C950 Algorithm Overview

Chloe Skorpikova - 001128673

Stated Problem:

The purpose of this project is to create an algorithm using Python 3.10 to assist Western Governors University Parcel Service (WGUPS) in finding the optimal routes for delivery trucks given a list of packages, their addresses, and additional criteria pertaining to each package. Due to a lack of consistency in WGUPS's ability to meet package delivery deadlines, this program is being implemented as a vital solution to not only these issues, but many more, including optimizing mileage traveled (and therefore, time spent) by each driver and truck. By utilizing this solution, WGUPS will see an increase in efficiency and ability to meet deadlines and adapt swiftly to adjustments.

Programming Environment: *Requirement B2*

The program was written using Python 3.10 in the PyCharm Community Edition 2021.3 developing environment on a Lenovo Flex laptop running Windows 10 Pro.

Algorithm Overview: *Requirement A, B1*

The routing function of the WGUPS delivery program utilizes both a greedy algorithm as well as nearest neighbor algorithm to produce the most optimal route for each truck to deliver. The greedy algorithm reads the package criteria and sorts the packages into trucks based on the best fit. Once the packages are loaded onto their respective trucks, each package address is considered in order to find the nearest deliverable address to the truck's current location. This assessment is repeated at each stop to ensure the truck travels the minimal amount of distance and therefore the minimal amount of time in order to meet package delivery deadlines.

The routing function is created in the following manner:

1. Create a table of all packages and note their addresses and criteria
2. Sort through all packages based on their criteria and place on the optimal truck
3. Set the truck's next deliverable package as that with the smallest distance between the package address and the truck's current location
4. Deliver package to location
5. Repeat steps 3 and 4 until all packages are delivered
6. Truck returns to hub

Package Sorting:

The package sorting function is based on any specialized notes provided for the packages through the comma separated value file. The packages are then sorted into respective lists based on the priority of packages, whether they are starting in house, paired, the correct address was given and more.

Package List Sorting Pseudocode:

```
for all packages in the package table:
        if package status is delivered:
                add package to delivered packages list
        if package status is en route:
                add package to en route packages list
        if package is delayed:
                add package to delayed list
        if package is paired, or is one of the mandatory pairs:
                add package to paired list
        if package deadline is before end of day:
                add package to prioritized list
        if package is required to be on a specific truck:
                add package to specific truck list
        if package has the wrong address:
                add package to wrong address list
        if none of the above apply:
                add package to standard list
```

Truck Loading:

Once the packages are sorted into their respective lists, the function then loads the trucks based on their start times, the package criteria and the truck capacity.

Truck Sorting Pseudocode:

```
if the truck is truck 1 and the truck is not full:
        load packages from paired list
        load packages from prioritized list
        load packages from standard list
if the truck is truck 2 and the truck is not full:
        load packages from specified truck list
        load packages from delayed list
if the truck is truck3 and the truck is not full:
        load packages from wrong address list
        load packages from standard list
```

Truck Routing:

Once all trucks are loaded, the function takes the addresses of the packages and finds the address which is nearest to the truck's current location and sets it as the next destination for the truck.

Nearest Neighbor Algorithm:

```
for each package on the truck:
        address = package's address
```

```
loc_num = package's address' location number
dist = distance between package's address and the truck's current address
if this distance is less than the previous package's distance:
        set this package as the next package for delivery
```

Programming Environment *Requirement B2*

This is a command-line interface in which the user interacts with a provided menu through the local console. The user utilizes the command-line interface to select options on the menu as well as provide input as needed. The program results are output to the console.

Space-Time Complexity *Requirement B3*

main.py:

Method	Time Complexity	Space Complexity
create_package_list	O(n)	O(n)
greeting	O(1)	O(1)
prompt	O(n)	O(n)

Distance.py:

Method	Time Complexity	Space Complexity
get_dist	O(1)	O(1)
get_loc_num	O(n)	O(n)
get_tot_dist	O(n)	O(n)

HashTable.py:

Method	Time Complexity	Space Complexity
get_hash	O(1)	O(1)
add	O(1)	O(n)
insert	O(1)	O(n)

lookup	O(1)	O(n)
fill_table	O(n)	O(n)
get_list_len	O(1)	O(1)

Package.py:

Method	Time Complexity	Space Complexity
get_id	O(1)	O(1)
get_address	O(1)	O(1)
get_status	O(1)	O(1)
get_spec_notes	O(1)	O(1)
get_deadline	O(1)	O(1)
set_status	O(1)	O(1)
set_address	O(1)	O(1)
is_prioritized	O(1)	O(1)
is_delayed	O(1)	O(1)
is_paired	O(1)	O(1)
is_wrong_address	O(1)	O(1)
is_spec_truck	O(1)	O(1)
load	O(1)	O(1)
deliver	O(1)	O(1)

Route.py:

Method	Time Complexity	Space Complexity
sort_packages	O(n)	O(n)
load_truck	O(n)	O(n)
load_trucks	O(n)	O(n)

unload_trucks	O(n)	O(n)
run_route	O(n^2)	O(n^2)
refresh_lists	O(1)	O(n)
get_all_packages	O(n)	O(n)

Truck.py:

Method	Time Complexity	Space Complexity
is_full	O(1)	O(1)
get_tot_dist	O(1)	O(1)
add_to_dist	O(1)	O(1)
return_to_hub	O(1)	O(1)
load_package	O(1)	O(1)
set_next_loc	O(n)	O(n)
get_next_package	O(n)	O(n)
unload_package	O(n)	O(n)
unload_truck	O(n)	O(n)
get_delivery_time	O(1)	O(1)
get_total_time	O(n)	O(1)
set_start_time	O(1)	O(1)
set_start_time_delta	O(1)	O(1)

Scalability and Adaptability

Requirement B4

As this program has a total time complexity of O(n^2), it is not necessarily the most scalable. As the amount of packages increases, so too would the time in an exponential manner. While the solution does offer minor increase in data size without much of a penalty, it is not the ideal algorithm for large-scale growth. Alongside the increase in time, the algorithm is suited specifically to the planned and understood specifications of the packages. It is not set up to be adaptable with specifications outside of those listed in the provided example. If any alterations were to fall outside of the accepted specifications, the program would no longer operate efficiently.

Efficiency and Ease of Maintenance *Requirement B5*

This program should be easy to maintain as needed through provided comments and conventional naming and formatting styles. The separate classes and methods should provide clear definitions and opportunity to easy alterations to any accessing programmer.

Self-Adjusting Data Structure *Requirement B6, D, K*

Hash Table

The hash table data structure accounts for the relationship between the data points by adjusting based on the size of the data and utilizing a unique key for each package, making searches linear rather than exponential, increasing optimization of the function. As long as the capacity of the hash table is accurate, the structure will automatically adjust and remain space and time efficient. Increasing the number of trucks or cities would maintain the same linear time and space usage.

Strengths

The strengths of utilizing the hash data structure is that the values each have a unique key identifier, making operations like lookup linear and highly efficient.

Weaknesses

The weaknesses of utilizing the hash table data structure is the space complexity as packages are added. The lack of scalability in adding more items at a rate of O(n^2) does not bode well for the future of the program as it is used by WGUPS.

Comparative Structures

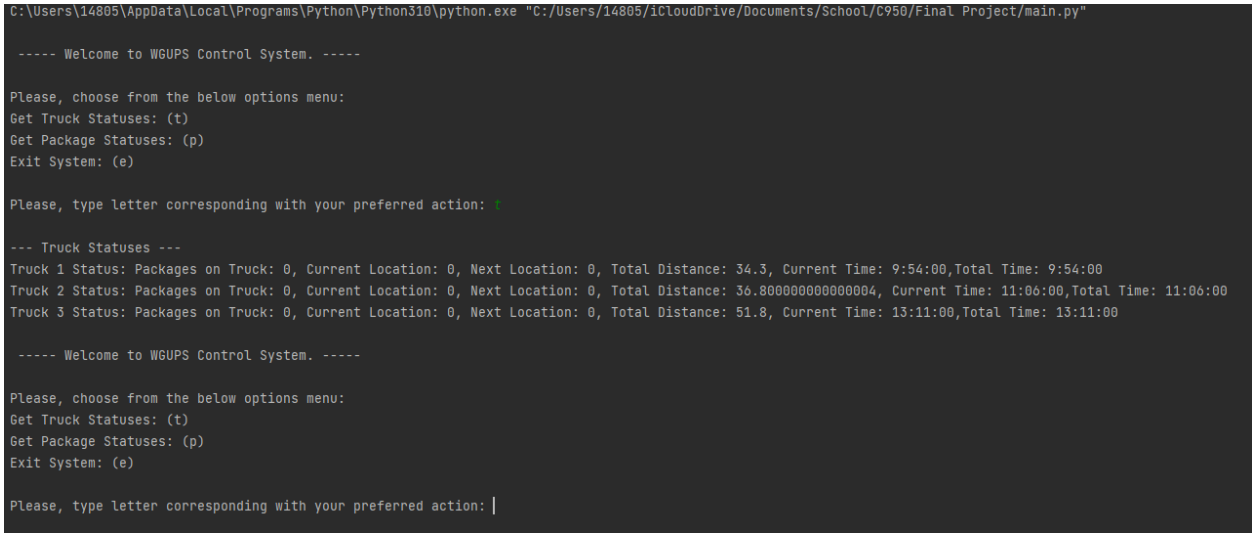
One other potential data structure could be locality based hash tables, with packages of nearby addresses being fed into the same hash buckets, allowing for better space complexity, but increasing time complexity.

Another potential data structure could be a k-d tree which would store the package addresses into a binary tree as nodes based on their locations. This would save on time complexity, but add to the space complexity of the program.

Screenshots of Package Status Based on Time *Requirement G*

Please see additional PDF attachment in WGU submission portal of console results based on three timestamp package status checks: 08:47:00, 09:47:00, 12:47:00.

Screenshots of Completed Code w/Truck Mileage *Requirement H*



Strengths of Algorithm *Requirement I1*

This algorithm's first strength is the optimal way it organizes the packages into trucks based on their criteria. This algorithm is also strong because of its ability to find the nearest deliverable package based on a truck's location. Both strengths combine to ensure that the least miles possible are traveled based on the adaptations needed for each package's unique situations.

Algorithm's Ability to Meet All Requirements *Requirement I2*

The algorithm is able to meet the requirements of directing the packages into the correct truck based on their individual needs. It also starts the trucks at the right time to fit both the package deadlines as well as the parameters that there are only two available drivers, certain packages are delayed and others have wrong addresses. This algorithm also ensures that all paired packages are delivered together. Lastly, the algorithm provides an optimized and self adjusting program that ensures the truck mileage stays under the allotted mileage.

Two Alternatives *Requirement I3*

One alternative algorithm could be using a k-d tree search, where each package would be a node in a binary tree with a location as a dimensional point. While this is a more time efficient method than that used, it is not as space efficient.

Another alternative algorithm would be to use dynamic programming by recursing the functions and adding the subproblems into a stored memory and utilizing those subproblems to complete the program. This approach might also be a more time efficient method, but would not be a more space efficient method.

What I Would Do Differently

Requirement J

In repeating the creation of this program, I would add more flexibility to the scheduling of the trucks in opposition to hard scheduling them based on the concrete parameters and criteria provided. In the future, I would want the program to be adaptable enough to retrieve the wrong address packages & the delayed information and utilize that to set truck start times. I would also seek alternative algorithms that provide better space and time efficiency than those selected, allowing for scalability.