Homework 7 - File I/O and CSV Files

CS 1301 - Intro to Computing - Fall 2024

Important

- Due Date: **Thursday, October 31st, 11:59 PM**.
- This is an individual assignment. High-level collaboration is encouraged, **but your submission must be uniquely yours.**
- Resources:
  - TA Office Hours or Ed Discussion
  - How to Think Like a Computer Scientist
  - CS 1301 YouTube Channel
- **Read the entire document before starting this assignment.**

The goal of this homework is for you to enhance your understanding of File I/O and practice reading and writing to files. Additionally, you will be able to work with important and frequently used data exchange formats like CSV by analyzing and writing data to and from CSV files. The homework will consist of **6 functions** for you to implement. You have been given `HW07.py` skeleton file to fill out. However, below you will find more detailed information to complete your assignment. Read it thoroughly before you begin.

Disclaimer: The autograder for HW07 may take about 30 to 40 seconds to fully run. The autograder will test your code with massive datasets, so please be patient!

Leaderboard: Since the autograder will be calculating the energy efficiency of some parts of your code, we have set up a fun, no-stakes, ungraded leaderboard of student submissions. When you submit HW07, you will be asked to provide a *Leaderboard Name*. **There is no obligation to put your actual name; a nickname would suffice. HOWEVER, inappropriate leaderboard nicknames will lead to no credit being awarded for HW07, and a misconduct report will be filed with the Office of Student Integrity.**

Hidden Test Cases: In an effort to encourage debugging and writing robust code, we will be including hidden test cases on Gradescope for some functions. You will not be able to see the input or output to these cases. Below is an example output from a failed hidden test case:

```
Test failed: False is not true
```

Written by: Nikhila Alavala, Matias Torres, Chris Liding, Saleh Alsedrah, Ananya Shetty, Priya Annapureddy, & Paige Holland

Helpful Information to Know

In industry, you may be asked to develop code that will be executed millions of times in energy-intensive data servers located all around the world. Thus, the approach you take to solve problems in computer science often has drastic consequences in code efficiency and carbon emissions. In this sustainability-themed homework, we will explore this concept by implementing functions that analyze data from huge datasets, the largest having over 1 million datapoints! We will be able to directly compare the carbon efficiency of different implementations, and we hope that you remain mindful of the energy consequences of the decisions you make while writing code throughout this homework and the rest of the semester.

Did you know?

Assigning multiple variables in one line is more energy efficient than in separate lines:

```
var1, var2, var3 = "happy", "earth", "day"
```

vs.

```
var1 = "happy"
var2 = "earth"
var3 = "day"
```

It is more energy efficient to sort a list **once**, outside of a for loop:

```
tips = [("***", "use cold water in laundry"),
        ("****", "unplug chargers from outlets"),
        ("**", "walk or bike to class")]
energyRatingList = []
for rating, tip in tips:
    energyRatingList.append((len(rating), tip))
    energyRatingList.sort()
```

vs.

```
for rating, tip in tips:
    energyRatingList.append((len(rating), tip))
energyRatingList.sort()
```

It is more efficient to import one function from a module instead of importing an entire module:

```python
from codecarbon import EmissionsTracker
```

vs.

```python
import codecarbon
```

Part 1 - File I/O

In Part 1 of this assignment, you will be reading and writing to text files that contain information about the carbon emissions of different forms of transportation. For the first function, you will need to analyze Flight Emission Data from the file `emissions.txt`, which is detailed below.

emissions.txt

For Part 1, the `emissions.txt` file being read from will be formatted as shown below. Be sure to download the provided file from Canvas, and **move it into the same folder as your `HW07.py` file**. Recall that if you are using a Mac, do not place HW07.py and `emissions.txt` in your Downloads or Desktop folder. To open the `.txt` file, you can use the built-in Notepad for Windows, TextEdit for Mac, or any other text editor of your choice.

You will be working with a text file that contains data about various flight routes and their associated carbon emissions. The data will contain the cities involved in each flight, the total distance traveled in miles, the number of miles per ton of carbon emissions, and the total tons of carbon emissions. Each flight's data will be separated by a newline. The code below shows how the text file will be formatted. See the provided `emissions.txt` file as an example.

```
city1 to city2 to city3
distance
miles per ton
tons

city1 to city4 to city3
distance
miles per ton
tons

...
```

Flying Green

Function Name: findGreenestFlight()
Parameters: start (`str`), end (`str`)
Returns: totalCarbonEmissions (`float`)
Description: The International Air Quality Commission is working to minimize carbon emissions in air travel, and your expertise is needed to help them identify the most eco-friendly flight options.

Write a function that takes in two parameters: a start and end city. Using the data provided in the `emissions.txt` file, your task is to identify the most efficient flight path between the two cities, which is the path that emits the least amount of carbon emissions whist traveling from the starting city to the ending city. If the most efficient flight emissions exceed 2.0 tons, **print** `"Who left the engine running?"` and don't return anything. Otherwise, **print** the `"Ready for takeoff!"` and **return** the amount of carbon emissions for the eco-friendliest route.

Note: The start city will always be `city1` and end city will always be `city3` in the `city1 to city4 to city3` line.

```
>>> findGreenestFlight("NYC", "LAX")
Ready for takeoff!
1.58
```

```
>>> findGreenestFlight("NYC", "MIA")
Who left the engine running?
```

fleetData.txt

You will also be working with a text file that contains information about an airline's fleet, including the name of the plane model, the maximum passenger capacituy, the maximum distance, and the emissions of CO2 per mile. Each plane's data will be separated by a newline. The code below shows how the text file will be formatted. See the provided `fleetData.txt` file as an example.

```
planeModel1
maxPassengers1
maxDistance1
lbs of CO2

planeModel2
maxPassengers2
maxDistance2
lbs of CO2

planeModel3
maxPassengers
maxDistance3
lbs of CO2
...
```

Picking Planes

Function Name: lowestEmissions()
Parameters: flights (`dict`)
Returns: None (`NoneType`)
Description: As a recent Georgia Tech graduate, Delta Airlines has decided to hire you as a software engineer. Your first task is determining which flight has the least carbon emissions per passenger.

Write a function that takes in a dictionary of flight routes mapped a tuple of related information, in the format `{"route": (distance, num_passengers), ...}`. For each route in the dictionary, find the **lowest** carbon-emitting plane model in `fleetData.txt` that satisfies the following two conditions:

1. The plane model is capable of flying the distance of the route.
2. The capacity of the plane model is large enough to hold the number of passengers on the route.

Write information about the plane model Delta should use for each route in a new file called `todaysFleet`.

Notes:

- If there aren't any available planes for a route, only include the route as shown in the test cases below.
- Be sure no newline characters (`\n`) are included at the end of `todaysFleet`.
- Each flight should be written in the order that it appears in the `flights` dictionary.

Below is the format for the output file `todaysFleet`:

```
The planes to be used for today's flights:

route1:plane1
route2:plane2
route3:plane3
...
```

The example test cases are on the next page.

```
>>> flights = {
    "ATL-JFK": (900, 120),
    "ATL-LAX": (1500, 90),
    "ATL-ORD": (850, 160),
    "ATL-SFO": (1200, 80),
    "ATL-BOS": (700, 130)
}

>>> lowestEmissions(flights)
```

Contents of `todaysFleet.txt` after the function above is executed:

```
The planes to be used for today's flights:

ATL-JFK:Airbus A321
ATL-LAX:Bombardier CRJ900
ATL-ORD:McDonnell Douglas MD-80
ATL-SFO:Bombardier CRJ900
ATL-BOS:McDonnell Douglas MD-80
```

```
>>> flights = {
    "ATL-JFK": (900, 120),
    "ATL-LAX": (1500, 90),
    "ATL-ORD": (1100, 200),
    "ATL-MIA": (400, 80),
    "ATL-SFO": (1200, 110)
}

>>> lowestEmissions(flights)
```

Contents of `todaysFleet.txt` after the function above is executed:

```
The planes to be used for today's flights:

ATL-JFK:Airbus A321
ATL-LAX:Bombardier CRJ900
ATL-ORD:
ATL-MIA:Bombardier CRJ900
ATL-SFO:
```

Part 2 - CSV Files

Instructions

In Part 2 of this assignment, you will be writing code that reads and analyzes international flight data that contains information on a flight's carbon emissions, with the end goal of creating functions that are able to query (or look up) and return the carbon emissions of a particular route.

To do so, you will have to read the data contained in the flight emissions dataset, and preprocess the data into a more useable data structure that is easier to work with. There are many different approaches and data structures you can use to store the data read from the dataset, but some approaches are much more energy efficient than others.

Even though the end goal of looking up specific flight routes is the same, you will need to implement two different approaches in this part of the homework: storing the data in the dataset in a list of tuples vs. storing the data in a dictionary. Are you able to tell which implementation is going to be more energy efficient? You'll soon find out.

Therefore, in Part 2 of this homework, you will be implementing two helper functions that read the dataset and store the data in a list or dictionary and implementing two main functions that are able to look up flight routes from these two data structures.

Our autograder for HW07 will test the carbon efficiency of your two implementations using a module called CodeCarbon. Please install CodeCarbon to keep track of the emissions of your code while you attempt this homework. See the CodeCarbon Guide Handout on Canvas to learn how to install and use it.

Note: While we encourage you to use CodeCarbon throughout your homework to test the carbon efficiency of your implementations, **please remove all CodeCarbon-related code from your homework file before you submit your code to Gradescope. CodeCarbon-related code will *significantly* slow down the speed of the autograder.** Without CodeCarbon, the autograder takes about 40 seconds to run. Be patient!

Extra Credit Opportunity (+5)

If your two implementations show a significant, favorable difference in carbon emissions, the autograder will award **up to 5 extra credit points** for this assignment, resulting in a maximum grade of 105/100 on this homework.

flight_emissions_data.csv

For Part 2, the `flight_emissions_data.csv` file being read from will be formatted as shown below, download the provided file from Canvas, and move it into the same folder as your `HW07.py` file.

By default, your computer will likely use Excel on Windows or Sheets on Mac to open the `.csv` file, but don't be alarmed: reading values from a `.csv` file is no different than a `.txt` file; the only difference lies in the formatting of the data. You take a look at the data using an Excel Sheet to understand the information a little easier.

The `flight_emissions_data.csv` file (and other `.csv` files like it) will contain global flight data covering top popular airports from Europe, Asia, America, and Africa. The file will have columns to denote the flight's origin airport, destination airport, and CO_2 emissions for a route.

origin_country	dest_country	co2_emissions
Columbia	Australia	2107000
Egypt	United Arab Emirates	331000
Greece	South Korea	1277000

- `flight_emissions_data_short.csv` contains 9 flights.
- `flight_emissions_data.csv` contains about one thousand flights.
- `flight_emissions_data_long.csv` contains about one million flights!

We recommend testing your code with `flight_emissions_data_short.csv`, since the other two datasets can take a considerable amount of time to read and process.

Helper Function #1

Function Name: csvToList()
Parameters: fileName (`str`)
Returns: flightsList (`list`)
Description: Helper Function #1 will be used to preprocess the flight emissions data into a list of tuples, which you will directly use in your List Implementation of Worst Routes. Write a function called `csvToList()` that takes in one parameter: the name of the flight emissions dataset file (`str`). This function should read the dataset and return a list of tuples, where each tuple represents one row of data structured as `(origin - str, destination - str, CO2 emissions - int)`. If the file name corresponds to a file that does not exist, return an empty list.

Note: The order of tuples in the list should correspond exactly to the order of flights in the flights emissions dataset passed in.

```
>>> csvToList("flight_emissions_data_short.csv")
[('Columbia', 'Algeria', 943000), ('Columbia', 'Germany', 889000),
 ('Columbia', 'Germany', 887000), ('India', 'Australia', 1302000),
 ('India', 'Australia', 996000), ('Ethiopia', 'United Kingdom', 560000),
 ('Ethiopia', 'United Kingdom', 560000), ('Ethiopia', 'United Kingdom', 756000),
 ('Ethiopia', 'United Kingdom', 729000)]
```

Helper Function #2

Function Name: csvToDict()
Parameters: fileName (`str`)
Returns: flightsDict (`dict`)
Description: Now instead of preprocessing the flights emission data into a list of tuples, Helper Function #2 will preprocess the data into a dictionary, which you will directly use in your Dictionary Implementation of Worst Routes. Write a function called `csvToDict()` that takes in one parameter: the name of the flight emissions dataset file (`str`). This function should read the dataset and return a dictionary mapping a tuple of the origin and destination as the key to a list of CO_2 emissions (`int`) from all flights with that specific origin and destination route. If the file name corresponds to a file that does not exist, return an empty dictionary.

Note: The order of emissions in all lists should correspond exactly to the order of flights in the flights emissions dataset passed in. The order in which the key-value pairs are located in the dictionary does not matter; recall that dictionaries are unordered data structures.

```
>>> csvToDict("flight_emissions_data_short.csv")
{('Columbia', 'Algeria'): [943000], ('Columbia', 'Germany'): [889000, 887000],
 ('India', 'Australia'): [1302000, 996000],
 ('Ethiopia', 'United Kingdom'): [560000, 560000, 756000, 729000]}
```

Worst Routes - List Version

Function Name: worstRoutes_list()
Parameters: originCountry (`str`), destCountry (`str`), flightsList (`list`)
Returns: message (`str`)
Description: Now we are ready to query (or look up) the emissions of a specific route from the flights emissions dataset thanks to the helper functions we wrote previously. Write a function called `worstRoutes_list()` that takes in three parameters: the country of origin (`str`), the destination country (`str`), and the list returned by the helper function `csvToList()` . By analyzing the data from the list returned by the helper function, find the flight that emitted the greatest carbon emissions on the specific route from the origin country to the destination country. **Return** a string in the following format: `"{origin} to {destination}: {worstEmissions} carbon emissions."` .

Note: You may assume that a route from the provided origin and destination country will always exist in `flightsList` .
Hint: Do not call `csvToList()` in your implementation of `worstRoutes_list()` . The helper function is meant to be called **outside** of the function, since the *return* value of the helper function is passed *into* this function.

```
>>> flightsList = csvToList("flight_emissions_data_short.csv") # Helper Function #1
>>> worstRoutes_list("Ethiopia", "United Kingdom", flightsList)
'Ethiopia to United Kingdom: 756000 carbon emissions.'
```

```
>>> flightsList = csvToList("flight_emissions_data_short.csv") # Helper Function #1
>>> worstRoutes_list("India", "Australia", flightsList)
'India to Australia: 1302000 carbon emissions.'
```

Worst Routes - Dictionary Version

Function Name: worstRoutes_dict()
Parameters: originCountry (`str`), destCountry (`str`), flightsDict (`dict`)
Returns: message (`str`)
Description: We are now going to implement the exact same functionality as the previous Worst Routes function, but with a dictionary this time. Take note of the change in efficiency and time between these two Worst Routes functions. Which one is more efficient?

Write a function called `worstRoutes_dict()` that takes in three parameters: the country of origin (`str`), the destination country (`str`), and the dictionary returned by the helper function `csvToDict()` . By analyzing the data from the dictionary returned by the helper function, find the flight that emitted the greatest carbon emissions on the specific route from the origin country to the destination country. **Return** a string in the following format: `"{origin} to {destination}: {worstEmissions} carbon emissions."` .

Note: You may assume that a route from the provided origin and destination country will always exist in `flightsDict` .
Hint: Do not call `csvToDict()` in your implementation of `worstRoutes_dict()` . The helper function is meant to be called **outside** of the function, since the *return* value of the helper function is passed *into* this function.

```
>>> flightsDict = csvToDict("flight_emissions_data_short.csv") # Helper Function #2
>>> worstRoutes_dict("Ethiopia", "United Kingdom", flightsDict)
'Ethiopia to United Kingdom: 756000 carbon emissions.'
```

```
>>> flightsDict = csvToDict("flight_emissions_data_short.csv") # Helper Function #2
>>> worstRoutes_dict("India", "Australia", flightsDict)
'India to Australia: 1302000 carbon emissions.'
```

Grading Rubric

Function	Points
findGreenestFlight()	25
lowestEmissions()	25
csvToList()	12.5
csvToDict()	12.5
worstRoutes_list()	12.5
worstRoutes_dict()	12.5
Extra Credit	5
Total	**105/100**

Provided

The `HW07.py` skeleton file has been provided to you. This is the file you will edit and implement. All instructions for what the functions should do are in this skeleton and this document.

Submission Process

For this homework, we will be using Gradescope for submissions and automatic grading. When you submit your `HW07.py` file to the appropriate assignment on Gradescope, the autograder will run automatically. The grade you see on Gradescope will be the grade you get, unless your grading TA sees signs of you trying to defeat the system in your code. You can re-submit this assignment an unlimited number of times until the deadline; just click the “Resubmit” button at the lower right-hand corner of Gradescope. You do not need to submit your `HW07.py` on Canvas.

Please remove all CodeCarbon-related code from your homework file before you submit your code to Gradescope. CodeCarbon-related code will *significantly* slow down the speed of the autograder.