Luke Williams

Preety Khatri

C950

10/20/2023

Title: C950 Task One Program Planning

A. The algorithm I will be using to solve the WGUPS issue is the Greedy Algorithm. I chose this because it is simple and efficient. It will choose the nearest address since that is the lowest cost, it is also dynamic as it won't have to reconsider the entire route when planning which address to go to next as packages are delivered by other trucks.

B. I will be using a hash table to store data.
   1. The package ID will serve as the key and the other data, such as the address, truck capacity, package weight, and zip code, will serve as the value. The hash table has a constant time complexity for most operation and can adjust size based on the number of elements, making it an efficient choice for the WGUPS problem. The relationship between data components is preserved as each package ID has associated data containing all the details about the package.

C. Here is an overview of the program in pseudocode:

```
FUNCTION findNearestDelivery(currentLocation, undeliveredPackages):
    nearestPackage = NULL
    shortestDistance = INFINITY

    FOR each package in undeliveredPackages DO
        distance = getDistance(currentLocation, package.address)
        IF distance < shortestDistance THEN
            shortestDistance = distance
            nearestPackage = package
        END IF
    END FOR

    RETURN nearestPackage
END FUNCTION

FUNCTION greedyDeliveryRoute(truck, undeliveredPackages):
    currentLocation = HUB

    WHILE truck is not empty AND undeliveredPackages is not empty DO
        nextPackage = findNearestDelivery(currentLocation, undeliveredPackages)

        IF nextPackage IS NOT NULL THEN
            travelTo(nextPackage.address)
            deliver(nextPackage)
            remove nextPackage from truck
            remove nextPackage from undeliveredPackages
            currentLocation = nextPackage.address
        END IF
    END WHILE
END FUNCTION

MAIN:
    // Load trucks with packages (other constraints considered)
    WHILE there are undelivered packages DO
        FOR each truck not in service DO
            greedyDeliveryRoute(truck, undeliveredPackages)
        END FOR
    END WHILE
END MAIN
```

2. I will be using the following software and hardware:
   Software:
   Python 3.9
   Visual Studio Code
   Windows 11
   Hardware:
   16 GB RAM
   I9-13900H Processor

3. I've identified the space-time complexity of each major segment of the code in big-O notation
   - Function 'findNearestDelivery'
     - The time complexity for looping through and checking the distance of 'undeliveredPackages' is O(n)
     - The space complexity for storing nearest package and shortest distance is O(1)
   - Function 'greedyDeliveryRoute'
     - The while loop iterating over packages in worst case scenario is O(n)
     - Within the loop 'findNearestDelivery' has a time complexity of O(n)
     - Overall, this function has a time complexity of O(n^2)
     - The space complexity for storing the current location and next package is O(1)
   - Main Segment(Overall)
     - While loop is time complexity O(n)
     - The nested for loop for each truck is time complexity O(1)
     - Within the loops, calling 'greedyDeliveryRoute' is time complexity O(n^2)
     - Overall, the main segment is time complexity O(n^3)
     - Overall, the main segment is space complexity O(n)

4. My space complexity for the program is linear O(n). So as the number of packages grows the required memory will grow linearly. Linear growth is manageable especially with modern tech, so this is no cause for worry. The time complexity for the program is O(n^3) so it is polynomial. This isn't as scalable as a linear or logarithmic complexity but should still be manageable with a reasonable number of packages. This could become an issue if the number of packages grows too large. Given that we have three trucks though, if the number of packages grows, adding another driver would help. The inherent nature of the greed algorithm allows for adaptablility and flexibility in its decisions. Lastly the solution is designed around a distance table, so it would not be difficult to use this program for other locations.

5. The software design is efficient for the following reasons: modularity, direct path optimization, dynamic adaptability, and data driven. The program is also easy to maintain for the following reasons: readable code with commenting, modularity, scalable data structure, flexible for future upgrades, and the modularity allows for easier unit testing.

6. My data structure I'm using has its weaknesses and strengths
   - Strengths
     - Constant average time complexity O(1): This data structure has this time complexity for search, insert, and delete operations. This is possible because it uses a unique key to directly access a location in memory.
     - Flexible Key-Pair Relationships: Data is stored in key-value pairs, meaning you can use almost any type of kata as a key and directly associate it with a specific piece of information.
     - Avoids wasted memory: Hash tables only allocate memory for elements that are added, which leads to more efficient memory usage.
     - Handles duplicates well: Since each key in a hash table is unique, it is easy to avoid inserting duplicate information. If you insert an element with a key that already exists, then it is either overwritten or rejected typically.
   - Weaknesses
     - Space complexity: Hash tables can sometimes use more memory than necessary.
     - Collision handling overhead: When two keys hash to the same index, the hash table will need a way to handle it. This will require overhead and comes with its own complexities
     - Ordering: Hash tables don't maintain any consistent order for stored keys, so if elements need to be retrieved in a specific order this may not be the most optimal choice

7. Package ID is the best choice for a key. Every package ID is guaranteed to be unique for each package, so no collisions happen because of the key. They can also be assigned values in a consistent manner, making it predictable and easy to manage. It also has good supporting relationship with the rest of the data which can be embedded/linked easily to the package ID without changing the structure or causing redundancy.

Works Cited

Data Structure and algorithms - hash table. Online Tutorials, Courses, and eBooks Library. (n.d.). https://www.tutorialspoint.com/data_structures_algorithms/hash_data_structure.htm

Greedy algorithm. Programiz. (n.d.). https://www.programiz.com/dsa/greedy-algorithm