Jaxon Adams

ID 011441603

jada900@wgu.edu

C950 Data Structures & Algorithms II

PA Task 1: WGUPS Routing Program Planning

A - Algorithm Identification

I will be using the nearest neighbor algorithm as the core algorithm in my routing program. The nearest neighbor algorithm is an algorithm in which for each node left to visit, the nearest node is always the next to be visited. More information on my choice of using the nearest neighbor algorithm can be found in part C.

In this program, the nearest neighbor algorithm will be used to find an optimal route which each truck should take to maximize the efficiency of their deliveries. It will plot a route from one package’s address to another, always choosing the nearest address to the previous delivery address when choosing the next location.

Being a greedy algorithm, the nearest neighbor algorithm may not always select the most optimal path for each delivery truck; however, the paths chosen will be sufficient for the purposes of this project.

B - Data Structure Identification

I will be using a hash table to store packages with their related data. A hash table is a self-adjusting data structure which stores key-value pairs. More information on my choice of using a hash table can be found in part C.

All package-related data will be held in the hash table. Each key in the table will be a package ID, and each value will be a “Package” object with all package-related data, including fields such as delivery address, delivery deadline, package weight, and delivery status.

C1 - Algorithm Overview

The core algorithm will plot the route for a single truck. Its logic can be described in the following pseudocode:

function PlotDeliveryRoute(truck)

    Initialize an empty list "route"

    Initialize a set "unvisited\_addresses" with each package address on the truck

    current\_location = HUB

    while length(unvisited\_addresses) != 0

        next\_address = null

        min\_distance = infinity

        for each address in unvisited\_addresses:

            distance = distance between current\_location and address

            if distance < min\_distance

                min\_distance = distance

                next\_address = address

            end if

        end for

        append next\_address to route

        remove next\_address from unvisited\_addresses

        current\_location = next\_address

    end while

    append HUB to route

end function

This function will be called once for each delivery truck before the packages are delivered. Once a route has been plotted, the truck will deliver its packages by visiting each address in the “route” list.

C2 – Programming Environment

IDE: VSCode v1.89.1

Language: Python v3.10.11

OS: Windows 11

Hardware: LG Gram; 12th Gen Intel® Core™ i7-1260P 2.10GHz; 32.0 GB RAM

C3 – Space-Time Complexity Analysis

The core algorithm in my program (the PlotDeliveryRoute function) contains a nested for-each loop which operates on the list of undelivered packages. This means the space-time complexity of the routing algorithm can be expressed as O(n^2).

Once a route has been plotted for a truck, a truck will deliver its packages by visiting each delivery address in its “route” list one at a time. This will be a linear operation; or in other words, package delivery can be expressed as O(n).

Packages will be read from a file and initialized one at a time, making it O(n). Similarly, packages will be loaded onto their designated trucks in linear (O(n)) time. As there is a set number of trucks in this scenario, the trucks will be initialized in constant (O(1)) time.

All operations on the package hash table, including insertion and retrieval, are constant operations and have a time complexity of O(1). The hash table has a space complexity of O(n).

Overall, my program will have a space-time complexity of O(n^2).

C4 – Program Scalability

With a few adjustments, my solution would scale well with a growing number of packages. Currently I plan on loading trucks manually; this isn’t scalable or sustainable, but it suits the purposes of this project. If I needed the solution to properly handle an arbitrary number of packages, I would first add functionality for loading trucks automatically.

The other major components of my solution will scale well without further adjustment. A hash table is a very efficient data structure for fast item insertion and lookup, and the “nearest neighbor” algorithm will produce suitably optimal routes for any number of packages on a truck.

C5 – Efficiency and Maintainability

The code for this program will be reasonably efficient and maintainable as I’ll try my best to follow software development best practices. The software will make use of object-oriented design, with an emphasis on making the code readable and easy to update. Additionally, the use of heuristics and self-adjusting data structures throughout the code will boost performance and efficiency.

C6 – Self-Adjusting Data Structures

A hash table as a self-adjusting data structure has many strengths and weaknesses. A major strength of this data structure is its speed – all its operations have an average-case runtime of O(1). This makes it a very appealing choice for storing data that will need to be looked up or updated often. A hash table’s main weakness is the space it requires. Usually, a hash table is initialized with a large amount of unused space. While this can be inefficient, for this project the benefits of the hash table’s fast operations outweigh the need for efficient space usage.