C950 WGUPS Algorithm Overview

Student Name: Bryan Yang

Student ID: #001303738

WGU Email: byang10@wgu.edu

Date: 12/06/22

C950 Data Structures and Algorithms II

# Introduction

This application program seeks to solve a business problem by implementing data structures and algorithms. Specifically, like traveling salesman problem (NP-complete problem), traveling trucks to load packages, and delivering them to different destinations with certain constraints given as the client requirements. This project implements and analyze high performance data structures and supporting algorithms including but not limited to self-adjusting data structures, hashing, dynamic programming, and set representation. This programming project was done using different python techniques related to memory management and data compression.

# A. Algorithm Identification

This application program uses Dijkstra as the main algorithm to solve the traveling trucks problem which is given as part of the class lecture resource and is called greedy algorithm. Please see a python file called “algo” for more details in the project folder.

# B1. Logic Comments

The project source codes have detailed comments for each function codes. This was done to explain the logic applied for each solution. The comments are in the above or below section of the code depending on the situations that were best suited for the most clarity in understanding the logic. Please see the python files for more details in the project folder.

# B2. Development Environment

This project was developed using PyCharm IDE in Python version 3.9 programming language. This application program runs on the local machine to provide portability and discretion. Network or SQL connection is required to run the code successfully. The project folder has two CSV files as data-frames for the project (WGUPS Distance Table.csv & WGUPS Package File.csv) which can be modified to add more data within the code or CSVs themselves with great flexibility to scale the project size.

# B3. Space-Time and Big-O

This project has detailed time complexity with Big-O notation on each block of code that is applicable as comments. The entire program has the upward time complexity (Big-O notation) of O(n^3).

# B4. Scalability and Adaptability

The chosen algorithm has an asymptotic behavior of O(E log V) meaning it can scale as the dataset grows. However, there may be a notable increase in the runtime as the dataset grows very large because the entire program’s Big-O notation is O(n^3). With the given dataset, the chosen data structure and algorithm are still optimal solution for realistic application, a delivery truck won’t have more than 400 packages for weight, space, and reasonable shift timeframe limitation reasons.

# B5. Software Efficiency and Maintainability

Despite the Big-O notation of the program being the O(n^3), the efficiency is still optimal range meaning it’s within maximum allowable distance traveled and meeting all the constraints/requirements. As mentioned above on the scalability section, it’s highly unlikely that a truck will carry enough packages to notice the exponential asymptotic behavior nature of the algorithm.

# B6. Self-Adjusting Data Structures

The self-adjusting data structure is a hash table in the program. This can be used to adapt any new data changes (addition, subtraction, or search). This data structure has a linear time complexity because its Big-O notation is O(n).

# C. Original Code

Please see the python files in the project folder for more detailed code and comment information of the project to assess the original code.

# C1. Identification Information

Each python file has author (student name), student ID, major with start date, student email address and date of the python file created.

# C2. Process and Flow Comments

All the python files have detailed comments to explain the process and flow of the code in the high and low level (top section and in between the codes) to meet this section requirement.

# D. Data Structure

This project used a hash table data structure because it’s well suited for the project scenario as described. Specifically, the hash table allows users to store unordered packages in the key-value pairs from the CSV files. The hash table has an insert time complexity of O(1). The search (look-up) and delete functions have a time complexity of O(n). Considering the size of the given data in the project scenario, this is still an optimal solution meeting all the requirements.

# D1. Explanation of Data Structure

The data structure using hash table is used to store package data in the key-value pairs via hashing each package ID in an empty array. Since every package ID is unique on its own, there is no collision without any duplication of hash. Insert function has a constant time complexity of O(1). Search (look-up) and delete functions have linear time complexity of O(n).

# E. Hash Table

The hash table used in the project include insert, search (look-up), and delete functions. The insert function being used as to hash package ID into an array or list. Search function being used as to look up those key-value pairs to find the appropriate match for the user. Delete function is used to delete packages from the hash table after the delivery. Please see the “hashmap.py” for more details.

# F. Look-Up Function

The search (look-up) function is used to look-up the package ID as the main parameter and display all data elements for a package. Please see the “main.py” starting in the line 42.

# G. Interface

The console or terminal ‘GUI’ interface give a detailed menus for users to navigate and use the application program. Please run the program and see the “main.py” for more details.

The main menu asks the user to enter a time in 24-hour format. The time is used to display a list of packages and the status of those packages based on the input time from the user.

Text

Description automatically generated

The second menu gives the user options to view complete package delivery list, searching for a specific package information, or exit the program entirely.

A screenshot of a computer

Description automatically generated with medium confidence

When the user chooses to search for a specific package, there is an additional menu with options to search specific package information based on the package ID. After the search is complete, the interface changes to the second menu.

Text

Description automatically generated

# G1. First Status Check

List of all packages between 8:35am – 9:25am (@9:00am).

Text

Description automatically generated A picture containing text

Description automatically generated

# G2. Second Status Check

List of all packages between 9:35am – 10:25am (@10:00am).

Text

Description automatically generated A picture containing text

Description automatically generated

# G3. Third Status Check

List of all packages between 12:03pm – 1:12pm (@12:30pm).

Text

Description automatically generated Background pattern

Description automatically generated with medium confidence

# H. Screenshots of Code Execution

The screenshots for the code execution.

Main menu display at 12:30am.

Text

Description automatically generated

After the initial menu was ran at 12:30pm, second menu appears with three options to choose from. Chose 2 to search for a specific package.

Text

Description automatically generated

A search menu appears with search options for further information based on the specific package ID. Address for package ID 9 displayed which was changed corrected at 10:20am to “410 S State St” from “300 State St”. Taken back to the second menu with options to choose from.

Text

Description automatically generated

Displaying option 1 from the second menu with list of all packages once the delivery is complete.

A screenshot of a computer

Description automatically generated with medium confidence

Displaying a completed pacakge delivery list.

Text

Description automatically generated Text

Description automatically generated with medium confidence

Below the completed package delivery list, the interface displays total distance traveled with miles and breakdown of each truck’s mileage. After which the application program exists on its own.

Text

Description automatically generated

# I1. Strengths of Chosen Algorithm

There are two strengths of the Dijkstra’s algorithm (Greedy algorithm). First, the algorithm is very optimal for this project since the algorithm can find the nearest path between two vertices in the graph. Second strength is its time complexity of O(E log V) meaning the algorithm can scale very well with bigger data-frame. The large number of packages will increase the runtime. However, realistically, trucks won’t be able hold much more than what was given in the project with many reasons such as shift timeframe, weight, and store limitations.

# I2. Verification of Algorithm

This algorithm was verified as shown on the screenshots in the section G through H where the application program was demonstrated thoroughly based on the given requirements. The program shows total miles, individual truck’s miles and most importantly timestamps of each package and its status at chosen the time.

# I3. Other possible Algorithms

Other algorithms can be used to meet the project requirements of this traveling salesman scenario. First is Bellman-Ford’s algorithm, this algorithm finds the shortest route to every vertex in the graph using one main vertex. Second algorithm is Prim’s Algorithm, it finds the minimal spanning tree. This algorithm builds a tree by choosing the least weight edge as it traverses the graph.

# I3A. Algorithm Differences

First other algorithm, Bellman-Ford’s algorithm differs from Dijkstra’s algorithm because Bellman-Ford supports the graph with negative edge weights whereas Dijkstra is positive. The time complexity also differs with O(EV).

Second other algorithm, Prim’s algorithm differs from Dijkstra’s algorithm because it finds the minimal spanning tree rather than finding the shortest route or path. Prim’s algorithm also finds the least weighted edge as it traverses the graph whereas Dijkstra’s algorithm works backwards to build the shortest path by accounting for the possible future paths.

# J. Different Approach

If I develop this project again, I think I want to try the Bellman-Ford’s algorithm because it would’ve been simpler to implement minus the time complexity being slower than Dijkstra’s algorithm. I also want to challenge myself to dynamically sorting packages and loading them onto the trucks based on the given restriction such as 10:20am delivery deadline for certain packages.

# K1. Verification of Data Structure

Each truck’s traveled miles are added to the total miles for the complete delivery of packages. The hash table implemented in the program has a search (look-up) function and the report is accurate. Please see the screenshots from G-H for more information of this requirement.

# K1A. Efficiency

A data structure using hash table works well with this project scenario since there are only three trucks delivering forty packages within a city. The time complexity is O(n) for the subject hash table search (look-up) and delete functions meaning data can be scaled linearly as additional packages are added to the dataset and flows through the program.

# K1B. Overhead

Despite the fact that insert function time complexity being the O(1). The computational time increases in a linear fashion based on the size of the input dataset given the fact that time complexity of the search (look-up) and delete functions is O(n). Therefore, the memory usage increases linearly as the dataset gets bigger with added data to the hash table. The bandwidth doesn’t need to be considered since the data files are stored locally in the CSV files.

# K1C. Implications

Since the program application has the time complexity of O(n^3) overall, scaling the program nationwide with various cities, trucks, and packages are not advised. If such use case is desired, optimizing the program with existing or new data structure and algorithm to find more optimal ways to avoid nested “for” or “while” loops will optimize the program further. Data structure such as binary search trees is needed since the function of the binary search tree has the time complexity of O(log n).

# K2. Other Data Structures

Other data structures can be used in this project such as Stacks (LIFO) and Queues (FIFO) to meet all the project requirements and based on the binary search tree data structure.

# K2a. Data Structure Differences

The binary search trees (BST) traverse other nodes via parent and child relationships. Each node in a BST has two child nodes at the most as references. This tree structure makes the tree well balanced and pre-ordered. Therefore, operation and use of this structure are faster than array list. It has the time complexity of O(log n).

The stack has the data structure of last in first out (LIFO) meaning the last item on the stack is the first one to go out. This data structure can be used to prioritize packages based on the LIFO accounting methodology by getting latest deadline package going out to be delivered first.

# M. Professional Communication

All the write-up in this overview document as well as the code comments are documented in a professional manner. Please see the python files for more information specific to code comments.

# L. Sources - Works Cited

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

Retrieved December 02, 2022, from <https://learn.zybooks.com/zybook/WGUC950AY20182019/>

Ashe, J. & Antonucci, A. (2020, March). *C950: Data Structures and Algorithms II*. Getting Started on the project – Long. Retrieved December 02, 2022, from

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=2ef1dff8-7cf5-4d9a-aa57-ad1900e87c2c>

Ashe, J. & Antonucci, A. (2020, March). *C950: Data Structures and Algorithms II*. Getting Started on the project – Short. Retrieved December 02, 2022, from

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=0112c9eb-10f5-42fe-853b-ab440168e97f>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar – Let’s go Hashing. Retrieved December 02, 2022, from

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=f08d7871-d57a-496e-a6a1-ac7601308c71>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar – Let’s go Hashing. Retrieved December 02, 2022, from

<https://westerngovernorsuniversity-my.sharepoint.com/:u:/g/personal/cemal_tepe_wgu_edu/EXaXbjKAci5EhnaWjPab6iMBc0zOUb_dOa_b-FwY4zeumg?e=1EN3Bl>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar 1 – Let’s go Hashing. Retrieved December 02, 2022, from

<https://westerngovernorsuniversity-my.sharepoint.com/:b:/g/personal/cemal_tepe_wgu_edu/EWjwOOwEwCdHsW83oQ2az00BkTCjFFiGXk8a6eXls74wLg?e=EhpKrF>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar 2 – Getting Greedy, who moved my data? Retrieved December 02, 2022, from

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=eee77a88-4de8-4d42-a3c3-ac8000ece256>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar 2 – Getting Greedy, who moved my data? Retrieved December 02, 2022, from

<https://westerngovernorsuniversity-my.sharepoint.com/:b:/g/personal/cemal_tepe_wgu_edu/EXzxjdXp41xBrIUCPPJne90Bpicz8Ss4Ao_QjPSNMkqfmA?e=dtO3Y8>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar 2 – Getting Greedy, who moved my data? Retrieved December 02, 2022, from

<https://westerngovernorsuniversity-my.sharepoint.com/:u:/g/personal/cemal_tepe_wgu_edu/EajOlRjaPoJFt1RRlqC94xUBp9IkW9tE5RQxX8CBtbPKwA?e=juK0Kc>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar 3 – How to Dijkstra. Retrieved December 02, 2022, from

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=aad71bd6-abf5-4cd4-8a78-ac7f01039c73>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar 3 – How to Dijkstra. Retrieved December 02, 2022, from

<https://westerngovernorsuniversity-my.sharepoint.com/:b:/g/personal/cemal_tepe_wgu_edu/Eb9uE_b8t5JGmrD0Smu6dfYBpQJwI7ygmWcUW7jLO_waWA?e=p2RcPv>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar 3 – How to Dijkstra. Retrieved December 02, 2022, from

<https://westerngovernorsuniversity-my.sharepoint.com/:u:/g/personal/cemal_tepe_wgu_edu/EYU77kH1kdpFl5Ay7J4g7x4BxUjMiSO4Jg906YR3GYZPIw?e=GB8eii>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar 4 – Python Modules. Retrieved December 02, 2022, from

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=a6e33b6d-9753-4ba4-a1b6-ac8000f5d250>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar 4 – Python Modules. Retrieved December 02, 2022, from

<https://westerngovernorsuniversity-my.sharepoint.com/:b:/g/personal/cemal_tepe_wgu_edu/EWR_MFMGFzVLgdTfstpUWY0Bm52QVcFZp6GHUKn5apXDHA?e=58vR7h>

Tepe, C. (2021, November). *C950: Data Structures and Algorithms II*. Webinar 4 – Python Modules. Retrieved December 02, 2022, from

<https://westerngovernorsuniversity-my.sharepoint.com/:u:/g/personal/cemal_tepe_wgu_edu/EYb_YoVd-h1EuicIv31KUEEB3JYm35Jyp_4_wSCXzXZmcw?e=rbH1VX>

James, J. (2014, May). *Bellman-Ford Single-Source Shortest-Path algorithm*.

Retrieved December 02, 2022, from

<https://www.youtube.com/watch?v=dp-Ortfx1f4&list=PLj8W7XIvO93oxLOZTi8JFghuRcKieIZU-&index=8>

James, J. (2014, May). *Prims Algorithm for Minimum Spanning Trees*.

Retrieved December 02, 2022, from

<https://www.youtube.com/watch?v=MaaSoZUEoos&list=PLj8W7XIvO93oxLOZTi8JFghuRcKieIZU-&index=5>

James, J. (2014, May). *Binary Search Trees (BST) Explained in Animated Demo*.

Retrieved December 02, 2022, from

<https://www.youtube.com/watch?v=mtvbVLK5xDQ>