Chris Loverchio
#000875972
C950 – Data Structures & Algorithms II

Algorithm Analysis and Overview

A. Identify a named self-adjusting algorithm that was used to deliver the packages.

This implementation of the project uses a greedy algorithm to facilitate the delivery of packages to their destinations. Using Dijkstra's algorithm, the application is able to make a decision about what the next best delivery location is going to be from the current location, therefore making a locally optimal choice at each location.

B1. Explain the algorithm's logic using pseudocode.

<u>Package Delivery</u>

Time complexity: O(N^2)

- **Delivery route: O(N^2) time**

Space complexity: O(N)

```
given a truck and its associated start time:
    set the truck's current time to the start time
    set the truck's current location to the starting location (hub)
    get the undelivered packages assigned to the truck
    create a "delivery route" using the starting location and the truck's assigned packages
    for each delivery in the delivery route:
        deliver the package using the truck
    get the shortest distance from the truck's current location back to the hub
    return to the hub
    update the total mileage using the truck's mileage
```

<u>Delivery Route Creation</u>

Time complexity: O(N^2)

- **Closest delivery: O(N) time**

Space complexity: O(N)

```
while there are assigned packages:
    find the closest delivery to the current location
    add the closest delivery to the delivery route
    update the current location to reflect the address of the closest delivery
    unassign the package associated with the closest delivery
```

Finding the Closest Delivery

Time complexity: O(N)
Space complexity: O(N)

```
for each package in the remainder of the assigned packages:
    find the shortest distance between the current location and the package address
    add the package and the calculated distance to a list of possible deliveries
return the possible delivery with smallest distance
```

Calculating the Shortest Distance

Time complexity: ~O(E log N)

- **Cache hit: O(1) time**
- **Cache miss: O(E log N) time - heap implementation of Dijkstra's algorithm**

Space complexity: O(N)

- **Cache: O(N) space**
- **Graph: O(V) space**

```
given an origin and destination address:
    if the shortest distance for the given origin and destination has been previously cached:
        return the shortest distance
    create a graph of locations and distances using the given origin
    find the shortest path within the graph (Dijkstra's algorithm)
    set the shortest distance to the sum of distances in the shortest path
    add the shortest distance to the cache for the given origin and destination
    return the shortest distance
```

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

The application was programmed on an Ubuntu OS using the Python3 programming language in conjunction with the PyCharm IDE.

B3. Evaluate the space-time complexity of each major segment of the program.

Space-time complexity analysis of the delivery logic is available in section B1.

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

The packages were manually loaded into the truck in this implementation; however, with minor changes to the loading functionality the application can function with any arbitrary distance and location data, provided it follows the same CSV formatting. Caching was also implemented in different spots within the application to reduce the impact of repeated queries and support scalability.

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

The application as a whole leverages object-oriented constructs and abstractions to promote maintainability. Service layer classes are named in accordance with functionality they provide. Functions are designed to be composable while having limited responsibilities.

B6. Discuss the strengths and weaknesses of the self-adjusting data structures.

There were two major self-adjusting data structures that were used in this implementation of the project. The first one being a hash table, which was largely used to store package data by its corresponding id but was also used to cache distance data as well as status queries from the console. The benefit to using a hash table in this context is that it provides the capability for constant time lookup operations. The disadvantages being the additional overhead in terms of spacial complexity and the risk of collisions during the insertion process. The second self-adjusting data structure that was used is a min heap. The heap is mainly used to facilitate Dijkstra's algorithm, providing a convenient way to access the smallest unvisited vertex from the distance graph in constant time

C1. Create an identifying comment within the first line of a file named "main.py" that includes your first name, last name, and student ID.

Identifying comment provided on line 1 of *main.py*

C2. Include comments in your code to explain the process and the flow of the program.

Single-line comments and multi-line docstrings are included on all notable functions.

D1. Explain how the data structures account for the relationship between the data points being stored.

As mentioned in section B6, the use of a hash table allowed for the mapping of package ids to the associated package data. Allowing for quick retrieval of all of the relevant data using just an identifier.

E. Develop a hash table, without using *any* additional libraries or classes, that has an insertion function that takes the package data as input and inserts the components into the hash table.

See *hashtable.py* for the initial implementation and *packaging_service.py* for its use with package data.

F. Develop a look-up function that takes the package id as input and returns the corresponding data elements.

See *hashtable.py* for the initial implementation and *packaging_service.py* for its use with package data.

G. Provide an interface for the user to view the status and info (as listed in part F) of *any* package at *any* time, and the total mileage traveled by *all* trucks.

See *main.py* for the console interface and *tracking_service.py* for the status functionality.

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.

Screenshots are included in their own directory within the project structure.

I1. Describe at least 2 strengths of the algorithm used in the solution.

The greedy algorithm outlined in previous sections has a few distinct advantages. It leverages the current location of the truck to make an informed decision about where to go next. Its fairly easy to reason about and discern as a user. There's also no need for backtracking or the analysis of previous choices.

I2. Verify that the algorithm used in the solution meets all requirements of in the scenario.

The implementation of the described algorithm was able to successfully deliver all 40 packages within ~120 miles, while taking into consideration certain deadlines and delays.

I3. Identify two other named algorithms that could have been used to meet the requirements.

Rather than choosing the next best delivery location at each step, a dynamic programming approach could have been used to build a more optimal route using previous steps in delivery process. An approach such as this would likely have a larger space complexity but can more accurately guarantee that an optimal route is generated. Recursive backtracking is another possible approach that could have yielded results for this kind of project. Using a recursive backtracking technique, each address starting from the hub could be explored, if one becomes unfeasible we could revert back to a previous address and explore other options. Similar to dynamic programming, this type of technique would likely require additional overhead compared to a greedy implementation but could potentially explore all permutations of delivery routes.

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

ZyBooks. (n.d.). Retrieved December 21, 2020, from https://learn.zybooks.com/

Edmonds, J. (2008). *How to think about algorithms*. Cambridge: Cambridge University Press.