Daniel LaForce #001119118
C950

Algorithm Overview

Stated Problem:

The purpose of this project is to create an application that can assign packages to three truck loads based upon the delivery deadline track packages and their delivery time using a self-adjust algorithm.

Algorithm Overview:

(Rubric Requirement A, B1, B2)

This application was coded in Python using PyCharm IDE by NetBeans. In Dijkstra's algorithm, we start from a source node and initialize its distance by zero. Next, we push the source node to a priority queue with a cost equal to zero. After that, we perform multiple steps. In each step, we extract the node with the lowest cost, update its neighbors' distances, and push them to the priority queue if needed. Of course, each of the neighboring nodes is inserted with its respective new cost, which is equal to the cost of the extracted node plus the edge we just passed through. We continue to visit all nodes until there are no more nodes to extract from the priority queue. Then, we return the calculated distances.

In the Bellman-Ford algorithm, we begin by initializing all the distances of all nodes with infinity, except for the source node, which is initialized with zero. Next, we perform V - 1 steps. In each step, we iterate over all the edges inside the graph. For each edge from node u to v, we update the respective distances of v if needed. The new possible distance equals to the distance of u plus the weight of the edge between u and v.

1. After V - 1 steps, all the nodes will have the correct distance, and we stop the algorithm.
2. Create a leaf for each character used in the sample file.
3. Dequeue the two leaves with the smallest values. Assign these two children to a new parent node. The value of the parent node is the sum of the two children.
4. Enqueue the parent node into the queue.
5. Repeat this process until one node remains on the root.

Operation time for the Huffman code depends on several factors. Is the frequency data sorted prior to running the program? What mechanism did the programmer use to implement the Huffman code? (priority queue, array, etc.) If the data has been sorted, then the operation time for running the program is O(*n*). If the data has not been sorted, then the operational time of the Huffman increases to O(*nlogn*). For this project, the Huffman algorithm was implemented using a priority queue. An analysis of the pseudo code used as a basis for this program will help explain the structure of the program and the operational time of the Huffman algorithm.

(Rubric Requirement B3)

Truck

set_d_time

return STRING X:XX AM/PM

init_loadtime

SET self.load_time

return: VOID

insert_package

APPEND package object memory location TO LIST self.all_package_info

l_truck

SET package object delivery_status IN :param: package_list TO 'in route'

SET package object on_truck IN :param: package_list TO self.name

SORT packages IN LIST :param: package_list

d_package

APPEND package_id TO LIST delivered_packages

SET package.delivery_status = 'delivered'

REMOVE package_id FROM LIST master_package_id_list

SET package.delivery_time

SET package_delivery_time

CHECK IF package IS delivered on time

path

CHECK for available drivers AND DISABLE 1 driver

CHECK for empty package list: if TRUE RETURN

SET EACH package IN :param: package_list TO 'in route'

FIND addresses for packages IN :param: package_list

APPEND EACH package_id TO LIST unvisited_queue

FIND the best route

ENABLE 1 driver

return best route

BIG Oh	**Functions**
O(1)	Set_d_time
O(1)	Init_load_time
O(N)	insert_package
$O(N^2)$	l_truck
O(N)	d_package
$O(N^6 \text{ Log } N)$	path
$O(N^5)$	find_address_list

TRUCK 1

Load Truck 1 first with lists:

T1 - not used for requirements of this program but exists for scalability

packages with delivery deadlines - load first because they are top priority

packages without deadlines - load last to fill the truck to max capacity

THEN find the route with the list all packages info

this list includes all package ids associated with previously loaded packages above

THEN SET the delivery time for Truck 1

TRUCK 2

Truck 2 shares a driver with Truck 1 and must wait for Truck 1 to return from deliveries

Truck 2 is set to leave at the time T1 returns (10:27)

Truck 2 is used to deliver delayed packages and #9 with the wrong address because it leaves after the address is reset

TRUCK 3

- delivers all remaining packages after Truck 1 and Truck 2 are loaded

- leaves the hub at 8:00 AM

- uses master package list to make sure no packages are missed during loading

Graph

BIG O	**Functions**
N^3	open the file and clean the unnecessary information, new lines, and spaces
N^2	adj_matrix
N^2	sort_adj_matrix
$O(KN^3)$	**Total**

Adj Matrix

The purpose of this block of code is to fill in the empty strings with the corresponding mileage from the matching

node. In order to do this, it is necessary to:

- hold the vertex id of the current node

- then look up the element number that matches the current node vertex id of the previous node

In other words the column_id (element number) that matches the previous address_id will hold the distance from the

current_address to the previous_address.

Sort Adj Matrix

The purpose of this function is to build a presorted adjacency matrix from distances in the format

Class Graph

Graph formats the distance data in dist_tbl_graph into bidirectional graphs

Package:

format the package data from the provided csv file into a hash table

Develop a hash table

BIG OH	Function
O(N)	mp_list_build
O(N)	new_pack
O(1)	pack_on_time
O(N)	**Total**

(Rubric Requirement B4)

While the truck is being loaded, the prioritization system is used by the WGUPS Package router to regulate which packages receive precedence over others. The packages are sorted into two parts, Ones with high priority and ones with regular priority. Both the sorting tags are considered mutually exclusive, such that ones that are not considered high priority are sorted into regular priority.

To be sorted as a high priority package it has to have an association of a deadline attached with it. Naturally the high priority packages are loaded onto the truck first and then any remaining truck space is to be utilized by the regular priority packets.

The minimization system just as the name suggests minimizes the route that the truck will take to deliver its cargo. This system determines the shortest possible route to deliver the truck cargo after the depot address is labelled as the current address, a greedy approach is used by the system to streamline by calculating the distance from current address hence sorting the packages for the next delivery as well. The algorithm always choses the shortest possible route while each delivery of the package updates current address to the current location of delivery.

Throughout the delivery period, the status of the package is updated in real time to allow for better updating of the package location. The status changes from loaded in truck to delivered with each milestone being achieved by the package.

For the same data input set the system always returns the data packets via the same route. However, this is not similar to other solutions for instance combinatorial optimization solutions that use metaheuristics to resolve this sort of problems. Despite the same data inputs the execution of the system might return slightly different solutions.

As far as the accuracy is concerned, the algorithm is in line to the provided requirements. The package restrictions are respected as well as on time package delivery also ensures the accuracy of the algorithm. This is also verified at different times to produce a package report to ensure accuracy.

Algorithm Scalability:

WGUPS could be classed as NP-Hard problem, problems like these are very hard to solve, and it makes getting a goal state even hard if solution has certain requirements attached to it. The core algorithm has a Time Complexity of Big Oh (n^6 Log N). Therefore, this algorithm is not recommended to be used for increasing number of packages because the time to get to the solution will increase exponentially with each new package to be delivered.

Since the solution has to have certain requirements it is inflexible algorithm will work only with the requirements given by the assignment. Therefore, scalability of this algorithm is not recommended.

(Rubric Requirement B5)

Maintenance

Efficiency and maintainability were kept in consideration throughout the development phase and therefore object-oriented approach was used to implement WGUPS. Classes and their data members, methods all of them are properly documented which explain the purpose of each block of code so that in future if any new developer starts working on the codebase, he/she can easily get along with it. Hence it simplifies maintenance.

(Rubric Requirement B6)

Program Strengths

Hash tables are used as the primary data structures because of their ability to store and retrieve values by a particular key. This ability for storage and retrieval of data for a particular key suits itself perfectly for WGUPS package data.

Program Weaknesses

Hash tables have a defined capacity they are not scalable whenever they run out of capacity, we have to rehash the whole. This can be a computationally expensive overhead with the number of increasing packages and since the time complexity is proportional to the entries of the table. Therefore, in future tables might need to be rehashed.

(Rubric Requirement D)

List of all package info

Strength:

1. The strength of a hash table as the primary mechanism of data storage in this application lies in its ability to store and retrieve values by a particular key. This characteristic lends itself quite nicely to storage of our package data,
2. information related to a given package can easily accessed in a hash table via the package's identifier. 2)

This hash table implementation is able to efficiently store and retrieve the specified package data according to the package identifier. This is evidenced specifically in the PackageTable class, where package data is stored and retrieved constantly throughout the application.

1) Dijkstra Algorithm
2) Bellman-Ford Algorithm

(Rubric Requirement J)

If I do it again, I might use A* algorithm to represent problem of delivering packages. It's a specialized version of Dijkstra Algorithm it uses heuristic to choose a path in the graph and is relatively faster than Dijkstra.

(Rubric Requirement K)

All the lists meet the requirements (ie: only 16 packages to be loaded on a truck), track of delivered packages and addresses of the package to be delivered.

A) Since it takes O(n^5) to find_address_list it's a big number so if we increase the number of packages to be delivered find address from the list will also increase.
B) There will be no difference since N will be changing space complexity and still remains the same in terms of big O
C) Data Structures

1. Stacks
2. Queues

Stack is last in first out and Queues is First in First Out while index can be used to find the value at a particular index List look up time complexity is O(1)