CS 2336 – PROJECT 4 – Amazon Prime Delivery Tracker (part 3)

Pseudocode Due: before 11/12 at 11:59 PM

Final Submission Due: before 12/4 at 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- The pseudocode will be submitted in eLearning as a Word or PDF document and is not accepted late.
- All project source code will be submitted in Zybooks.
    - Projects submitted after the due date are subject to the late penalties described in the syllabus.
- **Type your name and netID in the comments at the top of all files submitted. (-5 points)**

Objectives:

- Create and utilize a graph
- Complete a large scale project

Problem: Good news! Your project manager at Amazon is thrilled with your progress on the delivery tracker system. The project manager has asked you to expand on the tracker once more to verify the routes of the drivers. As routes are always changing, the delivery tracker application needs to verify that a given route exists in the network of locations. You have been asked to add this functionality into the system so that invalid routes can be identified, and the total mileage of valid routes can be calculated.

Pseudocode Details:

- Main.java
    - Detail the step-by-step logic of the main function
    - List other functions you plan to create
        - Determine the parameters
        - Determine the return type
        - Detail the step-by-step logic that the function will perform

Zybooks Information:

- You will have multiple source files
    - `Main.java`
    - `Graph.java`
- There is no core implementation
    - It is on you to manage your time wisely
- Final submission has unlimited submissions
- White space will not be checked

Classes:

- Graph class – `Graph.java`

    - Attributes
        - Size
        - Edges implementation
        - Vertices implementation
        - You may choose how to implement both of these
    - Methods
        - Size accessor
        - Insert edge
        - Insert vertex
        - Delete edge
        - Delete vertex
- **EXTRA CREDIT: Hashtable class (+10 points if implemented)**
    - Implement your own hash map to hold the vertices
        - This will speed up the process of finding a vertex index
    - The hashmap class will be a private class in Graph
    - The hashmap will use simple chaining for collision handling
    - The class must have a rehashing function
        - The load factor for the hashmap will be 4
            - If the average length of chains is over 4, rehash
        - The size of the new hashmap will be double the size of the old hashmap
    - You may use the built-in Java Linked List or Arraylists
    - The default size of the hashmap will be 5

User Interface:

- Prompt the user for the file containing the graph data
- Prompt the user for the file that contains all the routes
- Display all output to the console

Details:

- Determine if a given delivery route is valid based on the graph structure
    - Does the given path exist in the graph?
- The number of drivers in each file is unknown
- The number of destinations in the delivery route for each driver is unknown
- Driver names and location names will be single words
- All input will be valid.

Graph File:

- **Read this file first**
- Data from this file will be used to create the graph
- Each line will contain a vertex and a list of edges with weights
- The first location name of the line is the vertex
- Each edge connected to that vertex will be represented as `<adjacent vertex><comma><weight>`
    - Line example: `Hoth Endor,9 Coruscant,3 Tatooine,1 Dagobah,5`

- All vertices will have at least 1 edge

Routes File:

- **Read this file second**
- Each line in the file will contain the driver’s name followed by a list of locations
- The first string on the line is the driver name
- All subsequent strings are locations
- Each location will be separated by a space
  - Example: `Sully_Olvar Tatooine Dagobah Naboo Yavin Alderaan`
- The first vertex listed is the starting vertex of the delivery
- There will be a new line at the end of each line except for the last line
  - The last line may or may not have a new line character
- Each line in the file will represent a different driver.

Output:

- For each driver, display the following separated by tabs
  - Driver Name
  - Path weight
  - `valid` or `invalid`
    - whether or not the path is valid
- Drivers will be sorted from least to greatest path weight
  - In the event of a tie, order the driver names alphabetically
- If the path is invalid, the path weight will be zero
- Each driver’s data will be written on a separate line.