C950 WGUPS Algorithm Overview

Daniel Crites

ID #002152169

WGU Email: dcrite5@my.wgu.edu

Date

11/16/2023

C950 Data Structures and Algorithms II

# Introduction

This project was to make a program for the Western Governors University Parcel Service(WGUPS) to help deliver packages in Salt Lake City, UT. The requirements were that we could only use three trucks, and two could only be out to deliver packages. The total mileage of trucks had to be under 140 miles, and any package restriction must be met. A user interface is used to inspect the package statuses and total mileage. The UI also includes a search for specific times a package was delivered.

# A. Algorithm Identification

The algorithm that I used to deliver packages is the greedy algorithm. I use the min function in Python to find the closest route throughout the delivery process. As packages are delivered, they are removed from a list until no packages are left. Once that happens, the truck returns to the hub from the last delivery.

# B1. Logic Comments

Trucks are manually loaded and assigned to a specific truck. Once loaded, the trucks begin their deliveries. The following pseudocode demonstrates how I used the greedy algorithm.

Trucks are manually loaded based on certain criteria.

loadTruck1 = manually load packages

loadTruck2 = manually load packages

loadTruck3 = manually load packages

A total mileage is used to keep track milage once a truck is done with its route

total\_mileage = 0

The manually loaded trucks are inserted into an algorithm to be deliver

delivery\_process(truck)

Packages of the truck is turned into a list and a copy of that list is make

packages = truck.packages

copy\_list = packages

packageIDs = package keys

All addresses and delivery address of current truck load is created into lists

Addresses = all addresses

Delivery\_addresses = address on truck

Current position is tracked and is based off an index of the address list

current\_position = 0

The package list is used to until all packages are delivered an the length is 0

While length of package list is greater than 0

A list of all mileage is created given the current location and then cleaned

Cleaned\_list = row and column combined

Using the current position, mileages are loaded for current addresses on truck

mileage\_list = mileage distances for current locations on cleaned list

The shortest route is the smallest numerical number on the mileage list

shortest\_route = The smallest number on mileage list

Collisions are checked on the current truck route and all addresses

Collisions\_on\_route = tracks duplicate on mileage\_list

Collisions\_on\_all\_addresses = tracks duplicate distance on all addresses

A for loop is used to check the mileages on the cleaned list

If the mileage equals the shortest\_route and no collisions on the route the package is delivered and updated` in the hash table.

Truck\_mileage += shortest\_route

update package status and delivery time

remove package from packages

If mileage equals the shortest\_route and there are collisions on cleaned the list

Then, compare items on the comparative list to items on the collision list.

If they match then update the correct packages on hash table

Truck\_mileage += shortest\_route

update package status and delivery time

remove package from packages

If mileage equals the shortest\_route and there are collisions on mileage\_list

Grab the first collision on the mileage list then update package on hash table

Truck\_mileage += shortest\_route

update package status and delivery time

remove package from packages

If mileage equals the shortest\_route and there are collisions on cleaned\_list and mileage list. Then grab the first collision on collision and update the package on the hash table

Truck\_mileage += shortest\_route

update package status and delivery time

remove package from packages

return truck\_mileage

total\_mileage += delivery\_process(loadtruck1)

total\_mileage += delivery\_process(loadtruck2)

total\_mileage += delivery\_process(loadtruck3)

# B2. Development Environment

This program used Python 3.10 and Pycharm for the IDE. The OS was Windows 11, and GitHub was the repository. Using GitHub allowed me to work on the project on two different machines: my laptop and desktop.

# B3. Space-Time and Big-O

HashTable.py

def \_\_init\_\_() = Space-time complexity O(1)

def insert() = Space-time complexity O(n)

def search() = Space-time complexity O(n)

def remove() = Space-time complexity O(n)

Total = O(n)

Main.py

def address\_list() = Space-time complexity O(n)

def package\_list() = Space-time complexity O(n)

def distance\_list() Space-time complexity O(n)

def load\_package\_data() = Space-time complexity O(1)

def delivery\_process() = Space-time complexity O(n^2)

def remove\_delivered\_package = Space-time complexity O(n)

def find\_indices() = Space-time complexity O(n)

def ui() = Space-time complexity O(1)

Total = O(n^2)

Overall space-time complexity = O(n^2)

# 4. Scalability and Adaptability

This program is scalable for larger deliveries due to its ability to handle collisions and determine how to update those packages. Its space-time complexity is primarily linear, and the package increase would not alter much. The program’s adaptability could be improved by creating a more specific hash table; the current hash table is limited in what it can do.

# B5. Software Efficiency and Maintainability

This software is efficient since most of its space-time complexity is linear. The program should be easily maintained because of minimal code to source through. This makes it easy to quickly find classes, functions, variables, etc, and alter them as needed.

# B6. Self-Adjusting Data Structures

Self-adjusting data structures are beneficial in that they can handle large and small amounts of inputs and account for pattern changes. Self-adjusting data structures can be easy to read and increase updates' efficiency. One of the weaknesses of self-adjusting data structures would be data collisions. With these collisions happening, the cost and prevention can be complex and increase time complexity. Hash tables provide constant time for searching, inserting, or removal. This allows for fast data retrieval and altering. A weakness of hash tables is that they cannot accept null values, which may not make them a good fit for specific development.

# D1. Explanation of Data Structure

I used the chaining hash table from zyBooks 7.8.2. This hash table structure uses a key and value system, where the key is unique. These data points in the chaining hash table allow for easy retrieval and altering of data. The hash table has the initializing capacity to control the data it can use. The insert function allows one to input new data or update existing data. The search function employs a Key ID to look up packages and returns the information associated with the Key ID. The remove function removes data from the hash table by inputting the Key ID. I used the package ID as the key to store the package information. These numerical values as keys make inserting, updating, and removing packages easier. The other components would have issues with duplicate values and could be confusing as keys.

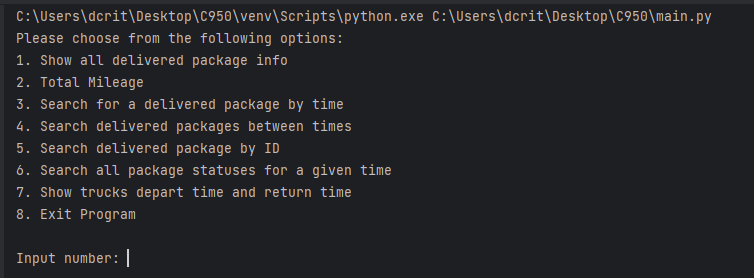
# E. Hash Table



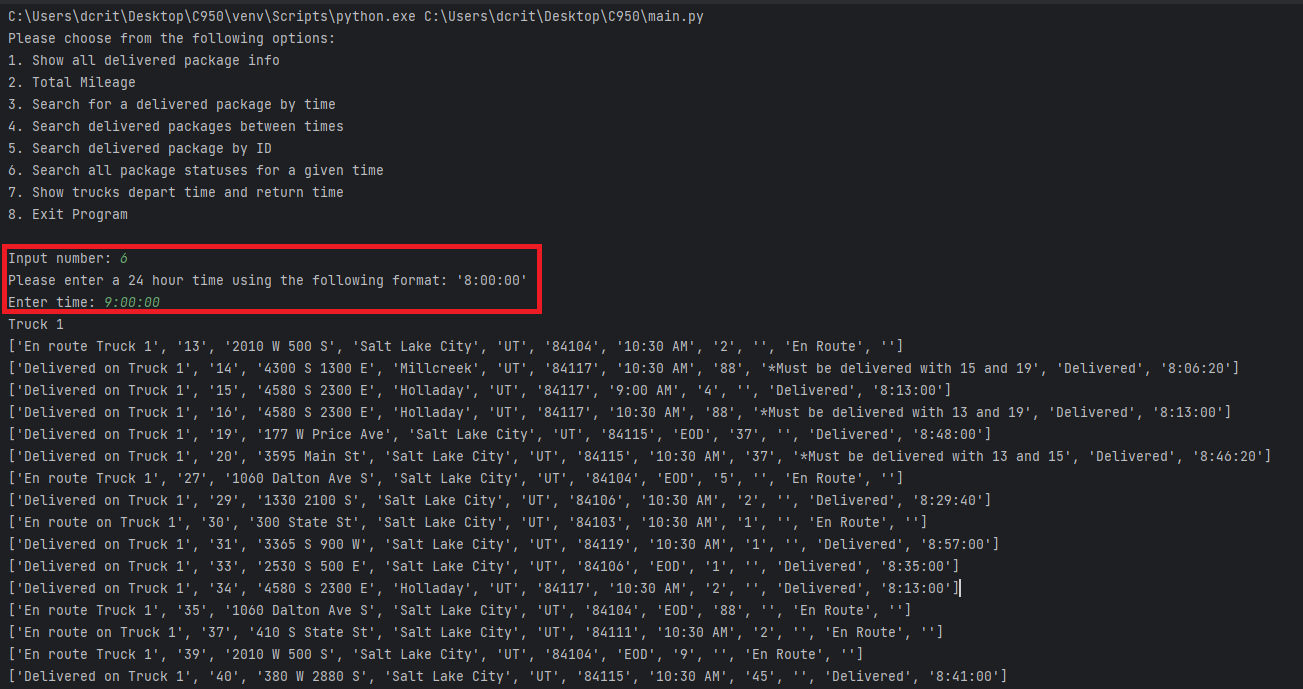
# F. Look-Up Function

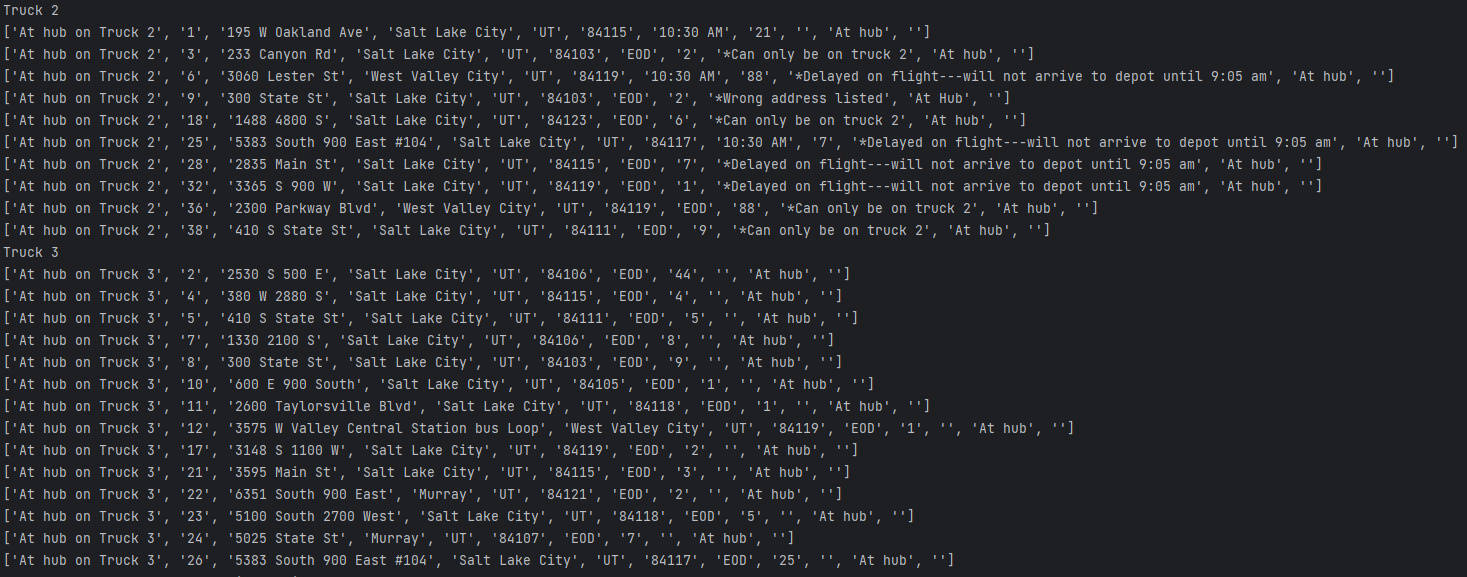


# G. Interface

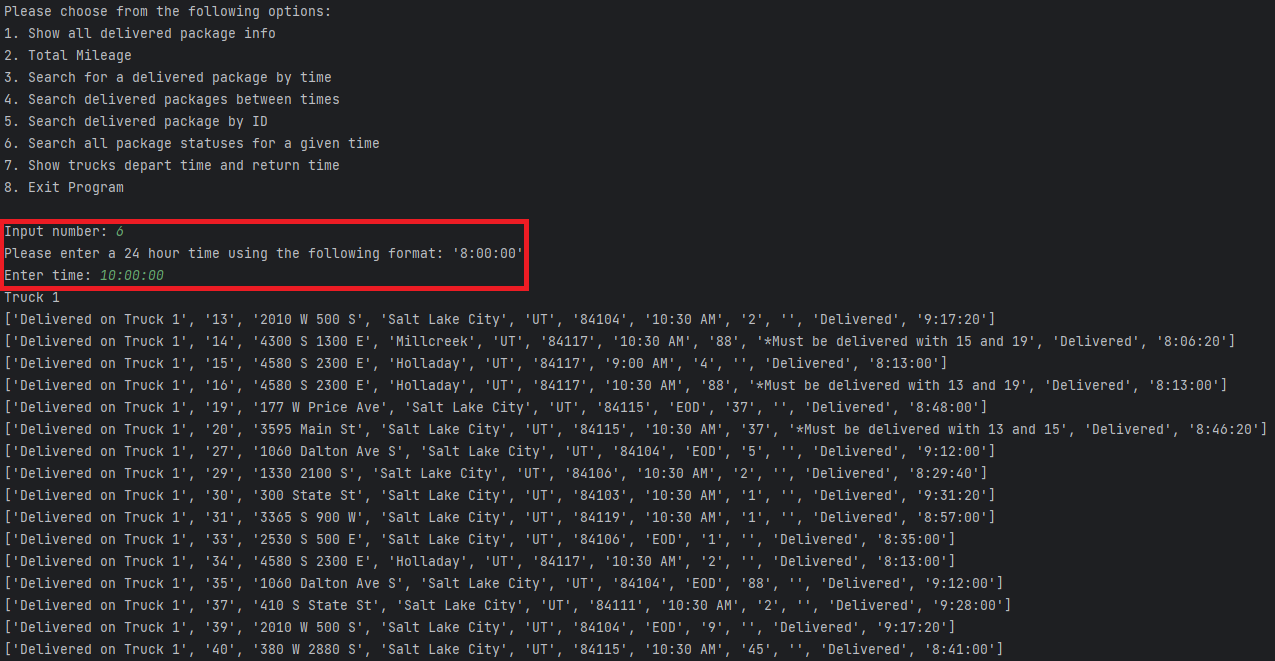


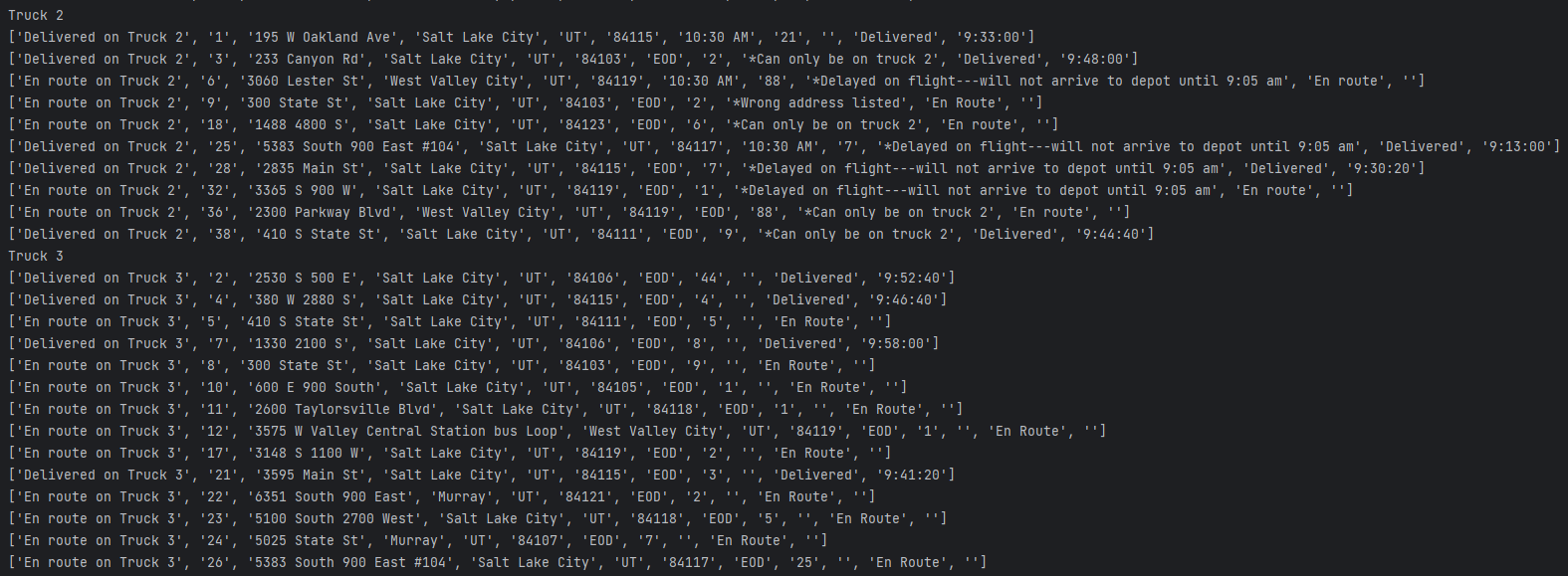
# G1. First Status Check



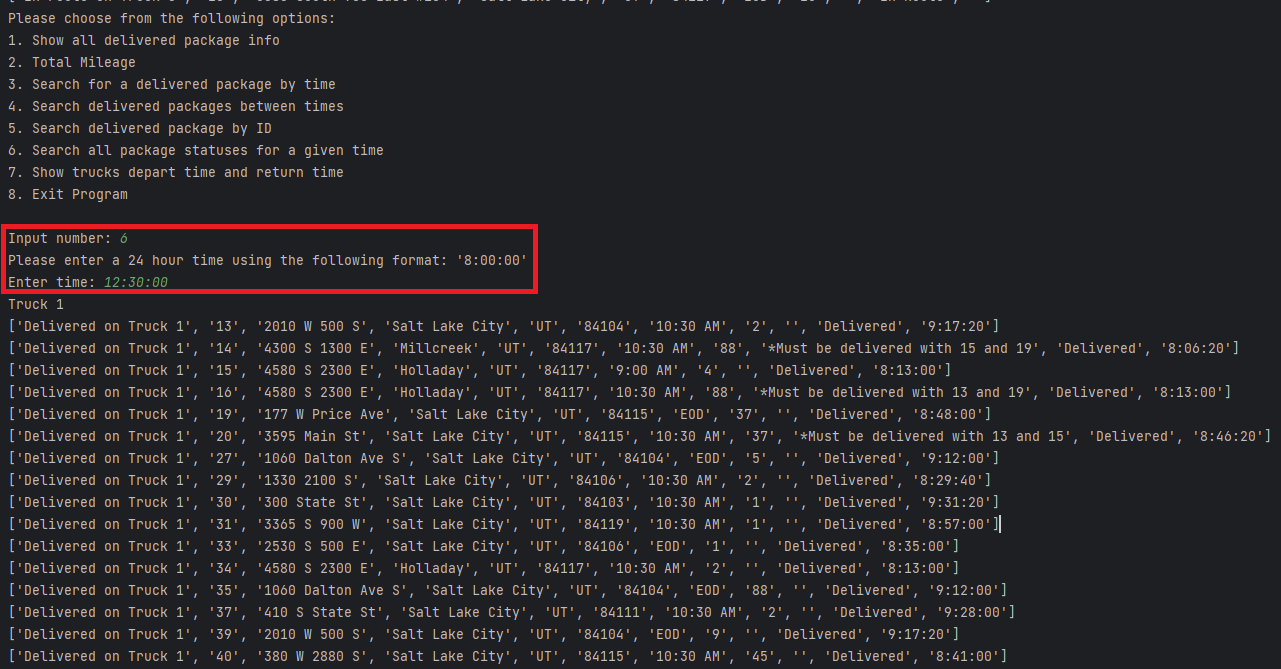
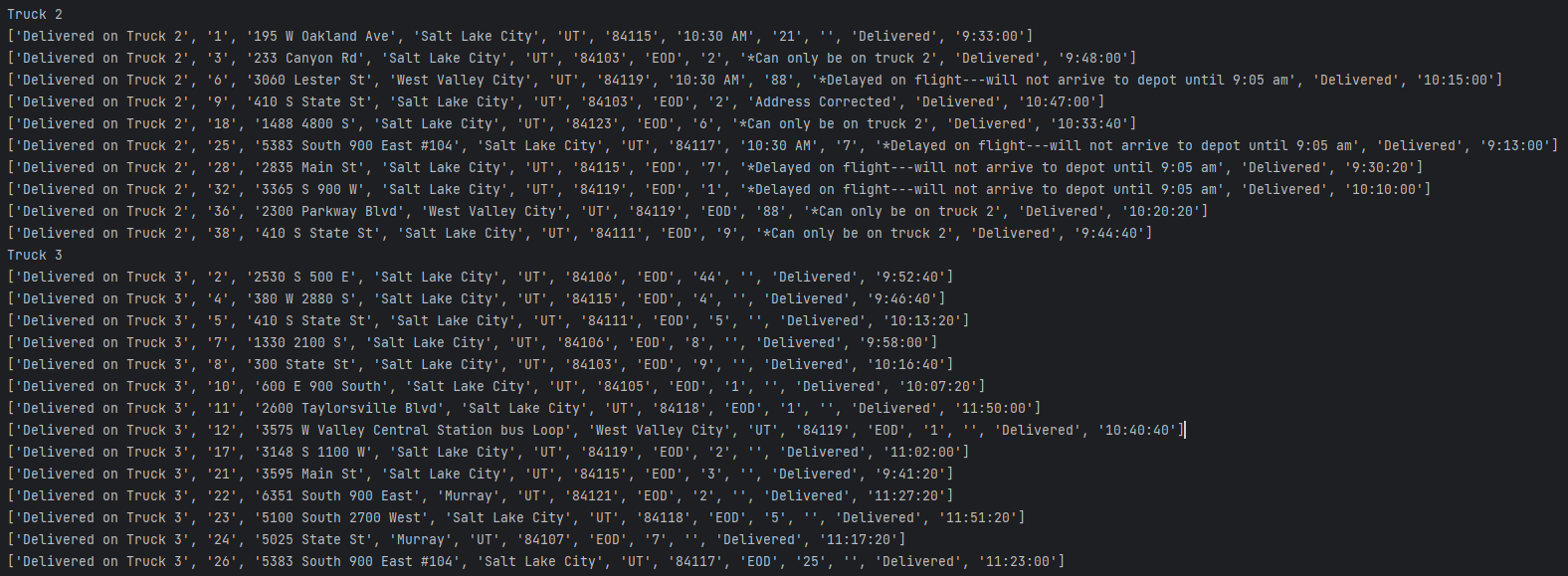


# G2. Second Status Check





# G3. Third Status Check

# H. Screenshots of Code Execution

A screenshot of a computer program

Description automatically generated

# I1. Strengths of Chosen Algorithm

I used the greedy algorithm to complete the project because of its simplicity. It finds the closest route from each location until deliveries are completed. The greedy algorithm scales well with a large or small amount of data. The efficiency is usually constant and makes it easier to predict space-time complexity.

# I2. Verification of Algorithm

My final mileage was 109.69, less than the 140-mile requirement. Package 9’s address was updated at 10:20 AM, and all packages were delivered on time. All packages were delivered to their specifications. All this can be viewed through the user interface.

# I3. Other possible Algorithms

Brute-force search algorithm

Farthest Insertion algorithm

# I3A. Algorithm Differences

One of the algorithms that I would use would be a brute-force search algorithm. This could be an optimal algorithm because it’s a guaranteed way to find the best solution. The other algorithm I would like to use would be the Farthest Insertion. This algorithm finds the farthest point from the starting location and repeatedly finds a point where it causes the smallest increase on the route.

# J. Different Approach

The first thing I would change would be to develop a function that automatically sorts packages in trucks and does not need to load them manually.

# K1. Verification of Data Structure

The total distance was 109.69, which is under the 140-mile requirement. All packages were delivered on time and to their specifications. An efficient hash table with a look-up function was used. The user interface reports all package's statuses at any given time.

A screen shot of a computer program

Description automatically generated

# K1A. Efficiency

The lookup function has a space-time complexity of O(n). Since n is constant, a large data set would take more time to search the entirety of the list. The more packages, the longer the list becomes and the longer it takes to search.

# K1B. Overhead

The space usage is only O(1) and would be consistent regardless of how many packages are added. This means the algorithm will continuously allocate memory at a constant of 1. The overhead is low for the given scenario.

# K1C. Implications

Adding additional trucks and cities would increase space and time usage. Since the look-up function is O(n), the larger the inputs, the more time it takes to find them. However, this would be an acceptable scalability feature that keeps time and space complexity reasonably low.

# K2. Other Data Structures

Binary Tree

Linked List

# K2a. Data Structure Differences

The binary tree data structure would work well with efficiency and searching large amounts of data since there can only be at most two nodes. The scalability would be better than the hash table I used for the project. The other data structure I would use would be a linked list. Linked lists allow for easy insertion and removal of data because the nodes are linked together. The linked list doesn’t require a fixed size or initial size. This would also be a better scalability option than a hash table.

# M. Professional Communication

None

# L. Sources - Works Cited

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

Retrieved October 22, 2023, from <https://learn.zybooks.com/zybook/WGUC950AY20182019/>