C950 WGUPS Algorithm Overview

Dat Huynh

ID 010536518

WGU Email: dhuyn25@wgu.edu

9/15/2023

C950 Data Structures and Algorithms II

# Introduction

This project aims to implement a system for a parcel service to deliver packages based on a set of constraints effectively. A chosen algorithm for the system must adhere to these conditions while delivering timely packages. In addition, optimal data structures should be used to keep track of location, truck, and package information.

# A. Algorithm Identification

The chosen self-adjusting algorithm combines Dijkstra’s algorithm and a greedy approach. When visiting each node, the algorithm will find the shortest path from the current node to unvisited nodes, which may have intermediate unvisited nodes. By selecting the closest node, the algorithm can optimally generate the most efficient route based on given nodes, which change depending on the package address.

# B. Data Structure Identification

A hash table with a linked list chaining implementation is the chosen data structure to keep track of parcel service information. The hash table bucket is an array of linked lists that can handle collisions. In this case, we do not expect collisions if utilizing package ID as the key since each ID is assumed to be unique. Instead of storing data as primitive types in lists or tuples, keeping the data as generic as possible as a class object allows us to be more flexible with our attributes and the type of data we want to store. This flexibility allows any information from any component or service in the system to be kept if it has a "next" attribute to transverse the linked list.

# B1. Explanation of Data Structure

The data type stored will be a generic class object that requires a "next" attribute. This method allows flexibility in adding more information about a package, truck, or location as we are not locked to one specific defined object. A key is utilized to access the hash table bucket, and then the linked list can perform operations on the class object based on unique attributes required for the data implementation. Using these two data structures gives us more control and complexity over the data but can provide implementation opportunities like implementing a priority queue.

# C1. Algorithm’s Logic

Given a start node and a list of nodes to visit, Dijkstra’s algorithm will return a list of visited nodes with all the previous vertex and the shortest distance from the start node. Initially, each node in our nodes list should have their distance set infinitely from the start node. They should also not have any previous nodes yet. The shortest distance from the start node to itself is 0 with no previous nodes. We also created a list to keep track of the visited and unvisited nodes, and it should also be mentioned that the start node should be in the unvisited node. The algorithm will then visit every unvisited node. It will find the node with the smallest known distance from the start node, select it as the current node, and remove it from the unvisited node lists. The current node is marked as visited and then begins to examine the neighbors, which could be any possible node. The distance between the current node and the subsequent potential node to visit is calculated. We need to update it if it is shorter than the shortest distance between the current node. Update the shortest distance if we find a shorter distance to the selected node. Finally, the algorithm returns a list of visited nodes with all their previous nodes and the shortest distance updated. Transversing the chains of previous nodes from any node back to the start node will give you the shortest path.

Pseudocode:

Function Dijkstra

Set distance of start node to start node to 0

Set distance of other nodes from start to infinity

While there are unvisited nodes

Visit the unvisited node with the smallest known distance from start node

Examine unvisited neighbors of the current node

Calculate distance of each neighbor from the start node

If the calculated distance of the node is less than the known distance

Update the distances of the current node and its previous node

Add the current node to the list of visited nodes

Return visited nodes list

# C2. Development Environment

The Python application will be developed on a Windows 11 22H2 x64 bit, Intel I7 12700K machine running Visual Studio Code Version 1.82.0 and Python 3.11.4.

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

The worse case time-complexity of the entire program should be O(N^2)

# C4. Scalability and Adaptability

It can scale but can’t scale too well.

# C5. Software Efficiency and Maintainability

We are running polynomial time and also using OOP so it’s easy to maintain

# C6. Self-Adjusting Data Structures

While the hash table can handle collisions if you wanted more buckets you’d have to extend the size of the array and that would be O(N). If you implemented it without collions there’s a chance you put two of the same thing in twice and the latest one will override what’s residing in the array. But if you did implement it with collisons you can handle it, but the search would be O(N) because there could be N amount of stuff in that bucket.

# C7. Data Key

I just use the package ID because it’s easier to implement a hashing function

# D. Sources - Works Cited

Text goes here

An example:

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

Retrieved March 22, 2021, from <https://learn.zybooks.com/zybook/WGUC950AY20182019/>