C950 WGUPS Algorithm Overview

Adam Gascoyne

ID #001254864

WGU Email: agasco1@my.wgu.edu

Date

Nov. 15th, 2022

C950 Data Structures and Algorithms II

# Introduction

The purpose of this project is to use python, through PyCharm, to create a program to deliver packages using the WGUPS delivery service. Within this program, we need to create an efficient method to deliver packages under 140 miles. I chose to use the Nearest Neighbor algorithm. Every package has varying attributes that make the delivery process more complex such as delivery deadlines, wrong addresses, delivery with limitations, and others. To get these limitations, and all the data for this project, we were given CSV files that we were instructed to pull data from. The code will output a user interface that will take in a user’s time input and provide the status of all packages at that point in time.

# A. Algorithm Identification

The Nearest Neighbor algorithm uses a simple method of solving the traveling salesman problem. Using the Nearest Neighbor algorithm we start at a specific hub, WGUPS, find the closest address to that, and deliver the package accordingly. We compare all the distances from the current address to the next possible hub on the list. the lowest distance will be the next destination. This will occur until the last destination on the list, and the truck will head back to the WGUPS hub.

# B1. Logic Comments

The most important code for time complexity in the nearest neighbor comes from my function “delivering\_packages”. It has a complexity of 0(n^2).

Nearest Neighbor:

Dictionary with hubs of comparable hubs and their different distances from each other.

Dictionary\_hub\_distances = {hub1: {{hub2: distance from other},{hub3: distance from other}}}

Use a distance-calculating formula to go through the dictionary to find the distance from one hub to another.

Def distance\_calculating(hub1, hub2)

There will be a comparison hub

Initial\_hub = 9999

Using a for loop we can go through all the package hubs.

For I in range(package amount):

Using an if statement we can compare different distances from the first hub.

If distance\_calculating(hub1, hub[i]) < initial\_hub):

The smaller distance will be saved to a variable and compared to the next hub until all hubs have been compared.

Next\_hub = distance\_calculating(hub1, hub[i])

After this, the “next\_hub” will be compared to the remaining destination hubs in the same fashion to find the next closest one it will become the “next\_going” hub.

This will create a loop with nested loops creating a 0(n^2) situation. The initial loop to iterate through the package hubs and then calling the “Def distance\_calculating” causes a nested loop.

# B2. Development Environment

The environment used was PyCharm Community edition 2021.3.3 coding with the Python 3 language. No outside extensions/libraries were used and the program was coded on a PC running windows 10.

# B3. Space-Time and Big-O

The program uses a maximum of O(n^2) and the code above examples the major components that use this level of time complexity.

# B4. Scalability and Adaptability

If the program uses the same data format that is in the CSV file, it should work. The program just compares all the data in the CSV files and creates a package object that is used to compare distance for the Nearest Neighbor method. The adaptability can be limited by the manual input of package objects into the truck objects for creating the list that will be compared in the “def delivering\_packages(truck):” function. If I created a function that sorted the packages into a list automatically, with certain criteria, this could make the whole process automated.

# B5. Software Efficiency and Maintainability

Using the Nearest Neighbor is somewhat efficient as it uses O(n^2). Other methods are more efficient than the Nearest Neighbor but the Nearest neighbor method is simple to implement on any scale. The software is easy to maintain as it is very simple and easily modified to add mor

# B6. Self-Adjusting Data Structures

The hash table makes it easy to store and update data. It makes it easy to store because regardless of the amount of data you have, you can use a simple insert function that allows you to loop through all your data and insert it into the hash table. Also, you can implement other functions such as a removal function that makes it easy to remove data by calling the function and entering the key. The weakness, of this specific problem, you don’t need it and could solve the issues of this situation with a simple dictionary or list of lists. Also, if the dataset was bigger collisions could cause the hash table to become inefficient

# C. Original Code

In python file.

# C1. Identification Information

In python file.

# C2. Process and Flow Comments

Text goes here

# D. Data Structure

A hash table was used for this project.

# D1. Explanation of Data Structure

The hash table has a functional insert, lookup, and removal function. It is used to store the data for all the packages from the CSV file. The package objects are read and inserted into the hash table by the insert function which takes the package id and then the entirety of the package info as a key, value pair. This hash table then hashes the key value and organizes the package info into a bucket to later be looked up by the get function. When the data needs to be manipulated you can use the insert function to reinsert the data using the same package id as the key, and the updated package info as the new value, and it should replace the previously stored data. Or you can use a remove function to delete an already inserted package data and then reinsert the package with new data and the same key.

# E. Hash Table

In python file.

# F. Look-Up Function

In python file.

# G. Interface

In python file.

# G1. First Status Check

Attached with this document.

# G2. Second Status Check

Attached with this document.

# G3. Third Status Check

Attached with this document.

# H. Screenshots of Code Execution

Screenshots (and possibly labels) go here

# I1. Strengths of Chosen Algorithm

The algorithm nearest neighbor is great as it is simple and can be used with a large dataset. This will work if the total package amount was increased from 40 to any number. It is simple to code and straightforward and intuitive, making the coding process easier.

# I2. Verification of Algorithm

In python file.

# I3. Other possible Algorithms

There were other options that would be useful such as the Random tour and the Nearest Insertion algorithm which is like the Nearest Neighbor but more complex. Both algorithms are usable in this scenario.

# I3A. Algorithm Differences

The Random Tour method is very simple as the hubs are randomized giving you a method that can give very efficient or inefficient results as each delivery route is going to be different. The Nearest Neighbor method will have a specific route dependent on what package is closest to the starting hub. The Nearest Insertion takes two initial hubs and adds the closest hub to either of them to the route and then again selects the closest hub to any of the hubs already in the route. This usually creates an overall shorter total distance than the Nearest Neighbor. Nearest Neighbor starts with a starting hub and finds the closest hub to that, moves to that new hub, and finds the next closest one from there. This happens until back at the initial hub.

# J. Different Approach

I would use the hash table to hold the time value of the package instead of only the package information given in the CSV file. By doing this I could more easily look up package information and it will be stored in one location. Also, I would make the delivering package function be more general, so it doesn’t have to look for specific trucks and just look for specific packages. I could do this by specifically looking for the notes in each package rather than the truck first making it so a package could be on any truck. My code is slightly longer because of this.

# K1. Verification of Data Structure

Attached with this document.

# K1A. Efficiency

Increasing the number of packages in the program would increase the time to lookup function because my lookup function relies on a loop to search for the packages in the hash table. The time complexity is O(n) as it is a single loop not nested. It would be a very insignificant increase in time depending on the number of packages used.

# K1B. Overhead

The hash table makes space for 64 buckets. I used 64 as an arbitrary number even though there were 40 packages. Adding more to this hash table would cause a collision of data and some packages would populate the same bucket.

# K1C. Implications

The number of truck objects would directly increase or decrease the space usage based on how many truck objects are initialized. The lookup time with more trucks could be more efficient with more trucks but it would be insignificant. Increasing the number of cities without increasing the number of packages wouldn’t change anything for space complexity as there will still be only 40 package objects. It wouldn’t change the lookup time either as my program loops through package data by the total number of packages in the package list. Changing the package data, such as the city hub, would not change the lookup time.

# K2. Other Data Structures

A list of lists could fulfill the requirements of a hash table with how simple the data being stored is. Also, a dictionary would suffice.

# K2a. Data Structure Differences

My hash table is a dictionary that has functions built-in and also uses a bucket system where keys are hashed to determine where the data rests in the data structure and in memory. A normal dictionary would essentially be the same thing but with no functions in its own class. Dictionaries and my hash table use a key, value pair to locate packages in the dictionary/hash table. A list of lists could be used as it is simple and we don’t necessarily need to hash any keeps to get to the same implementation of the provided data. A list of lists would be a list comprised of lists of each package in itself. You could still lookup packages by iterating through the list for whatever package you need. Dictionaries, hash tables, and lists are mutable.

# M. Professional Communication

Attached with this document.

# L. Sources - Works Cited

# References

Geeksforgeeks. (2022, 6 22). *Traveling Salesman Problem (TSP) Implementation*. Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/

Lysecky, R., & Vahid, F. (2018, 06). *C950: Data Structures and Algorithms II.* Retrieved from learn.zybooks.com: https://learn.zybooks.com/zybook/WGUC950AY20182019?modal\_name=about-zybook

Weru, L. (2022, 11 15). *11 Animated Algorithms for the Traveling Salesman Problem*. Retrieved from stemlounge.com: https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/#:~:text=3%3A%20Nearest%20Insertion&text=One%20implementation%20of%20Nearest%20Insertion,to%20be%20the%20shortest%20possible.

WGU. (2022, 11 15). *NHP2 — NHP2 TASK 1: WGUPS ROUTING PROGRAM*. Retrieved from WGU.edu: https://tasks.wgu.edu/student/001254864/course/15900034/task/2782/overview

Wikipedia. (2022, 11 15). *Travelling salesman problem*. Retrieved from wikipedia.org: https://en.wikipedia.org/wiki/Travelling\_salesman\_problem#Heuristic\_and\_approximation\_algorithms