|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Data Structures Laboratory (using C/C++)** | **Semester:** | **III** |
| **Date of Performance:** | **10 / 10 / 2023** | **Batch No:** | **A-3** |
| **Faculty Name:** | **Om Goswami** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No: 6**

**Title: Graph Traversal**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Write a program for BFS and DFS. |

|  |
| --- |
| **COs to be achieved:** |
| **CO1:** Understand and implement the different data structures used in problem solving.  **CO2:** Apply linear and non-linear data structure in application development. |

|  |
| --- |
| **Books/Journals/Websites referred:** |
| Virtual Lab |

|  |
| --- |
| **Tools required:** |
| **DEV C/C++ compiler/ Code blocks C compiler** |

|  |
| --- |
| **Theory:** |
| Graph traversal is a technique used for a searching vertex in a graph. The graph traversal is also used to decide the order of vertices visited in the search process. A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into a looping path.  There are two graphing traversal techniques and they are as follows:   1. DFS (Depth First Search) 2. BFS (Breadth First Search)   VLAB Link for experiment:  DFS: <https://ds1-iiith.vlabs.ac.in/exp/depth-first-search/index.html>  BFS: <https://ds1-iiith.vlabs.ac.in/exp/breadth-first-search/index.html> |

|  |
| --- |
| **Implementation details:** |
| 1. **Enlist all the Steps followed and various options explored.**   Graph traversal is a fundamental algorithmic concept in computer science used to explore and navigate graphs. There are two main graph traversal techniques: Depth-First Search (DFS) and Breadth-First Search (BFS). Here are the steps followed and various options explored in graph traversal:  Depth-First Search (DFS):  Steps:   * Start: Pick a starting node. * Mark as Visited: Mark the starting node as visited. * Explore: Explore an adjacent node not yet visited. * Recursion: If there are adjacent nodes to the current node, repeat step 3 recursively for those nodes. * Backtrack: If you reach a node with no unvisited adjacent nodes, backtrack to the previous node and explore any unvisited nodes from there. * Repeat: Repeat steps 3-5 until all nodes are visited.   Options Explored in DFS:   * Recursion: Implement DFS using recursion, allowing the algorithm to explore as deeply as possible along each branch before backtracking. * Stack: Use an explicit stack data structure to keep track of nodes to visit, simulating the recursive call stack. * Visited Array: Maintain a Boolean array to keep track of visited nodes to avoid revisiting nodes. * Parent Pointers: Keep track of parent nodes to reconstruct the traversal path if needed.   Breadth-First Search (BFS):  Steps:   * Start: Pick a starting node. * Queue: Enqueue the starting node into a queue. * Mark as Visited: Mark the starting node as visited. * Explore: Dequeue a node from the queue and explore all its adjacent nodes. * Enqueue: Enqueue any unvisited adjacent nodes into the queue. * Repeat: Repeat steps 4-5 until the queue is empty.   Options Explored in BFS:   * Queue: Utilize a queue data structure to keep track of nodes to visit, ensuring nodes are visited in the order they were discovered. * Visited Array: Maintain a Boolean array to keep track of visited nodes, preventing revisiting nodes. * Shortest Path: BFS can be used to find the shortest path between two nodes by stopping the algorithm when the destination node is reached. * Level Information: Keep track of the level of each node from the starting node to determine the shortest path or other level-specific information.  1. **Explain your program logic and methods used.**   Logic:   * Recursion: One common way to implement DFS is by using recursion. The algorithm starts at the root node and explores as far as possible along each branch before backtracking. * Visited Nodes: DFS maintains a Boolean array to keep track of visited nodes to avoid revisiting nodes and getting stuck in cycles. * Backtracking: If a node has no unvisited neighbors, the algorithm backtracks to the previous node where there are unexplored paths.   Depth-First Search (DFS):  DFS explores as far as possible along each branch before backtracking. It can be implemented using recursion or an explicit stack:  Recursive DFS:   * Start at a node. * Mark the node as visited. * Recur for all adjacent nodes not yet visited.   Iterative DFS (Using Stack):   * Start with a stack. * Push the initial node onto the stack. * While the stack is not empty: * Pop a node from the stack. * Process the node. * Push all unvisited neighbors onto the stack.   Breadth-First Search (BFS):  BFS explores all the neighbor nodes at the present depth before moving to nodes at the next depth level:  Using Queue:   * Start with a queue. * Enqueue the initial node into the queue. * Mark the node as visited. * While the queue is not empty: * Dequeue a node from the queue. * Process the node. * Enqueue all unvisited neighbors into the queue. * Mark the neighbors as visited.  1. **Explain the Importance of the approach followed by you.**   Some reasons why the approach followed in graph traversal is crucial:   1. Problem Solving:  * Pathfinding: Graph traversal is essential in finding paths between nodes, which is crucial in navigation systems, GPS, and routing algorithms. * Connectivity Analysis: It helps in understanding the connectivity between nodes, which is vital in social network analysis, communication networks, and internet connectivity analysis.  1. Data Representation:  * Data Modeling: Many real-world scenarios can be represented as graphs, including social networks, transportation networks, and dependencies between tasks. * Data Mining: Analyzing relationships between entities in a graph structure helps discover patterns and insights in large datasets.  1. Optimization:  * Shortest Path Problems: Graph traversal algorithms like Dijkstra's and BFS are used in finding the shortest path, minimizing travel time, or resource usage. * Resource Allocation: Graph traversal assists in optimizing resource allocation in various applications such as job scheduling, network flow optimization, and logistics. |

|  |
| --- |
| **Simulation screenshots:** |
|  |

|  |
| --- |
| **Output:** |
|  |

|  |
| --- |
| **Post Lab:** |
| 1. **Discuss DFS and BFS with real world applications?** 2. DFS:  * Maze Solving: DFS can be used to find a way out of a maze by exploring paths until a solution is found. * Topological Sorting: DFS can be employed to find a topological ordering of tasks in a project, ensuring that tasks are executed in the correct order. * Connected Components: DFS can identify connected components in a graph, which is useful in social network analysis and community detection.  1. BFS:  * Network Broadcasting: BFS is used in computer networks to broadcast messages to all nodes. It ensures that the message reaches all nodes in the shortest possible time. * Puzzle Solving: BFS can be used to solve puzzles such as the sliding puzzle or the Rubik's Cube by exploring possible states and finding a solution. * Optimizing Social Networks: BFS can be applied to optimize social networks by finding the shortest introduction chain between two individuals, known as the "six degrees of separation". |

|  |
| --- |
| **Conclusion:** |
| In conclusion, we've thoroughly grasped the workings of DFS and BFS algorithms, their core principles, strengths, and practical applications. These versatile techniques, whether for maze-solving or network analysis, are invaluable tools in the realm of graph-based problem-solving. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |