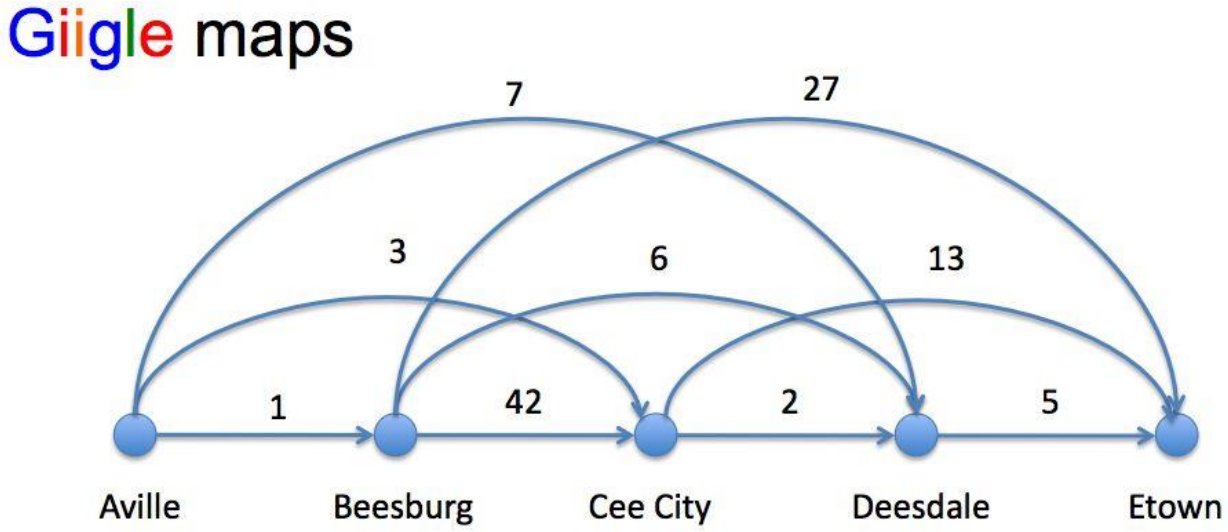
Problem 4: Giigle Maps

Copied from:
https://www.cs.hmc.edu/twiki/bin/view/CS5/GiigleMaps
on 3/15/2017

Upto +10 pts. optional bonus problem (individual or pair)

In class we discussed the problem of finding the shortest path from a start vertex to a destination vertex in a map in which all roads "point east". (The more general problem in which one can go both east and west is a bit more complicated.)
Here is an example of an "east-to-west" map.



We can represent a map using a Python "dictionary", as described in class. For example, the map above would be represented by the dictionary below (which we've called `FiveDists`), where the city names have been changed to be `"A"`, `"B"`, `"C"`, `"D"`, and `"E"` rather than `"Aville"`, `"Beesburg"`, etc.

```
Inf = float("inf")
```

```
fiveDists ={("A","A"): 0, ("A","B"): 1, ("A","C"): 3, ("A","D"): 7,
("A","E"): Inf,
           ("B","A"): Inf, ("B","B"): 0, ("B","C"): 42, ("B","D"): 6,
("B","E"): 27,
           ("C","A"): Inf, ("C","B"): Inf, ("C","C"): 0, ("C","D"): 2,
("C","E"): 13,
           ("D","A"): Inf, ("D","B"): Inf, ("D","C"): Inf, ("D","D"): 0,
("D","E"): 5,
           ("E","A"): Inf, ("E","B"): Inf, ("E","C"): Inf, ("E","D"): Inf,
("E","E"): 0
          }
```

Notice that `Inf` is a variable that we've defined to be `float("inf")` so that it behaves like infinity (in particular, it is larger than any other number, so that taking the `min` of `Inf` and any other number will always return the other number!). When a road between two cities is missing, we represent that by a road of length `Inf`. In addition, the distance from a city to itself is necessarily 0.

Recall that in a dictionary like `fiveDists`, we can get the value associated with a "key" like `("A", "B")` by asking for `fiveDists[("A", "B")]`.

In addition, we would package list of cities in a Python list like this:

```
fiveCities =["A", "B", "C", "D", "E"]
```

Part 1 (up to +5 extra-credit points)

`shortestPath`

The first part of this bonus problem is to write the a function called `shortestPath(cities, distances)` that takes a list of cities (e.g., `fiveCities`) and a dictionary of distances between these cities with all roads pointing east, and

returns the length of the shortest path from the first city in the list to the last city in the list.
For example:

```
In [1]: shortestPath(fiveCities, fiveDists)
Out[1]: 10
```

Of course, your function should work for any list of cities and any distance dictionary (as long as the only finite distances are the ones on roads that point east).
Your function should be only four lines of Python code long (not including the `def shortestPath` line, of course). In particular, it should be of the following form:

```
def shortestPath(cities, distances):
    if BLAH:
        return BLAH BLAH
    else:
        return BLAH BLAH BLAH
```

In addition to recursion, you may wish to use the built-in `min` function, which can accept an *entire list* and returns the minimum element in that list. You may also wish to use `map` and `len`. If so, you will need to use an anonymous function (a `lambda` expression) within `map`.

Part 2 (up to +5 extra-credit points)

findShortestPath

If you've got the time and inclination, now write a function called `findShortestPath(cities, distances)`, which returns a list that contains two items: the shortest path from the first to the last city in `cities`, **and** a list of the cities on this path. This function can be a bit longer than your `shortestPath` function, but it won't be very long!

```
In [1]: findShortestPath(fiveCities, fiveDists)
```

```
Out[1]: [10, ["A", "C", "D", "E"]]
```

Submit

Submit your function(s) in a file called `hw2pr4.py`.