Nathan Perkins

8/30/22

C950 Assessment

1. The main algorithm used in my program is called load\_packages it is a Greedy algorithm because it finds the next best packages to load based on the previously loaded package.
2. 1. Algorithm Overview:

This algorithm works by checking for the next nearest destination to the last loaded package in the deque. It loops until the truck is full or all destinations are visited or it runs out of eligible destinations.

Psuedocode:

Start loop while truck has room or there are destinations still to visit:

If package list size > 0:

then find last added package

then find next nearest destination

else find nearest destination to the HUB

new package list = list of packages going to the same destination

If package list + new package list > Max size of truck

Then do not add packages to the list

If the packages do not have special instructions

Then add to the truck

Else if special instructions are for truck 2 and the truck is truck 2

Then add to the truck

Else if there are special instructions and the truck is truck 3

Then add to the truck

Else add the destination to ineligible list

This function has a worst case time complexity of N^2 for number of destinations. The worst case space complexity is N for the number of packages going to the same destination.

Psuedocode for nearest() algorithm:

Smallest = 10000000

Nearest id =. Given id

For loop in size of destination list:

If value in array is not Null:

Then if distance value < smallest and distance != 0:

Then smallest = distance value

Nearest id = new destination

Else if other side of array is not null:

Then if distance value < smallest and distance != 0:

Then smallest = distance value

Nearest id = new destination

Return nearest id

This function has a time complexity of N for the number of destinations. The space complexity is O(1).

Psuedocode for same\_destination finction:

New list = empty list

For each package in the Hash Table:

If destination id = given id:

Then add package to new list

Return new list

This function has a time complexity of N for the Number of packages. The space complexity is O(N).

B2. PyCharm 2022.2 (Community Edition)

Build #PC-222.3345.131, built on July 27, 2022

Runtime version: 17.0.3+7-b469.32 x86\_64

VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

macOS 12.4

GC: G1 Young Generation, G1 Old Generation

Memory: 2048M

Cores: 8

Metal Rendering is ON

Python 3.10

B3. Nearest() has a time complexity of N and a space complexity of O(1)

Same\_destination() has a time complexity of N for the Number of packages and a space complexity of O(N)

Load\_packages has a worst case complexity N^2 and a space complexity of O(N)

The whole program has a worst case run time of O(N^2) and a space complexity of O(N + K + J) where N is the number of packages and K is the number of destinations and J is the number of trucks.

B4. The controls in the algorithm are based on the sizes of the data structures and are not hard coded in. Also the Hash table can adjust size with chaining. The size of the hash table can be adjusted as the amount of packages increases. This will increase the amount of buckets in the hash table. However it isn’t necessary to change the size of the hash table since it supports chaining, and the buckets in the hash table will dynamically allocate memory for each package added.

B5. The software is easy to maintain because the values that control the loops are not hard coded. The algorithm will adjust as the size of the data structures change. Also the main algorithm is well commented and the flow is easy to understand.

B6. The strength of the hash table used is that it is easy to look up data using a key. It requires very few operations in order to retrieve data. Also it is self-adjusting so chaining data is supported. A weakness of the hash table is that it can be difficult to iterate through and it can be difficult to change data values in the hash table.

D1. I am using a self adjusting hash table to store package data. Package data is stored using the package id as the key and then the package type object is stored in the value. The package object stores all the data related to the package. The hash table also supports chaining in the event that there are multiple values with stored in the same bucket.

I1. One strength of the algorithm that I used is that it finds a fairly good route for each truck. All trucks were able to finish deliveries before 12pm.

Another strength of the algorithm is its readability. The core algorithm is fairly simple and I think it is easy to understand.

I2. The algorithm meets all the requirements of the scenario by loading and delivering all the packages while accounting for special instructions and the limitations with each truck.

I3. One other algorithm that could be used is a Shortest Path algorithm. This algorithm would check every possible path for delivery and then select the shortest one. Compared to the Greedy algorithm it would be much less efficient program, but would find the most optimized solution.

Another algorithm that would work is a Nearest Neighbor algorithm. This algorithm would use a weighted graph to store destination information with the distances as values on the weighted edges. It would iterate through all neighbors on the graph to find the next shortest distance. Compared to the greedy algorithm this algorithm would come up with a similar solution and have a similar run time.

J. What I would do different if I did this project again is try to find a more efficient algorithm for finding packages going to the same destination. I also would include the time calculations when adding the packages to the trucks. I also could group packages together by city or similar destination and then load them on the trucks that way.

K1. The data structure used is a hash table and meets the requirements of the assignment by storing data in key value pairs with a set and look up function.

K1a. The time needed to execute the look up function would increase slightly as the number of packages increases. Though it would not significantly increase because the data structure supports chaining.

K1b. Space usage would increase as the number of packages increases because each bucket in the hash table has a list inside of it. The list is what would increase in size and not the number of buckets in the hash table. You could also increase the size of the hash table in the code itself if you wanted to reduce the amount of chaining.

K1c. The number of trucks or cities would not affect the time or space usage of the hash table data structure.

K2. Another data structure that would work in this scenario is a dictionary. Dictionaries support key value pairs and easily can look up data. They are also self adjusting as you add elements to them. This would work basically the same as the hash table used in the program. Since the python dictionary is a built in hash table for python.

A different data structure that would also work is a list. The list index would serve as the key while each element in the list would be the package data. Compared to the hash table used in this program as long as the package id was the same as the list index then it would work similarly. However if packages were added or removed then it keeping track of the list index would become very difficult.

L. Sources:

https://www.w3schools.com/python/python\_reference.asp