Greg Layton

ID #001189396

C950

A.  Greedy algorithm was used to compare the distances between the current location and the available “Next” locations to find the closest stop to go to.

B. The problem this algorithm is set to solve is as follows.

There are several packages on a truck to be delivered. We need a way to minimize the distance traveled between points of delivery. Using Python, I implemented a version of a “Greedy” algorithm.

1. Greedy algorithm overview

(The location the truck is currently at = start, A list of packages on the truck = list)

Loop to compare the package destinations to find the closest

Current closest = (An infinite value that will be replaced in the first loop)

For each package in the list:

Get the distance to the package’s destination from the start

If this distance is less than Current closest: save distance as Current closest

Save the current package as the next package to deliver

The next package to deliver is the package that is the closest

Code example:

Greedy\_Algorithm(Start\_Location, List\_of\_Packages):

Current\_closest = float(“inf”)

Next\_Package\_to\_Deliver = None

For Package in List\_of\_Packages:

Distance\_to\_Check = Get\_Distance\_Between(Start\_Location, Package.Location)

If Distance\_to\_Check < Current\_closest:

Current\_Closest = Distance\_to\_Check

Next\_Package\_to\_Deliver = Package

return Next\_Package\_to\_deliver

1. This program was created using JetBrains PyCharm community 2021.1.1 IDE. Using Python 3.7 on an AMD Ryzen 5 with 32gb of ram running windows 10.
2. All code methods are noted with their individual space-time complexities.

4.  Overall, the program is very scalable requiring average case O(N^2) when performing the Greedy algorithm comparisons. Most of the actions are broken into smaller calls to help limit the big O of any one function at a time. The main draw back would be the manual sorting of the packages. As it is now, there is no automatic sorting of packages to routes. Doing so would increase the big O but would actually be faster as a larger number of packages would not be able to be sorted by hand.

5.  For the most part, the software breaks each primary action into its own function. This is a pure example of the term “efficiency through simplicity” because in the absence of any multiple nested loops, the code becomes much easier to debug. Locating the incorrect data or data type being passed is easier when you locate the error to a single loop.

6.  Chaining hash tables are relatively simple to code compared to other data types. They still provide scalability and speed with fairly large number sets because it starts to slow only once there is a large number of items in the buckets. Keeping the number of buckets roughly equal to the length of the bucket lists would provide for optimal lookup speed.

C.  Please see the attached Python Project. Included in the folder are supporting documents “Salt Lake City Downtown Map,” a screen shot folder containing the required images for section G and H, “WGUPS Distance Table,” its .csv counterpart and the “WGUPS Package File” with its .csv counterpart.

1.  Please see lines 1-3 of main.py.

2.  All members of the code are commented at their location in the code.

D.  In HashTable.py you will find a chaining hash table that stores the data from the “WGUPS Package File”. This data is accessed and used to populate the sorted lists for delivery.

1.  The Chaining Hash Table uses a key value system at its base.

The unique key is used to identify an associated value.

This unique key is Hashed, arithmetically modified to alter, then inserted into an array.

In the case of hash collision, where multiple keys have the same value after hashing, this data structure stores the data inside another array at the hash value location.

E.  Please see the class HashTable.py in my project for this item.

F.  The look-up function is called from the main menu in Cli.py.

It takes in a number in range of the packages and searches for a matching package id.

If found, this will return the package info to the CLI.

Please see Cli.py line 50 for more information.

G. Cli.py provides for the Command Line Interface for this project.

The Run time-based report option satisfies the requirement for any package at any time.

1.  See 8\_45am\_report.png - Please find the screen shot in the screen shots folder.

2.  See 10\_00am\_report.png - Please find the screen shot in the screen shots folder.

3.  See 12\_30pm\_report.png - Please find the screen shot in the screen shots folder.

H.  See Screenshot\_for\_section\_H.png - Please find the screen shot in the screen shots folder.

I.  Justification for Greedy algorithm choice:

1.  The Greedy algorithm has an advantage of speed as it only is needed to iterate through the packages available and do a single comparison. This algorithm is also rather simple at its root. Though possibly beneficial when large data sets are used, complex algorithms are harder to code/maintain and require more system resources.

2.  See 12\_30pm\_report.png - Please find the screen shot in the “screen shots” folder.

At this time, all packages have been delivered. You can see all constraints of the packages listed in the notes of each.

1. Dijkstra’s algorithm or A\* could also have been used to deliver the packages with potentially lower total miles.
2. Dijkstra’s is used as a “brute force” method to determine the best route. This builds a path between every possible location and generates the shortest possible path between them. This is incredibly slow and consumes a lot of resources.

A\* is similar to Dijkstra’s Algorithm, but it differs because it considers “directionality.” If the distances continue to escalate in a branch, the algorithm will stop exploring every point on that branch and focus on the branch with lower distances. This saves time and processing power. A\* is overly complex to code compared to a Greedy algorithm.

J.  If I were to do this project again, I think I would like to create a better method for storing the data to parse into a .csv.

For example: if there is a delay known, a qualifier for it and an entry for the time available.

This would allow for dynamic sorting with minimal code.

K.  Justifying the use of the Chaining Hash Table identified in part D:

1.  Please see the code base to verify that all requirements for the scenario have been met.

a.  The look-up function is affected by changes in the number of packages to be delivered in a Big O(N^2) time complexity. As the data set grows larger, the complexity grows exponentially. O(N) for the search of Buckets and a nested O(N) to then search through that buckets list.

1. The data structure space usage is affected by changes in the number of packages to be delivered in a linear fashion O(N). Each extra item only requires one additional spot to be appended to a list.

1. The number of trucks would not affect the look-up time. However, the number of cities would affect the look-up time and the space usage of the data structure drastically.

Each city would need its own distance table defined so that the packages would not need to be compared to locations far outside of a sensible range. This would benefit from an initial run of the greedy algorithm using the hubs to determine which locations should be allocated to which hub.

2.  T**wo** other data structures that could meet the same requirements in this scenario are Graphs and Trees.

1. A graph is much like what is currently used for navigation software. It uses versions of A\* to locate the fastest path between objects while taking directionality into account. It saves resources and time by eliminating branches to check if they move away from the destination. This is a resource-intensive process and would have to be rerun every time a new location was introduced to create a new path table.

A Tree could be used to create a more specific route tailored to a set of locations. Because each node branches off from another, this structure would help limit the resources needed to search through. In comparison to the Greedy algorithm, this presents a much narrower field of locations to search through and still maintains acceptable performance.