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)

Brandon Nguyen

ID #003964281

WGU Email: bnguy80@wgu.edu

Date: 8/11/23

C950 Data Structures and Algorithms II

**Stated Problem:**

The objective of this program is to establish efficient routes and distribution plans for package deliveries within the Western Governor University Parcel Service (WGUPS), utilizing Python 3.11. The task involves handling 40 packages scheduled for delivery by the end of the day, each with unique delivery deadlines. Challenges include limit of three trucks and two available drivers for delivery, accounting for delays in package arrival, adhering to specific truck loading constraints, and accommodating special requirements like grouped package deliveries and designated trucks for certain packages. To address these challenges a systematic approach is proposed.

**Proposed Solution and Approach:**

Our approach consists of several strategic steps aimed at optimizing package deliveries for the Western Governor University Parcel Service (WGUPS):

1. **Truck Prioritization:** The three available trucks will be prioritized based on package delivery deadlines. High priority(truck1) for packages with a delivery deadline of 9:00 AM, medium priority(truck2) with delivery deadline of 10:30 AM, and low priority(truck3) for packages with deliver deadline by EOD which will be defined as 5: 00 PM. This prioritization will for the foundation for efficient route planning and distribution.

2**. Nearest Neighbor Greedy Algorithm:** The core of our approach lies in the implementation of the self-adjusting heuristic nearest neighbor algorithm. The nearest neighbor algorithm is used to create an initial delivery route for each truck. It starts from the hub (starting point) and iteratively selects the nearest package or destination to visit next. This process continues until all packages are assigned to the route.

The nearest neighbor algorithm is relatively simple and quick but may not always result in the most optimal route. In the context of the code, the nearest neighbor algorithm is used to create an initial route for each truck, which is then further optimized using the two-opt algorithm and Dijkstra's shortest path algorithm.

3. **Two-Opt Greedy Algorithm:** The two-opt algorithm is then applied to the initial routes created by the nearest neighbor algorithm. This algorithm attempts to improve the route by iteratively swapping pairs of edges to eliminate intersections and reduce the overall route distance. It helps refine the initial routes and make them more optimal.

After applying the two-opt algorithm to the initial routes, Dijkstra's algorithm is used once again to calculate the shortest paths for the refined routes. This ensures that the routes remain optimized even after the two-opt optimization.

4. **Dijkstra's Shortest Path Greedy Algorithm:** Dijkstra's algorithm is a graph traversal algorithm used to find the shortest paths from a source vertex (hub) to all other vertices (package delivery locations) in a weighted graph. In the code, Dijkstra's algorithm is used to calculate the shortest route from the hub to each package's delivery location, taking into account the distances between vertices (locations).

Dijkstra's algorithm ensures that the calculated routes are truly optimized in terms of minimizing the distance traveled. This algorithm provides a more accurate measure of the shortest routes compared to the initial routes generated by the nearest neighbor algorithm and two-opt algorithm.

**Core Algorithm Overview:**

The self-adjusting heurist algorithm is executed by doing the following:

**Space Complexity:**

**Sorted\_Packages and Sorted\_Route:** Our implementation is dependent of the number of packages, The space complexity is O(n) where 'n' is the number of packages on the truck.

**Time Complexity:**

The outer while loop runs at most m times, where m is the maximum number of packages. In practice, this loop will run significantly fewer times as packages are delivered. The inner for loop iterates over n packages initially, but the number of iterations decreases with each package that is removed. In the worst case, it could iterate over all n packages. However, on average, the number of iterations is likely to be significantly smaller than n, especially as the truck gets closer to being empty. Inside the inner loop, there are constant-time operations such as distance calculations and comparisons. Finally, there's an operation to append packages to sorted\_packages and sorted\_route, which takes constant time.

Considering all these factors, the time complexity of the function can be approximated as O(m \* n), where m is the maximum number of packages and n is the number of packages on the truck. In practice, as packages are delivered, the number of iterations in both loops decreases, leading to faster execution

**Best and Worst Case Time Complexity:**

**Best Case:** The best-case scenario occurs when there is only one package on the truck. In this case, the outer while loop will run only once, and the inner for loop will iterate over the single package to calculate the distance and update the nearest package. After that, the function will append the package to sorted\_packages and sorted\_route. The function then adds the hub vertex to sorted\_route and completes

In this best-case scenario, the time complexity is O(1), constant time.

**Worst Case:** The worst-case scenario occurs when the truck is carrying the maximum number of packages (m) and there are many packages with long distances between them. This would result in the outer while loop running m times. For each iteration of the outer loop, the inner for loop will iterate over all n packages on the truck to find the nearest package. Additionally, distance calculations and updates are performed in the inner loop.

In this worst-case scenario, the time complexity is approximately O(m \* n), where m is the maximum number of packages and n is the number of packages on the truck.

**Dijkstra's Shortest Path Algorithm Overview:**

The greedy algorithm is executed by doing the following:

1. For each truck, find the final otimized routes for minimum distances traveled.

2. Utilize a min-heap as a priority queue to prioritize processing vertices with the shortest known distance. This queue is continually updated to ensure efficient shortest path determination during execution.

3. Iterate over neighbors of a vertex and calculate the distance. If the calculated distance is smaller than the current known distance, update the minimum distance.

4. The algorithm returns minimum distances for all addresses associated with each truck package.

**Space Complexity:**

**Priority Queue:** Our implementation a priority queue (min-heap) to manage vertices by their distances. The space complexity the priority queue is O(V), where V is the number of vertices.

**Best and Worst Case Time Complexity:**

**Priority Queue (Min-Heap) Operations:** The core operation in Dijkstra's algorithm involves extracting the minimum element from a priority queue (min-heap) and updating its neighbors. The number of times this operation is performed depends on the number of vertices and edges in the graph.

Best Case: O((V + E) \* log V), where V is the number of vertices and E is the number of edges. In the best case, the priority queue operations may be more efficient due to the specific distribution of edge weights and vertex connections.

Worst Case: O((V + E) \* log V), where V is the number of vertices and E is the number of edges. In the worst case, all vertices and edges need to be processed.

**Updating Neighbors:** In each iteration, the algorithm updates the distances of neighboring vertices. This operation involves checking the distances and possibly updating them.

Best Case: O(V), where V is the number of vertices. In the best case, only a few neighbors need to be updated.

Worst Case: O(V), where V is the number of vertices. In the worst case, all neighbors need to be updated.

Considering these factors, the overall time complexity of Dijkstra's algorithm in this context is O((V + E) \* log V). As of now the number of packages is very small and there is

.**Dijkstra's Algorithm Pseudocode:**

1. Takes two parameters graph object(associated packages with address acting as vertices and edge weights acting as distance to each address) and src the current location(by default always at hub)

2. Initializes 'distances' dictionary of all vertices in the graph to infinity, except for the src, which is set to 0.

3. A min-heap represented as a list of tuples(distances, vertex), is created to prioritize processing of vertices with the shortest known distance first. Initially, the src and distance of 0 are added to the min-heap.

4. Enters while loop that continues until the min-heap is empty. In each iteration, the vertex with the smallest distance is popped from the min-heap. If the vertex is already been visited and its current distance is greater than the distance stored in 'distances' dictionary then the iteration is skipped.

5. Popped vertex is marked as visited and its neighbors are iterated over. For each neighbor, the function calculates the distance from the source vertex to the neighbor by adding the weight of the edge connecting them to the current distance. If this calculated distance is smaller than the current known distance to the neighbor, the `distances` dictionary is updated with the new, smaller distance

6. Finally, the updated distance and neighbor vertex are pushed into the min-heap, ensuring that the neighbor vertex will be processed later with its updated distance.

7. Once loop is finished, function 'distance' dictionary will contain the shortest distances for packages to be returned

A. Hash Table

Hash Table Screenshot goes here

B. Look-Up Functions

Look-up function screenshot goes here

C. Original Code

Major code blocks screenshots go here showing implementation

C1. Identification Information

main.py screenshot goes here showing Student ID

C2. Process and Flow Comments

Some code blocks screenshots go here showing comments

D. Interface

Interface screenshot goes here

D1. First Status Check

Screen shot goes here

D2. Second Status Check

Screenshot goes here

D3. Third Status Check

Screenshot goes here

E. Screenshot of Code Execution

Screenshot goes here

F1. Strengths of the Chosen Algorithm

Text goes here

F2. Verification of Algorithm

Text goes here

F3. Other Possible Algorithms

Text goes here

F3a. Algorithm Differences

Text goes here

G. Different Approach

Text goes here

H. Verification of Data Structure

Text goes here

H1. Other Data Structures

Text goes here

H1a. Data Structure Differences

Text goes here

I. Sources

Text goes here

An example:

Lysecky, R., & Vahid, F. (2018, June). *C950: Data Structures and Algorithms II*. zyBooks.

Retrieved March 22, 2021, from <https://learn.zybooks.com/zybook/WGUC950AY20182019/>

J. Professional Communication