**Package Routing Optimization Utilizing Nearest-Neighbor Algorithm**

Duncan Gwin

ID #008698673

C950 – Data Structures & Algorithms II

January 23, 2023

**PROBLEM STATEMENT**

The Western Governors University Parcel Service (WGUPS) needs to determine an efficient delivery route for their daily package deliveries. There are three trucks, two drivers, and special requirements that must be met for certain packages including priority deliveries and packages that have a delayed arrival to the hub due to a late flight. Using the knowledge gained from the C950 Data Structures & Algorithms course, a program must be developed that utilizes and efficient data structure for looking up and inserting data as well as an algorithm that will determine an optimal route for the delivery of those packages. The total route distance should not exceed 140 miles and all deliveries should be within the constraints of their special requirements. Along with this document, you will find the following files:

|  |  |
| --- | --- |
| File Name | Purpose |
| package\_data.csv | Package information including ID and delivery address |
| distance\_map.csv | Table of distances between all of the delivery destinations |
| HashTable.py | Module that includes Hash Table implementation |
| packagemgmt.py | Module with code to manage package data and objects |
| locations.py | Module with code to manage the location and distance data, graph object |
| main.py | Main program with user interface |

1. **ALGORITHM IDENTIFICATION**

The algorithm that was chosen for this project is the Nearest-Neighbor algorithm. The reasoning behind selecting this algorithm was due to the simplicity and effectiveness for a complete graph, that is a graph in which all vertices have an edge shared with all other vertices.

**B. PROGRAM OVERVIEW**

B1. The way that the nearest-neighbor algorithm functions is by taking in a list unvisited destinations and an arbitrary starting point and then visiting the closest destination (“neighbor”) of the remaining destinations. Once a destination is visited, it is removed from the list and then placed into a separate array representing the best route. This process is repeated until the unvisited destinations list is empty, representing that all destinations have been visited.

The pseudocode for this algorithm is as follows:

// the function takes the list of packages, a graph with edge weights, and

// a hash table with package data as arguments.

def create\_route(list\_of\_packages, graph, package\_hash\_table)

// Initialize empty array of unvisited destinations

unvisited\_destinations = []

// Initialize an empty array for the route

route = []

// This portion will add packages to unvisited\_destinations array

// get id from list\_of\_packages array

for id in list\_of\_packages:

// use id as the key to get the package object from hash table using // look\_up function

package = package\_hash\_table.look\_up(id)

// add package to array

unvisited\_destinations.append(package)

// This portion will determine the best-route using Nearest-Neighbor

// Loop continues until unvisited destinations is empty

while len(unvisited\_destinations) > 0:

min\_distance = 500 // arbitrary high value for comparison

next\_package = null

for package in unvisited\_destinations:

// compare\_distance returns the edge weight of two vertices

// if value is less than min\_distance, assign to min\_distance

if graph.compare\_distance(current\_location,

package\_location) <= min\_distance:

min\_distance = graph.compare\_distance(current\_location, package\_location)

// assign the package that had lowest value to next\_package

next\_package = package

// Once the package with the minimum distance is identified the package is // added to the route array

route.append(package)

// The package is then removed from the unvisited\_destinations array

unvisited\_destinations.remove(package)

// The while loop continues until there are no elements in the // unvisited\_destinations array

B2. The programming environment for writing this code is below:

**Software:**

PyCharm CE IDE

Python 3.11

**Hardware:**

Macbook Pro OS 10.13.6

2.66GHz Intel Core 2 Duo

4GB Ram

B3. The program consists of 4 major components: HashTable.py, locations.py, packagemgmt.py, and main.py.

The HashTable.py module is the code that implements the HashTable class. The insert function of the HashTable is a constant time operation and has O(1) time-complexity. The look\_up function is also a constant time operation and has O(1) time-complexity. A print\_all function was implemented to facilitate in printing out the hash table to display in the user interface. This function will iterate through the entire hash table of N elements. This has a time-complexity of O(N) for N elements of the hash table. The Big-O notation for this section is O(N).

The locations.py module is what contains code to manage the distance table and implement the graph object with the supplied data. The constructor functions for the graph class are all constant-time operations with O(1) time-complexity. The get\_distance\_data function loops through N rows of the supplied CSV, so the function has O(N) time complexity. The most expensive function in this module is the create\_graph function due to a nested loop. The outer loop iterates through N rows and the inner loop will iterate through N objects, each representing a cell of the table. The Big-O notation for this function is O(N2) so the resulting big-o for the module is also O(N2).

The packagemgmt.py module contains code to process a CSV file containing details of the packages to be delivered as well as the main algorithm to determine the best route to deliver the given list of packages. The create\_package\_table function takes the CSV as input and iterates through N rows to build the package objects. The time-complexity of the function is O(N). The most expensive function of the packagemgmt.py module is the create\_route function that determines the best route utilizing the Nearest-Neighbor algorithm. The big-o notation for create\_route is c

The last module is main.py which runs the main program and the user interface. This module sets up the main constraints outlined in the project requirements and initializes the command line for the supervisor to interact with the program. The interface function contains one while loop, which would give a run time O(N), with N being the number of options selected by the user. Technically this loop could be infinite if the user never selects the third option to quit the program. The main.py program calls functions from the other module, which would cause main.py and the overall program to have the worst-case runtime of O(N2).

B4. The solution is scalable since the hash table and code would adapt to any number of package inputs, however this may not be the optimal solution for large values of N due to the O(N2) space-time complexity.

B5. The software is easy to maintain since the code is broken into modules relating to the specific functional areas of HashTable, Package Management, Location Management. There are comments discussing the usage of the functions and their implementation which would allow future developers to easily make changes.

B6. The hash table implementation is based on the Chaining Hash Table model which allows for flexibility and a fairly simple implementation. However the table is not scaled properly to the number of inputs or if the data provided results in collisions of key values, this may prove to be a challenge for looking up and displaying package information.

**C. ORIGINAL CODE**

C1. The initial comment in main.py contains my name as well as my student ID. Both of these items can also be found on the cover page of this document.

C2. Please see the attached python code for comments on the flow of the program.

**D. DATA STRUCTURE**

Outside of the standard Python library, the program utilizes two primary data structures: Hash Tables and Graphs. The HashTable is a self-adjusting data structure that will scale based on the number of inputs and provide a hash value to place data in the appropriate bucket. The hash table is used to store the package objects to provide a quick and efficient lookup method. The graph structure is used in the location management to track the edge weights, or distances between the nodes of graph that can be quickly referenced in the Nearest Neighbor algorithm.

**E. HASH TABLE**

Please review the HashTable.py file the hash table code and the packagemgmt.py file for the code relating to processing the package CSV data. The create\_package\_table function first creates package objects using the created Package class and then places the resulting objects into the hash table.

**F. LOOK-UP FUNCTION**

The HashTable.py file contains the look\_up function of the hash table class and returns the package object which has the attributes outlined in the project requirements.

**G. INTERFACE**

G1. Status check at 9:20 a.m.

**Text

Description automatically generatedText

Description automatically generated**

G2. Status check at 10:15 a.m.

**Text

Description automatically generatedText

Description automatically generated**

G3. Status check at 10:15 a.m.

**Text

Description automatically generatedText

Description automatically generated**

**H. SCREENSHOTS OF CODE EXECUTION**

The code execution outputs the total length of the three truck routes and the time the last truck returns to the hub. The user can select an option from the menu or select ‘3’ to exit. In the following screenshot, 3 was selected to show to code executing and completing successfully.

Text

Description automatically generated

**I. CORE ALGORITHM JUSTIFICATION**

I1. The nearest-neighbor algorithm was chosen for this project due to the simplicity of it’s implementation and it’s effectiveness for a complete graph.

I2. As shown in the screenshots above, the algorithm meets all of the requirements that were necessary for the project. It was able to achieve a total mileage of less than 140 and adjusts based on the provided input.

I3. There are additional algorithms that could have been chosen to complete this project such as Depth First Search or Dijkstra’s Shortest Path algorithms. The depth first search algorithm would have a more difficult implementation and a more complex runtime, but the resulting route would be more efficient than nearest-neighbor. Dijkstra’s algorithm can have varying degrees of difficulty depending on the chosen data structures to use. Depending on the chosen data structures the Dijkstra algorithm efficiency will be better than the nearest neighbor method and have a more optimal route.

**J. DIFFERENT APPROACH**

Possible improvements to this project would include implementing an automated approach to package sorting based on deadline and special requirements. In the current form of the code, the packages are loaded manually which is suitable for small quantities, but as this value grows it would become unmanageable and result in inefficiencies with route planning.

**K. DATA STRUCTURE VERIFICATION**

K1. The project required that a hash table data structure be implemented into the code without using Python’s built-in dictionary data structure. The HashTable class that was used in the project only contains the python list structure and implements buckets based on the hash value of a provided key. The hash table class takes in the size as a positional argument, with a default value of 40 if no size is provided. If the number of packages to be delivered changes, the hash table will scale accordingly and place packages into unique buckets if the keys are unique. The look\_up function finds the exact bucket for the provided key and returns none if there is no value. The look\_up function would not be altered by changes in the number of packages if the hash table is sized appropriately. The space complexity is O(N) for N packages. Changes in the number of trucks or cities would not affect the overall lookup time or space usage of the hash table.

K2. Other data structures that could be used in place of the Hash Table include Binary Search Trees, or a Hash Table with linear probing. Binary Search Trees would allow for an efficient means of storing package objects with a relatively low time-complexity for insertion and search of O(logN). The Hash Table with linear probing would help deal with any collisions for key values that are not unique but would increase the overall insertion and search time-complexity if collisions are present. For this project, the Hash Table that was used is the best choice to meet the given project requirements.

**L. SOURCES**

Tepe, C. (2020). *How To Dijsktra*. Western Governor's University. Retrieved January 18, 2023.

*Time - Time Access and conversions*. Python Documentation. Retrieved January 19, 2023, from https://docs.python.org/3/library/time.html

Vahid, F., & Lysecky, R. (2018). *zyBooks C950: Data Structures and Algorithms II*. Zyante Inc.