search starting from the given `start` vertex.

10. `if visited is None:`: Checks if the `visited` set is `None` and initializes it to an empty set if so.

11. `if start not in visited:`: Checks if the `start` vertex has not been visited yet, and if so, prints it, marks it as visited, and recursively explores its neighbors.

12. `for neighbor in self.adj_list[start]:`: Iterates over the neighbors of the `start` vertex and recursively calls `dfs` on each unvisited neighbor.

13. `def bfs(self, start):`: Performs a breadth-first search starting from the given `start` vertex.

14. `visited = set()`: Initializes a set to keep track of visited vertices.

15. `queue = [start]`: Initializes a queue with the `start` vertex.

16. `visited.add(start)`: Marks the `start` vertex as visited.

17. `while queue:`: Loops until the queue is empty.

18. `vertex = queue.pop(0)`: Dequeues a vertex from the queue.

19. `for neighbor in self.adj_list[vertex]:`: Iterates over the neighbors of the dequeued vertex.

20. `if neighbor not in visited:`: Checks if the neighbor has not been visited yet.

21. `visited.add(neighbor)`: Marks the neighbor as visited.

22. `queue.append(neighbor)`: Enqueues the neighbor for further exploration.

23. `Example usage`: Creates a `Graph` object, adds edges between vertices, and demonstrates DFS and BFS starting from vertex `0`.

This code provides a basic implementation of a graph and two common graph traversal algorithms (DFS and BFS).