**Summary:**

**Breadth-First Search (BFS):**

Breadth-First Search (BFS) is a graph traversal algorithm designed to explore and search nodes within a graph systematically. It starts from a chosen source node, moves to its neighbors, and explores their child nodes. BFS is ideal for finding the shortest path in unweighted graphs and checking graph connectivity.

**Depth-First Search (DFS):**

DFS is a graph traversal algorithm that explores one path as deeply as possible before backtracking, making it useful for tasks like finding connected components. It can be implemented using a stack or recursion.

**Data structures used:**

**1.Arrays:**

I used Boolean arrays to keep track of visited vertices while doing BFS( Breadth First search) and DFS(Depth First Search)

**2.ArrayLists:**

Used to store lists of vertices and adjacency lists for the graph. **vertexList** is an **ArrayList** for storing vertices, and **adjacencyList** is an ArrayList of ArrayLists used to represent the graph's adjacency lists. These ArrayLists provide dynamic storage for vertices and their connections.

**3.LinkedList-based Queue:**

Utilized to implement BFS (Breadth-First Search). **BFSqueue** is a Queue based on a LinkedList structure, which ensures that vertices are visited in the order they were encountered during BFS traversal. This data structure aids in managing the order of exploration in a breadth-first manner.

**4.Stack:**

Utilized for DFS (Depth-First Search) to manage the order of vertices to explore and backtrack during traversal.

**Time and space complexity analysis:**

**For breadthFirstSearch() method:**

**1. Time Complexity:**

In BFS, each vertex and edge is visited once.

* Initializing data structures like the **visited** array, **BFSqueue** (a queue), and **visitedVertices** (a list) takes constant time, O(1).

* While Loop, The core of the BFS algorithm is the `while` loop. In the worst case, this loop can run through all vertices and edges in the graph, which results in a time complexity of O(V + E), where V is the number of vertices, and E is the number of edges.
* Inside the `**while**` loop, I iterated through the neighbors of the current vertex. The number of neighbors visited depends on the degree of the vertex, which is typically bounded by a constant factor. So, visiting neighbors is O(1) on average.

Therefore, the overall time complexity of the `**breadthFirstSearch**()` method is O(V + E), where V is the number of vertices, and E is the number of edges in the graph.

**2. Space Complexity:**

- The space complexity of BFS is determined by the additional data structures used, mainly the queue and the boolean array to track visited vertices.

- The queue can store at most all vertices at one level of the graph, which is the maximum number of vertices at any level.

- In the worst case, the space complexity of BFS is O(V), where V is the number of vertices in the graph.

**For deapthFirstSearch() method:**

The depthFirstSearch method uses a depth-first search (DFS) algorithm to traverse a graph.

* Initializing the **visitedVertices** list, the **dfsStack**, and the **visited** boolean array all have constant time complexity, O(1), as they involve basic variable assignments.
* The main work of the DFS traversal occurs within the `while` loop, where vertices are pushed onto and popped from the **dfsStack**.
* In the worst-case scenario, where the graph is a tree and we traverse all vertices and edges, the loop runs **O(V + E)** times, where V is the number of vertices and E is the number of edges in the graph.
* Within the **while** loop, there is an inner `**for**` loop that iterates over the neighbors of the current vertex. In the worst case, if all vertices are connected to each other, this inner loop can run O(V) times.
* However, since each edge is visited only once (because we mark vertices as visited), the overall time complexity of this inner loop across all iterations of the outer loop is O(E).
* Combining the time complexities of the outer loop and the inner loop, the overall time complexity of the **depthFirstSearch** method is O(V + E).

**2. Space Complexity:**

- The space complexity of DFS is determined by the additional data structures used, mainly the stack (or the call stack in the case of a recursive implementation) and the boolean array to track visited vertices.

- The maximum depth of the recursion stack or the maximum number of vertices that can be stored in the stack is equal to the depth of the deepest branch in the graph.

- In the worst case, for a graph with a single linear path, the space complexity of DFS is O(V), where V is the number of vertices. However, in a more complex graph, it can be less than O(V).

**Summary of important methods of Graph class:**

1. **Constructor (public Graph()):**

- Initializes the `Graph` object by creating empty lists for `vertexList` and `adjacencyList`. It's called when creating a new graph.

2.**public void addVertex(int vertex):**

- Adds a vertex to the graph by appending it to the `vertexList` and creating an empty adjacency list for the vertex.

3. **public void addEdge(int source, int destination):**

- Adds an edge between two vertices (source and destination) in the graph. It updates the adjacency list for both vertices since the provided graph is undirected.

4. **public ArrayList<Integer> getNeighbors(int vertex):**

- Retrieves and returns a list of neighbors (adjacent vertices) for the specified vertex. This method is useful for finding the neighbors of a particular vertex.

**5.public int getVertexCount():**

- Returns the total number of vertices in the graph. It is based on the size of the adjacency list, which corresponds to the number of vertices.

6.**public ArrayList<Integer> breadthFirstSearch(int startVertex):**

- Performs a Breadth-First Search (BFS) traversal of the graph, starting from the specified `startVertex`. It returns a list of visited vertices in BFS order.

7. **public ArrayList<Integer> depthFirstSearch(int startVertex)**:

- Performs a Depth-First Search (DFS) traversal of the graph, starting from the specified `startVertex`. It returns a list of visited vertices in DFS order.

8.**Main Method:**

The `main` method serves as the program's entry point. It reads graph data from a specified file, performs connected components analysis using BFS and DFS, and prints the results.

9.**readFile Method**

The readFile method reads a graph data file, parses its content, and creates `Graph` objects. It returns a list of these objects for further analysis.