Aaron Artz
Student: #000871650
October 5, 2020

C950 Core Algorithm Overview

Stated Problem:

The Western Governors University Parcel Service (WGUPS) needs to determine the best route and delivery distribution for their Daily Local Deliveries (DLD). There will be an average of 40 packages delivered per day with varying constraints such as, deadlines, incorrect addresses, and packages that must be shipped together. We begin to solve this problem by manually loading each truck based on the conditions then applying a sorting algorithm to the list of packages on each truck to find the best route. The algorithm used is a Greedy algorithm since it sorts the packages based on the next nearest neighbor for each package in the list.

Algorithm Overview:

The greedy algorithm has been implemented in through the following steps:

1. The method takes in a list of packages, a current location, and a truck ID number.
2. The method then checks the distance between each location in the list and the current location.
3. When a new lowest distance is found it updates the lowest package to that location.
4. After it iterates through all locations, the lowest package is then appended to a new sorted list, and removed from the input list
5. The method then calls itself again using the list with the removed previous location, and continues to do so until all locations are added to the sorted list.
6.

The complexity of this algorithm has a worst case runtime of O(n^2). The only factor that could improve this runtime is manually sorting the packages prior to adding them to the input list, since that is unrealistic we assume the worst case runtime will be the most common case. The following pseudo code will help explain the algorithm and runtime.

Greedy Algorithm:

1 .Variables declared outside of the method:

A. unsorted_list (this will be populated with packages.)

B. sorted_list (this is an empty list, where the method will append packages.)
C. (These will be declared for each truck shipment.)

Complexity for part 1: O(1)

2. Method takes in:

A. unsorted_list (This is the list of packages that needs to be optimized.)
B. current_location (This defaults to the HUB)
C. truck_id (This will keep each package list separate.)

Complexity for part 2: O(1)

3. Check truck_id: (to determine what sorted_list to append to)

```
    A. If unsorted_list is empty:
            exit the method (this will end the recursion when there are no
            Unsorted packages left.)

    B. nearest_package = unsorted_list[0] (first package in list by default)
    C. distance = distance_between(current_location, nearest_package)
```

Complexity for part 3: O(1)

```
4. For row in length(unsorted_list):
    if distance_between(current_location, unsorted_list[row]) <= distance:

            nearest_package = unsorted_list[row]
            lowest_row = row
```

Complexity for part 4: O(n)

```
5. sorted_list.append(nearest_package)
        A. delete unsorted_list[row]  #    (This deleted the newly appended
           package from the original list)

        B. Call method again using (unsorted_list, nearest_package, truck_id)
```

This will continue until all packages have been removed from the unsorted_list and appended to the new sorted list.

Complexity for part 5: O(n^2)

Greedy Algorithm total complexity: O(n^2)

Methods:

LinkedList.py

Method	Line	Space-Time Complexity
__str__ (Link)	14	O(1)
__init__ (Link)	21	O(1)
push	30	O(n)
pop	39	O(n)
insert	57	O(n)
__str__ (LinkedList)	77	O(1)
__init__ (LinkedList)	84	O(1)

Total Complexity O(n)

HashTable.py

Method	Line	Space-Time Complexity
put	16	O(n)
division_hash	34	O(1)
get	38	O(n)
insert	51	O(1)
hash	55	O(1)
__str__	59	O(n)
__init__	69	O(1)

Total Complexity O(n)

Distance.py

Method	Link	Space-Time Complexity
get_distance	24	O(1)

get_loc_id	31	O(n)
get_total_distance	39	O(n)
add_time	50	O(1)
load_trucks	57	O(n)
ship_truck	65	O(n)

Total Complexity O(n)

Packages.py

Method	Line	Space-Time Complexity
sort_truck_packages	110	O(n^2)
search_by_address	208	O(n)
search_by_id	230	O(n)
status_by_time	252	O(n)
total_mileage	266	O(1)
mileage _by_truck	275	O(1)

Total Complexity O(n^2)

Main.py

Method	Line	Space-Time Complexity
----	11	O(n)

Total Complexity O(n)

Algorithm Justification:

The Greedy Algorithm accomplished the goals of the program by creating a route that delivers all packages with a total distance of ~120 miles. The benefits of using the greedy algorithm include having a speedy run time compared to other algorithms, O(n^2), being able to implement it with quick and easy to understand code, as well as easily being scalable with any number of additional packages.

A different algorithm choice could be a heuristic algorithm. “A heuristic algorithm is an algorithm that quickly determines a near optimal or approximate solution.” (Zybooks 3.1) This could work by starting at the hub and choosing the next closest location, then checking to see what packages need to be delivered there, adding them to the truck, then going to the next closest location. Locations that have already been visited will be skipped. This is very similar to my approach in that it chooses locations based on the smallest distance.

Another algorithm I could have used is Dynamic Programming. “Dynamic programming is a problem solving technique that splits a problem into smaller subproblems, computes and stores solutions to subproblems in memory, and then uses the stored solutions to solve the larger problem.” (ZyBooks 3.5) Since dynamic programming stores values along the way it would be possible to check to see if there is a quicker path by first going to a different location. This could give us a shorter total distance traveled at the cost of increased runtime.

Data Structure:

The data structure I chose to implement is a linked list. This data structure works well in the application since it is fairly easy to implement and offers speedy delete and insertion methods at O(1) and slightly slower access methods at O(n). Since we know the amount of data the data structure will need to hold, it is easy to create an appropriate amount of buckets and hashing algorithm to evenly split the packages up amongst buckets. If the business were to expand or had unexpected added packages, the linked list would easily be able to handle any additional data. One issue with using a linked list structure is it makes it difficult to perform actions on specific packages without first exporting them to a temporary list to work with.

A different data structure I could have chosen would be a binary search tree(BST). These offer a great average complexity of O(log(n)) for all access, deletion and insertion methods which makes for a very consistent data structure. This is very important when N is a large number, but in the case of this program and reasonable expansion of the business, this is unlikely to make a large difference in runtime. Another possible data structure would be a simple Hash Table. Since we are aware of how many packages a day the business will be delivering we could have easily built a table with that many available storage locations and place a package in each one. This would give us the quickest possible runtime a O(1) for all operations. The down side to this is that expanding the business would require adding more buckets to the hash table which does not allow for expansion or any unforeseen additional packages.

Efficiency and Maintainability:

A majority of the runtime of this program comes from the sorting algorithm, this algorithm must be called once for each truck. Although it has a complexity of O(n^2), each truck consists of at most 16 packages so it still runs quickly. There are a few more methods that must be called for each truck with a complexity of O(n). These calculate the total distance and update the status of the packages. The maintainability of this program is improved by having descriptive method names and frequent comments that explain the proceeding code.

Adaptability:

The program was built with the understanding that business will likely increase over time. The choice in data structure was made so that if there is an increase or decrease in total packages the program would not falter. A way to sort packages and load them on trucks automatically would be ideal, but small differences in the wording in the original excel file could cause issues for example, searching for "10:30:00" in the package delivery restriction to find packages that must be delivered by that time, but the user entered in "10:30" This could leave to packages being overlooked. This could be resolved by giving restriction codes and drop down choices for end users but that is outside the scope of this program. With manually loading packages I was able to avoid having a 16 package restriction on the trucks which means if the business purchases larger shipping equipment or has an outlier that needs more than that amount, it will perform equally as well. For that reason, this program will handle an increase in business demands up to a point where manually loading packages is no longer viable.

Hindsight:

If I were to have another attempt at this project I would like to challenge myself further by implementing a method to load packages on each truck. This could go a step further by loading trucks based on lowest total distance along with package restrictions. This would increase the runtime of the program substantially but would decrease the total mileage. This change could also help in scaling the business as the more packages the company must ship the less likely you will be able to manually sort and load packages.

Sources:

The only sources used in this project were the course materials provided by ZyBooks.

Learn.zybooks.com Data Structures and Algorithms 2 retrieved at:
https://learn.zybooks.com/zybook/WGUC950AY20182019/chapter/3/section/1

Learn.zybooks.com Data Structures and Algorithms 2 retrieved at:
https://learn.zybooks.com/zybook/WGUC950AY20182019/chapter/3/section/5