Project 3
Subway Routing

Due May 10th

Summary

Create a network of subway stations, then an application to generate a route between any two stations.

Input

Resource

A detailed list of stations and lines of the MTA subway system can be found here

URL Reader

Java has a built-in URL connection class (import java.net.* required). The URL object need to first initialize by calling its .openConnection() function, then it can return an inputstream by calling its .getInputStream() function. The stream will return the contents in the URL. In the case the URL is a web page, the input stream will return the HTML code for the web page.

HTML

HTML is a markup language that uses so-called "HTML tags" to format text. The basic idea of HTML tags are surrounding certain text with matching tags so that the text inside would be applied a certain property. For example

<h1> sample text </h1>

would apply the property "h1" to the sample text.

There are many different tags and the tags can be nested with each other.

jsoup

jsoup is a free Java Library that can be downloaded and imported to your Java project to read HTML files much easier.

You can download the jsoup library here, and find a tutorial with some examples here.

Unfortunately, repl.it does not support importing .jar libraries into the project, so this must be done on your own IDE. See special submission guidelines below for more info.

Input file

By utilizing the URL and jsoup functions, you should be able to read some tokens and save them into a / a few file(s). Due to the fact that the MTA's code itself is messy and inconsistent, you are expected to run into some exceptions. You are not expected to handle all the exceptions and write a perfect reader, in other words, feel free to manually fix the files that you've generated. You should balance the time between trying to improve the automated reading function vs manually fixing the exceptions.

Other problems (optional)

Many stops have multiple different names on different lines (e.g. Jackson Height - 74th Street (7) / Roosevelt Avenue (EFMR)), and there are transfer stops that are really far away (e.g. Lexigton Av/ 63rd St (FQ) to Lexington Av/59th St (NRW)). Due to these problems, your routing program could provide unrealistic results. You can remedy problems like these by manually adding a "transfer" line that links

stations like this together (with 0 weight for all its edges), or adding alternative names to the stations.

Building the network

Data Structure

We learned two types of data structure in class to represent graphs: edge-list and adjacency matrix. You can choose whichever that seems appropriate to you or use a customized data structure to represent the network.

Additionally, you might be required to create additional classes to better structure your project.

Creating objects

Similar to Project 1 and Project 2, you are expected to create objects based on the input file.

Eventually, you should have an object that resembles the subway network.

Scalability

In terms of scalability concerns for better code writing practices, you can expect more train lines and stops, but you don’t need to worry about multiple train networks.

Execution and output

Main Routine

This time, your program will be taking in inputs from the console: It will ask the user to input an origin station and a destination station, then find the shortest path between the two stations, and report it back to the user.

At minimum, the report should include the list of stops in the shortest path. Alternatively, if you are feeling fancy, you can create a more readable output

that resembles modern navigation systems - by only highlighting the stations required to make a transfer.

Shortest Path

Practically, the shortest path between two subway stations should be defined by time. Since we don't have that data (unless we implement some type of data query from the Google Map API), we will assume the "time" between each station is the same unit time. Therefore, essentially, the path with minimum number of stations/stops will be the shortest path. Notice that express trains will bypass local stops, and therefore have less stops and more likely to be a part of the solution.

Rubrics/Grading Guide

This project (as well as all the other projects) will be graded "max = 10 point", then scaled by multiplying 1.5 or 2.0 depending on your performance on the other projects.

(2 points) Creating the stations list

Using the jsoup library and/or URL class to generate file(s) that contains all the station names for all the lines

(3 points) Network related objects

Station/Line object(s)

Helper function(s)

Shortest path algorithm

(2 points) Main routine and console IO

Console menu and IO

(1 point) General code structure / readability

(1 point) Submission format

(1 point) Documentation

(? points) Extra credits possible

More realistic results
Fancy “GPS style” output
Fuzzy search: station name autofix recommendation

Special Submission guidelines

Due to the fact that repl.it does not support .jar library imports, the station list generation code won’t be compiled on repl.it. To remedy this:

On repl.it

Include the station generation code class but remove the .java extension (otherwise the code could affect compilation)

Include all the station name file(s) AFTER your manual adjustments

Include all other files as usual

In the attachment zip

Include all java files as usual, you can assume that I have the library installed on my machine

Include all the station name file(s) BEFORE your manual adjustments