**DSA2 WGUPS Program Planning Part 1**

**Stated Problem:**

Determine an efficient route delivery distribution for WGUPS’s daily local deliveries. Write and present a solution where all 40 packages will be delivered on time while meeting each package’s requirements and keep the combined distance traveled by the three trucks under 140 miles.

**Part A**: Identify a named self-adjusting algorithm that could be used to create your program to deliver the packages.

* I will be creating an algorithm named findNearestNeighbor

**Part B:** Identify a self-adjusting data structure, such as a hash table, that could be used with the algorithm identified in part A to store package data.

* To store all package data, I will be creating a hash map to parse through a given CSV file containing package information. The ID listed in the CSV will be used as a key for the hash map and the corresponding key value will be a Package object with necessary package data as private fields.
* A hash map will ensure that a package list of N size will be able to be properly distributed within the data structure and allow for easy retrieval using the package ID value.

**Part C:** Write an overview of your program in which you do the following.

Explain the algorithm’s logic using pseudocode.

Nearest Neighbor Psedocode:

**nearestNeighbor(Current Address, Package List, Package Hash)**

**minDistance = 1000** <=minimum distance is initialized to a large value

**minPackageID = 0** <= package ID with the minimum distance is initialized to a value that does not exist as a Package ID

**for (Package ID) in Package List:**

retrieves the package object stored in the package hash with the package ID as key

**package = Package Hash.get(Package ID)**

Distance is calculated by the getDistance function which refers to the saved distance matrix to search for the distance between two addresses

**distance = getDistance(Current Address, package.address, distanceMatrix, addressHash)**

Sets the new minimum distance and package ID if it is lower than the current minimum

**If distance < minDistance:**

**minDistance = distance**

**minPackageID = packageID**

Function returns the packageID and the distance traveled to that package in order to calculate total distance traveled by the truck.

**return [minPackageID, minDistance]**

Get Distance Pseudocode:

**getDistance(Address A, Address B, Distance Matrix, AddressHash):**

Key A and B are integer values retrieved from a hash map of addresses in order to retrieve distance data from the distance matrix.

**keyA = AddressHash.get(Address A)**

**keyB = AddressHash.get(Address B)**

returns the distance between two addresses

**return distanceMatrix[keyA][keyB]**

Process Delivery Psedocode:

Process delivery lives inside the Truck class and processes the packages that are loaded in the truck.

**process Deliveries(self, Package List, Desired Time)**

while loop loops through the package list until it is empty

**while(len(Package List) > 0):**

Uses the nearestNeighbor function to get the closest package out of the remaining packages. Passes in the truck’s current location

**nextTargetAndDistance = nearestNeighbor(self.currentAddress, Package List)**

returned value contains miles in [1] and adds the miles to the total driven.

**self.milesDriven += nextTargetAndDistance[1]**

**r**eturned value contains packageID in [0] and removes it from the package list

**Package List.remove(nextTargetAndDistance[0])**

Adds the delivered packageID to the delivered list

**Self.delivered.appen(nextTargetAndDistance[0])**

**package = packageHash.get(nextTargetAndDistance[0])** retrieves the package from the package hash

**package.setStatus(“Delivered at:” + self.time)** <= sets the delivery status of the package to delivered including the current time.

Updates the truck’s current address to the address of the package that was just delivered

**self.currentAddress = package.address**

Package Hash parsers

**Loops through Package CSV**

**Create Package Object and Address Object for each row and add them to the corresponding hash map**

Address Hash/Distance Matrix Creator

**For every row in Distance CSV**

**Create Address Object for each row and add them to the corresponding hash map**

**matrixRow = []**

**For every column in distance CSV**

**Append cell data to matrixRow**

Mirrors distance data to their corresponding cells so that distance to and from different addresses can be easily retrieved

**If(cell data is not empty)**

**Also add the cell value to the distanceMatrix[column][row]**

**Append matrixRow to Distance Matrix**

Space-Time Complexity Analysis

Time:

* getDistance: O(1)
  + getDistance has a constant runtime. It retrieves two values from the Address Hash Map and uses the two values to retrieve distance between the addresses from the distance matrix
* nearestNeighbor: O(N)
  + The nearest neighbor algorithm has a run time of O(N) since it loops through the entire list of the remaining packages each time it looks for the closest neighbor
* processDeliveries: O(N^2)
  + Process delivery as a runtime complexity of O(N^2). The algorithm loops through the package list with a runtime complexity of O(N) since it loops N times in a list of length N. Each loop, nearestNeighbor algorithm is called which has a runtime complexity of O(N)
  + Multiplying the two runtime complexity gives us a algorithm time complexity of O(N^2)
* packageCSVParse: O(N)
  + the CSV parser for the package file has a runtime complexity of O(N) since it has to evaluate every package row once and add it to the Package hash Map. Adding to the hash map is a O(1) operation. The runtime of the entire algorithm is O(N)
* DistanceCSVParse: O(N^2)
  + The Distance CSV parse loops through every row in once. O(N)
  + On each pass, the address is parsed and added to a address hash map O(1)
  + In addition, every column is looped through per row to create the distance matrix and add the distance value to the corresponding cell of the matrix O(N)
  + Combining the O(N) and O(N) gives a total time complexity of O(N^2)
* Total: The total run time complexity can be simplified to a O(N^2) and it will depend on which CSV file is bigger. More packages means that processing deliveries will outgrow the address. More addresses means that the creation of the distance matrix will take longer.

Space:

Data structures that grow with the size of the data were the following

* Package Hash
  + O(N)
* Address Hash
  + O(N)
* Distance Matrix
  + O(N^2)
* Package List(Truck)
  + O(N)
* The total space complexity of the program is O(N^2) because the space for the Distance Matrix grows at a speed of N^2. The matrix has rows and columns for each distance value.

Explain the capability of your solution to scale and adapt to a growing number of packages:

* This solution to package management scales and adapts easily since each package data can be bundled into a Package object and stored in a hash map with just the packages’ ID as the key value. The program can easily retrieve the package and alter it using the ID value associated with every package.

Discuss why the software design would be efficient and easy to maintain:

* The software design is efficient since creating and adding the package object to the hash map has a runtime complexity of O(1). This means that the entire process of adding the item to the hash map would have a runtime complexity of O(N) since it just loops through the list of packages once.
* The design is easy to maintain because the hash maps can insert and access values with a run time of O(1). This means that any time the program must manipulate package data, the algorithm’s total run time won’t be impacted much. Something like an array or a list would require the program to execute a search that would take longer than looking for a value with a hash map.

Describe both the strengths and weaknesses of the self-adjusting data structure (e.g., the hash table):

* Strengths of a hash table structure is that it has a run time of O(1) for inserting and retrieving values.
* A weakness of the hash map is that it is non-ordered nor is it easily iterable.
  + This makes it to difficult to alter the data if every single package has to be alter in some way.

Justify the choice of a key for efficient delivery management from the following components:

* I chose to use Package ID as a key for efficient delivery management since it is a unique and random key that exists with every package. This means that the hash map will grow evenly for the most part as more and more packages get added.