C950 Task-1 WGUPS Algorithm Overview

Kerrie Abrams

ID # 010894830

WGU Email: kabra70@wgu.edu

C950 Data Structures and Algorithms II

# Introduction

Western Governors University Parcel Service requests the development of a program that ensures their delivery system delivers 40 packages on time, under 140 miles and meets all package requirements. The program should include a user interface to allow checking the status of any or all packages at a given time.

# A. Algorithm Identification

This scenario is an example of the Traveling Salesman Problem, a problem that requires finding the shortest path of a given set of locations. A valid algorithm to solve this problem is the 2-Opt Algorithm. The 2-opt algorithm is a simple, local search algorithm that creates a path without any intersecting edges, effectively, creating the shortest path (Wikipedia, 2024).

# B. Data Structure Identification

This scenario calls for the use of a self-adjusting data structure to store the data of packages that must be delivered. This information must be stored and retrieved in a fast and efficient manner. One possible data structure would be a hash table to store package objects, or instances of a package class.

# B1. Explanation of Data Structure

Hash tables store unordered key-value pairs within buckets using a hash function. This means, data can be retrieved, updated, or inserted with simply knowing the key. In this case, the key is a package ID, unique to every package and the value is the package object.

# C1. Algorithm’s Logic

2-Opt Algorithm (Wikipedia, 2024)

*# First, load the truck with packages it should deliver*

TruckPackages = LoadTruck(list\_of\_packages)

*# Then, create a random route for the truck.*

RandomRoute = GenerateRandomRoute(TruckPackages)

*# Pass the RandomRoute to the 2-Opt Algorithm to find the best route*

2-Opt Algorithm(RandomRoute):

*# Allow the algorithm to assume the random route is the best route*

BestRoute = RandomRoute

*# Find the total distance of that route and assign it to Best Distance*

BestDistance = FindTotalDistance(BestRoute)

*# Let the algorithm assume there is improvement so it may begin.*

FoundImprovement = True

*# The algorithm will keep looping, searching for the best route until it no longer # finds improvement*

While FoundImprovement = True:

*# Set FoundImprovement to False so that it will stop if no improvement is # found.*

FoundImprovement = False

*# For every possible combination of two points, do the following.*

For each combination of locations in BestRoute as (pointA, pointB):

*# Perform a 2-Opt Swap on the route and return it as a new route.*

*# A 2-opt swap breaks the route into three parts and forms a new # route from them.*

*# part 1 is from the start of the route to point A*

part\_1 = BestRoute[start] to BestRoute[pointA]

*# part 2 is from point A to point B in reverse.*

part\_2 = Reverse(BestRoute[pointA] to BestRoute[pointB])

*# part 3 is from point B to the end of the route*

part\_3 = BestRoute[pointB] to BestRoute[end]

*# Put the parts together into a new route*

NewRoute = part\_1 + part\_2 + part\_3

*# Calculate the total distance of the new route*

NewDistance = FindTotalDistance(NewRoute)

*# Compare the new route distance with the best route distance and keep the better route that is shorter.*

If NewDistance < BestDistance:

*# The new route becomes the best route*

BestRoute = NewRoute

BestDistance = NewDistance

*# Let the algorithm know that improvement was found so that # it may keep searching for an even better route.*

FoundImprovement = True

*# Once the algorithm determines there can be no more improvements, return the # best route.*

return BestRoute

# C2. Development Environment

The following hardware and software will be used for the development of this application.

Hardware Specifications:

Computer: AMD Ryzen 5 5600x 6-Core Processor, 32 GB RAM

Display: Dell 24 Monitor

Keyboard: Logitech MK270

Mouse: Logitech M240

Software Specifications:

OS: Windows 11 version 23H2

IDE: IntelliJ IDEA Ultimate 2023.3.4

Language: Python 3.12

Version Control: GitHub

# C3. Space and Time complexity using Big-O notation

The following table displays the time complexity in Big-O notation of every major code block in the program.

|  |  |  |  |
| --- | --- | --- | --- |
| File | Code Block | Time Complexity | Space Complexity |
| Truck.py | \_\_init\_\_() | O(1) | O(n) |
| Truck.py | \_\_str\_\_() | O(1) | O(1) |
| Package.py | \_\_init\_\_() | O(1) | O(1) |
| Package.py | \_\_str\_\_() | O(1) | O(1) |
| Package.py | update\_status() | O(1) | O(1) |
| CreateHashTable.py | \_\_init\_\_() | O(n) | O(1) |
| CreateHashTable.py | \_get\_bucket\_list() | O(1) | O(n) |
| CreateHashTable.py | insert() | O(n) | O(n) |
| CreateHashTable.py | remove() | O(n) | O(n) |
| CreateHashTable.py | search() | O(n) | O(n) |
| Main.py | load\_package\_data() | O(n) | O(n) |
| Main.py | load\_file\_data() | O(1) | O(n2) |
| Main.py | get\_index() | O(n) | O(n) |
| Main.py | get\_distance() | O(1) | O(1) |
| Main.py | generate\_ran\_tour() | O(n) | O(n) |
| Main.py | two\_opt\_distance() | O(n2) | O(n) |
| Main.py | two\_opt\_swap() | O(1) | O(n) |
| Main.py | two\_opt\_implement() | O(n2) | O(n) |
| Main.py | deliver\_packages() | O(n2) | O(n) |
| Main.py | print\_package() | O(1) | O(1) |
| Main.py | print\_trucks() | O(n) | O(n) |
| Overall | - | O(n4) | O(n2) |

# C4. Scalability and Adaptability

Theoretically this application should perform for any number of packages. By creating a hash table with lists for buckets, the storage capacity is essentially limited only by hardware memory. In addition, the 2-Opt algorithm is self-adjusting as well, meaning it would be able to determine the best route for any number of packages. However, in large enough numbers, there could be hash collision and excessive runtime due to the suboptimal time-complexity (aayushi24…, 2023).

# C5. Software Efficiency and Maintainability

Ultimately, the program should be efficient and easy to maintain. By incrementally developing methods, minimalizing “hard coding”, providing informative comments, and ensuring code readability, any adjustments that need to be made in the future should be relatively easy to achieve.

# C6. Self-Adjusting Data Structures

Hash tables provide a fast and efficient way to store, delete, and find data. However, hash tables are created with a finite amount of space, meaning they can eventually reach max capacity. In addition, because they must be created with a finite amount of space, they can be a poor use of memory (aayushi24…, 2023). However, in Python, lists do not have a fixed size. Thus, by creating a hash table using lists for buckets, they essentially become infinite in size.

# C7. Data Key

For efficient delivery management, packages must be stored within the hash table using a key and value. As described in section B1, the package ID of a package is a unique identifier that can be used as the key in a key-value pair insertion into the hash table. Since every package has a different package ID, this should eliminate and chance of hash collision.

# D. Sources

1. Wikipedia contributors. (2024, March). *2-opt.* Retrieved from Wikipedia: <https://en.wikipedia.org/w/index.php?title=2-opt&oldid=1216337937>

2. aayushi24….(2023, March) *Applications, Advantages and Disadvantages of Hash Data Structure.* Retrieved from Geeks for Geeks: <https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-hash-data-structure/>