Assignment 9 New York City 2015 Street Tree Data

1 Overview

In 2016, New York City made public the results of *TreesCount!*, the *2015 Street Tree Census*, conducted by volunteers and staff organized by the NYC Department of Parks & Recreation as well as partner organizations. The data includes information about more than 680,000 trees on the streets of New York City. This is a large dataset, with over 683,000 lines of text totaling more than 193 MB of data. Although it will fit into your computer's memory, it will make many computers behave sluggishly, depending on what you do with the data. The original data set has 41 columns; I have edited the data set so that it has only nine columns. I also removed entries from the data set that corresponded to dead trees and stumps in order to simplify the tasks that you will need to do for this project. Even so, the data set is still about 55 MB in size. The data set is stored in a comma-separated-values file, i.e., a *csv* file. This means that each field in each row is separated from the adjacent fields by a single comma, and that, at least in this file, no field contains any commas. *(The version of the data set that I provide on the server has its first row, containing column headers, deleted.)*

This programming project is designed to give you practice in processing datasets and in using complex data structures. In particular, your program will store tree records as Perl hashes and store these hashes in an array. It will then process the stored data to answer specific questions about the data set.

The details of the assignment are specified in the *Detailed Requirements* section below.

2 More About the Original Data Set

The data set is part of the NYC OpenData website and can be found here:

`https://data.cityofnewyork.us/Environment/2015-Street-Tree-Census-Tree-Data/uvpi-gqnh`

You may find it interesting to take a look at an online visualization project based on an older New York City tree census data set at `http://www.cloudred.com/labprojects/nyctrees/`.

The *NYC OpenData* website for this tree census data gives you the means to download the data in various formats. A file in *csv* format, in case you are not familiar with it, is a ***comma-separated-values*** file. A comma-separated-values file is a plain text file in which each line represents a single record, and within the line, commas separate the individual fields of the record. (Fields can also contain commas if they are within quoted strings, e.g., "Brooklyn, New York" is a single field.) Spreadsheet applications let you import *csv* files to view their contents by rows and columns. The tree data file that can be downloaded contains records for over 680,000 trees. Each row represents a single tree (or tree stump) and has 41 columns, which means that there are 41 different pieces of information for each tree. *The data set that is downloaded will have as its first row, the labels of its columns.*

A detailed description of the meaning and form of every column[1] in that dataset can be found in the ***data dictionary*** described here:

`https://data.cityofnewyork.us/api/views/uvpi-gqnh/files/8705bfd6-993c-40c5-8620-0c81191c7e25?download=true&filename=StreetTreeCensus2015TreesDataDictionary20161102.pdf`

This data dictionary is also available on our server in the `data` subdirectory of the `cs132` directory. Each valid line in the dataset contains 41 columns. Some of these columns may be empty. An empty column is

[1] This description is missing the description of the column with index 14 that appears between the *user_type* and *root_stone* columns in the dataset.

represented by two commas with no intervening characters. The columns are determined by the commas separating each entry. This means that a valid line has to contain at least 40 commas separating the entries (even if the entries are empty), and maybe more, if the fields contain embedded commas. While there should not be invalid lines in the file, if any are found, the program should handle them by ignoring them.

3 Detailed Requirements

The program must be run with *one* command line argument, which is the name of the *csv* file containing the data. The program should be named `treequery.pl`. If the tree data were stored in a file named `nyctreedata.csv`, then proper usage of this program would be

```
treequery.pl nyctreedata.csv
```

which will cause the program to read the `nyctreedata.csv`, and for line in the file, it will process that line according to the instructions below.

As mentioned above, each row in the data file has 9 fields. Following is a brief description of each field and what it contains.

- `tree_id`; a non-negative integer that uniquely identifies the tree
- `tree_dbh`; a non-negative integer specifying tree diameter
- `health`; a string, valid values: "`Good`", "`Fair`", "`Poor`", or the empty string
- `pc_common`; the common name of the tree, such as "`white oak`" or a possibly empty string
- `zipcode`; a positive five digit integer (This means that any number from 0 to 99999 is acceptable. The values that are shorter should be treated as if they had leading zeroes, i.e., 8608 represents zipcode 08608, 98 represents zip code 00098, etc.)
- `boroname`; valid values: "`Manhattan`", "`Bronx`", "`Brooklyn`", "`Queens`", "`Staten Island`"
- `nta_name`: a string, corresponding to the New York City neighborhood, such as "`Williamsburg`" or "`Astoria`"
- `latitude`; specifies GPS latitude of the tree point, in decimal degrees
- `longitude`; specifies GPS longitude of the tree point, in decimal degrees

The `latitude` and `longitude` values are what GPS systems use for location coordinates. Your program should define a hash named `gps_coordinates` that has two members, `latitude` and `longitude`. It should also define a hash named `tree`. The `tree` hash should contain a reference to `gps_coordinates` rather than storing the latitude and longitude as individual members.

A line might look like this:

```
179366,281,Fair,pin oak,11375,Queens,Forest Hills,40.73407358,-73.85075192
```

Your program should read all of the lines in the file, creating a `tree` record for each line and storing that into an array named `nyc_trees`. Once it has done that it must enter an interactive loop that prompts the user for the species common name of a tree, such as "`paper birch`" or "`white oak`", e.g.

```
Enter the name of a type of tree, or enter "quit" to quit:
```

to which the user might respond

```
white oak
```

If the user enters a valid tree name, meaning one for which there is at least one tree in the data set, the program will compute the following information:

- the total number of such trees in the data set
- a list of the zip codes in which this tree is found, separated by commas
- the borough containing the largest number of such trees and what that number is,
- the average diameter of all of the trees of this type in the data set.

This information should be presented on the standard output stream like this, with fictional data:

```
total number of such trees: 642
zip codes in which this tree is found: 10011,11103,11375,10002,10463
borough containing the largest number of such trees: Bronx, with 432
average diameter: 2.4
```

It should then redisplay the prompt. If the user entered "`quit`" then the program should exit.

Notice that the program does not use all of the data. There is an extra credit problem that uses the other data fields, but this is posted separately.

4 Error Checking

If the program is called without a command-line argument it is an error and the program should report it. If the file on the command line cannot be opened, the program should display a specific message related to the error and then exit. The program does not have to check that the data file is in the correct format.

5 Program Considerations

- As a reminder, the program must have proper documentation and comply with the Programming Rules.
- Your program should not make more than one pass across the array in order to compute the above pieces of information. If it does it is not designed well.
- The output should be what is described above and nothing else.

6 Testing Your Program

All programs must be thoroughly tested before they are released to users. I suggest that you create small subsets of the data to test your program. To do this you can use various filters. Which ones and how to use them I leave to you to figure out. The full data set is in the directory

```
/data/biocs/b/student.accounts/cs132/data/open_datasets
```

Do not copy this entire directory into your hoem directory on our system. It is wasteful and may use up your valuable disk space allocation. Instead make small subsets of it in your home directory. You can of course copy it onto your personal computing device.

Create some sample input files of a small size and manually figure out what the outputs should be. Run your program and make sure that your output matches the one you manually computed.

7 Rubric

This homework is graded on a 100 point scale. The program will be graded primarily on its correctness, and then on its efficiency. This means that it does exactly what the assignment states it must do, in detail. Correctness is worth 60% of the grade. Efficiency is 10%. Then it is graded on its documentation and design. Comments and appropriate naming of variables are worth 20%; good design another 8%. Naming all files and directories correctly is 2%.

8 Submitting the Program

This assignment is due by the end of the day (i.e. 11:59PM, EST) on Thursday, December 13; it is okay to submit it by 6:00 AM of the following day. To submit your project, you must follow the instructions below exactly! Do not deviate from these instructions.

To be precise:

1. Login using `ssh` to `eniac.cs.hunter.cuny.edu` with your valid username and password, and then `ssh` into any `cslab` host. Do not forget this step. You will not be able to run the `submithwk` command on `eniac`.

2. If you did not do the work on one of the computers in our network, then upload the Perl program file into your home directory. Create a directory named `your-username-hwk9` and put the Perl program into it.

3. Run the command

   ```
   zip -r your-username-hwk9.zip your-username-hwk9
   ```

   This will create the file `your-username-hwk9.zip`. The `zip` command is a special command that compresses the files in the directory and creates a new file that can later be extracted by the `unzip` command. So it will create a "zip file" named `your-username-hwk9.zip` containing your `your-username-hwk9` directory and any files it contains. For example, I would run

   ```
   zip -r sweiss-hwk9.zip sweiss-hwk9
   ```

4. Run the command

   ```
   /data/biocs/b/student.accounts/cs132/bin/submithwk 9  your-username-hwk9.zip
   ```

   Do exactly this. Do not mistype it. The command will create a copy of the file `your-username-hwk6.zip` in the directory

   `/data/biocs/b/student.accounts/cs132/hwks/hwk9`

   It will be named `hwk9_`*`username`*, where *`username`* is your username on the network. You will not be able to read this file, nor will anyone else except for me. If you decide to make any changes and resubmit, just do all the steps again and it will replace the old file with the new one. I will be able to unzip the file, extracting whatever files you created. Do not try to put your file into this directory in any other way - you will be unable to do this.

5. **Do not put anything in this directory other than the file treequery.pl. You will lose one point for each file that is in this directory that does not belong there.**

Although these instructions may seem complicated, they simplify the way you submit your work and the way I can retrieve it. If you make mistakes, just start over. If things don't seem to work out, post a question on Piazza with the details included.