**Core Algorithm Overview**

**Stated Problem:**

This project is designed to help the WGUPS (Western Governors University Parcel Service) determine an efficient route and delivery distributions for their daily local deliveries because packages are not being currently delivered by their promised deadline. The local delivery route has three trucks, two drivers and an average of 40 packages to deliver each day. Each package has specific criteria and delivery requirements. This project will be developed using the Python programming language and CSV files that have been provided containing package and location information.

**Algorithm Overview:**

1. A list of packages on the truck is passed in along with the associated truck number and current location
2. Current location is compared to the locations of all packages present in the truck to find the nearest location
3. Lowest value will be determined if all objects in the truck have been compared. The value is removed from the truck list, and the truck moves to that location
4. Algorithm is repeated with the new current location

The time complexity of the greedy algorithm is O(N^2) which scales well with 16 package limit per truck. Pseudocode is given below

truck\_list (Nested list of packages on a given truck)

truck\_num (The truck you are working with)

curr\_location (Recursive variable that is used to show where the truck is)

if length of truck\_list is 0 then return the empty list

set lowest\_value to 50.0

set new\_location to 0

if length of truck\_list is not 0:

for index in truck\_list:

if check\_current\_distance of curr\_location and the next index in the truck package list less or equal to lowest\_value:

new lowest\_value exists then update lowest\_value and current\_path

for index in truck\_list:

if check\_current\_distance of curr\_location and next index in the truck package list is equal to lowest\_value:

if truck\_num is equal to 1, 2, or 3

truck.append(current package)

truck\_index.append(current package index)

pop\_value = truck\_list.index(current package)

truck\_list.pop(pop\_value)

curr\_location = new\_location

find\_quickest\_route(truck\_list, truck\_num, curr\_location)

**Environment:**

I used the PyCharm community edition IDE to create this project. Using virtualenv (venv). The applications uses data from CSV files from the project folder on the local machine (Dell Windows 10 Pro, 8GB RAM computer). There is no need for a network connection because data exchanges are restricted to the interactions of the application and the local machine.

**Algorithms:**

I used the Greedy Algorithm to design the program to deliver packages

Greedy algorithm solves a problem by assuming that the optimal choice at a given moment will be the optimal choice overall. They tend to be efficient and produce optimal solutions for a lot of problems. They are also intuitive and tend to run faster than dynamic programming. The greedy algorithm performs all required functions within given constraints. This

algorithm can also scale with any set of data provided to it. An alternative to greedy algorithms would be to use a dynamic programming approach. It would run slower, and not be as intuitive as its greedy counterpart, but will still meet the requirements. A heuristic programming approach would also be able to meet these requirements though it may not always find the optimal solution as is usually the goal of the greedy algorithm. A heuristic approach accepts a non-optimal solution in order to improve execution speed.

**Data Structures:**

The data structures that were used in the algorithm were a list of lists that contained basic information. This data structure is easy to work with. Using a list made it convenient for easy and quick access to information and resulted in fast search, insertion and removal of data elements. Alternatives to this data structures would be trees or graphs. Graphs are made up of nodes (or vertices) and edges that connects one node (or vertex) to another. Trees are similar, but have more restrictions compared to graphs which include but are not limited to: no loops, only one path between vertices. One thing I would do differently attempting this project again would be to consider using Dijkstra’s algorithm on a graph data structure.

**Space & Time complexity**

HashTable.py

|  |  |  |
| --- | --- | --- |
| Method | Space complexity | Time complexity |
| make\_hashkey | O(1) | O(1) |
| insert | O(1) | O(1) |
| \_inIt\_ | O(N) | O(N) |
| get | O(N) | O(N) |
| remove | O(N) | O(N) |

TruckDistance.py

|  |  |  |
| --- | --- | --- |
| Method | Space complexity | Time complexity |
| get\_location\_address | O(1) | O(1) |
| calculate\_distance | O(1) | O(1) |
| calculate\_current\_distance | O(1) | O(1) |
| calculate\_time | O(N) | O(N) |
| find\_quickest\_route | O(N^2) | O(N^2) |
| truck1\_index | O(1) | O(1) |
| truck1\_list | O(1) | O(1) |
| truck2\_index | O(1) | O(1) |
| truck2\_list | O(1) | O(1) |
| truck3\_index | O(1) | O(1) |
| truck3\_list | O(1) | O(1) |

CSVReader.py

|  |  |  |
| --- | --- | --- |
| Method | Space complexity | Time complexity |
| delivery1 | O(1) | O(1) |
| delivery2 | O(1) | O(1) |
| delivery3 | O(1) | O(1) |
| display\_map | O(1) | O(1) |

Packages.py

|  |  |  |
| --- | --- | --- |
| Method | Space complexity | Time complexity |
| overall\_distance | O(1) | O(1) |

As can be seen from the table above, the space and time complexity of the methods are mostly linear with a good amount of constant time. This ensures the inputs can scale fairly easily.

**Sources:**

Learn.zybooks.com (n.d). zyBooks. [online] Available at: https://learn.zybooks.com/zybook/WGUC950AY20182019 [Accessed 29th March 2022]