Final Project: Data Structures and Algorithms (Optimizing a Delivery System)

Objective

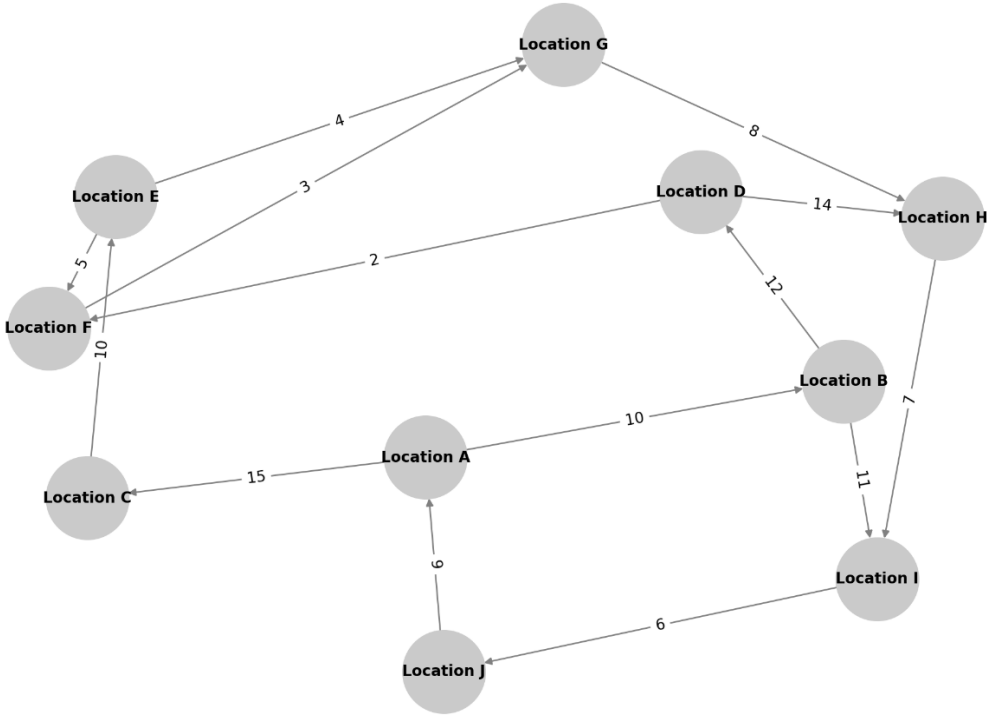
The goal of this project is to integrate key concepts from the course into a cohesive coding and analytical experience. Students will demonstrate their understanding of data structures and algorithms by implementing, testing, and analyzing their performance in a real-world-inspired context

Task

Build a system to manage and optimize delivery routes for a company. The system should use arrays, linked lists, skip lists, and various sorting and searching algorithms to organize and process data. Furthermore, your project should incorporate tree-based structures and graph algorithms to handle route optimization and efficiency

Data

Here is the data that you will need for the project, the graph represents delivery routes, and the table represents delivery orders



Order ID	Priority	Destination
ORD1	1	Location A
ORD2	2	Location B
ORD3	3	Location C
ORD4	4	Location D
ORD5	5	Location E
ORD6	1	Location F
ORD7	2	Location G
ORD8	3	Location H
ORD9	4	Location I
ORD10	5	Location J
ORD11	1	Location K
ORD12	2	Location L
ORD13	3	Location M
ORD14	4	Location N
ORD15	5	Location O
ORD16	1	Location P
ORD17	2	Location Q
ORD18	3	Location R
ORD19	4	Location S
ORD20	5	Location T
ORD21	1	Location U
ORD22	2	Location V
ORD23	3	Location W
ORD24	4	Location X
ORD25	5	Location Y
ORD26	1	Location Z
ORD27	2	Location A
ORD28	3	Location B
ORD29	4	Location C
ORD30	5	Location D
ORD31	1	Location E
ORD32	2	Location F
ORD33	3	Location G
ORD34	4	Location H
ORD35	5	Location I
ORD36	1	Location J
ORD37	2	Location K
ORD38	3	Location L
ORD39	4	Location M
ORD40	5	Location N
ORD41	1	Location O
ORD42	2	Location P
ORD43	3	Location Q
ORD44	4	Location R
ORD45	5	Location S
ORD46	1	Location T
ORD47	2	Location U
ORD48	3	Location V
ORD49	4	Location W
ORD50	5	Location X

Part 1: Data Management with Arrays and Lists

- Use an array to store a static list of delivery orders
- Use a single-linked list to handle dynamic additions of new orders
- Convert the singly linked list into a doubly linked list for bi-directional traversal
- Implement a skip list to optimize the search for a specific delivery order

Detailed Description

1. **Implement an Array**
    - Create an array to store all 50 delivery orders. Each order should include the following details:
        - Order ID (e.g., "ORD1")
        - Priority (1–5)
        - Destination (e.g., "Location A")
    - Implement functions to:
        - Add a new delivery order at the end of the array
        - Display the entire list of orders
        - Search for a specific order using the Order ID. Use a linear search
2. **Implement a Singly Linked List**
    - Convert the array into a singly linked list
    - Each node should contain:
        - Order ID
        - Priority
        - Destination
    - Implement functions to:
        - Add a new delivery order at the end of the list
        - Traverse and print all orders in the linked list
        - Search for an order using the Order ID

3. **Convert to a Doubly Linked List**
    - Upgrade the singly linked list to a doubly linked list to allow bi-directional traversal
    - Implement additional functions to:
        - Remove a delivery order from the list
        - Traverse the list in reverse order
        - Update the priority of an existing order
4. **Implement a Skip List**
    - Design and implement a skip list for the delivery orders, with Order ID as the key for ordering.
    - Use the skip list to efficiently:
        - Insert a new delivery order
        - Search for a delivery order by Order ID
        - Delete an existing order

Part 1: Expected Output and Deliverables

- **Code Implementation**
    - Separate files or functions for each data structure (arrays, singly linked-lists, doubly linked-lists, skip-lists)
    - Well-commented code demonstrating the functionality of each data structure
- **Test Cases**
    - At least 3 test cases for each operation (add, remove, search, update) for all data structures
- **Documentation**
    - A brief explanation (in the summary document) of the time complexity for each operation across these data structures
    - Insights on which data structure performs best for each type of operation
- **Presentation Slide**
    - Include one slide in the presentation that explains:
        - The benefits and drawbacks of using arrays, singly linked-lists, doubly linked-lists or skip-lists
        - Insights into when to use one data structure over another

Part 2: Sorting Algorithms

- Sort the delivery orders by priority using selection sort, bubble sort, merge sort, and quick sort
- Compare and analyze the time complexity of each sorting algorithm

Detailed Description

1. **Generate a random list of priority levels**
   - Length of the list = 10,000
   - Values should be between 1 to 5
2. **Implement Sorting Algorithms**
   - Write implementations for the following sorting algorithms:
     - Selection Sort
     - Bubble Sort
     - Merge Sort
     - Quick Sort
   - Each implementation should sort the delivery orders array based on the Priority field
   - Sort the provided delivery orders table by priority level using each algorithm
3. **Compare Performance**
   - Measure and record the execution time of each sorting algorithm using the randomized list you generated:
     - Use the first 1,000 numbers from the list
     - Use the entire 10,000 numbers from the list
   - Ensure the datasets are randomized before testing each algorithm!
4. **Analyze Time Complexity**
   - Provide a written explanation of the time complexity for each sorting algorithm:
     - Best-case, average-case, and worst-case scenarios
   - Discuss why some algorithms are faster than others for this specific use case
5. **Visualization**
   - Create a simple visualization (graphs) to show the progress of sorting for one of the algorithms

Part 2: Expected Output:

- **Code Implementation**
    - Each sorting algorithm should be implemented as a separate function.
    - Functions should take the random list of priority levels as input and return a sorted array.
    - Code should include comments explaining each step of the sorting process.
- **Test Cases**
    - A table displaying 10 run times for each algorithm for small (1,000) and large (10,000) inputs
- **Documentation**
    - Include in the summary document:
        - Observations about the efficiency and suitability of each algorithm for sorting delivery orders
- **Presentation Slide**
    - Include one slide in the presentation that explains:
        - The differences in performance between the algorithms
        - Insights into when to use one algorithm over another

Part 3: Binary Search and Data Structures

- Use binary search to quickly find a delivery order in the sorted list
- Implement a Binary Search Tree (BST) to store delivery data for hierarchical searches
- Convert the BST into an AVL Tree to ensure balanced searches and analyze the difference in search efficiency

Detailed Description

1. **Binary Search Tree (BST)**
    - Implement a BST to store delivery orders
    - Each node should contain:
        - Order ID (key for searching)
        - Priority
        - Destination
2. **Search**
    - Implement a search function in the BST to find a delivery order by Order ID
    - Return the details of the order (Order ID, Priority, Destination) if found, or a message indicating that the order does not exist
3. **Insert, Delete, and Traverse**
    - Implement functions for:
        - Inserting a new delivery order. Search for ORD51
        - Deleting a delivery order.
        - Traversing the tree in-order (to display the orders in sorted order by Order ID).
4. **AVL Tree**
    - Convert the BST into an AVL Tree to ensure it remains balanced
    - Implement rotations (left, right, left-right, right-left) as necessary during insertions and deletions
    - Implement the same operations as the BST (insert, search, delete, in-order traversal)

- Compare the height of the AVL Tree and the original BST for the same dataset.

5. **Performance Analysis**
   - Measure and compare the time required to:
     - Insert all 50 delivery orders into the BST and AVL Tree
     - Search for a specific order in the BST and AVL Tree
   - Discuss the differences in performance and tree structure (balanced vs. unbalanced)

Part 3: Expected Output:

- **Code Implementation**
    - Separate functions for:
        - BST operations (insert, search, delete, in-order traversal)
        - AVL Tree operations with balancing
    - Include comments to explain the logic, particularly for rotations in the AVL Tree
- **Test Cases**
    - Test binary search with at least 3 Order IDs (existent and non-existent)
    - Insert all delivery orders into the BST and AVL Tree, then perform searches for 5 specific Order IDs
    - Delete a few orders from the BST and AVL Tree, then verify the structures
- **Documentation**
    - In the summary document:
        - Provide a comparison of insertion, search, and deletion times for the BST and AVL Tree
        - Explain the importance of tree balancing and how the AVL Tree ensures better performance
        - Include diagrams (hand-drawn) of the BST and AVL Tree for a the first 10 orders in the list
- **Presentation Slide**
    - Include one slide explaining:
        - The differences between the BST and AVL Tree in terms of structure and performance
        - Insights into why balancing is crucial for efficient operations

Part 4: Route Optimization with Graph Algorithms

- Represent delivery routes as a graph
- Use Depth-First Search (DFS) and Breadth-First Search (BFS) to explore routes
- Implement Dijkstra's algorithm to find the shortest path between the delivery hub and destinations

Detailed Description

1. **Graph Representation**
   - Represent the delivery network as a directed graph using an adjacency list or adjacency matrix. Each node represents a delivery location, and edges represent routes with associated distances (weights)
2. **Depth-First Search (DFS) and Breadth-First Search (BFS)**
   - Implement DFS to explore all reachable delivery locations starting node Location A
   - Implement BFS to explore all reachable delivery locations starting node Location A
   - Compare and discuss the results of DFS and BFS
3. **Dijkstra's Algorithm**
   - Implement Dijkstra's algorithm to find the shortest path (based on edge weights) from the starting node (Location A) to all other nodes
   - Output the shortest path and total distance for at least 3 destination nodes
   - Highlight how Dijkstra's algorithm calculates the shortest path
4. **Performance Analysis**
   - Measure the time taken to execute DFS, BFS, and Dijkstra's algorithm on the graph from Location A to Location J
   - Analyze the time complexity of each algorithm and discuss which is most suitable for optimizing delivery routes
5. **Visualization**
   - Visualize the computed paths using console output or a basic graphical representation (displaying visited nodes in order)

Part 4: Expected Output:

- **Code Implementation**
    - Separate functions for:
        - adjacency list/matrix
        - DFS, BFS, and Dijkstra's algorithm.
    - Code should be well-documented with comments explaining the logic of each algorithm.
- **Test Cases**
    - Use at least 3 test cases:
        - Run DFS and BFS from the starting node (Location A) to various destinations
        - Compute shortest paths using Dijkstra's algorithm for multiple destinations
- **Documentation**
    - Include in the summary document:
        - A brief explanation graph representations
        - A comparison of results from DFS, BFS, and Dijkstra's algorithms
        - Observations on the performance and suitability of each algorithm for route optimization
- **Presentation Slide**
    - Include one slide in your presentation that explains:
        - The differences between DFS, BFS, and Dijkstra's algorithm
        - Why Dijkstra's algorithm is preferred for weighted graphs in route optimization scenarios

Other information

Deliverables

1. **Code Folder**
    - A folder containing the implementation of all required components, organized into separate files or classes for clarity
    - Code should be well-documented with comments explaining functionality
2. **Summary Document**s
    - A PDF report summarizing:
        - The implementation process
        - Key findings and observations
        - Challenges faced and how they were addressed
        - Analysis of time complexity for each implemented algorithm
3. **Presentation**
    - A PowerPoint or PDF document that:
        - Explains approaches used and highlights results
        - Discusses the comparison of algorithms employed
        - Reflects on the use of different data structures in solving problems

Other Information

1. **Timeline and Expectations**
    - Estimated Time Commitment: ~10 hours
    - Due Date: December 18, 2024

2. **Evaluation Criteria**

Component	Weight
Code Implementation	75% of course project points
Summary Document	25% of course project points
Presentation	100% of course presentation points

   - Code will be assessed on correctness, efficiency, and clarity
   - Summary Document will be evaluated for depth of analysis and clarity
   - The presentation will assess understanding of the material

3. **Sample Data**
   - A list of 50 delivery orders, each represented by an ID, priority level (1–5), and destination address
   - A graph representation of delivery routes (nodes for locations and edges with distances)

4. **Guidance and Resources**
   - Revisit course materials on each algorithm and data structure
   - Use online resources for code references
   - Collaborate responsibly (discuss concepts but write **your own** code)