CSE 143: Computer Programming II — Spring 2015

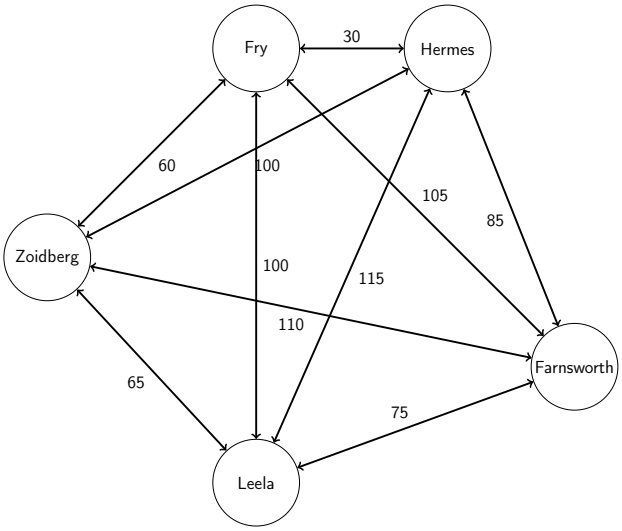
Extra Credit: Travelling Salesperson Problem (due Friday, June 5, 2014 11:30PM)

This extra credit assignment is far harder than the *two* points it is worth. Do it for fun, not for the points. BECAUSE THIS IS EXTRA CREDIT, YOU MAY NOT ASK FOR HELP IN THE IPL ON THIS ASSIGNMENT. It focuses on more recursive backtracking. Turn in the following files using the link on the course website:

- `TSP.java` – A class that uses recursive backtracking to find all "tours" of locations for an input file.

Travelling Salesperson Problem

The *travelling salesperson problem* asks the question "what is the shortest route I can take that starts and ends in the same place and visits every location on my list exactly once?" For example, suppose I am Fry and I'd like to visit all my friends using the following routes:



One route Fry could take is:

Fry → Farnsworth → Leela → Hermes → Zoidberg → Fry

Note that this route visits each other person *exactly once* and starts and ends in the same place. Furthermore, the total distance of this route is $105 + 75 + 115 + 100 + 60 = 455$mi.

Another route Fry could take is:

Fry → Leela → Zoidberg → Farnsworth → Hermes → Fry

This route also visits each person exactly once and starts and ends at Fry. However, the total distance of this route is $100 + 65 + 110 + 85 + 30 = 390$mi, which is significantly better than the first route.

The goal of this problem is to identify all routes, along with how much distance they take. This allows us to select the best possible route (e.g., the one with the shortest distance).

In this assignment, you will create a class called `TSP` that reads in an input file and prints all possible "tours" of all places/people in that file (along with their distances) when given a starting point. You will use *recursive backtracking* to implement your algorithm.

Since this is extra credit, you are *not* provided with a client program. We recommend you write one called `TSPMain` that prompts the user for a starting point and input file and passes them to your `TSP` object. It asks your object to print all tours given those parameters.

`TSP` should have the following constructor:

`public` **`TSP`**`(String filename)`

This constructor should initialize a new `TSP` object that will use the given input file as its data. The input file is guaranteed to have lines of the form:

`<distance in miles>,<location1>,<location2>`

where `<distance in miles>` is a placeholder for a `double`, and `<location1>` and `<location2>` are placeholders for names of locations. You are guaranteed that each location will have a distinct name, but many lines will reference the same location.

You should read the file in, line by line, and store the file in a `Map`. Note that your program will need to find the distance between pairs of locations *very frequently*. Also, any pair in the file should be considered `symmetric`. In other words, if there is a line:

`100.0,Adam,Riley`

Then, your `Map` should show that Adam is 100mi from Riley AND that Riley is 100mi from Adam.

Note that this constructor will have to have a `throws FileNotFoundException` clause.

`TSP` should also implement the following method:

`public void` **`printTours`**`(String start)`

This method should use recursive backtracking to print all possible tours that visit each location in the input file exactly once, except `start` which it should visit at the very beginning and the very end.

You will find it *necessary* to create a `private` helper method that has *two* accumulators: the *distance* you are building up and the *tour* itself.

For the example tours mentioned above, this method would print the following two lines (among others...):

```
>> [Fry, Farnsworth, Leela, Hermes, Zoidberg, Fry] (455.0 mi)
>> [Fry, Leela, Zoidberg, Farnsworth, Hermes, Fry] (390.0 mi)
```

You may output the lines in any order, as long as they are all there. Do not round the distances.

Hints for Your `Map`:

Your map should be able to answer the following question:

How far is A from every other person/location?

Consider a similar situation in which people are numbers, instead of `Strings`:

If we had the following input:

```
10.2,0,1
50.55,0,2
30.5,1,2
```

We could use a *two-dimensional array* to store this:

```
1 double[][] friends = new double[2][2];
2 friends[0][1] = 10.2;
3 friends[1][0] = 10.2;
4 friends[0][2] = 50.55;
5 friends[2][0] = 50.55;
6 friends[1][2] = 30.5;
7 friends[2][1] = 30.5;
```

Then, the idea is to come up with a similar, nested `Map` structure to support the same idea for `Strings`.