DATA STRUCTURES AND ALGORITHMS 2

KYLE HOGUE

000931190

Program Overview

**A:** I used a combination of a Greedy algorithm and a Nearest Neighbor algorithm to sort and route the delivery of packages within the specified parameters. Package priority, special notes as well as total distance were taken into account. I used a Greedy algorithm to sort the packages into truck loads. I used a Nearest Neighbor algorithm to route the truck through package delivery.

**B:** The purpose of the algorithm as a whole was to deliver packages within the specified time range while taking special considerations (notes) into account all while achieving an acceptable level of efficiency (distance traveled).

**1.** I began with a hash table for the packages to be stored. This is O(n) in terms of complexity as the table adjusts with no modifications up to 100 packages rather than the given 40. Any more and a slight modification to add more buckets or a more complex hash function would be needed. The filling of the table with the packages is also O(n) (Lysecky & Vahid, 2018).

**hash\_table():**

**for x in range(10)**

**add\_bucket**

**fill\_table():**

**for line in csv:**

**insert(hash\_package)**

Once the packages were stored and accessible, sorting could begin. I began with a Greedy sort method to separate the packages between the three available trucks. It begins with getting a list of all available package IDs. This is O(n^2) complexity. The sorting itself is a series of O(n) complexity actions, slowly narrowing down where each package goes.

**get\_IDs():**

**for I in bucket:**

**for x in I:**

**get\_ID**

The first sort takes care of a specific set of special notes. Namely, a set of packages that must be delivered on a certain truck. This is O(n) complexity.

**first\_sort():**

**for in in list:**

**if special\_note:**

**truck4.add**

After the minor sort above for special cases, this is the first major sort. This sort is of O(n) complexity. Priority packages without notes are separated and the remainder are added into another truck load.

**second\_sort():**

**for id in list:**

**truck1.move(priority & no\_notes**

**truck3.move(everything else)**

This next sort is of O(n^2) complexity as it must compare each item in one list to each item in another list (Lysecky & Vahid, 2018). This adds packages with matching addresses into the primary load to facilitate efficiency.

**third\_sort():**

**for id in truck3:**

**for id in truck1:**

**truck1.move(matching\_delivery\_addresses & no\_notes)**

At this point, the primary load is complete. This sort separates the priority packages that have been delayed into another load. This section will be O(n) complexity.

**fourth\_sort():**

**for id in truck3:**

**truck2.move(priority & delayed\_notes)**

This last sort is of O(n^2) complexity as it must compare each item in one list to each item in another list.Similar to third\_sort() above, this section checks for other packages that can be delivered at the same time as the existing package list in order to improve efficiency.

**fifth\_sort():**

**for id in truck2:**

**for id in truck3:**

**if not specified for truck 2:**

**truck2.move(matching\_addresses & matching\_zipcode)**

The package manifests for each truck are then routed using a Nearest Neighbor method. This uses O(n^2) complexity. The priority packages, should they exist, are separated and delivered using the Nearest Neighbor algorithm. This ensures that package deadlines are met even with delays. Then the Nearest Neighbor algorithm is applied to the remainder of the packages.

**deliver\_packages():**

**separate\_priority\_packages**

**while manifest\_not\_empty:**

**while priority\_manifest\_not\_empty:**

**find\_closest()**

**deliver\_and\_record**

**remove\_package\_from\_manifest**

**find\_closest()**

**deliver\_and\_record**

**remove\_package\_from\_manifest**

**2.** I created the program using VS Code v1.62 and Python 3.9.6

**3.** (See above for complexity of specific parts) The overall complexity of the application should be O(n^2) at a minimum. The method I chose requires multiple comparisons from multiple lists. This is largely due to the unsorted nature of the package information. Pre-sorting priority packages, or special note packages could improve the complexity to O(log n) (Lysecky & Vahid, 2018).

**4.** As mentioned above, the hash table can accommodate more than double the number of packages provided without modification while maintaining the same complexity (Lysecky & Vahid, 2018). While it will continue to sort and route packages with reasonable efficiency, its accuracy will decrease with more variables. Example: The project specifies a max of sixteen packages per truck. My application doesn’t check for this. That said, it could easily be adapted by adding more checks or sorting characteristics.

**5.** Generous use of comments contributes the most to ease of maintenance. Often re-used code sections are separated into their own functions. Further while splitting the packages is done to the whole package list. The routing only takes into account the current truck inventory reducing the overall complexity.

**6.**  For the hash table I used a list of lists and key:value pair was stored in a tuple. One strength is ease of understanding. However it has many more weaknesses. Without being able to use a dictionary, we are also restricted from the many useful dictionary functions for traversal of the table. The hash table itself uses chaining but for such a small number of packages this is not required. It does however allow easy expansion for larger numbers of packages regardless of whether the sorting algorithm will break down or not (Lysecky & Vahid, 2018).

**C.**  (See main.py, My\_Hash.py, and Package.py)

**D.**  My\_Hash hash table and function uses a list of lists storing the hashed (key:value) pairs in tuples. The function itself uses a modulus operation on the package ID attribute to determine the bucket and then is added to the table. Chaining is not needed for such a small number of packages and actually increases the complexity of the algorithm but allows but expansion of larger numbers of packages. For such a small number using the unique package ID as the key would have worked more efficiently but seemed too simple (Lysecky & Vahid, 2018).

**E.** (See main.py, My\_Hash.py, and Package.py)

**F.** (See main.py, My\_Hash.py, and Package.py)

**G.** (See main.py, My\_Hash.py, and Package.py) Packages default to ‘at the hub’ status. Once on a truck they are considered ‘en route’. Once delivered, their status is updated to ‘Delivered (time)’

(See attached file ‘Delivery Status 9\_00am.png’)

(See attached file ‘Delivery Status 10\_00am.png’)

(See attached file ‘Delivery Status 12\_30am.png’)

**H.** (See attached file ‘Total Mileage.png’)

**I. 1.** My combination of Greedy and Nearest Neighbor algorithms for package delivery worked well given the Assessment limitations. The first strength is in using a simple choice for each part of the total process. Namely, the sorting and the delivery routing. This allows for each section to be changed independently and a solution found much quicker than if only one was chosen. A strength of the Greedy algorithm sorting is the ability to take many items into considerations and narrow the field. In this case, priority packages shouldn’t wait for special cases or delay issues. Then, with the field narrowed, further sorting can proceed. The Nearest Neighbor algorithm part of the delivery routing also benefits from such things like packages sharing an address. Which in turn can be brought into consideration of the sorting section.

**2.** (See attached file ‘Total Mileage + Package ID Lists.png’)

The file above provides the same overview of Part H. However I modified the software to include the truck manifests of package IDs. Comparing the special notes to truck departure times and final delivery times from Part G should verify all scenario requirements.

**3.** Two other algorithms that would provide a solution are a Depth-First Search (DFS) Algorithm or a Breadth-First Search (BFS) Algorithm.

**a.** The DFS algorithm would likely function by instead of sorting first it would chose a single path to a far delivery address, delivering packages along the way (nodes) then on the way back, stopping at ones slightly off the path (children). A BFS algorithm would trace a path more like a spiral around the delivery hub. This approach could be useful given smaller loads, as the nearness of the hub would allow for frequent restocking. A combination of both would likely yield the best results (Lysecky & Vahid, 2018).

**J.** The major change I would make, should I have to do this project again, would revolve around planning. I wrote a lot of code based on a quick scan of the package list and address map. Once such tests failed, I found that much of the code wasn’t applicable to alternative approaches and was wasted. Even now there are a few lines I can spot that are one off usages that could be much more efficient but exist now to cover previous mistakes. (One wasted patch line vs. having to re-write an entire section.)

**K. 1.**  My program contains a hash table and listed functions. I used only in-built data structures excluding a dictionary as specified.

**a.** The lookup function is O(n) complexity. It takes correspondingly more time given a larger number of packages to be delivered (Lysecky & Vahid, 2018).

**b.** My hash table is of consistent size using buckets for chaining. The usage of lists allows this to grow as needed up to a certain size (~100 packages) before software modifications are needed (Lysecky & Vahid, 2018).

**c.** My hash table won’t be affected by additional cities or trucks at all. Addresses are data attributes of packages and not relevant to data storage. Routing and distance to deliveries may affect routing but the data structure would be unaffected. The same reasoning applies to the number of trucks in use.

**2.** Linked Lists or a Tree could be used to fulfill the project requirements.

**a.** Linked Lists would work similarly to the provided solution but would scale better with a larger data set. While potentially using a list .insert() will be O(n) complexity, using a linked list will allow insertion at a constant O(1) (Lysecky & Vahid, 2018).

A Tree could allow consolidation of package data with address and distance data. Such as a nodes placement being contingent on distance from the hub. The affect this would have on the Big-O complexity is similar the Linked List insertions mentioned above (Lysecky & Vahid, 2018).

**L.**  Lysecky, R., Vahid, F. (2018). Data Structures and Algorithms. Retrieved from https://learn.zybooks.com/zybook/WGUC950AY20182019

**M.** (See Above)