CS2230 Computer Science II: Data Structures
Homework 6
Using Sets and Maps to answer Queries

Table of Contents

1 Goals for this assignment

- Use Sets and Maps to write efficient algorithms
- Debug programs using JUnit tests

2 Introduction

Oftentimes a dataset is stored in a table format, where the rows (horizontal) are ***records*** and the columns (vertical) are ***attributes***. An example of such a table is a comma-separated values (CSV) file. When we have a lot of data, we often want to ask questions that have a human-digestible answer. We call these questions ***queries***.

In this assignment, you'll be working with a dataset from the United States Department of Transportation's (USDOT) Bureau of Transportation Statistics (BTS). Specifically, the dataset contains airline flights in the US from one year. You'll answer interesting queries about this dataset by writing small Java programs.

3 Setup

1. Clone from https://research-git.uiowa.edu/cs2230-assignments/hw6-flights-sp21.git
2. Push the initial code to your private repository: https://research-git.uiowa.edu/cs2230-sp21/hw6-hawkid.
3. Set up an IntelliJ project with the hw6-flights-sp21 folder.
4. Download commons-csv-1.8.jar from ICON. Put commons-csv-1.8.jar into your hw6-flights-sp21 folder.
5. Download flights.zip from ICON. Put flights.zip into your hw6-flights-sp21 folder. Then unzip it. The result is that flights1990.csv, flights2005.csv, and flights2020.csv are in hw6-flights-sp21 folder.
6. You should now be able to run the tests in QueryTest.java.

Just an FYI: Why didn't we include the jar and csv files in the git repository?

- Typically, a git repository is only meant for source code. It is not a good place to put data (e.g., the large csv files) or external libraries (e.g., the jar file). We call such files "dependencies", and they should be distributed in a separate way from the source code.

- We have set things up so that you do not accidentally commit the jar or csv files. To do so, we've included a .gitignore file that tells git what types of files never to commit. You do not need to modify that file.

4 Structure of the dataset

Look at the first few lines of flights1990.csv, flights2005.csv, or flights2020.csv to see the format of the data. Each row is one flight, and each column is one of the attributes of that flight, such as the three-character airport code where the flight originated.

In the provided Java code, there is a class FlightRecord, which represents one record of the dataset. It has one instance variable for each attribute. All of its instance variables are public and final. Therefore, you can read a record's attributes but you cannot change its attributes' values.

5 Structure of a query

Each query takes an Iterable<FlightRecord> as input. See Query0.java for an example of a completed query. The query for Query0 is

How many flights were there from Cedar Rapids (CID) to Chicago (ORD) in the month of August (8)?

Some queries return a single integer (e.g., Query 0). Others return a single String (e.g., Query 5). And others have multiple results, so they return an Iterable (e.g., Query 2). For those that return an Iterable, you'll need to return an object whose class implements Iterable. Examples of Iterables include LinkedList, ArrayList, TreeSet, and HashSet. In fact, since interfaces List and Set extend the Iterable interface, any List or Set is an Iterable, including those returned by Map's methods entrySet and keySet.

For each query, you can check your answer by running the corresponding test in QueryTest.java.

Except for Query 0 and Query 1, all of these queries require key-based operations, such calculating distinct[1] results, counting how many times a key appears, or matching two records on a key. ***Use a Set or a Map*** as much as possible for operations like these. Good choices of Set or Map data structures to use are HashSet, TreeSet, HashMap, or TreeMap.

6 Query 1

[1] Distinct means no duplicates. For example, if you see "LAX" multiple times, you should include it just once.

How many flights out of Cedar Rapids (CID) were there in total? Return your result as a single integer.

This is the only query you'll write that does not require at least one Set or Map. All other queries will.

7 Query 2

What are the distinct destinations that can be reached in one flight from Cedar Rapids (CID)? Return each result as a String with the format DEST, DEST_STATE_ABR, for example, "HPN, NY".

If you are unsure how to create and return an Iterable, read Section 5.

8 Query 3

How many distinct destinations can be reached in one flight from Cedar Rapids (CID)? Return your result as a single integer.

9 Query 4

How many flights out of Cedar Rapids (CID) were there for each destination? You only need to include destinations for which there was at least 1 flight from CID. Return each result as a String with the format DEST=number of flights. For example, if there were 10 flights from CID to LAX, then "LAX=10".

10 Query 5

What month had the most flights (and how many flights was that)? Return your result as MONTH had NUM flights. For example, if the month was 7 and there were 500 flights then "7 had 500 flights".

11 Query 6

What pair of states has the most flights between them in either direction? Return your result as a single String with the two states in alphabetical order. For example, if the two states are IA and IL, then "IA, IL".

12 Query 7

What distinct states can you NOT get to from an Iowa (IA) airport in one flight? Only include states that are present somewhere in the dataset (whether as an origin or destination).

13 Query 8

For each state, what is the percentage of flights that stay within that state? For example, CID to DSM stays within Iowa. Return each result as STATE=PERCENT. For example, if the percentage of within-state flights in Florida is 5%, then “FL=0.050”. Round and display the percentage with three digits. Here is example code to do that.

```
float y = 0.1234567
DecimalFormat df = new DecimalFormat("#.000");
String withThreeDigits = df.format(y);
```

14 Query 9

For each state, what is the airline (UNIQUE_CARRIER_NAME) with the most flights into that state? Return each result as STATE,AIRLINE. For example, if the airline most flying to Kentucky is United Parcel Service, then “KY,United Parcel Service”.

15 Query 10

What distinct routes are there from CID to LAX that consist of exactly two flights (i.e., has a “layover”)? Such a route exists if the destination of the first flight is the origin of the second flight. Return each result as a String with the format $ORIGIN_1$->$DEST_1$->$DEST_2$. For example, if there is a flight from CID to ALB and a flight from ALB to LAX, then “CID->ALB->LAX”. We don’t include $ORIGIN_2$ in the output because it is the same as $DEST_1$.

Tips

- You are giving all “theoretical” routes. That is, we are ignoring the times of the first and second flight. The records don’t include takeoff and landing times, so we can’t know whether a passenger could really make the transfer from the first flight to the second flight.
- You’ll need to go over the input more than once! There are various algorithms for performing the matching between $DEST_1$ and $ORIGIN_2$. Algorithm 1 is simpler but less efficient than Algorithms 2 and 3.
- For any of the algorithms, if computation time is too high, you can also consider changing where in your loops you check for “CID” and where you check for “LAX”. Why does moving these checks change the computation time?

15.1 Algorithm 1: Nested loops

```
for all flights k in the input:
    for all flights m in the input:
        if k.DEST equals m.ORIGIN then you found a match!
```

15.2 Algorithm 2: Map

Uses a Map where the key is an airport and the value is a List<FlightRecords>.

```
for all flights k in the input:
    add k to the list whose key is k.DEST

for all flights m in the input:
    L = the list for key m.ORIGIN
    for all flights k in L:
        you found a match!
```

15.3 Algorithm 3: Sort and merge

Uses two SortedMaps where the keys are airports and the values are List<FlightRecords>

```
for all flights k in the input:
    in map1, add k to the list whose key is k.DEST

for all flights m in the input:
    in map2, add m to the list whose key is m.ORIGIN

iterate through the maps in sorted order, outputting the matches within a key by using nested loops
```