DATA STRUCTURES AND ALGORITHMS II — C950

WGUPS Routing Program

APPLICATION OVERVIEW

Josh Gaweda

1-12-2021

WESTERN GOVERNORS UNIVERSITY

Joshua Gaweda
C950: Data Structures & Algorithms II Student # 001150997

A. Identify a named self-adjusting algorithm (e.g., "Nearest Neighbor algorithm," "Greedy algorithm") that you used to create your program to deliver the packages.

The named self-adjusting algorithm selected for this project was a "Greedy algorithm" , better known as "Primms algorithm"

B. Write an overview of your program

This projects purpose is to find the fastest route of distribution for the Western Governors University Parcel Service (WGUPS) using Python. There are two trucks and three different drivers that are available to deliver 40 packages and each driver stays with thew truck for its duration of service. Some of the packages being delivered also have additional constraints and delays to consider. Additionally, one of the packages being delivered has the wrong address on it, so it will need to go back out for delivery on the second truck.

To loads the trucks we first create some conditional statements based on the provided data. Next, I created a greedy algorithm to optimize the delivery of each package on the route. The reason this algorithm is greedy is because it will first determine the shortest path available and then continue doing that until there are no additional packages left. This document will analyze how the algortihm is used and will describe the functions and components inside the algorithm.

The following steps represent the execution of the self adjusting greedy algorithm:

1. The associated truck # and the trucks current location will be passed in along with the package list for that truck. The truck will always start at the hub by default.
2. All of the packages on the truck will be compared to the trucks current location to determine the closest delivery point.
3. The closest delivery point is determined after all packages on the truck are compared. Once the point is determined it is removed from the trucks delivery list amd the truck will move to that location.
4. Once the truck is at the new location the point is appended to a list of optimized locations.
5. The current location is updated with the new address and the Function gets called again using the smaller list.

B1. Explain the algorithms logic using pseudocode

1) Truck Loading
   a) The packages are first sorted based on the conditions of their requirements. Delayed packages, packages that must be delivered on truck 2, packages that must be delivered

with other packages and urgent packages are separated into groups. The remaining packages are then placed into their own group.

b) Packages that must be placed on truck 2 are as well as packages that go to the same address. Urgent packages are also placed on truck 2.
c) Delayed packages are placed on truck 1 as well as packages going to the same address.
d) If there is remaining space on truck 1 or 2, packages with the closest address to an exisisting address on the truck are added until the truck is full.
e) All remaining packages are placed on truck 3.

1) Route Planning
   a) Each truck starts at the hub (address 0).
   b) Primm's path finding algorithm is used to create a minimal spanning tree:
      i) Two counters are created: num_address & num_edges.
      ii) A list selected is created with length equal to num_address.
      iii) The first address index (hub) in selected is set to True.
      iv) The program loops, checking the distance between each address to all other address. It stores the smallest distance between the addresses if one of the addresses is currently in selected.
      v) Once the smallest distance is found, an edge is created with data on the from address, to address and distance between. The edge is stored, and the to address is added to selected.
      vi) Steps 4 & 5 repeat until num_edges is equal to num_address + 1.
   c) A depth first search algorithm is then used to find a route through the minimal spanning tree created in the previous step:
      i) A placeholder for current address and two lists are created: visited and unvisited.
      ii) The visited has the hub address added, the unvisited has all other addresses added.
      iii) The program checks each edge of the minimum spanning tree. If the from address in the edge matches the current address and the to address is not in the visited list, the to address is added to visited and current address is updated to that address.
      iv) If no addresses meet the criteria of the previous step, the current address is a leaf in the tree and the program backtracks, adding the address it's back tracking to into the visited list and updating the current address.
      v) Steps 3 & 4 repeat until there there no addresses left in the unvisited list. This will produce a path through the minimal spanning tree.
   d) The path found in the previous step is then converted from a list into a dictionary, then back into a list. This removes all duplicate addresses from the list while preserving it's order.
   e) The hub address is then appended to the end of the path creating a Hamiltonian Cycle through the minimum spanning tree. This is the route to deliver packages.

B2. Describe the programming environment you used to create the Python application.

The application was written using Visual Studio Code and Python 3.9. I developed the application on Windows 10.

B3. Evaluate the space-time complexity of each major segment of the program, and the entire program, using big-O notation.

The self adjusting greedy algorithm has a best-case runtime of O(1) and worst-case run time of $O(N^2)$. Since the best case is only possible when the truck is empty before the package list is loaded, we can generally expect for the worst-case. The pseudo code for the algorithm is detailed below:

The algorithm has a total space-time complexity of $O(N^2)$

Where each individual section has a space-time complexity of:

A. O(1)
B. O(1)
C. O(N)
D. $O(N^2)$

Main.py

Function	Line #	Space- Time Complexity
start_deliveries	19	O(N)
time_of_day_prompt	55	O(N)
print_status	71	$O(N^2)$
main	85	$O(N^2)$
Total		**$O(N^2)$**

Package.py

Function	Line #	Space-Time Complexity
__str__	22	O(1)
__eq__	31	O(1)
Total		**O(1)**

Joshua Gaweda
Student # 001150997

HashTable.py

Function	Line #	Space-Time Complexity
__repr__	28	O(N)
address_count	51	O(N)
Insert_package	49	O(1)
retrieve	85	O(1)
Init1	100	O(N)
Init2	122	O(N)
init3	1143	O(N)
table_from_csv	165	O(N)
graph_from_csv	194	$O(N^2)$
update_package	217	O(1)
pack9	237	O(1)
lookup_package	256	$O(N^2)$
Total		**$O(N^2)$**

Main.py

Function	Line #	Space-Time Complexity
deliver	19	O(N)
deliver_to_time	55	O(N)
print_status	71	O(1)
main	85	O(N)
Total		**O(N)**

Truck.py

Function	Line #	Space-Time Complexity
__repr__	28	O(1)
distance_to_next_stop	47	O(1)
deliver	62	O(1)
move	74	O(1)
sort	99	$O(N^2)$
get_packages_from_address	121	O(N)
find_minimum_spanning_tree	140	$O(N^2)$

get_dfs_path	199	$O(N^2)$
num_addresses	248	O(N)
travel	267	O(1)
Total		$O(N^2)$

The computational time and memory remaining mostly linear throughout allows the available set of inputs to scale without worrying about the availability of memory. We also do not need to be concerned with bandwidth as the application is run locally.

B4. Explain the capability of your solution to scale and adapt to a growing number of packages.

Adaptability was a core concern when designing this program. The hash table has the ability to grow dynamically to accept new packages, the package and distance data can be loaded from different CSV files, and placing the path finding algorithm in the truck class allows the program to create more trucks as the company scales. Each step of the process is it's own method, so if different data structure or algorithm is needed in the future, only the step it's replacing should need to be rewritten.

B5. Discuss why the software is efficient and easy to maintain.

Many of the steps to increase adaptability also increased maintainability. Each method is responsible for one step of the delivery process and is heavily commented. This allows future maintainers to understand the thought process that went into making this program in order to correct potential bugs or add/remove features.

B6. Discuss the strengths and weaknesses of the self-adjusting data structures (e.g., the hash table).

A direct hash was chosen as it is the fastest available hash table. The downside to direct hashing is that it also creates the largest hash table, as each bucket can only store one package. However, since the company averages only 40 packages per day, this will still create a small memory footprint. The hash table by default is initialized with 100 buckets, but this can be increased/decreased by changing the capacity when initializing the table.

D. Identify a self-adjusting data structure, such as a hash table, that can be used with the algorithm identified in part A to store the package data.

The main data structure used in the project is a direct hash table.

Joshua Gaweda
C950: Data Structures & Algorithms II Student # 001150997

E,F,G. See Attachments

I. Justify the core algorithm you identified in part A and used in the solution by doing the following:

1. Describe *at least* two strengths of the algorithm used in the solution.
2. Verify that the algorithm used in the solution meets *all* requirements in the scenario.
3. Identify two other named algorithms, different from the algorithm implemented in the solution, that would meet the requirements in the scenario.
   a. Describe how *each* algorithm identified in part I3 is different from the algorithm used in the solution.

The greedy algorithm chosen meets the project constraints and performs all required functions to deliver all packages within a set range of 140 miles. The user interface also features the ability to lookup all package details at a given time and check on a specific package with its package ID number. The algorithms greatest strength is how quickly it can find the best path for a truck and how well it can scale with any dataset provided to it.

I could have also used a Dynamic Programming Approach (Data Structures - Dynamic Programming, n.d.) to optimize package delivery that uses a splitting technique to break a program down into smaller methods. The advantage of coding the application this way is the ability to store paths all along a route and check to see if there is a faster total path by first traveling another location. Its possible that using this approach would have created a much larger space complexity but it may have ultimately resulted in a shorter path.

The 2nd type of algorithm that could have been used is a "Self-Tuning Heuristic". This approach would work by:

1. Starting from the hub and determining which path is closest to the hub.
2. Determine which packages need to be delivered to and loaded on that truck.
3. Start at the new location and determine the shortest path from that location. If the location was previously visited than the truck would move to the next closest location. This will last until the 40 packages have been assigned to a truck and a path.

The self-tuning heuristic shares a similar approach to my own in the sense that both choose the shortest path available.

J. Describe what you would do differently, other than the two algorithms identified in I3, if you did this project again.

If I were to do this project again I would have gone with an heuristic approach to decide which packages go on each truck. By automating this process it could allow the application to easily adapt to other environments and also could potentially provide a better path for the algorithm to work with. That being said, the greedy algorithm already handles limited data sets very efficiently and has great potential to scale as is.

I also started working on the project before full understanding the instructions, so I did spend more time fixing/adjusting then I did actually coding the application. I would certainly study the instructions before getting started if I were to do this project again.

K1. Justify the data structure you identified in part D by doing the following:

1. Verify that the data structure used in the solution meets *all* requirements in the scenario.

a. Explain how the time needed to complete the look-up function is affected by changes in the number of packages to be delivered.

b. Explain how the data structure space usage is affected by changes in the number of packages to be delivered.

c. Describe how changes to the number of trucks or the number of cities would affect the look-up time and the space usage of the data structure.

When a package is added to the table from the CLI, a package ID is dynamically created based on available space in the hash table. When a free bucket is found the package ID will be set to it's index, and that package will be stored in that bucket. The address of the new package is also checked against current addresses, and the address ID is assigned if found. If no matching address is found, then a new address ID is created. This allows the hash table to self adjust to new packages created at run time.

Regardless of it's size the direct hash table will have a Big O(1) for package insert, deletion, and lookup by ID. If another package attribute is used for lookup instead of ID (such as city name), the direct hash table will have an efficiency of Big O(N) as it will need to perform a linear search through all buckets. The size of the direct hash table will be N, where N is the number of packages. Since most of the methods used on the direct hash table relies on package ID, the speed of runtime will not be affected since those methods run at Big O(1). The only functionality of the program that will be affected is lookup by attributes other than ID (city, address, zip, weight, deadline, status), as that runs at Big O(N). Using these functions will see a slowdown when more packages are added.

Joshua Gaweda
C950: Data Structures & Algorithms II Student # 001150997

K2. Identify two other data structures that could meet the same requirements in the scenario.

a. Describe how *each* data structure identified in part K2 is different from the data structure used in the solution.

In retrospect there are many types of data structures that could have been used instead that meet the project constraints. One popular data structure is a Binary Search Tree(BST). The BST is a node-based binary tree that has unique ordering properties allowing me to presort packages based on attribute and the access it through a tree like structure If the project had required the use of larger quantities of data I think this would have been the best way to go. What was used was lists because they made the most since before I got started on the project and they are pretty straightforward.

Another possible data structure that could have been used is a stack. Stacks are linear data structures that only allow data access from one end via “push” and “pop” operations. I could have used stacks to sort the packages onto the truck in their optimal delivery order by distance and “pop” the item upon delivery. This method would type of structure may have made schedule/address changes difficult.

L. In-Text Citations

Data Structures - Dynamic Programming. (n.d.). Retrieved from Tutorials Point: https://www.tutorialspoint.com/data_structures_algorithms/dynamic_programming.htm