Bailey Borchers

C950 Write Up

Student ID:

000925699

4/18/2023

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.

      I used a Greedy algorithm approach to deliver the packages. This can be found in the greedy.py file in my submission. In the “greedyAlgorithm” function I had a priority list of packages that needed to be delivered first and a master list of all the packages that had not been delivered yet. This was decided by giving each package a Boolean value titled priority that would be True or False based on their special instructions. Focusing the priority packages first, the greedy algorithm would then decide which package would be the closest. There are some additional parameters here regarding how many packages can fit on a truck, adding packages to the priority queue when certain conditions are met, and dynamically changing the distance to each package after one has been delivered.

B.  Write an overview of your program, in which you do the following:

1.  Explain the algorithm’s logic using pseudocode.

Without going into detail on how a package is marked as priority or what truck it will be applied to; we will be focusing on just how the Greedy Algorithm functions.

We have two classes that are being used. The important variables are noted:

unvisited queue [ ]

Package:

id

address

distance

Address:

address

distance

adjacency distance [address, distance]

Package class holds individual package data and will assign its distance value as the algorithm runs. Address class distance hold the distance from the most recently delivered package and has the distance from each package to itself in the adjacency\_distance list.

The unvisited queue list will hold each package and remove them as they are delivered. This first pseudocode adds each package to the unvisited queue list.

for package in Package

unvisited queue. add(package)

This next bit of pseudocode has two variables smallest distance and shortest package that will be used to store the package that is the closest from our current destination. After finding this package, it is removed from the unvisited queue and added to the package order list. After that all current distances for the address objects are updated, and the while loop starts again until there are no more packages that need to be delivered.

package order = [ ]

while unvisited queue > 0

smallest distance = inf

shortest package = Package object

for package in Package

for address in Address

if package.address == address.address

package.distrance = address.distance

if smallest distance > package.distance

smallest.distance = package.distance

unvisited queue. pop(shortest package)

package order.add(shortest package)

for address in Address:

if shortest package.address == address. adjacency distance[1]

address.distance = adjacency distance[2]

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

This program was created using Python with the PyCharm IDE version information below:

Name: PyCharm

Version: 2022.3.3

Build: 223.8836.43

Python version 3.9

This was also created on a desktop using Windows 10 with an intel 11th gen processor.

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

For each major section of code I added big O notation in the comments. The worst this ran at was O(n^3) so that will be the estimate for the entire program. I did add a time that it will take for the greedy algorithm to fully implement and added that to the start of my program. Looks like average time is 0.007 seconds to run one iteration of the algorithm.

For space complexity for this program, this is much harder to define for each segment. We will have N values held by our main data structures being the Packages hash table and the adjacency list of addresses in the Graph Class. The adjacency edge weights will hold N^2 memory as for each address added we will need to hold the edge weights to and from every other address.

For each Package there are 13 attributes that will need memory, 2 for each Address, and 3 for each Truck. That also being said each Truck will have a list of packages that will be maximum of 16.

For the greedy algorithm trying to calculate only Auxiliary space needed. There are 17 variables used with 5 of them being arrays that will hold additional bits of memory. Inside the while loops of the function, we are mostly just comparting the values of the different packages/ address and updating those data already stored. I do see 5 new variables that will need memory allocated when running the while loop to assign packages.

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

I do not see an issue in the ability to scale given more packages being added. The biggest issue I would have to tackle would be adding a more dynamic way of tracking the trucks. The greedy algorithm will still find the closest distance but there is nothing currently in place if we needed to add more trucks/ drivers.

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

When all is said and done, if the package data is stored the same way in an excel file, the program should be able to sort them without too much hassle. If we need to scale up and add additional hubs or cities that would be easy to manage as all you would need to do is add another class and additional trucks.

The software is efficient for the scope of this project. The project runs in worst case O(n^3) runtime but that is only for one specific part of the algorithm. This section is parsing whether packages need to be delivered together in the special instructions. That loop runs only three times in the case of this assignment and should not affect the overall efficiency of the software that much. Using a hash table, it is quite efficient to access package data. The bulk of the time used in the project will be used when running the greedy algorithm and finding the shortest route to deliver the next package.

The algorithm does need to iterate over each package in the hash table and then each address in the graph list to find the distance associated. This is because multiple packages can be delivered to the same address. We need to separate these objects so that we can continue to add additional packages to be delivered to the same address in the future. That being said, an average runtime of O(n^2) is perfectly acceptable when solving this problem.

I did add a time that it will take for the greedy algorithm to fully implement and added that to the start of my program. Looks like average time is 0.007 seconds to run one iteration of the algorithm.

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

Hash tables are very useful when you need to sort through a large amount of data. It can quickly find an object with an average time of O(1). It is also rather easy to add an item to the hash table as you do not need to define the length of the data structure like you would have to in a list.

Hash tables are a little complicated when it comes to accessing the data in programming. This is because you cannot simply iterate over them and must iterate over each bucket and array in that bucket. Another disadvantage of using hash tables is when you are scaling a project. If you are hard defining how many buckets are in your hash table, you may run into issues. If you have too many buckets the memory usage is poor. If you have too few then the runtime gets worse.

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.

1.  Explain how your data structure accounts for the relationship between the data points you are storing.

There are variables for each package, address, and truck object that are used to route our packages. A hash table was used to store package information. Using a [key, value] form to store each package, they were inserted into the table hashing their unique id number. For this case there were 40 packages with id values from 1-40. When parsing the .csv file, any relevant package information stored under deadline or special instructions were considered. These values were used to flag each package using a Boolean value as whether they were a priority to be delivered or unavailable to be delivered that time.

To store Address data, I used a graph with vertices. For each vertex created (Address class), the name of the address being delivered to was used. Then I would assign the distance to each address from the Hub or starting position when parsing through the .csv that holds the distances from each address. After that using the adjacency list in the Graph class, edge weights were held from the distances .csv that can be accessed by entering any two vertices/addresses.

This data is accessed by using the address attribute held in the package class and to find a matching label in the address class. When we have a match, we can return those edge weights by using the adjacency list in the Graph class. The greedy algorithm is constantly updating the package priority and unavailable status based on what truck we are using and what the time of the day is.

After the greedy algorithm has finished, it returns the order of packages delivered, the total distance, and the time finished for each truck and creates those truck objects.

Using a hash table, the packages were inserted after being parsed from the excel data file. Hash table was used here so that we can add any number of packages without issue. Address data was stored from the distance excel file, and distance to and from each address was stored in array.

Iterating through the packages hash table, we were able to find matching addresses and assign distances after they were parsed through the algorithm.

Note: Use only appropriate built-in data structures, except dictionaries. You must design, write, implement, and debug all code that you turn in for this assessment. Code downloaded from the Internet or acquired from another student or any other source may not be submitted and will result in automatic failure of this assessment.

H.  Provide a screenshot or screenshots showing successful completion of the code, free from runtime errors or warnings, that includes the total mileage traveled by all trucks.

Option 1

Text

Description automatically generated

A picture containing text

Description automatically generated

Option 2

Text

Description automatically generated

Option 3

9:00 a.m.

Text

Description automatically generated

Text

Description automatically generated

9:50 a.m.

Text

Description automatically generated

A picture containing text

Description automatically generated

12:10 p.m.

Text

Description automatically generated

Text

Description automatically generated

Option 4

Text

Description automatically generated

Exception handling

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

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.

Greedy algorithm is very useful as it is easy to conceptualize when explaining to the people who need the algorithm implemented. It is also very easy to code and runs quickly.

2.  Verify that the algorithm used in the solution meets all requirements in the scenario.

This was verified by using the interface and making sure all the packages were delivered at the correct times given their special instructions.

3.  Identify **two** other named algorithms, different from the algorithm implemented in the solution, that would meet the requirements in the scenario.

Dijkstra’s shortest path and Bellman-Ford algorithm.

1. Describe how each algorithm identified in part I3 is different from the algorithm used in the solution.

Dijkstra’s and Bellman-Ford algorithm involves finding if there are any shorter paths that can be taken by travelling to specific vertices first. They calculate if it is shorter to go from A -> C -> B compared to A -> B -> C regardless of whether B or C are initially closer to A. The difference between these two algorithms is that Bellman- Ford can handle negative edge weights where Dijkstra’s cannot. Although not important to this assignment, it could be useful to have that capability.

The greedy algorithm is more efficient as you only require one loop to find the smallest value from a current point. When using the Dijkstra or Bellman – Ford algorithm, you must loop through the adjacency values of each potential path to find if it would be quicker to travel a different direction. Whereas the greedy algorithm is more efficient, these other two algorithms will return a better result to the problem.

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 spend more time creating the classes used. Parsing the data was simple enough but I was not always thinking about how it would be used after holding it in my hash table. I would probably create the trucks and packages first and then figure out what data is needed to get the end user results. When it comes to the algorithm, I would probably try the Dijkstra’s method as a challenge to myself. I tried it initially but was running into some issues on getting it implemented. I believe I would be able to get it to work now and in the end, it should be able to get the total mileage of trucks down even if it is a slight improvement.

K.  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.

As touched on earlier, average case for a hash table is O(1) and worst case is O(n).

Where this can run into issues is when you are defining how many buckets you need in your hash table. For example, if you only have one bucket then the program will have to iterate over every package in that bucket to find the right one. As the program gets larger you would want to increase the buckets in order to have runtime be efficient.

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

My hash table is an array of items in key: value pairs. When creating a hash table and assigning how many buckets of memory will be used, the keys are created to be mapped to. When adding a package to the hash table, you are adding memory as a value associated with that key that will be stored. For this project unique id’s were hashed and then packages were stored at those values. If packages have the same hash value based on the modulo, then an array is created to hold the additional values that map to that key.

1. 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.

If I were tackling this issue, I would assign trucks to different cities based on class object variables so that would not be an issue. After that I parse the zip codes of each package address and assign which zip codes are to be delivered at which city/hub. The additional lookup time would then be increased on this additional parsing to make sure that the package is assigned to the right area.

An example of how I would map this issue is as followed.

Class Truck

cityId = “”

Class City/Hub

listTrucks = []

listZip = []

hashPackages = new.hashTable

addTruck(truck)

listTrucks.add(truck)

When parsing through package .csv files I would then assign them to the City/Hub and store them in hashPackages for each instance.

for row in csv reader

package = (row1, row2, row3, etc…

for cityHub in Class City/Hub

if package.zip in cityHub.listzip

cityHub.hashPackages.add(package)

These hash tables are now stored in relation to which City/ Hub they are associated to. When running the algorithm for these now, you would have to individually run the greedy algorithm for each City/Hub. So, the lookup time will not be affected to find a package as it should still run in average O(1) time for a hash table. But we should see an increase in space as we are created new hash Tables for each instance and buckets to hold these values.

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

You could still use a list, or a linked list to store the data. You could also use a queue after defining the order packages need to be delivered. It all depends on how you set up the program.

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

A list would be different as the runtime would increase dramatically as you would have to iterate over the whole list to find which member needs to be found. I would also only use a queue to store the package order that needs to be delivered as you can only add to the queue and pop out the first entry. Not the most dynamic option but it would be fun to implement.

L.  Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

Zybooks course material was used to help with the hash table and graph/ vertices classes.

No other outside material was used.