Assignment 3 Report
Hassan ElMasry
52-2481 // 13000828

1- BFS

Breadth-First Search (BFS) algorithm for traversing a graph represented by an adjacency matrix. The method bfs takes two parameters - a starting vertex v and the adjacency matrix e. It uses a queue (q array) to systematically visit and process vertices in a breadth-first manner. The algorithm initializes a queue with the starting vertex, marks it as visited, and then enters a loop where it dequeues a vertex, prints it, and explores its adjacent vertices. For each unvisited adjacent vertex, it enqueues the vertex into the queue and marks it as visited. The process continues until all reachable vertices have been visited. This BFS traversal order is then printed to the console. The use of the boolean array visited ensures that each vertex is processed only once, preventing cycles and ensuring an efficient exploration of the graph in a breadth-first fashion.

2- DFS

Depth-First Search (DFS) algorithm for traversing a graph represented by an adjacency matrix. The method dfs is overloaded, with one variant taking a starting vertex v and the adjacency matrix e, and another variant taking the same parameters along with a boolean array visited to keep track of visited vertices. The DFS algorithm starts from the given vertex, marks it as visited, prints its value, and then recursively explores its unvisited adjacent vertices. The recursion allows the algorithm to delve deeper into the graph, backtracking only when there are no more unvisited neighbors. The base case of the recursion occurs when a vertex has no unvisited neighbors, and the algorithm systematically backtracks, printing the values of vertices in a depth-first order. The use of the boolean array visited ensures that each vertex is processed only once, preventing infinite loops and ensuring an efficient exploration of the graph's connected components using DFS.

3- Cycles

cycle detection algorithm for a directed graph using Depth-First Search (DFS). The method findCycles initiates the process by iterating through the edges of the graph. For each unvisited starting vertex, it invokes the dfsWithCycleDetection method. The latter performs a DFS traversal, keeping track of the visited vertices, the vertices in the current traversal stack (inStack), and the current path being explored.

During the DFS traversal, when a vertex is encountered that has already been visited and is in the current stack, it indicates the presence of a cycle in the graph. The printCycle method is then called to display the nodes involved in the detected cycle. The cycle is printed by identifying the starting index of the cycle in the current path and printing the corresponding vertices until the cycle completes.

This algorithm helps identify and print cycles within the directed graph, providing valuable information about its structure and potential issues such as feedback loops or unintended connections. The use of DFS and the tracking of visited vertices and the traversal stack contribute to an efficient and accurate cycle detection process.

4- Bipartite

algorithm to determine if a given directed graph is bipartite. A bipartite graph is one in which the vertices can be divided into two disjoint sets such that every edge connects a vertex from one set to a vertex in the other set. The method isBipartite takes an adjacency matrix e and an array color to represent the color assigned to each vertex in the graph.

The algorithm employs Breadth-First Search (BFS) in the bfsForBipartiteCheck method to traverse the graph and assign colors to the vertices. It starts by initializing the color array with -1 (indicating uncolored vertices) and then iterates through each vertex. For each uncolored vertex encountered, it assigns a color (0 or 1) and enqueues the vertex. During the BFS traversal, it ensures that adjacent vertices receive a different color from their current vertex. If the algorithm finds an adjacent vertex with the same color, it indicates a violation of the bipartite property, and the method returns false. If the traversal completes without conflicts, the graph is deemed bipartite, and the method returns true.

The overall isBipartite method iterates through all vertices, invoking the BFS-based color assignment for uncolored vertices. If any invocation returns false, the method concludes that the graph is not bipartite and returns false. Otherwise, it confirms that the graph satisfies the bipartite property and returns true. This algorithm efficiently leverages BFS and the concept of graph coloring to determine the bipartiteness of a directed graph.

5- Print Tree

prints the tree structure of a graph based on its adjacency matrix representation. The method first initializes an empty list, adjacencyList, to represent the adjacency list of the graph. It then iterates through each edge in the given adjacency matrix e. For each edge, it extracts the source vertex from and the destination vertex to. The code ensures that the adjacency list is appropriately sized to accommodate the vertices by dynamically adding new lists as needed. The from vertex is used as an index to add the to vertex to its corresponding adjacency list.

After constructing the adjacency list, the code proceeds to print the tree structure. It starts the iteration from index 1, assuming that the vertices in the graph are numbered starting from 1. For each vertex, it prints the vertex number along with its adjacent vertices in the format "Vertex x -> y, z, ...". The inner loop handles the printing of the adjacent vertices with proper comma separation. The resulting output displays the tree structure, showing the relationships between vertices in the graph. This method is particularly useful for visualizing the connectivity and hierarchy of tree-like structures in a graph.

6- To determine if it's a tree or not we need to
   a- Check if there are Cycles
   b- Check if during BFS or DFS all vertices are visited
   c- Check that there is n-1 edges for n vertices graphs
   If any of the following is not true then it is not a tree

7- Running time would be O(V+E) as the BFS and DFS have a time complexcity of O(V+E)