C950 WGUPS Algorithm Overview

Barnabas Bulpett­­­­

ID #001480927

WGU Email: bbulpe1@wgu.edu

January 2, 2022

C950 Data Structures and Algorithms II

# **Introduction**

The application determines efficient route and delivery distributions for a list of packages and corresponding location distance data, loaded from CSV files, for a package delivery service. Packages are sorted and distributed among three delivery trucks with the following restrictions:

* The service has 3 available trucks and 2 available truck drivers.
* Each package will have a unique identifier and the maximum capacity for each truck is 16 packages.
* Maximum truck speed is 18 miles per hour, and it is assumed that trucks have unlimited fuel will not need to stop.
* Each truck’s route must have no collisions with other truck routes and may leave the hub location starting at 8:00 AM.
* Trucks may return to the hub location to load additional packages.
* Packages may include one special note, indicating a contingency (delayed arrival at hub), a truck restriction (must be delivered on a specific truck), or a grouping restriction, meaning that the package must be delivered with certain other package(s).
* In addition, a package note may indicate that the wrong delivery address is listed. For packages with this designation, the delivery address will not be known until 10:20 AM and therefore cannot depart the hub facility until that time.
* If a package includes a delivery deadline time, it must arrive at the destination address prior to the stated time.

# **A. Algorithm Identification**

The primary algorithm used in scheduling package deliveries is Dijkstra’s shortest path algorithm. Given a graph of nodes and a starting node, this algorithm determines the shortest path from the starting node to each node in the graph. While doing so, a pointer is maintained to reference previously visited shortest path nodes.

For further clarification, this solution uses all three of the available delivery trucks:

* The first truck delivers only priority (early deadline) and packages annotated to be grouped. Upon completion, the driver of this truck returns to the hub facility so that they can drive the delayed truck (see “truck three” below).
* Truck number two delivers packages it alone can deliver, based on package notes. In addition, packages that are both delayed *and* have a priority deadline are loaded onto this truck. It departs the hub facility at 9:05 AM, once delayed priority packages have arrived.
* Finally, the “delayed” truck (truck three) departs the facility no earlier than 10:20AM, as it must await delayed packages and those for which an address is not yet known. No priority packages are loaded onto this truck.

# **B1. Logic Comments**

|  |
| --- |
| INPUT  CSV files for packages and delivery locations  User input via command line interface  OUTPUT Detailed information about package delivery status  READ Package data from CSV file and store in a chaining hash table  SET Objects for each Truck, defining requirements and limitations of each  SET Graph object to store distance data to and from each location node as a  collection of weighted edges  FOR each package object  Categorize based on deadline and annotated restrictions and load onto appropriate truck  FOR all packages in each loaded truck  Form a collection of corresponding location nodes based on address  FOR each loaded truck  Execute Dijkstra’s shortest path algorithm, beginning with the hub facility location to determine the shortest path for delivery to  all locations  FOR each package object  CALCULATE the estimated time of arrival (ETA) for delivery of each package and assign to the object’s delivery time property  CALL to prompt user for keyboard input  Show options menu and display package delivery details based on selection with appropriate handling for invalid input |

# **B2. Development Environment**

* The IDE used to develop this program was JetBrains PyCharm, version 2021.2
* The computer system used to develop this program was an Apple MacBook Pro using a 2.6 GHz Quad-Core Intel Core i7 processor with 16 GB of 2133 MHz LPDDR3 RAM.

# **B3. Space-Time and Big-O**

Overall program: O(n^2 log n)

Load CSV packages file: O(n)

Load CSV distances file: O(n^2)

Create distances graph: O(n^2)

Sort packages: O(n^2)

Determine shortest paths: O(n^2)

Schedule truck routes: O(n^2 log n)

Calculate delivery times: O(n log n)

Update package delivery data: O(n^2)

Display command line interface: O(1)

Display details for all packages: O(n)

Display all package details: O(n)

# **B4. Scalability and Adaptability**

The program is adaptable to a growing number of packages by allowing a specific truck to return to the hub to collect more packages. In addition, it ensures that any number of packages with deadlines or annotated constraints are be loaded onto the appropriate trucks.

# **B5. Software Efficiency and Maintainability**

Efficiency for the overall program is O(n^2 log n), where n is the number of packages to be delivered. Classes and modules of the program have been organized in a way that promotes maintainability, following object-oriented design principles. Object structures and related functions are grouped into their own classes and files. Service processes to be executed on the objects are categorized into their own files as well, containing only the required functionality for the execution of the process.

# **B6. Self-Adjusting Data Structures**

Packages are stored in a chaining hash table, which is a self-adjusting data structure that will expand to accommodate a large list of packages. However, this can also become a negative aspect since the list can potentially become very large and cause performance to suffer when iterating its contents (Preshing, 2011)1.

# **D1. Explanation of Data Structure**

The chaining hash table data structure used in this solution (named “Packages”), stores data using the **insert*(key, item)*** function where the arguments passed are a key and a value. Since keys must be unique, the function first searches the hash table for the key to see if it already exists. If so, it is updated with the given value. If it is not present, a new key-value pair is appended in the form of a 2-element tuple.

To retrieve package information, two functions are given in the Packages data structure. The first, **find(*key*)**, is used to find a single package given its key (corresponds to the package ID). In addition, the **all()** function will retrieve all packages currently present in the chaining hash table. The chaining hash table has a constant time complexity for insertion and lookup of data, however since the hash table expands automatically, this can increase to linear (O(n)) time if there are too many data collisions (Kankowski, 2009)2.

# **Screenshots**

# **G. Interface**

*User Interface*

Graphical user interface

Description automatically generated with medium confidence

# **G1. First Status Check**

*Provide screenshots to show the status of all packages at a time between 8:35 a.m. and 9:25 a.m.*

Graphical user interface

Description automatically generated with medium confidence

# **G2. Second Status Check**

*Provide screenshots to show the status of all packages at a time between 9:35 a.m. and 10:25 a.m.*

Graphical user interface

Description automatically generated with low confidence

# **G3. Third Status Check**

*Provide screenshots to show the status of all packages at a time between 12:03 p.m. and 1:12 p.m.*

Graphical user interface

Description automatically generated with medium confidence

# **H. Screenshots of Code Execution**

*Provide a screenshot or screenshots showing successful completion of the code, free from runtime errors or warnings, that includes the total mileage traveled by all trucks.*

Display all package data and total mileage traveled

Graphical user interface

Description automatically generated with medium confidence

Search for specific package and display status

Text

Description automatically generated

# **I1. Strengths of Chosen Algorithm**

Dijkstra’s algorithm has several benefits over other greedy algorithms. It has a time complexity of O(n^2), so it can be used with fairly large data sets without significant impact to performance. In addition, it is able to find the shortest path to all nodes using a single graph, without the need to construct multiple graphs (Jaliparthy Venkat, 2014)3.

# **I2. Verification of Algorithm**

*Verify that the algorithm used in the solution meets all requirements in the scenario.*

The following screenshot demonstrates that the algorithm successfully delivers all the packages for the use case in this scenario within the required limit of 140 miles.

*Graphical user interface

Description automatically generated with medium confidence*

# **I3. Other possible Algorithms**

* The **A\*** algorithm (which can be implemented as a variant of Dijkstra’s shortest path) could be used to this project.
* **Prim’s Minimum Spanning Tree (MST)** could be used to satisfy the project requirements and is an example of a greedy algorithm.

# **I3A. Algorithm Differences**

* The **A\*** algorithm may be slightly more efficient than Dijkstra’s due to its heuristic nature – that is, it only visits nodes which appear to be good choices to reach the target (Roy, 2019)4.
* **Prim’s Minimum Spanning Tree (MST)** bears some similarities the Dijkstra’s implementation used here, as it maintains sets of visited and unvisited nodes. The main difference is that Prim’s considers all edges between the two sets and connects them with the shortest edge distance (Geeks for Geeks, 2021)5.

# **J. Different Approach**

A different approach to the Dijkstra’s shortest path algorithm used in this solution would be to incorporate a *Fibonacci Heap* algorithm to store path nodes. Research shows that, although it is more difficult to implement and requires more space, it can be significantly more efficient than using lists or priority queues to store path nodes (LeetCode, 2019)6.

# **K1. Verification of Data Structure**

The “Packages” chaining hash table satisfies the requirements of this scenario by providing a flexible data structure to accommodate any number of data items using the **insert** function. The **find** function efficiently retrieves a single package data item (see screenshot above in section *H. Screenshots of Code Execution*). Moreover, the **all** function successfully and quickly retrieves details for all package data items (see screenshot above in section *G. Interface*).

# **K1A. Efficiency**

Barring any collisions (caused by the same index being used for the same key), hash tables can store and retrieve data in constant time. However, when collisions do occur, hash tables become slightly less efficient and in the worst case can become as bad as O(n) (linear time complexity) (Chresfield, 2019)7.

# **K1B. Overhead**

The Chaining Hash Table used in this solution can expand to accommodate any number of packages and as more packages are added, the space required will increase in a respectively to the number of key-value tuples stored. Although hash tables are considered a good solution for large sets of data, this overhead could impact performance for extremely large data sets. As Kankowski (2009) explains, “chaining requires more memory for the next item pointers” (Kankowski, 2009)2.

# **K1C. Implications**

When data indices correspond directly to unique keys, time complexity for the “Packages” structure will remain constant. This is dependent on the notion of a package being stored in its most atomic form to avoid index collisions, in other words “no multi-part packages”. The space complexity needed for this data structure is linear, O(n). Since the data stored in the given scenario is relatively small, the implications for memory usage are negligible. However, care should be taken to ensure that the size of an individual package object remains reasonable (Chresfield, 2019)7.

# **K2. Other Data Structures**

The Python dictionary abstract data type is data structure that could be used instead of the hash table for this scenario. A dictionary is part of the Python standard library and can be implemented without the need for manual setup.

# **K2a. Data Structure Differences**

Since items are stored in memory as an array of key-value pairs (Lerner, 2019)8, its memory consumption and time complexity (O(n)) is slightly worse than the chained hash table used in this solution. For this use case, however, the data set is not large enough to have a significant performance impact, and the programmer would benefit from the Python dictionary’s variety of helpful built-in functions (Lerner, 2019)8.

# **L. Sources - Works Cited**

1. Preshing, J. (2011, June). *Hash Table Performance Tests*

Retrieved December 24, 2021,

From <https://preshing.com/20110603/hash-table-performance-tests/>

1. Kankowski, P. (2009, March). *Hash Functions – An Empirical Comparison*

Retrieved December 1, 2021,

From <https://www.codeproject.com/Articles/32829/Hash-Functions-An-Empirical-Comparison>

1. Jaliparthy Venkat, R. (2014, December). *PATH FINDING – Dijkstra's Algorithm*

Retrieved December 1, 2021,

From <http://cs.indstate.edu/~rjaliparthive/dijkstras.pdf>

# **L. Sources - Works Cited (continued)**

1. Roy, B. (2019, September). *A-Star (A\*) Search Algorithm*

Retrieved December 11, 2021,

From <https://towardsdatascience.com/a-star-a-search-algorithm-eb495fb156bb>

1. "Geeks for Geeks”. (2021, November). *Prim's Minimum Spanning Tree (MST)*

Retrieved December 11, 2021,

From <https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/>

1. “sykKo” via LeetCode. (2019, April). *Dijkstra’s algorithm with Fibonacci Heap in Python*

Retrieved December 11, 2021,

From <https://leetcode.com/problems/network-delay-time/discuss/276349/dijkstra-algorithm-with-fibonacci-heap-in-python>

1. Chresfield, K. (2019, January). *Pros & Cons of Hash Tables*

Retrieved December 11, 2021,

From <https://medium.com/@kaelyn.chresfield/pros-cons-of-hash-tables-bc5d3097ffa7>

1. Lerner, R. (2019, May). *Python Dicts and Memory Usage*

Retrieved December 24, 2021,

From <https://lerner.co.il/2019/05/12/python-dicts-and-memory-usage/>