Nicholas Warner
C950 - Algorithms and Datastructures II
Attempt #3
19.11.2021

Analysis of Self-Adjusting Algorithm in C950 Parcel Delivery Final Assessment

A. This submission uses the “Nearest Eligible Neighbor” algorithm to determine the order in which parcels are loaded and delivered for the final assessment. The “Nearest Eligible Neighbor” algorithm can be found predominantly in the load_truck method in controller.py.

B. An overview of the algorithm’s logic is as follows:

1. BEGIN

```
For package in allPackages
    If (package is available AND package is allowed on truck)
        Add package to eligiblePackages

For package in eligiblePackages
    If (package.deadline is within two hours)
        Add package to priorityPackages

While (truck is not full AND eligiblePackages is not empty)
    If (groupPackages is not empty)
        Load the nearest group package onto the truck
    Else If (priorityPackages is not empty)
        Load the nearest priority package onto the truck
    Else
        Load the nearest eligible package onto the truck

    If (the loaded package is part of a group)
        Add each package in the group to groupPackages

    Remove loaded package from all relevant sets

Deliver packages on truck in the order they were loaded

END
```

2. The program was written with Python 3.9.6 using Notepad++ v8.1.5.

3. The space-complexity of the algorithm is $V(P + T + D^2) = V(D^2)$ where P is the number of parcels, T is the number of trucks, and D is the number of destinations. Space-complexity of destinations is exponential because each destination stores the distance to every other destination.

   The space-complexities for other major segments of the program such as the

controller and UI are V(1), as their space and overhead requirements are independent from the number or parcels, trucks, or destinations.
Thus, the space-complexity of the program overall is also $V(D^2)$.

The time-complexity of the algorithm is $O(N\ *\ N)\ =\ O(N^2)$ where N is the number of parcels. Time-complexity is exponential because for each parcel, the algorithm will search roughly every remaining parcel for the nearest neighbor.

The time-complexities of other major segments of the program, the user interface and reading of xml files, are both O(N). While the user interface is able to lookup and display information for individual trucks and parcels in O(1) time, displaying information for all packages still has O(N) complexity, where N is the number of parcels.

Thus, the overall time-complexity of the program as a whole is also $O(N^2)$.

4. Space-complexity of the application is dominated by the number of delivery destinations, so the program will scale nicely with an increase in the number of parcels alone. The time-complexity is exponential however, and adding more packages will quickly increase the required amount of processor resources. The code itself is capable of adapting to a large increase in parcels, but could run into trouble meeting deadline requirements if the number of trucks is not increased to accommodate an increase in the number of parcels with deadlines.

5. The program is efficient and easy to maintain for a couple reasons. First, using XML format to load data into the program makes adding additional packages or editing existing packages easier and more flexible than hard coding values. Second, the algorithm itself is written to handle a largely variable number of packages, so the user can add or remove packages without experiencing any changes in expected behavior. Lastly, all constants such as truck capacity, average truck speed, and even hours of operation are kept together in the same file, allowing operational parameters to be edited quickly and in one place.

6. Notably, the algorithm is self-adjusting in two main places. First, the algorithm will reserve a truck at the hub if it notices any parcels are delayed within a specified time after the start-of-day. Second, after loading a parcel onto a truck, the algorithm checks if that parcel is part of a group. If it is, all members of the group will be moved to the top of the loading priority.

   The strengths of these self-adjusting structures are that they ensure deadlines can be met even for packages which are delayed and they also ensure that if a package must be delivered with another package, it will be. Unfortunately, prioritizing “group” packages over potentially nearer packages will undoubtedly increase truck mileage, and reserving trucks will extend the length-of-day, though That particular parameter has no requirements in this case.

D. In addition to using python’s built in dictionaries for storing parcels, trucks, and destinations, the program uses a self-adjusting data structure called “PackageGroup.” The PackageGroup class can be found in packagegroup.py and itself uses a dictionary as its underlying data structure to store groups or packages which must be delivered together or other PackageGroups.

1. The PackageGroup data structure accounts for relationships between stored data points using a dictionary. Both parcels and PackageGroups have an overridden hash function, and items are inserted to the PackageGroup's dictionary attribute as both the key and value to allow for O(1) lookup (zyBooks 7.1).

I. Justifications for the chosen algorithm are as follows:

1. One strength of the algorithm is its flexibility allowing for variable numbers and packages and editing of parameters. A second strength of the algorithm is its $O(N^2)$ complexity achieved by using dictionaries for O(1) storage and access times.

2. The algorithm meets the scenario requirements. All parcels are delivered by their deadlines, parcels follow truck restrictions, parcels which must be delivered with other parcels are delivered together, and at end-of-day the trucks are driven a total of 124.5 miles which is within the 140 mile parameter.

3. As an alternative to "nearest neighbor," one could use heuristics. In this case, one might place the parcels into 'buckets' by area code and then load these buckets together onto trucks. This heuristic approach is different from the algorithm utilized in the program because it does not find the nearest package to the last one; instead it assumes packages with the same area code are relatively close together. This reduces time-complexity but undoubtedly increases total mileage driven.

   Alternatively, instead of storing distances between delivery destinations as a matrix in each destination, one could organize the destinations into a graph and then use a "Breadth First Search" to plot an optimal delivery route. Such an approach would increase the time-complexity, especially if the graph was not sparse, but would decrease the space-complexity.

J. If I were to do this project again, do away with the Time and Truck classes to save overhead and just calculate when each package was delivered.

K. Justification for the PackageGroup data structure are as follows:

1. PackageGroup meets all scenario requirements by using a hash table. Lookup time is constant, so it is not affected by the number of packages. However, the space usage of the data structure is linear. Again, lookup time for delivery destinations and trucks is constant, so is unaffected by additional trucks or cities.

2. An array could meet scenario requirements. An array differs from the hash table used because finding a specific element would require iterating through the stored elements thus making the time-complexity linear.

   A set could also meet scenario requirements. In some languages, such as java, a HashSet, like the hash table, will have O(1) lookup time in the best case scenario, but in Python, a set, like an array, will have O(N) lookup time because it must iterate through all elements to search for a specific element.

L. Sources

1. “C950: Data Structures and Algorithms II.” zyBooks, zyBooks, 2018. *learn.zybooks.com/zybook/WGUC950AY20182019*