C950 – Task 1 Writeup

# Author: Ian Crisp

# Student ID: 010377033

# Project: C950 – Task 1

# A. Named Self-Adjusting Algorithm

The algorithm used in the program is the Nearest Neighbor Algorithm. This algorithm is particularly useful for solving the Traveling Salesman Problem, which closely resembles the package delivery problem faced by WGUPS. The algorithm starts at the depot and selects the nearest package to deliver based on the current location, thereby minimizing the distance traveled.

# B. Self-Adjusting Data Structure

## Data Structure Identification

The data structure used for storing package data is a Hash Table.

Hash tables, used for package storage in the program, are highly efficient for quick look-ups and insertions (Cormen et al., 2009).

## 1. Detailed Explanation of Hash Table Structure

The Hash Table is implemented as an array of 'buckets', each containing one or more key-value pairs. The key is the package ID, and the value is a package object that stores details such as address, delivery time, and other constraints. A hashing function is used to map each key (package ID) to an index in the array, which is where the key-value pair will be stored. This ensures that look-up operations are, on average, performed in constant time \(O(1)\).

Collision Resolution: In cases where two keys hash to the same index, a collision is resolved using chaining. Each 'bucket' at an index can contain a linked list of key-value pairs. This allows for efficient handling of collisions, although the worst-case time complexity can degrade to \(O(n)\) if many collisions occur.

Overall, the hash table is designed for quick access and updates, making it ideal for a package delivery scenario where time is crucial.

# C. Core Program Logic and Functionality

## C1. Algorithm's Logic

Stated Problem:

The objective is to create an algorithm that efficiently plans the delivery of packages from a single depot to various locations in a city.

Requirements:

1. The algorithm should minimize the total distance traveled by the delivery trucks.  
2. The algorithm should take into account package-specific constraints such as delivery deadlines, delays, and required delivery times.

Pseudocode:

1. Initialize the starting point as the WGUPS depot.  
2. Create an empty list called 'delivered\_packages'.  
3. WHILE there are packages to be delivered DO:  
 a. Initialize 'nearest\_distance' to a large number (e.g., infinity).  
 b. Initialize 'nearest\_package' to None.  
 c. FOR EACH package 'p' in the list of packages to be delivered DO:  
 i. Calculate the Euclidean distance 'd' between the current location and the location of 'p'.  
 ii. IF 'd' < 'nearest\_distance' THEN:  
 - Update 'nearest\_distance' to 'd'.  
 - Update 'nearest\_package' to 'p'.  
 d. Deliver 'nearest\_package'.  
 e. Remove 'nearest\_package' from the list of packages to be delivered.  
 f. Add 'nearest\_package' to 'delivered\_packages'.  
 g. Update the truck's current location to the location of 'nearest\_package'.  
4. RETURN 'delivered\_packages'.

The nearest-neighbor algorithm is optimized for smaller data sets and provides a reasonable trade-off between efficiency and optimality (Wikipedia, 2022).

## C3. Space-Time Complexity Using Big-O Notation

In this section, we discuss the space-time complexity of each major segment of the program:

1. Hash Table for Package Storage:

Space Complexity: \(O(n)\), where \(n\) is the number of packages. Each package requires a constant amount of space in the hash table.

Time Complexity: Average \(O(1)\) for look-up and insertion operations.

2. Nearest Neighbor Algorithm:

Time Complexity: \(O(n^2)\) in the worst case, as for each package, we may have to scan through all remaining packages to find the nearest one.

3. Package Delivery:

Time Complexity: \(O(n)\), where \(n\) is the number of packages to be delivered. Each package is visited once for delivery.

## C4. Scalability and Adaptability

The program is designed with scalability and adaptability in mind, taking into account various factors that contribute to its performance and ability to handle an increasing number of packages.

1. Scalability:

The hash table used for package storage allows for efficient package look-up, even as the number of packages increases. Its average time complexity for look-up operations remains constant at \(O(1)\), ensuring rapid access. However, the nearest-neighbor algorithm has a time complexity of \(O(n^2)\) in the worst case. This means that as the number of packages grows, the time required for package sorting and delivery could increase quadratically.

2. Adaptability:

The program's modular design allows for easy incorporation of additional features or algorithms in the future. For instance, more advanced algorithms like k-d trees or Dijkstra's algorithm could be integrated for efficient nearest-neighbor searches or route optimization, respectively. This adaptability ensures that the program can evolve to meet changing requirements and performance expectations.

## C5. Software Efficiency and Maintainability

The program is designed to be both efficient and maintainable, with various features contributing to each aspect:

1. Efficiency:

The use of a hash table for package storage contributes to the program's efficiency by enabling quick look-up and insertion operations with an average time complexity of \(O(1)\). Furthermore, the nearest-neighbor algorithm, despite its \(O(n^2)\) time complexity, is optimized for smaller data sets and provides a reasonable trade-off between efficiency and optimality. The use of Python’s in-built libraries for data manipulation also contributes to reducing the program's execution time.

2. Maintainability:

The program's modular design, which includes separate classes for packages, trucks, and the hash table, makes it easier to manage and extend. This modularity allows for easier debugging, testing, and future expansion of the program's functionalities.

## C6. Self-Adjusting Data Structures

The hash table used in the program has both strengths and weaknesses, which are outlined below:

1. Strengths:

The hash table is highly efficient for quick look-ups and insertions, with an average time complexity of \(O(1)\). This is particularly beneficial in a package delivery scenario where rapid access to package data is crucial.

2. Weaknesses:

The hash table is susceptible to collisions, where two different keys hash to the same index. Collisions can degrade the performance of the hash table by causing look-up and insertion operations to take longer. In the worst-case scenario, the time complexity could degrade to \(O(n)\), where \(n\) is the number of keys that collide. While techniques like open addressing or chaining can be used to mitigate this weakness, it remains a challenge that needs to be carefully managed.

# D. Sources

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.  
  
2. Wikipedia. (2022). Nearest Neighbor Algorithm. Retrieved from https://en.wikipedia.org/wiki/Nearest\_neighbour\_algorithm