Welcome to Pie in the Sky Airlines technology division!

OVERVIEW:

We are a fast growing new airline and are looking to enhance our scheduling software. We would like you to implement a new route finder that will serve as the cannonical source of all routes (where we fly) in our system. This software has two operations: to store and update information about the cities we service and to query its data store for pathing information to determine routes between those cities.

The software should be able to store are retrieve information about the distance and flight time between cities. For example, a flight from St. Louis to Kansas City is 250 miles and it takes 1.3 hours to complete the flight. Think of this as creating a database of EDGE COMMANDs (see below).

We should also be able to query the system (EDGE database) for available routes and the cost of that trip. For example, if I have flights (EDGEs) from:

1) St. Louis to Kansas City
2) Kansas City to Denver
3) St. Louis to Denver

and I query the system for flights between St. Louis to Denver I should retrieve two routes

1) St. Louis to Kansas City to Denver
2) St. Louis to Denver

See the INPUT / OUTPUT section for more information on the specifics of how to accept input and format output.

DESIGN:

The goal of this design is to be modular, easy to integrate with other systems, and language agnostic. To facilitate these goals this software is designed to follow the linux philosophy and will use pipes for inter-process communication. It will use the standard streams for all communication.

NOTE for more information see the wikipedia article on Standard Streams or look up how to read / write to stdin, stdout, stderr in your language of choice.

OPERATING SYSTEM:

You can assume that the operating system will be compatible with a brand new Ubuntu 20.04 installation. Your program can be written in any language, however you will need to provide detailed instructions for the installation of any dependencies. If your language of choice is compiled please provide the source as well as a compiled executable. If you have any challenges using a linux based operating system please contact neal@pricepointmoves.com to discuss other options.

INPUT / OUTPUT:

All input and output (refered to as a COMMAND from this point forward) will have a similar format. The name of the COMMAND followed by a single whitespace, then a comma separated list of parameters, then a new line. All COMMAND names will contain only uppercase characters, the list of parameters may or may not contain spaces.

INPUT:

There are two input COMMANDs, ADD and QUERY.

ADD:

An ADD COMMAND has exactly 4 parameters:
1) the name of originating city (string)
2) the name of the destination city (string)
3) the mileage for the leg (float)
4) the duration of the flight (float)

An ADD COMMAND will add or update the records stored by the system and will output an EDGE COMMAND (see the output section for deatils) to stdout to report a successful update.

QUERY:

A QUERY COMMAND has exactly 2 parameters:
1) the name of originating city (string)
2) the name of the destination city (string)

A QUERY COMMAND will seach all of the stored records to determine all paths from the originating city to the destination city. If at least one path from the originating city to the destination city exists a RESULT COMMAND should be written to stdout followed by PATH COMMANDS, in ascending order of cost, for each possible route.

If at least one path from the originating city to the destination city does not exists the QUERY should be considered malformed (see the ERROR section).

OUTPUT:

EDGE:

An EDGE COMMAND has exactly 4 parameters:
1) the name of originating city (string)
2) the name of the destination city (string)
3) the mileage for the leg (float)
4) the duration of the flight (float)
For each successful ADD COMMAND and corresponding EDGE COMMAND should be written to stdout.

RESULT:

A RESULT COMMAND has exactly 2 parameters:
1) the name of originating city (string)
2) the name of the destination city (string)
For each successful QUERY COMMAND and corresponding RESULT COMMAND should be written to stdout.

PATH:

A PATH COMMAND has at least 3 parameters:
1) the cost of the path (float) (see NOTE below)
2) the originating city (string)
3, 4, ... n-1) the name of the next city in the path (string)
n) the destination city (string)
For each successful QUERY COMMAND a corresponding PATH COMMAND for all available paths should be written to stdout. These paths should be written in ascending order by the cost of the path.

NOTE: A path is made up of "legs" from an originating city to a destination city. The cost of an individual leg is defined as the mileage of the leg multiplied by 15 added to the flight time multiplied by 30 rounded to the

nearest hundredth. The cost of a path is the sum of the cost of each "leg"
in the path.

ERROR:

An error should be generated for all malformed input COMMANDs. A command is considered malformed if the name of the command is not listed in the INPUT section, the COMMAND's parameters do not match the expected parameters, or the COMMAND is otherwise considered malfored (see QUERY). All errors should write a MALFORMED command to stderr.

MALFORMED:

A MALFORMED COMMAND has at least two parameters:

1) the name of the COMMAND that caused the error
2, 3, ... n) the parameters of the command that caused the error

TESTING:

To test your solution you have been provided three files: input.txt, output.txt, and error.txt. The output.txt and error.txt files are the expected output to stdout and stderr respectively if the program is given input.txt on stdin. For example, if you have a python solution solution.py in the same directory as these three files the following terminal command should have no output.

NOTE: the quoted portion is one line, the line break is for readibility

```
"python solution.py < input.txt > actualOutput.txt 2> actualError.txt && diff
 -q output.txt actualOutput.txt && diff -q error.txt actualError.txt"
```

To test a solution in another language replace only the python specific potion of the command.

```
"{invoke language specific command here} < input.txt > actualOutput.txt 2> actualError.txt &&
diff
 -q output.txt actualOutput.txt && diff -q error.txt actualError.txt"
```

This command redirects stdin to the file input.txt, stdout to the file actualOutput.txt, and stderr to the file actualError.txt. It then compares the results with the expected output/errors using diff. If you wish to see more detailed output on the differences remove the -q flag from the diff commands.