C950 Task-2 WGUPS Write-Up

(Task-2: The implementation phase of the WGUPS Routing Program).

(Zip your source code and upload it with this file)

Livio Angelim Mororo

ID #011953539

WGU Email: lange70@wgu.edu

27-10-2024

C950 Data Structures and Algorithms II

# A. Hash Table

Hash Table Screenshot (1/2):

A screenshot of a computer program

Description automatically generated

Hash Table Screenshot (2/2):

A screen shot of a computer

Description automatically generated

# B. Look-Up Functions

Look-up function screenshot:

A screenshot of a computer program

Description automatically generated

# C. Original Code

Major code blocks go here showing implementation.

The Main Function orchestrates the entire program:

* Loading package and distance data,
* Managing the delivery simulation (with its package assignments and time update)
* Displaying the total mileage after deliveries

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

# C1. Identification Information

main.py screenshot showing Student ID

A screenshot of a computer

Description automatically generated

# C2. Process and Flow Comments

The comments help the reader understand how packages are filtered, constraints are applied, and how assignments are made.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

# D. Interface

The interface presents a menu with four options. Users can select an option by entering the corresponding number. The program then prompts for additional input if necessary (e.g., time or package ID) and displays the requested information.A screen shot of a computer

Description automatically generated

# D1. First Status Check

As an example, at **8:55 AM**, the 40 packages are in various statuses.

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated | A screenshot of a computer  Description automatically generated |

Packages with earlier deadlines are **Delivered**, some other are **En Route** and the delayed packages remain **At Hub** if they haven't arrived yet.

# D2. Second Status Check

At **10:20** **AM**, more packages have been delivered. Delayed packages that arrived at 8:55 AM now are En Route or Delivered.

|  |  |
| --- | --- |
| A screen shot of a computer  Description automatically generated | A screen shot of a computer  Description automatically generated |

# D3. Third Status Check

At **12:30 PM**, all packages are **Delivered**, with delivery times indicated.

|  |  |
| --- | --- |
|  |  |

# E. Screenshot of Code Execution

After finishing the simulation, the program displays the **total mileage of 138.30** traveled by all trucks and the message indicating it is within the limit *(by the requirement: 140.0)*. After exiting through the user interface *(option 4)*, the program is finished without error.

A screenshot of a computer

Description automatically generated

# F1. Strengths of the Chosen Algorithm

The Nearest Neighbor Algorithm offers several strengths:

* **Simplicity and Ease of Implementation:** The algorithm is straightforward to understand and implement. It selects the nearest unvisited location at each step, making it intuitive for route optimization problems like package delivery.
* **Efficiency for Small Datasets:** For problems with a relatively small number of nodes (packages and destinations), the Nearest Neighbor Algorithm provides quick and reasonably efficient solutions. It reduces computational overhead compared to more complex algorithms, allowing for faster execution times.

# F2. Verification of Algorithm

The Nearest Neighbor Algorithm meets all requirements in the scenario:

* **Delivery Deadlines**: By prioritizing the nearest packages, the algorithm helps ensure that packages with earlier deadlines are delivered promptly. In the implementation, packages with urgent deadlines are loaded first, and the route is optimized to meet those deadlines.
* **Special Constraints Handling**: The algorithm accounts for special notes, such as packages that must be delivered together or those that can only be on a specific truck. Delayed packages are only loaded after their arrival time, and package #9's address is updated at the correct time.
* **Total Mileage Under 140 Miles**: The algorithm efficiently minimizes the total distance traveled by always selecting the closest next destination. As shown in the program output, the total mileage is **138.30 miles**, which is within the 140-mile limit.

# F3. Other Possible Algorithms

* **Dijkstra's Algorithm:** An algorithm for finding the shortest paths between nodes in a graph, particularly suitable for graphs with non-negative edge weights. It systematically selects the node with the smallest tentative distance, updating distances to neighboring nodes, and builds the shortest path tree from the starting point to all other nodes.
* **Dynamic Programming Approach (e.g., Held-Karp Algorithm)**: A method used to solve the Traveling Salesman Problem (TSP) optimally by breaking it down into smaller overlapping subproblems. It stores the results of subproblems to avoid redundant calculations, considering all possible routes to find the minimum total distance.

# F3a. Algorithm Differences

**Comparison of Nearest Neighbor Algorithm, Dijkstra's Algorithm, and Dynamic Programming Approach in four aspects:**

1. **Approach**

* **Nearest Neighbor Algorithm**: A greedy heuristic that builds a delivery route by always selecting the nearest unvisited location. It focuses on immediate proximity without considering the overall route efficiency.
* **Dijkstra's Algorithm**: Calculates the shortest paths from a single source to all other nodes by considering cumulative distances. It optimizes individual paths from the starting point but doesn't create a complete tour visiting all locations.
* **Dynamic Programming Approach**: Solves the Traveling Salesman Problem (TSP) optimally by evaluating all possible routes through recursive subproblem solving and memorization, ensuring the shortest possible route covering all destinations.

1. **Complexity**

* **Nearest Neighbor Algorithm**: Time complexity is **O(n²)**, practical for small to medium datasets like the 40-package scenario.
* **Dijkstra's Algorithm**: Time complexity ranges from **O(V²)** to **O(E + V log V)** (with a priority queue), where **V** is the number of nodes and **E** is the number of edges.
* **Dynamic Programming Approach**: Time complexity is **O(n² \* 2^n)**, making it computationally infeasible for larger datasets due to exponential growth.

1. **Optimization and Suitability**

* **Nearest Neighbor Algorithm**: Fast and easy to implement but may not find the optimal route. Suitable for the WGUPS scenario due to its ability to handle constraints efficiently and generate routes quickly.
* **Dijkstra's Algorithm**: Finds optimal paths from the hub to each location individually but doesn't solve the TSP. Not suitable for creating a complete delivery route that visits all locations once.
* **Dynamic Programming Approach**: Guarantees the shortest possible route but is impractical for the WGUPS scenario because of its high computational demands with many packages.

1. **Handling Constraints**

* **Nearest Neighbor Algorithm**: Easily incorporates delivery deadlines and special requirements by adjusting selection priorities during routing.
* **Dijkstra's Algorithm**: Not inherently designed to handle delivery constraints or to construct a full route; would require significant adaptation.
* **Dynamic Programming Approach**: Can handle constraints within its formulation but adds to computational complexity, making it unsuitable for timely deliveries in the scenario.

**Conclusion**

The **Nearest Neighbor Algorithm** offers a practical balance between efficiency and complexity for the WGUPS delivery problem. It allows for timely route planning and effectively manages delivery constraints, despite not always providing the optimal route. **Dijkstra's Algorithm** is not suitable for creating a complete delivery tour, and the **Dynamic Programming Approach**, while optimal, is too resource-intensive for the number of packages involved. Therefore, the Nearest Neighbor Algorithm is the most appropriate choice among the three for the given scenario.

# G. Different Approach

If I were to do this project again, I would consider implementing a more advanced routing algorithm, such as the Clarke-Wright Savings Algorithm, to potentially reduce the total mileage further.

I would explore using a priority queue to manage packages based on delivery deadlines, ensuring that packages with earlier deadlines are prioritized more effectively.

I would refactor the code to enhance modularity and maintainability by separating concerns more distinctly between data loading, routing logic, and user interface.

Improving the user interface to be more interactive or user-friendly by adding features like real-time tracking of package statuses or a graphical representation of the delivery routes.

# H. Verification of Data Structure

The custom hash table provides O(1) average-case time complexity for insertions and lookups, essential for real-time package tracking. It allows for immediate updates to package information, such as correcting addresses, which is crucial for meeting delivery deadlines. The data structure scales well with the number of packages, ensuring consistent performance even as the dataset grows.

# H1. Other Data Structures

**AVL Tree:**

* Efficient for ordered data storage and retrieval.
* Maintains balance to ensure O(log n) time complexity for operations.
* Suitable for applications requiring sorted data or range queries.

**Trie (Prefix Tree):**

* Efficiently handles dynamic sets of strings, such as addresses.
* Provides fast retrieval, especially for prefix-based searches.
* Useful if the application needs to perform searches or auto-completion on addresses or other string data.

# H1a. Data Structure Differences

**Comparison of Hash Table, AVL Tree, and Trie:**

1. **Time Complexity**

* **Hash Table**: O(1) average-case for insertion and lookup; O(n) worst-case if many collisions occur.
* **AVL Tree**: O(log n) for all operations due to self-balancing.
* **Trie**: O(k) for operations, where **k** is the length of the key.

1. **Data Ordering**

* **Hash Table**: Unordered; does not maintain any sequence.
* **AVL Tree**: Ordered; maintains elements in sorted order.
* **Trie**: Hierarchical; organizes data based on key prefixes.

1. **Memory Usage**

* **Hash Table**: Efficient but may have overhead from empty slots or resizing.
* **AVL Tree**: Moderate; each node stores extra balance info.
* **Trie**: High; consumes more memory due to numerous nodes for key characters.

1. **Suitability for Package Management**

* **Hash Table**: Ideal for fast access using unique package IDs; efficient updates.
* **AVL Tree**: Useful if ordered data access is needed; consistent performance.
* **Trie**: Best for prefix searches on strings; less efficient for numeric IDs.

1. **Ease of Implementation**

* **Hash Table**: Simple; requires collision handling.
* **AVL Tree**: More complex; balancing logic needed.
* **Trie**: Complex; manages multiple node levels.

1. **Conclusion**

* **Hash Table**: Most suitable for quick, direct access and frequent updates in the delivery system.
* **AVL Tree**: Adds complexity without significant advantage for this application.
* **Trie**: Not ideal for package IDs; better for applications needing prefix searches on strings.

# I. Sources

* **C950 WGUPS Project Implementation Steps - Example - Nearest Neighbor**. Western Governors University. Retrieved from [WGU Course Material](https://srm--c.vf.force.com/apex/coursearticle?Id=kA03x000001DbBGCA0)
* Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
* GeeksforGeeks. (n.d.). *Hashing Data Structure*. Retrieved from https://www.geeksforgeeks.org/hashing-data-structure/
* Lysecky, R., & Vahid, F. (2018). *Data Structures and Algorithms II*. zyBooks. Retrieved from <https://learn.zybooks.com/zybook/WGUC950AY20182019/>
* Python Software Foundation. (n.d.). *Python Documentation*. Retrieved from <https://docs.python.org/3/>
* Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.
* W3Schools. (n.d.). *Python Tutorial*. Retrieved from https://www.w3schools.com/python/

# J. Professional Communication