22C:16 (CS:1210) Homework 5

Due via ICON on Friday, April 19th, 4:59 pm

What to submit: Your submission for this homework will consist of just one file called `hw5Solution.py`. This file should contain the definitions of two functions `createDistanceMatrix` (for Problem 1) and `createCityNetwork` (for Problem 2). As always your file should start with a header consisting of your name and section. Also, make sure that your code is well-documented (i.e., has plenty of helpful comments), uses appropriate variable-names, is not unnecessarily complicated or inefficient.

Introduction

For this homework, we have posted a file called `miles.dat` that contains geographic data on 128 cities in the U.S. and Canada. The data is very old — from a 1949 Rand McNally mileage guide, so the population numbers are definitely out of date. The highway mileage may also be out of date, but the data serves our purposes quite nicely. For each of the 128 cities, the data file contains (i) the name and state of the city, (ii) the latitude and longitude of the city, (iii) the population of the city, and (iv) the distance from that city to each of the cities listed before it in the data files. For example, Wilmington, Delaware is the tenth city in the data file and here is how information about Wilmington, DE appears in the data file.

```
Wilmington, DE[3975,7555]70195
466 168 1618 430 934 299 2749 1305 345
```

The two numbers in square brackets after "DE" are the latitude and longitude of the city. Following this, we see the population of the city, which was 70,195 in 1949. In the next line are the distances between Wilmington, DE and the 9 cities that appear before it in the file. For example, 466 is the number of miles between Wilmington, DE and the city Wilmington, NC that appears just before Wilmington, DE in the data file. Similarly, 168 is the distance between Wilmington, DE and Winchester, VA which appears just before Wilmington, NC in the data file.

Problems

1. Write a function called `createDistanceMatrix` that reads the geographic data given in `miles.dat` and returns two data structures: (i) a list of city names and (ii) a 128×128 matrix that stores distances between pairs of cities.

   Here are some notes that should help clarify, what you are being asked to do.

   - The list of city names should be a list of 128 strings. Since the names of cities are not all distinct (e.g., there are two Wilmington's) you should store the name of the city along with the name of the state that the city belongs to. One way to do this would be to represent each city by a string "cityName stateName". For example, the first city in the data set is Youngstown, which is a city in Ohio. You would represent this city by the string "Youngstown OH" with the city name and the two-letter state code being separated by a singe blank. The city names should by stored in the list in the order in which they appear in the data file. For example, the element in position 0 should be "Youngstown OH", the element in position 1 should be "Yankton SD" etc.
   - The matrix of distances (let us call this `distances`) should be a list of lists such that `distances[i][j]` should be the distance between the city that is position i and the city that is in position j in the list of city names. In particular, `distances[0][0]` should be the distance between Youngstown, OH and itself, which is 0; `distances[0][1]`

should be the distance between Youngstown, OH and Yankton, SD, which is 966; `distances[0][2]` should be the distance between Youngstown, OH and Yakima, WA, which is 2410, etc.

- Once you have correctly defined the function `createDistanceMatrix`, we should be able to call it as

```
        [cityList, distances] = createDistanceMatrix()
```

  and work with the data structures `cityList` and `distances`, as we please.

Here are some hints with regards to some of the issues you will run into while working on the problem.

- First of all, notice that the distances between a city and all previous cities appear over several lines. For example, consider Uniontown PA which has 37 cities before it in the file. So we should expect to read 37 distances after the line containing Uniontown PA and indeed there are 37 numbers appearing over three lines. One easy way to read these distances might be to have a variable called `expectedDistances` that tracks the number of distances you still expect to read before reading the line containing the next city. So for Uniontown PA, `expectedDistances` will initially have value 37. Now let us say you have read the first line of distances (there are 16 distances in this line), split this line, and stored the distances. The variable `expectedDistances` should now have value 11. Since this variable has not reached 0 yet, it means that there are more distances to read. Once this variable reaches value 0, you know that the next line (if it exists) will contain a new city.
- You will need to think a little bit about which "slot" in the distance matrix each distance you read from the file goes into. Again, consider the example of Uniontown PA. Uniontown PA appears in position 37 in the list of city names. Therefore, the first distance after the line containing Uniontown PA (which is 417) should go into slots [37][36] and into slot [36][37] in the distance matrix.

2. Write a function `createCityNetwork` that takes three parameters: (i) a list of city names, (ii) a distance matrix containing pairwise distances between cities, and (ii) a positive real number r. The function should create a *dictionary-based* data structure of a network of cities in which pairs of cities that are at distance at most r are considered to be neighbors. For example, we should be able to call this function as

```
        cityNetwork = createCityNetwork(cityList, distances, 400)
```

   and then expect `cityNetwork["Uniontown PA"]` to be the list of all cities (in `cityList`) that are atmost 400 miles from Uniontown PA. By "dictionary-based" data-structure, we simply mean the type of data-structure that we used to represent the word network in the word ladders game example, discussed in the lectures.

   Note that the typical usage for `createCityNetwork` would be after we have called the function `createDistanceMatrix` to construct the list of cities and the distance matrix extracted from the text file `miles.dat`. However, `createCityNetwork` should work fine even with arguments of appropriate size, that have nothing to do with `miles.dat`. For example, the call

```
  cityNetwork = createCityNetwork(["A", "B"], [[0, 10], [10, 0]], 15)
```

   should be able to correctly construct a city network with two cities.