Daniel Mudge
Student id: 004124446

Program Overview:

Use a self-adjusting Nearest Neighbor algorithm to plan truck routes of delivering packages

Logic

- Populate a list of packages to deliver for each truck
- Load the packages on the trucks
- Set the set time of the truck
- Delivery packages to the Nearest Neighbor
- Move the truck back to the hub
- Show the data to the user
- If necessary change the packages on the trucks to verify all packages can be delivered on time

Load trucks

listOfPackages = []

for p in listOfPackages

    truckPackages.Add(p)

Find Nearest Neighbor

For p in truckPackages

    Distance = getDistanceData(truckLocation,p.Address)

    If Distance < SmallestDistance Then

        SmallestDistance = Distance

Deliver Package

Move the trucks location to the neighestbor

Truck.Location = newPoint

truckPackages.remove(p)

Update Package Status

p.Update(Status, DeliveredTime, WasOnTime)

Programming environment

The Hardware used was:

Intel i7

32GB Ram

Software:

Windows 10

Visual Studio Code version 1.56.2

Python version 3.9.5 64-bit libraries

Space-time complexity of each section in Big-O notation

The worst case space-time complexity is commented in each major section of the code.

Capability to scale

The algorithm chosen can scale appropriately for a limited number of packages, but since the complexity of the algorithm at worse case is O(n^2), it may be ideal to look for alternatives if the number of packages per truck increases. The imported packages, addresses and distance address are designed to scale to any number of these items. The lookup function for these items is quick and efficient with space-time complexity of O(n). The GUI will not scale well without being written to handle more items.

Software efficiency and maintainability

On the current hardware the application loads the data and runs the algorithm in under 13 milliseconds. The code itself can be easily maintained it is well commented, modulated with re-used code pieces being functions or methods in a single spot to debug.

Strengths and weakness of self-adjusting data structures

The application uses a custom hash-table and a few lists to plan a truck route. The strengths of the hash-table are the look up time and reusability of the code for creating a hash-table. A few weaknesses of the hash-table are having to manually code it and creating iterations for it. Another data structure used was a list. Some strengths of lists are, they are easy to implement and easy to iterate through items. Lists also have drawbacks, looking up items requires iterating through and you can't specify a key an item in a list

Strengths chosen algorithm

One of the strengths of the nearest neighbor algorithm is it's simplicity. Given a tight deadline to complete a project like this, implementing this method makes sense logically and is easy to think through. Another strength is this can be scaled easily. If we add more addresses the algorithm can find the shortest path without updating any code in the codebase. The simplicity and scalability allows a person to easily update packages on loaded on each truck to test the route to ensure all packages can be delivered on time.

Other possible algorithms and algorithm differences

An alternative would have been to use a Greedy Algorithm that identifies the shortest path between all the points until a path is mapped. The key difference between this and the presented solution is this algorithm would identify the shortest distance between the paths before creating a route, while the solution presented routes each truck to the shortest point as it gets to the next point. Another alternative would have been using a 2-opt algorithm. This could have met the requirements by considering all the paths that wouldn't cross. This would ensure the truck doesn't have to cross a path it had gone through. The route may have been more optimal with this algorithm.

Different Approach

If I were to attempt this project again. I would want to update the nearest neighbor algorithm used to escalate package delivery based on the deadline. For example, given all the packages and the delivery requirements the algorithm can route the truck outside of the nearest neighbor to ensure the package is delivered on time.

Verification of Data Structure

The data structures used meet all the requirements. The total delivery distance is 117.36 miles, all packages were delivered on time. The codebase includes a hash-table with a lookup function. All data reported in the GUI is accurate and efficient.

Efficiency

The packages are stored in a hash table with the package id being the key. If we increased the number of packages the time to complete the lookup function in the hash table won't change if each key is unique

Overhead

As we add more packages to the hash table it will require more memory to run.

Implications

Add trucks would increase space as each truck has a copy of packages stored to it in the solution. The lookup time for packages on a truck would remain the same.

Adding cities (or more addresses) would increase the space requirements since these are stored in a hash table and increase the time since there exists a look up for addresses by address name which is not hashed and the alternative to this is an iterative loop through all addresses.

Other Data Structures and differences

The solution could have used a list or a set to meet the requirements. A list or set could have met the requirements by the package in either data structure. A key difference would have been searching for packages, in either case the package would need to be iterated through to get the correct lookup.