Data Structures and Algorithms Report (Rubric sections B1, B3)

Austin Green

ID: #000946771

Email: [Agre313@wgu.edu](mailto:Agre313@wgu.edu)

04/27/2022

**Stated Problem:**

The purpose of this project is to write a program using Python in order to help the Western Governors University Postal Service (WGUPS) deliver a set of 40 packages to different locations in and around Salt Lake City, Utah in an efficient manner. The packages have certain requirements which determine the truck they can be loaded on, which packages they must be delivered with, what time they must be delivered, and what time they are able to leave the shipping hub. In addition to providing an efficient method of delivering the packages, users must be able to retrieve shipment information using a console interface.

**Data Structures and Algorithms Report (Rubric sections B1, B3)**

In this program, the packages are manually loaded on each of the two trucks available. However, the program is able to analyze these packages after they have been loaded and determine which route to take based on the closest location relative to its current position, also called the “nearest neighbor algorithm”. As mentioned earlier, the packages are first manually selected and divided among the two active trucks. The program then uses the current location of the truck and the destination location of each package and determines the distance the truck is from each location. Ultimately, the program selects the package with the least amount of distance between the two locations as the next one to be delivered. This can be represented in pseudocode as the following:

* **Create Truck objects to hold packages to be delivered**
  + **Load package objects manually to truck. Limited to 16 packages per truck.**
  + **Change delivery status of packages on truck to ‘in transit’**
* **Find optimal route**
  + **Use the current location of the truck, starting with the hub address, and run for loop on the packages loaded on truck to pull address information. Compare using distance table to find the distance between current truck location and package destination address.**
    - **Selection sort method used. As each package is evaluated for distance from the current location, the package with the smallest distance is saved to a placeholder variable and is replaced upon finding the next shortest distance.**
    - **The object in the placeholder variable is eventually moved to a new, ‘optimized list’ once it has been determined to be the package that is the shortest distance away and placeholder variable has data removed.**
    - **Original list has the package object removed from its contents**
  + **Time to travel between locations is calculated in seconds and added to the truck departure time.**
  + **Current location of truck is updated to the destination address of the package that was selected as the ‘nearest neighbor’**
  + **Delivery status of package is updated to ‘delivered’ and time of delivery is recorded on the package object.**
* **Process is repeated until all packages have been delivered from the truck.**

# Operation Time

There are a few different functions that are run to complete the task described in the pseudocode. The first is manually loading the packages to the trucks which can be represented by:

**Load Truck Function**

**Truck.add(package1)**

**Truck.add(package2) …**

**…**

**…**

**Truck.add(package16)**

This has a runtime of O(1) since each item is added individually and manually.

The next section utilizes a for loop to scan the address table to compare the index of the trucks current location and the package address of all packages to determine which package has the closest destination:

**Check Distance Function**

**Get index of current location of truck from addressTable**

**For address in addressTable:**

**If packageAddress == addressFromTable:**

**DestinationIndex == indexOfAddressFromTable**

**If addressTable[currentLocationInd][destinationLocationInd] == ‘ ‘:**

**Distance = addressTable[destinationLocationInd][currentLocationInd]**

**Else: Distance = addressTable[currentLocationInd][destinationLocationInd]**

**Return Distance**

As the function only a single for loop to accomplish this, the run time of this function is O(N). The next step in the process is to optimize the packages list based on the distances that were just found and to plan the route based on the nearest next package. This is accomplished using a **selection sort** method which utilizes a placeholder to hold the package with the current smallest distance while traversing the list, checking the other addresses of the packages on the truck. The logic of this step is as follows:

**Optimize Package List Function**

**BaselineDistance = high number the packages are guaranteed to be lower than**

**temporaryPlaceholder = None**

**optimizedList = new List [ ]**

**while length(optimizedList) <= number of packages on truck:**

**for package in packageList:**

**distance** = **CheckDistanceFunction(truck.currentLocation, pkg.address)**

**if distance < baseline:**

**placeholder = package**

**baseline = distance**

**optimizedList.add(package)**

**truck.currentAddress = package.address**

**reset Baseline to 0**

**if placeholder is in packageList:**

**remove placeholder from packageList**

**return optimizedList**

The way this works is by using a while loop to place a condition on the for loop to stop running when the number of packages in the optimized list reaches 16 packages. Since we have a for loop nested within a while loop, and the total number of packages is decremented each iteration, the time complexity of this function is O(n Log n).

The main function acts as the field space for all these functions mentioned above to be implemented and their results used in a loop of its own to perform the update and delivery of the packages. It is here that the optimized list for each truck is iterated over and the figurative action of ‘delivering the package’ is performed, the time needed to travel between locations is calculated, and the trucks current location is updated to that of the just-delivered package. The time complexity for these actions remains to be O(N) and the pseudocode for these actions is as follows:

**For package in truck.optimizedList:**

**travelTimeInSeconds = (((distance / 18mph) \* 60) \* 60)**

**updatedTime = truck.departureTime + travelTimeInSeconds**

**package.setDeliveredTime = updatedTime**

**package.deliveryStatus = ‘Delivered’**

**distanceToLocation = CheckDistanceFunction(truck.currentLocation, package.address)**

**truck.currentLocation = package.address**

**truck.totalDistanceTravel = totalDistanceTraveled + distanceToLocation**

Once all the packages from each truck have been delivered, the main function then finds which truck is closest to the hub location in order to determine which truck will return to the original hub location to pick up and deliver remainder of the packages. The truck that is closest to the hub location returns to the hub, the packages list for the truck is cleared, and the time it took to travel is added to the total time traveled for the truck. The truck is then loaded with the remaining packages and the steps explained above are reimplemented. The time complexity for this step is also O(N) as there is only one for loop.

**If truck1.distanceFromHub < truck2.distanceFromHub:**

**Truck1.totalDistance += truck1.distanceFromHub**

**timeToHubInSeconds = (((truck1.distanceFromHub/18) \* 60) \* 60)**

**truck1.updateTimes = truck1.updateTimes + timeToHubInSeconds**

**truck1.optimizedList = none**

**truck1.packagesOnTruck = none**

**truck1.loadPackages(remainder of packages at starting hub)**

**for packages on truck1:**

**set deliveryStatus = ‘in transit’**

**truck1.optimizePackageListFunction**

**for packages in truck1:**

**calculate travelTime to package address**

**update recorded time in seconds**

**set delivery status of package to ‘delivered’**

**calculate distance between current truck location and destination address**

**update truck1.currentLocation**

**update total distance traveled by truck1**

**else:**

**Truck2.totalDistance += truck2.distanceFromHub**

**timeToHubInSeconds = (((truck2.distanceFromHub/18) \* 60) \* 60)**

**truck2.updateTimes = truck2.updateTimes + timeToHubInSeconds**

**truck2.optimizedList = none**

**truck2.packagesOnTruck = none**

**truck2.loadPackages(remainder of packages at starting hub)**

**for packages on truck1:**

**set deliveryStatus = ‘in transit’**

**truck2.optimizePackageListFunction**

**for packages in truck2:**

**calculate travelTime to package address**

**update recorded time in seconds**

**set delivery status of package to ‘delivered’**

**calculate distance between current truck location and destination address**

**update truck2.currentLocation**

**update total distance traveled by truck2**

By examining the time complexity of its parts, we can also deduce the time complexity of the program as well by combining the sections listed above and using the combined time complexities of the parts to find what makes up the whole. We can see that the time complexities of the program’s parts are as follows:

**O(1) + O(N) + O(N Log N) +O(N) + O(N)**

**O(3N + N Log N + 1)**

**Environment (Rubric section B2):**

This program was written using the PyCharm IDE runtime version 11.0.13 and is written in Python 3.7 programming language. This application is written to run locally on the computer and is not utilizing any external databases or hosts. Data used is imported in the form of a CSV file and the CSVReader function in Python is used to parse the info.

**Scalability (Rubric section B4):**

The program is written to account for a maximum package capacity on the trucks, which limits the program to using the type of truck specified in the project details, however, the number of overall packages can be scaled to include as many deliveries as needed. Coupled with the run time of the program mentioned above, the application would be able to handle a fairly high number of deliveries before seeing any performance declines.

**Efficiency and Maintainability (Rubric section B4, B5, B6):**

Based on the calculated time complexity of the program, I would argue that this program is efficient in its programming. The only limitation I see is not based on the program but on the physical truck capacity which is out of the control of the program. The number of total packages is scalable and able to include as many as needed to still maintain the speed needed for fast results. As for maintainability, the CSV files used were specific in the information it gave, so as long as the same information is given in the same format, it can be uploaded to the program and used to perform the same calculations, making maintenance slim to none on the program itself. There might be editing needed for any info that is being uploaded to the program, but that is outside the scope of the actual program.

**Data Structure Analysis and Justification (Rubric section K):**

In this program, a hash table was built from scratch instead of using the built-in Python function. The strengths of using this type of data structure are that allows the number of items to be scaled to the need of the user, meaning as WGUPS grows their business and have more packages to deliver, the data structure will not need to be changed to meet those needs. In addition to being scalable, another strength of this data structure is the ease of identifying an item using only the key in the key value pair. This allows for easy lookups of the items and limits the amount of data that is needed to be stored throughout the program to only a single integer, rather than an entire object. A weakness of using a hash table is that if the exact key is not known for the item, it become more difficult to be able to find a specific item in the table. Much of the information about the object is stored within the value of the key value pair, which is not information that is able to be used as a search parameter. In this situation, the package ID’s, which were chronological and in order, which would not always be the case, especially when scaled to work with more information, such as a database. If a database were ever planned to be used with a hash table, it is necessary to look at the hash function, as well as how the database is configured. The reason for this is that hash tables can cause issues if there are many collisions of information, the program becomes inefficient, and the database can degrade due to this. One last weakness of using a hash table is that this data structure will not accept ‘null' as a key’s value, as it is not a value nor is it unique.

**Justification of Algorithm (Rubric section I):**

In this project, the nearest neighbor algorithm was used to find a solution to the problem. A strength of this algorithm is that the package with the closest delivery address to the current location of the trucks is selected, which keeps the distance of each individual trip short. This solution is also straight forward in its solution and is easy to implement. However, just because the closest address to the current location of the truck is used, this does not necessarily mean that this is the shortest overall distance for the truck to travel. Since the next destination is selected based on the current location of the truck, it is a short-sighted solution that only solves the immediate problem instead of looking at the entire situation, leaving us with a total distance that is more than likely sub-optimal.

Another approach that could have been used here is a dynamic programming approach, which would use a recursive function to calculate the distance between locations and return the minimum distance as the solution. This is a more complex solution that would not ultimately be a more successful solution regarding optimizing the route. The time complexity for this algorithm ends up being which is still an exponential run time and much slower than the nearest neighbor that was used above.

Another option that could have been used here is Dijkstra’s Shortest Path algorithm which finds the shortest paths from the source to all vertices in the constructed graph (GeeksforGeeks, 2022), instead of the nearest neighbor to the current location of the truck.

# Works Cited

GeeksForGeeks. (2018, Sep 06). *Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming)*. Retrieved from Geeksforgeeks.com: https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/?ref=lbp

GeeksforGeeks. (2022, Feb 22). *Dijkstra’s shortest path algorithm | Greedy Algo-7*. Retrieved from GeeksforGeeks.com: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/?ref=leftbar-rightbar

Vahid, R. L. (2018). *C950: Data Structures and Algorithms II.* Zybooks.

WGU. (2022). *NHP2 TASK 1: WGUPS ROUTING PROGRAM.* Salt Lake City: WGU.