C950: Tasks

A : ALGORITHM SELECTION

*Identify by name the self-adjusting algorithm used to create a program to deliver the packages and meet all requirements specified in the scenario.*

The program will use a two opt algorithm to calculate the best tour from a given list of addresses. It adjusts depending on the addresses input as well as their associated weights. It will also include a function that evaluates whether the tour meets all package deadlines.

B1: LOGIC COMMENTS

*The submission accurately explains the algorithm’s logic using pseudocode.*

ARGS: tour[0, 1, 2, 3, 4], distanceMatrix[5][5]

GET length of tour

GET distance of tour by using distanceMatrix

SET improved to True

WHILE tour is improved

FOR each number in tour (i)

FOR each number in tour +1 (j)

CREATE new tour by swapping numbers at i and j

e.g. (0,1,2,3,4)->(0,2,1,3,4)->(0,3,2,1,4)

GET distance of new tour

IF new distance is more than current distance THEN

UPDATE tour equal to new tour

UPDATE current distance equal to new distance

UPDATE improved to true

CONTINUE WHILE

ELSE continue

ENDWHILE

B2: DEVELOPMENT ENVIRONMENT

*The submission accurately describes the software and hardware used to create the Python application.*

Programming Language: Python 3.12.4

Text Editor: VS Code version 1.91.1

OS: Windows 10 Home, version 22H2

Hardware: Dell XPS 13, Processor: Intel i7-8550U, RAM: 8GB

B3: SPACE-TIME AND BIG-O

*The (code) submission accurately describes the space-time complexity for each major block of code and the entire program using Big-O notation.*

Additional details about these and other functions can be found commented within the code

|  |  |  |  |
| --- | --- | --- | --- |
| Function | File | Time Complexity | Space Complexity |
| updatePkgsStatus() | Main.py | O(m\*n) | O(m\*n) |
| getPackageDataList() | Package.py | O(n) | O(n) |
| organizePackages() | Packages.py | O(n^3) | O(n) |
| matrixAttributes() | DistanceMatrix.py | O(r\*(v+e)) | O(v^2) |
| bestTour() | DistanceMatrix.py | O(m\*n^3) | O(v+n) |
| populateTable() | HashTable.py | O(2n^2) | O(2n^2) |
| Insert(), resize() | HashTable.py | O(2n) | O(2n) |
| Update(), find() | HashTable.py | O(n) | O(n) |
| orderPackagesByRoute() | Truck.py | O(m\*n) | O(n) |
| Entire Program | ALL | O(m\*n^3) | O(2n^2) |

B4: ADAPTABILITY

*The submission application accurately explains the application’s capability to scale and adapt to an increasing number of packages.*

Self-adjusting data structures are exhibited in the HashTable and the DistanceMatrix. The HashTable is initialized with a size of 40, however once it reaches 39, it will double in size to ensure that there is more room. The Distance Matrix is also self-adjusting in that it will only create a matrix with weights and vertices which correlate to the distance and address for whatever list of packages that you put in. That could be the packages on each truck or all packages. The bestTour function incorporates a two opt algorithm that is also self-adjusting. It can take any list of edgeIndices (which correspond to the row/col of an address in the DistanceMatrix) and update the truck object associated with those packages as well as return a list of addresses that exhibit the best path.

B5: SOFTWARE EFFICIENCY AND MAINTAINABILITY

*The discussion addresses how the software is efficient and easy to maintain. By “efficiency,” they mean the Big-O time complexity of your entire program. Your program must run in polynomial time or better to be efficient.*

In terms of efficiency, the Big-O time complexity is relatively efficient. Using .pop() to minimize loops and including collision checking in Hash Table functions add a level of inefficiency that could be improved. The efficiency of the algorithm is also not optimized due to an added meetDeadline() constraint that changes the complexity from O(m\*n^2) to O(m\*n^3)

In terms of maintainability, the code is compartmentalized into classes to improve its organization. It includes comments on unusual aspects such as an index being (index+1)%n so that it is more easily understood.

B6: SELF-ADJUSTING DATA STRUCTURES

*The discussion addresses the strengths and weaknesses of the selfadjusting data structures (including the hash-table).*

**Strengths of HashTable**: automatically increases in size based on number of used buckets. It incorporates collision checking to improve functionality.

**Weaknesses of HashTable**: collision checking in the find(), update(), and insert() method causes the time Complexity to be increased by a factor of n. The bucket packages are tuples of values from package objects, it may be easier to reference the values if this was a package object

**Strengths of DistanceMatrix**: it excludes unnecessary distances and addresses to reduce errors in accessing values. It is self-adjusting

**Weaknesses of DistanceMatrix**: it is designed to have Euclidean distances, if distances were not Euclidean, adjustments would be required to add edges. Space Complexity is significantly larger than a dictionary or linked list of edges and distances

**Strengths of bestTour algorithm**: It adapts its result depending on the deadline of the packages as well as the distance between edges. It calculates whether the tour meets the required parameters rather than just 2 edges.

**Weaknesses of bestTour algorithm**: Time Complexity could be improved. Doesn’t scale well with large data inputs

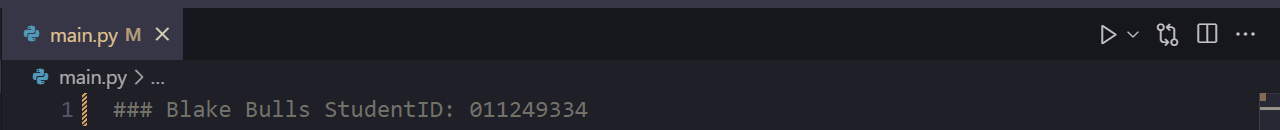
C: ORIGINAL CODE

*The code is original and runs without errors or warnings.*

All code referenced elsewhere is cited and adapted to fit my code. Code runs without errors or warnings: See H, which references file G1-3 and H Screenshots file in C950 DSA2 folder

C1: IDENTIFICATION INFORMATION

*The initial comment within a file named “main.py” includes your first name, last name, and student ID*



C2: PROCESS AND FLOW COMMENTS

*Include comments within the code adequately explaining the process and flow of the program.*

Extensive comments are included to elaborate on flow, process, as well as an additional details that may help with understanding

D: DATA STRUCTURE

*The submission identifies a self-adjusting data structure that can store the package information and perform well with Part A’s algorithm.*

The Distance Matrix is a self-adjusting data structure that stores the address and distances of packages used in the bestTour() algorithm. It also has a dictionary of deadlines that is mapped to address keys which allow the algorithm to check if the tour has met all deadlines for a package going to said address. The self-adjusting aspect of this data structure is due to its versatility. When given any list of packages, the data structure will adjust and only include the addresses and distances related to the packages in it. The Hash Table is updated based on the results of the bestTour() algorithm and the time input from the user. Using the update() function in Hash Table, It will adjust the packages in the Hash Table to exhibit the correct address and update the status to include ‘delivered’, ‘en route’, or ‘at the hub’ plus the time provided by the user.

D1: EXPLANATION OF DATA STRUCTURE

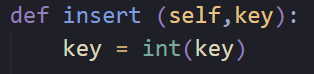
*The submission accurately explains how the data structure (hash-table) uses package IDs to store and retrieve package information.*

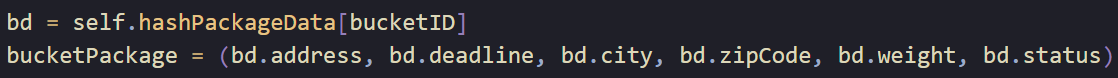
The Hash Table uses direct mapping (e.g. key – 1) to reference package data held within buckets. The buckets consist of a key, a tuple of relevant package data, and self.next=None(used for collision management). Upon initialization the Hash Table populates itself with packages using a for loop to iterate through a list of all packages and then using the insert() function to input the package objects into the Hash Table using the package ID. Within the insert() function a bucket is created using the package ID as the key, and package data is extracted from the package and stored within a tuple in the bucket. This bucket is then stored at index (key-1) in the Hash Table. The find() function is able to look up a bucket using the package ID because the bucket key is equal to the package ID. It simply returns the bucket that has a key that’s equal to the package ID and is located at index (key-1) in the Hash Table.

E: HASH TABLE

*The hash table has an insertion function that stores all of the given components (listed in Part D) using the package ID as the key.*

The Hash Table has an insert function, it stores all the given components into a tuple. The package ID is the key to this bucket.



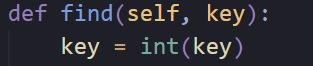


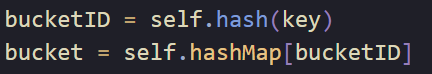


F: LOOK-UP FUNCTION

*The provided hash table should include a look-up function that can use a package's ID to retrieve all of the same package’s components from the hash table (listed in Part D).*

The find() function uses the package ID as the key to retrieve all of the contents of that package which were stored into a bucket with a key that is equal to the package ID located in the Hash Table at an index that is key-1 (found using hash() function).



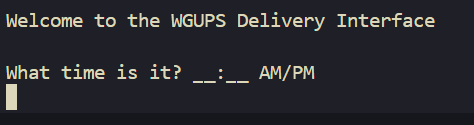


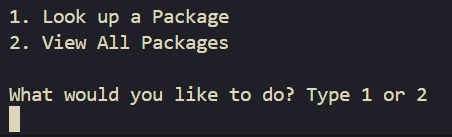


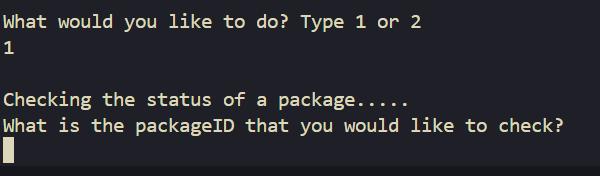
G: INTERFACE

*Provide an intuitive interface for the user to view the status and information (as listed in Part D) of any package at any time and the total mileage traveled by all trucks.*

Interface requests for the time from the user. And then asks what the user wants to do. If the user chooses Look up a Package, the interface will request a package ID. If the user chooses View All Packages, the interface will print all packages.







H: SCREENSHOTS OF CODE EXECUTION

*Provide a screenshot or screenshots showing the total delivery mileage and successful completion of the code free from runtime errors or warnings.*

See G1-3 and H Screenshots file in c950 DSA2 folder

I1: STRENGTHS OF THE CHOSEN ALGORITHM

*Identify at least two specific strengths of the algorithm from Part A relevant to finding a solution.*

**Strengths of bestTour algorithm**: It determines whether the entire tour meets the required parameters rather than just 2 edges. It adapts it’s result depending on the deadline of the packages as well as the distance between edges

I2: VERIFICATION OF ALGORITHM

*Verify that the algorithm used in the application meets all the requirements by:*

*● Provide the total combined miles traveled by all trucks. It must be less than 140.*

* milesDriven is stored in the truck object, it’s updated based on the distance traveled up to the time that is input by the user. This attribute is updated in the truck object based on the best distance from the bestTour algorithm.

*● State that all packages were delivered on time.*

* Within the bestTour algorithm, the tour will only be approved if it meets all deadline requirements associated with the address of each package in the tour. You can verify that each package was delivered on time by checking the deadline of the package and its status in the interface.

*● State that all packages were delivered according to their delivery specifications.*

* Packages are sorted onto trucks based on their delivery specifications in organizePackages(). This includes wrong address, delayed arrival, special notes, groups of packages, and deadline priorities. In main.py the parameters are verified and print statements are displayed according to whether these specifications are met.

*● Describe how all the above points are verifiable through the user interface.*

* totalMiles adds all the milesDriven variables stored in truck objects and prints the sum. The deadline of each package is displayed next to the status of the package so that the user can verify that it meets the deadline. A check is done in main.py to verify that packages meet special requirements and it prints a message in the interface if all requirements are met or not.

I3: OTHER POSSIBLE ALGORITHMS

*The submission identifies two algorithms different from the one provided in Part A that could meet the scenario’s requirements.*

Dijkstra’s algorithm or a genetic algorithm could have been used to meet the scenarios requirements

I3A: ALGORITHM DIFFERENCES

*The description includes attributes of each algorithm identified in Part I3 and how the identified attributes compare to the algorithm’s attributes from Part A.*

Dijkstra’s algorithm is used for finding the shortest path in weighted directed or undirected graph. It will provide the exact shortest path. Although both are deterministic, meaning they will continue searching as long as they find an improvement, two-opt differs in that it is a heuristic algorithm. Meaning, two-opt will find approximate solutions that aren’t guaranteed to be as precise as Dijkstra’s algorithm (Navone, 2020).

A genetic algorithm has a population of potential solution that incorporates operators to “evolve” the population over generations. These generations can be evaluated based on the quality of the solution provided. This is very different from two-opt, which simply checks the distance of two edges and keeps the shortest one. Genetic algorithms can also have lots of variety in their solutions, whereas two-opt iteratively makes small local changes that lack variety (MathWorks, 2019).

J: DIFFERENT APPROACH

*The description includes at least one aspect of the process that the candidate would do differently and includes how the candidate would modify the process.*

One aspect that I would do differently is the distance matrix design. As it is, it seems awkward in the way it chains together multiple functions. Perhaps a dictionary of addresses as keys and tuples of neighboring addresses and their distances would have been a more concise and less space consuming option. Another thing that could’ve have been better is better integration of the hash table with the algorithm.

K1: VERIFICATION OF DATA STRUCTURE

*Verify that the data structure used in the application meets all the requirements by:*

● Provide the total combined miles traveled by all trucks. It must be less than 140.

* miles driven is stored in the truck object, it’s updated based on the distance traveled up to the time that is input by the user. This attribute is updated in the truck object based on the best distance from the bestTour algorithm.

● State that all packages were delivered on time.

* Within the hash table, the buckets store the deadline and the status, Upon using the interface, the status is updated based on the time input using the update() function in the hash table. The deadline and status is accessed from the hash table using find() and visible in the interface to see if packages were delivered on time.

● State that all packages were delivered according to their delivery specifications.

* Packages are sorted onto trucks based on their delivery specifications in organizePackages(). Updates are also made in the hash table based on the time input to correct addresses or mark delayed packages as arrived. In main.py the parameters are verified and print statements are displayed according to whether these specifications are met.

● State that an “efficient” hash-table (Part D) with a look-up function (Part F) is present.

* The hash table is efficient. It has a find() function that allows the user to access package data by inputting a package ID.

● State that the “reporting” (package statuses and information) can be verified through the user interface and that all information is accurate (Part G).

* The user interface contains information for all packages on each truck and allows the user to very whether deadlines are met or whether package specifications are met.

K1A: EFFICIENCY

*The discussion accurately explains how adding packages directly affects the time needed to complete the look-up function.*

Adding packages would not have any effect on looking up a package. The hash table incorporates direct mapping which utilizes package IDs to easily find packages that have been input.

K1B: OVERHEAD

*The discussion accurately explains how adding packages directly affects the data structure space usage.*

Adding packages would increase the space usage of the hash table by 1 for each package inserted. The space usage of the hash table will double upon reaching the capacity – 1.

K1C: IMPLICATIONS

*The discussion accurately explains adding trucks or cities would affect lookup time and space usage.*

Adding trucks and cities should not effect the hash table space usage directly. The implication is that the packaging company will be larger and thus have more packages delivered to other cities, which would of course call for an increase in the size of the hash table. This would not really affect the time complexity of the look-up function as it will still be bounded by the total number of packages. The space usage will similarly be affected as it is also bounded by the total number of packages.

K2: OTHER DATA STRUCTURES

*The submission identifies two data structures other than the one used in Part D to meet the requirements in the scenario.*

A trie or a binary search tree are 2 possible data structures that could be used.

K2A: DATA STRUCTURES DIFFERENCES

*The submission accurately describes attributes of both data structures identified in part K2 and compares these to Part D’s data structure attributes.*

A binary search tree guarantees that all operations will work in O(log n), whereas with hashing, some operations can be O(n^2). Binary search trees require less memory and computing (Advantages of BST over Hash Table, 2015).

A trie, like a hash table, prioritizes quick retrieval of data. This seems to be a likely choice for this project due to the frequency of looking up different packages for their status. Searching for the existence of a particular string like “ wrong address” or “truck 2” would have a much faster time complexity than in the current hash table implementation (Hash Table vs Trie, 2022).

L: SOURCES

*The submission includes in-text citations for properly quoted sources, paraphrased, or summarized and a reference list that accurately identifies the author, date, title, and source location as available.*

*2-opt algorithm to solve the Travelling Salesman Problem in Python*. (n.d.). Stack Overflow. <https://stackoverflow.com/questions/53275314/2-opt-algorithm-to-solve-the-travelling-salesman-problem-in-python>

*Advantages of BST over Hash Table*. (2015, April 1). GeeksforGeeks. https://www.geeksforgeeks.org/advantages-of-bst-over-hash-table/

*Create an incremental ID in a Python class with subclasses each maintaining their own ID series*. (n.d.). Stack Overflow. Retrieved July 22, 2024, from <https://stackoverflow.com/questions/71520394/create-an-incremental-id-in-a-python-class-with-subclasses-each-maintaining-thei>

*Hash Table vs Trie*. (2022, December 15). GeeksforGeeks. https://www.geeksforgeeks.org/hash-table-vs-trie/

‌

MathWorks. (2019). *What Is the Genetic Algorithm?* Mathworks.com. https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html

‌ Navone, E. C. (2020, September 28). *Dijkstra’s Shortest Path Algorithm - A Detailed and Visual Introduction*. FreeCodeCamp.org. <https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/>

‌ Oggi AI - Artificial Intelligence Today. (2016, April 16). *Python: 2 Ways to Represent GRAPHS*. YouTube. <https://www.youtube.com/watch?v=HDUzBEG1GlA&t=486s>

‌ PageKey. (2017, November 26). *Hash Tables in Python*. YouTube. https://www.youtube.com/watch?v=zHi5v78W1f0&t=628s

‌