C950 WGU Routing Overview – Addison Ashworth ID: 001393715

**Problem:**

Deliver WGUPS packages using an algorithm to determine an efficient route. The two trucks cannot exceed a combined distance of 140 miles. After the algorithm determines the best route for our packages I will sort them further as there are only two trucks available to take the packages along with additional restraints (ie: delivery time, change of address mid route) that will not be represented in the algorithm. The output is the end result of our algorithm, as well as a solution and simulation of how the packages should be delivered. The following assumptions are considered when sorting the packages (taken from the project text):

•   Each truck can carry a maximum of 16 packages, and the ID number of each package is unique.

•   The trucks travel at an average speed of 18 miles per hour and have an infinite amount of gas with no need to stop.

•   There are no collisions.

•   Three trucks and two drivers are available for deliveries. Each driver stays with the same truck as long as that truck is in service.

•   Drivers leave the hub no earlier than 8:00 a.m., with the truck loaded, and can return to the hub for packages if needed.

•   The delivery and loading times are instantaneous, i.e., no time passes while at a delivery or when moving packages to a truck at the hub (that time is factored into the calculation of the average speed of the trucks).

•   There is up to one special note associated with a package.

•   The delivery address for package #9, Third District Juvenile Court, is wrong and will be corrected at 10:20 a.m. WGUPS is aware that the address is incorrect and will be updated at 10:20 a.m. However, WGUPS does not know the correct address (410 S State St., Salt Lake City, UT 84111) until 10:20 a.m.

•   The distances provided in the WGUPS Distance Table are equal regardless of the direction traveled.

•   The day ends when all 40 packages have been delivered.

**Algorithm Overview:**

A.  Identify a named self-adjusting algorithm that you used to create your program to deliver the packages.

In my program I chose to use the Nearest Neighbor algorithm.

B.  Write an overview of your program, in which you do the following:

1.  Explain the algorithm’s logic using pseudocode.

1. Get data from csv files
   1. Import package data
      1. Package data on each row converted to a package object
      2. Package\_id is used as hash for the self adjusting hashtable that it will be stored in
   2. Import distance data
      1. Distance data is stored in a custom distance graph object that has values representing vertices and directed edges
      2. All edges from excel file are duplicated in both directions as all are undirected lines
2. Initialize other data (creates truck objects that represent trucks and algorithm object that handles the nearest-neighbor algorithm)
3. Query user for input
   1. look up a package id, use their input as a hash to look up the package in O(1) time from our hashtable
   2. look up status of packages at certain time
      1. load trucks with predetermined packages (picked out from previous runs of nearest-neighbor algorithm plus some hand sorting)
      2. run each truck from start time (8 am) until end time specified by user
      3. print out all package data
   3. determine best route using algorithm
      1. nearest neighbor algorithm
         1. starts at hub
         2. determines nearest package using distance graph
         3. removes package from list, and adds it to final list
         4. moves current position to last package grabbed
         5. repeats 2 through 3 until package list is empty
         6. prints final package list (as list of package id ints)

2.  Describe the programming environment you used to create the Python application.

I used PyCharm Community Edition version 2020.3.5 with Python version Python 3.8.10 on a windows 10 computer. I used GitHub for version control.

3.  Evaluate the space-time complexity of each major segment of the program, and the entire program, using big-O notation.

**Hash Table**

**Inserting package into hashtable O(1)**

* hash package\_id O(1)
* insert at index O(1)
* increment int of filled slots O(1)
* check if hash table is getting full O(1)
  + if it is getting full, double the size O(1)

**Getting package from hashtable O(1)**

* hash package id O(1)
* retrieve package from index of hash O(1)

**Print package status O(n)**

* iterate hashtable linearly O(1 … n)
  + print status of each package object stored O(1)

**Distance Graph Dictionary**

**Calculating distance between two points O(1)**

* convert strings to vertices O(1)
* use vertices of two points as key in dictionary O(1)

**Nearest Neighbor Algorithm**

**Calculating Path given n packages O(n^2)**

* iterate through all points O(1 … n)
  + at each point, compare to all points not yet visited O(n … n – 1 … n - 2)
    - distance calculations take O(1) time

4.  Explain the capability of your solution to scale and adapt to a growing number of packages.

My program will scale in O(n^2) time with the number of packages, to get better results consider sorting packages within their own zip codes first

5.  Discuss why the software is efficient and easy to maintain.

Using a HashTable allows me to access any package’s information in O(1) time making it very fast to find the status of a package. Very easy (linear) to calculate storage needs based on number of packages expected in a day.

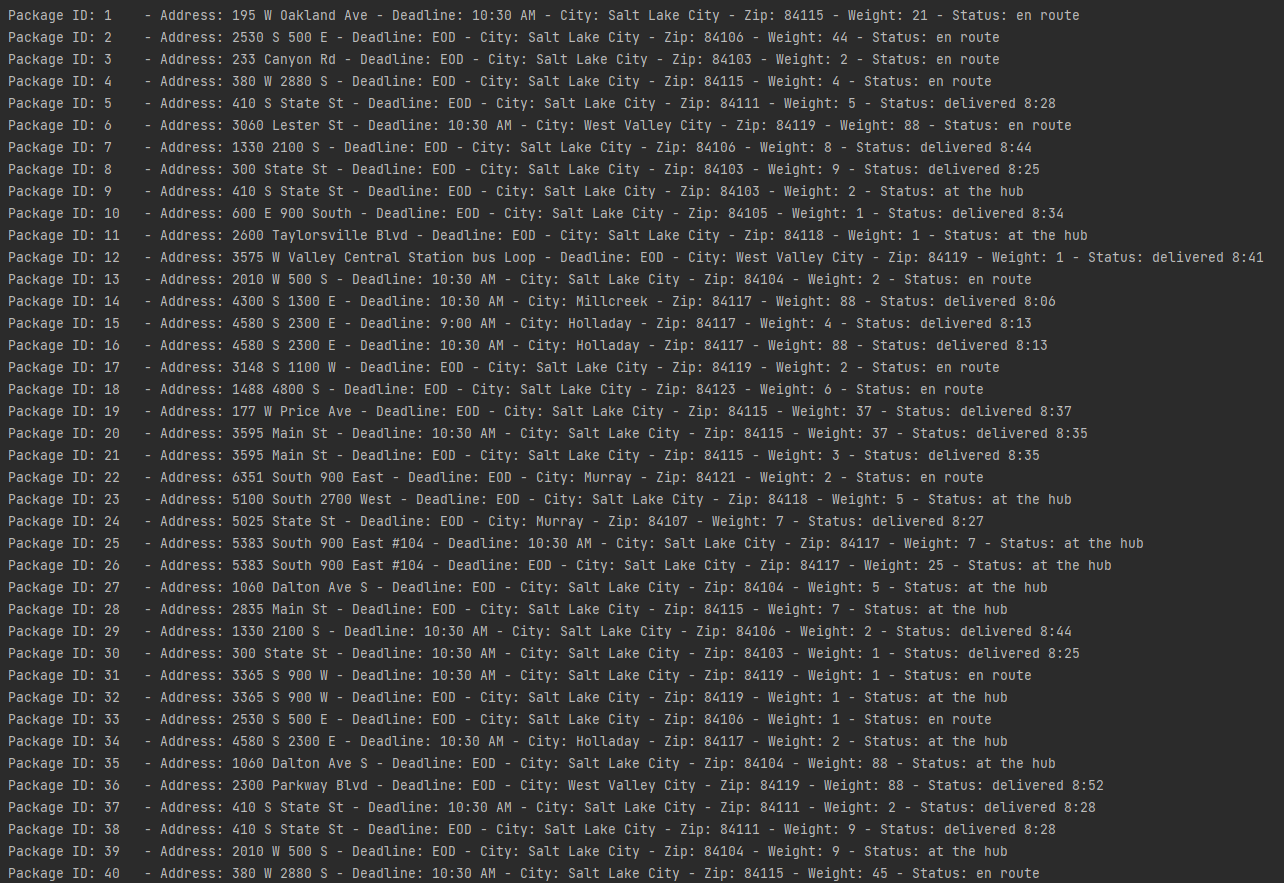
6.  Discuss the strengths and weaknesses of the self-adjusting data structures (e.g., the hash table).

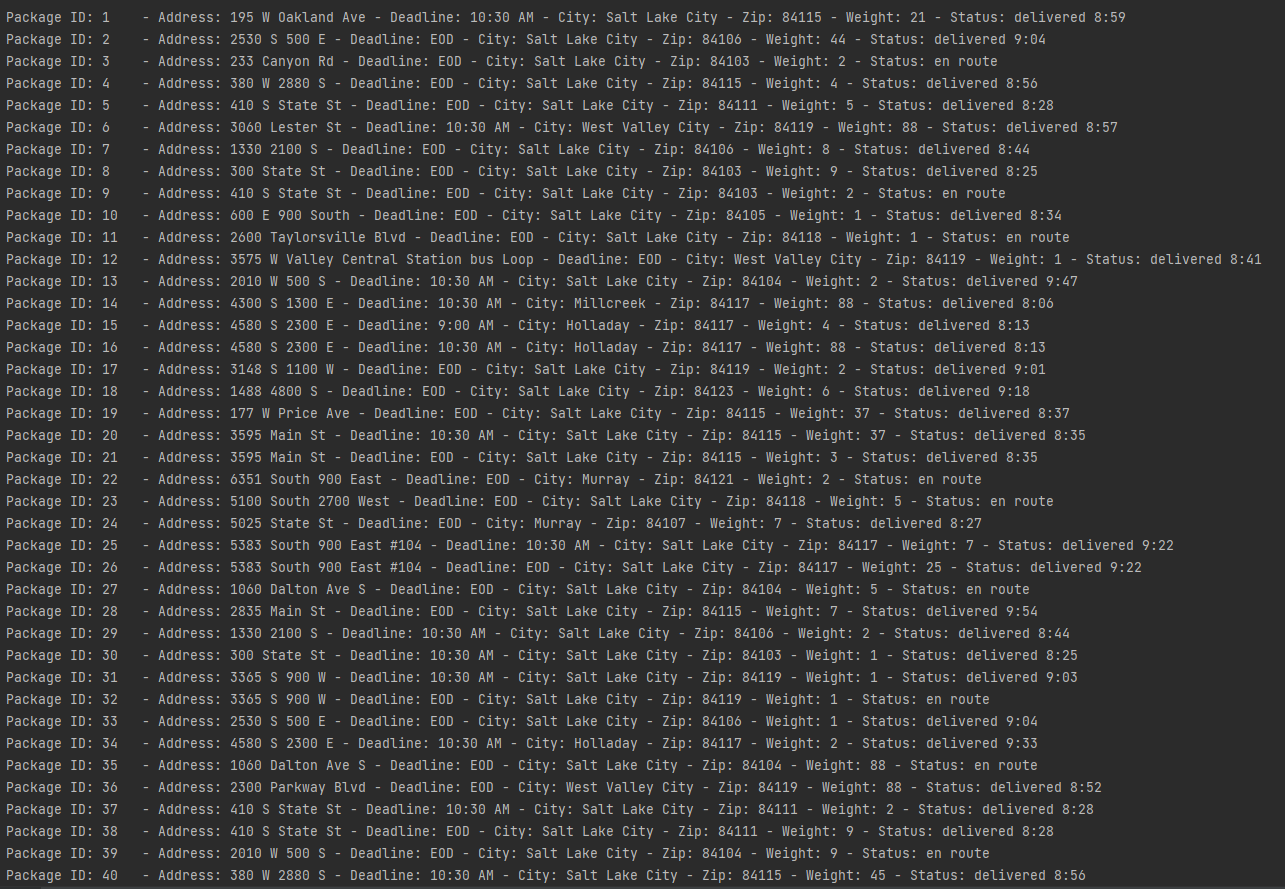
The HashTable allows me easy access to any package in O(1) time but isn’t efficient when id’s begin to be skipped, leaving holes in my dictionary.

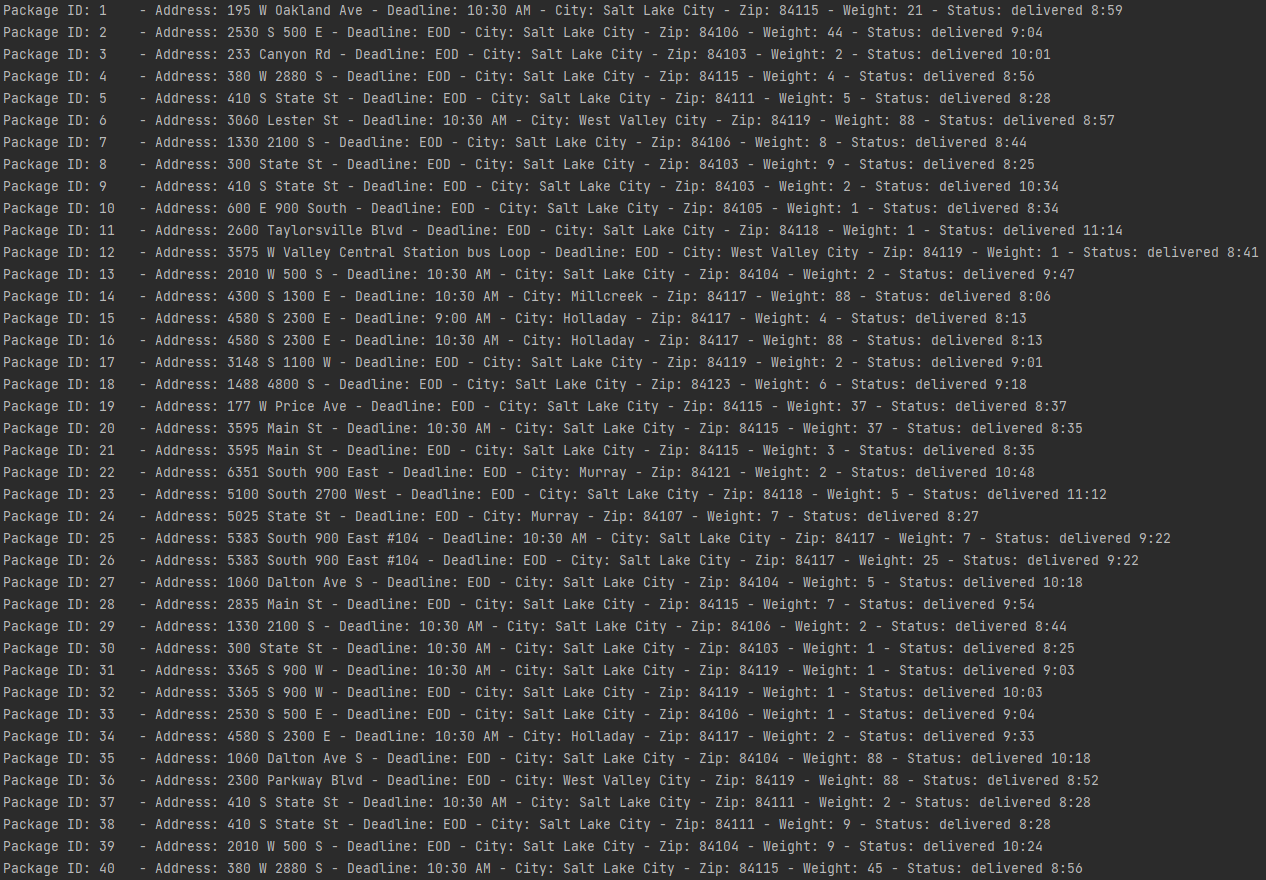
D.  Identify a self-adjusting data structure, such as a hash table, that can be used with the algorithm identified in part A to store the package data.

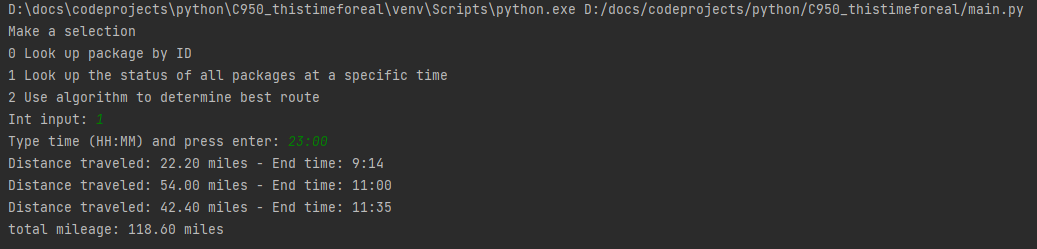
1.  Explain how your data structure accounts for the relationship between the data points you are storing.  
  
My data structure stores objects by hashable package ID, making it easy to access and package that you know the ID of.

G.  Provide an interface for the user to view the status and info (as listed in part F) of *any* package at *any* time, and the total mileage traveled by *all* trucks. (The delivery status should report the package as *at the hub*, *en route*, or *delivered*. Delivery status *must* include the time.)

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

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

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

H.  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.

I.  Justify the core algorithm you identified in part A and used in the solution by doing the following:

1.  Describe *at least***two** strengths of the algorithm used in the solution.

Its easy to justify to employee’s and manager’s. It just makes sense from a driver’s perspective, you deliver one package, then you go to the next closest and deliver that package.

Its easy to implement, the program simply selects the most optimal solution at the time, and then reevaluates with the remaining options until it has run out of options.

3.  Identify **two** other named algorithms, different from the algorithm implemented in the solution, that would meet the requirements in the scenario.

Brute force and cheapest insertion  
  
To brute force this problem would mean to examine every possible solution in order to be sure that you found the fastest route. This is not optimal because it takes an increasingly long O(n!) time the more paths there are to choose from, this would be very slow  
  
Cheapest insertion makes essentially a loop or “tour” that connects all dots in a circle. It considers the current loop and adds the next point to the loop that will keep it as small as possible, making the smallest “tour” it can from the selected starting nodes, good for truck routes because it calculates in the entire route, there and back, unlike the nearest neighbor algorithm I used.

J.  Describe what you would do differently, other than the two algorithms identified in I3, if you did this project again.

I would adjust the nearest neighbor algorithm to only look for neighbors within a zipcode similar to how graphics in video games are optimized by only rendering objects in a scene if you are looking at them. I would perform the nearest neighbor first on each zip code and then assign the zip codes to trucks for them to go to, that would hopefully save some processing power when WGUPS starts getting a lot of packages.

K.  Justify the data structure you identified in part D by doing the following:

1.  Verify that the data structure used in the solution meets *all* requirements in the scenario.

a.  Explain how the time needed to complete the look-up function is affected by changes in the number of packages to be delivered.

There is no change in time with any change in packages.

b.  Explain how the data structure space usage is affected by changes in the number of packages to be delivered.

The data structure increases as space is needed, doubling the size every time it is nearly full.

c.  Describe how changes to the number of trucks or the number of cities would affect the look-up time and the space usage of the data structure.

More trucks and cities would not affect the look up time of packages, but would linearly increase the space usage just like adding packages anywhere

2.  Identify **two** other data structures that could meet the same requirements in the scenario.

AVL tree or a linked list

1. Describe how *each* data structure identified in part K2 is different from the data structure used in the solution.

An AVL tree is a self balancing binary search tree that could provide O(log N) search times (zyBooks 5.1). Each package would be stored in a node that could potentially have two branches. the tree is self balancing to keep the depth of the tree at a minimum giving optimal search times.

A linked list is a node – branch relationship that can extend indefinitely, only using as much space as needed. Unfortunately due to each node being linked to the previous it requires O(n) search time to find what you are looking for, because worst case if your object is at the end of the line it could take O(n) time as you must iterate through every object in the list.

**References**

Project text and requirements

zyBooks c950 data structures and algorithms II home, accessed 2021