C950 Data Structures and Algorithms II
Date: May 1st, 2024

Introduction

An effective algorithm is required by Western Governors University's Parcel Service, known as WGUPS, to enhance their package delivery system. The goal is to create software that ensures deliveries are on schedule, adheres to package specifications, and keeps the total trip distance to less than 140 miles. To be used in other cities, this system needs to be scalable, manage exceptions like wrong addresses, and track package status.

A: Algorithm Identification

For planning the truck routes of WGUPS' deliveries, a self-adjusting algorithm that can be used is called the "Nearest Neighbor Algorithm." The Nearest Neighbor Algorithm works by starting at a specific point. Then it begins to find the nearest neighbor, by choosing the nearest, unvisited location based on the shortest path available from its current location. It updates its current location to the nearest location it has moved to and then repeats the process of moving to the nearest, unvisited location until all destinations have been visited. Lastly, once all locations have been visited, it returns to the starting point to complete the cycle.

B & B1: Data Structure Identification and Explanation of Data Structure

A hash table can be used as the primary data storage structure. A hash table in itself is inherently self-adjusting, which makes it the best choice for managing and accessing large data sets. For a data structure to be considered "self-adjusting", it refers to the ability to rearrange itself after operations are performed on it. Hash tables can resize or rehash their data to accommodate more entries to maintain efficient operations.

The hash table is designed for storing package information. Each package is given a unique "Package ID" that will be used as the key for the hash table, allowing for quick data retrieval. When a new package is inserted into the hash table, its "Package ID" is used by the hash function to determine the bucket it will be stored in. The hash function calculates the index by taking the remainder of "Package ID" divided by the current capacity of the hash table. Hash tables can also handle collisions, which occur when there are different "Package IDs" that are hashed into the same index. To manage this, the hash table uses chaining where each bucket contains linked lists of entries that share the same index. If a collision occurs, the package is added to the end of the list at that bucket. This enables hash tables to effectively handle increased data size without compromising a lot on performance.

C1: Algorithm Pseudo Code

```
Algorithm: Nearest Neighbor

Procedure deliver(truck)
    #Initialize list of packages to be delivered
    Initialize toDeliver list

    # Load packages onto the toDeliver list from the truck
    Call loadPackage(toDeliver, truck)

    #Clear the truck's package list after loading them to toDeliver
    Clear truck's package list

    # Loop until all packages are delivered
    While toDeliver list is not empty
        Set nextAddressDist to a large number (e.g., 9999)
        Initialize nextPackage to none

        # Iterate through each package in toDeliver to find the closest one
        For each package in toDeliver
            Calculate destDiff as the distance from truck's currentAddress to package's address
            If destDiff is less than or equal to nextAddressDist
                Set nextAddressDist to destDiff
                Set nextPackage to current package

        # Check if a next package to deliver is determined
        If nextPackage is not null

            Append nextPackage's ID to truck's packages list
            Remove nextPackage from toDeliver list

            Update truck's milesDriven by adding nextAddressDist
            Update truck's currentAddress to nextPackage's address

            Calculate transitTime as the time taken to travel nextAddressDist at truck's speed
            Increment truck's timeElapsed by transitTime
            Increment truck's timeDelivered by transitTime

            Save the delivery time and duration in nextPackage for UI data lookup

End Procedure
```

C2: Development Environment

- **Hardware:** MacBook Air M2 Chip, 16 GB RAM
- **Operating System:** Sonoma 14.4.1
- **Language:** Python 3.9.13
- **Integrated Development Environment (IDE):** Visual Studio Code 1.88.1

C3: Space and Time Complexity

The overall time and space complexity of my entire program is O(n^2)

Hashtable.py	Function	Time Complexity	Space Complexity
Constructor	`'__init__'`	O(1)	O(n)
Hash Function	`'hash_index'`	O(1)	O(1)
Insertion/Update	`'insert'`	O(1)	O(1)
Search	`'search'`	O(1)	O(1)
Removal	`'remove'`	O(1)	O(1)

Package.py	Function	Time Complexity	Space Complexity
Constructor	`'__init__'`	O(1)	O(1)
Status Update	`'updateStatus'`	O(1)	O(1)
String Representation	`'__str__'`	O(1)	O(1)

Truck.py	Function	Time Complexity	Space Complexity
Constructor	`'__init__'`	O(1)	O(n)
String Representation	`'__str__'`	O(1)	O(1)

Matrix.py	Function	Time Complexity	Space Complexity
Constructor	`'__init__'`	O(1)	O(n^2)
Getter	`'getAddressID'`	O(1)	O(1)
Getter	`'getAddressName'`	O(1)	O(1)
Distance Calculation	`'distanceBetween'`	O(1)	O(1)
Package Loading	`'loadPackage'`	O(n)	O(n)
Nearest Neighbor Loop	`'deliver'`	O(n^2)	O(n)

Main.py	Function	Time Complexity	Space Complexity
Reading package CSV and inserts into hash table	`'decodePackageCSV'`	O(n)	O(n)
Converting CSV into a 2D adjacency matrix	`'decodeDistanceCSV'`	O(n)	O(n^2)
Reads Addresses from CSV and Maps to a Dictionary	`'decodeAddressCSV'`	O(n)	O(n)
Formats and Prints Package Information	`'printPackageUIFormat'`	O(1)	O(1)
Outputs Delivery Route of a Truck	`'printDeliveryRoute'`	O(n)	O(n)
Outputs Packages for a Specific Truck at a Specified Time	`'printTruckPackage'`	O(n)	O(n)
Outputs Information for All Packages on All Trucks	`'printAllPackages'`	O(n)	O(n)
Prints Details of a Single Package at a Given Time	`'printSinglePackage'`	O(1)	O(1)
Calculates Total Mileage of Three Trucks	`'printTotalMileage'`	O(1)	O(1)
User Interface Function to Select a Truck	`'selectTruckUI'`	O(1)	O(1)
Main function	`'main'`	O(n)	O(n^2)

C4: Scalability and Adaptability

Using the nearest neighbor algorithm and a chaining hash table allows for the program to effectively adapt to a growing number of packages. Hash tables are designed specifically for the package class. Through collision handling by chaining and dynamic resizing, it maintains performance even as the size of packages grows. The nearest neighbor algorithm is used to find the shortest path by selecting the nearest package based on certain criteria and operates independently from the hash table. Their integration increases efficiency for quick data access and improved delivery routes. Additionally, the truck class manages the distribution of packages to different trucks, allowing the system to scale up by adding more trucks as needed. By manually loading trucks, it contributes to scalability by providing flexibility to adapt to growing package volumes and delivery routes without having to adjust the program.

C5: Software Efficiency and Maintainability

The software is designed by using an object-oriented approach that incorporates a chaining hash table and the nearest neighbor algorithm that allows for efficiency and maintainability. The hash table provides an average constant access time for package data that is needed for the nearest neighbor algorithm to quickly compute its delivery routes. With the use of object-oriented principles, it designates data and operations within classes such as Package, Truck, Matrix, and Hashtable. This allows for easy data management, route computation, fast data access, and maintainability. Developers can easily find and modify codes due to it being separated, where each class handles a specific part of the program. Additionally, the use of comments combined with the object-oriented approach, makes it easier for developers to understand, maintain, and perform updates on.

C6: Self-Adjusting Data Structures

Some, but not all, of the advantages of using a hash table with changing are:

- **Handling Collisions:** Chaining is an example that can effectively manage collisions. This is done by storing elements in a linked list within each bucket (Lysecky, R., & Vahid, F., 2018). As the size of entries increases, it can maintain performance as long as the load factor is kept at a reasonable level.
- **Dynamic Resizing:** Because chaining is more flexible, it can handle various data sizes by adding more entries into the chain without having to resize the entire table. This allows for more scalability for larger data sets.
- **Fast Data Retrieval:** With a well designed hash function that allows for keys to be uniformly distributed across buckets, the average time complexity for retrieving data is only O(1), known as constant time.

Some of the weaknesses of implementing a hash table with chaining include:

1) **Increased Memory Usage:** Each entry for chaining requires additional space due to pointers in each linked list element. This increases the amount of memory needed especially when the amount of entries grows. This can be an issue where memory is limited.
2) **Access Time:** While chaining has an average time complexity of O(1) for access time, this is not the case if many keys are hashed into the same number of buckets. The access time complexity increases to O(n), making this the worst-case scenario.
3) **Hash Function Dependent:** The performance of a chaining hash table relies on an effective hash function to uniformly across buckets. If not, it can result in buckets being overloaded and can increase access time and reduce overall performance.

C7: Data Key

Package ID was selected as the key for storing and retrieving package data. This is due to it being unique to each package, ensuring that no two or more packages are mistaken for one another and eliminating any potential confusion. Other potential keys like delivery address, city, and status, can be the same amongst other packages which makes the Package ID the optimal choice. Additionally, Package IDs are also numeric. This allows for a more convenient way to compute the hash value and for quicker access time. This efficiency is good for operations such as querying or updating package information that might be needed for the routing algorithm.

D: Citations

Lysecky, R., & Vahid, F. (2018, June). *C950: Data Structures and Algorithms II*. zyBooks. Retrieved March 22, 2021, from https://learn.zybooks.com/zybook/WGUC950AY20182019/