Data Structures

Overview

The world's air quality is deteriorating, and you've been chosen as Rutgers' leading data analyst to track air quality across various locations. Your task is to develop a robust system called "AirQuality" that efficiently monitors and analyzes air quality data using hash tables and other data structures.

In this assignment, you will implement a hash table to store air quality data for states and for the counties within each state! There are two levels of hash tables, both levels use separate chaining to resolve collisions.

Implementation

Overview of files provided

- The AirQuality class contains the methods you will use to work with the states and counties, base on the input file you'll be using; you will submit this file to Autolab.
- The **Driver** class is used to test your methods interactively. You can implement the driver by following the instructions indicated in the file, but **do not edit or submit to Autolab.**
- The County class contains information about a SINGLE county, including constructors and getter and setter methods. **Do not edit or submit to Autolab.**
- The State class contains information about a SINGLE state, including constructors, a and getter and setter methods. **Do not edit or submit to Autolab.**
- The MapView class provides visuals for AQI for different counties. **Do not edit or submit to Autolab.**
- **BEFORE YOU BEGIN:** Read the constructors and methods for each given class, to understand how they can be used, and how they connect to each other. Need help with understanding constructors, getters, and setters? Read this OOP guide.

The input File Format

The input file consists of states, counties, AQI (Air Quality Index), the county's location, the pollutant name, and color. This is a modified subset of data from the EPA.

The provided method **buildTable()** reads one line at a time from the input file and calls:

- **addState()** to insert the state into the hash table, and
- **addCountyPollutant()** to insert a county (there are many per state) and pollutant (there are many per county).

In the methods addState() and addCountyPollutant() below, you will need to parse a line from this file. Read the description for later methods to understand what indices you will need to read from.

You may use the following:

```
String[] token = inputLine.split(",");
String stateName = token[0];
String countyName = token[1];
int aqi = Integer.parseInt(token[2]);
...
```

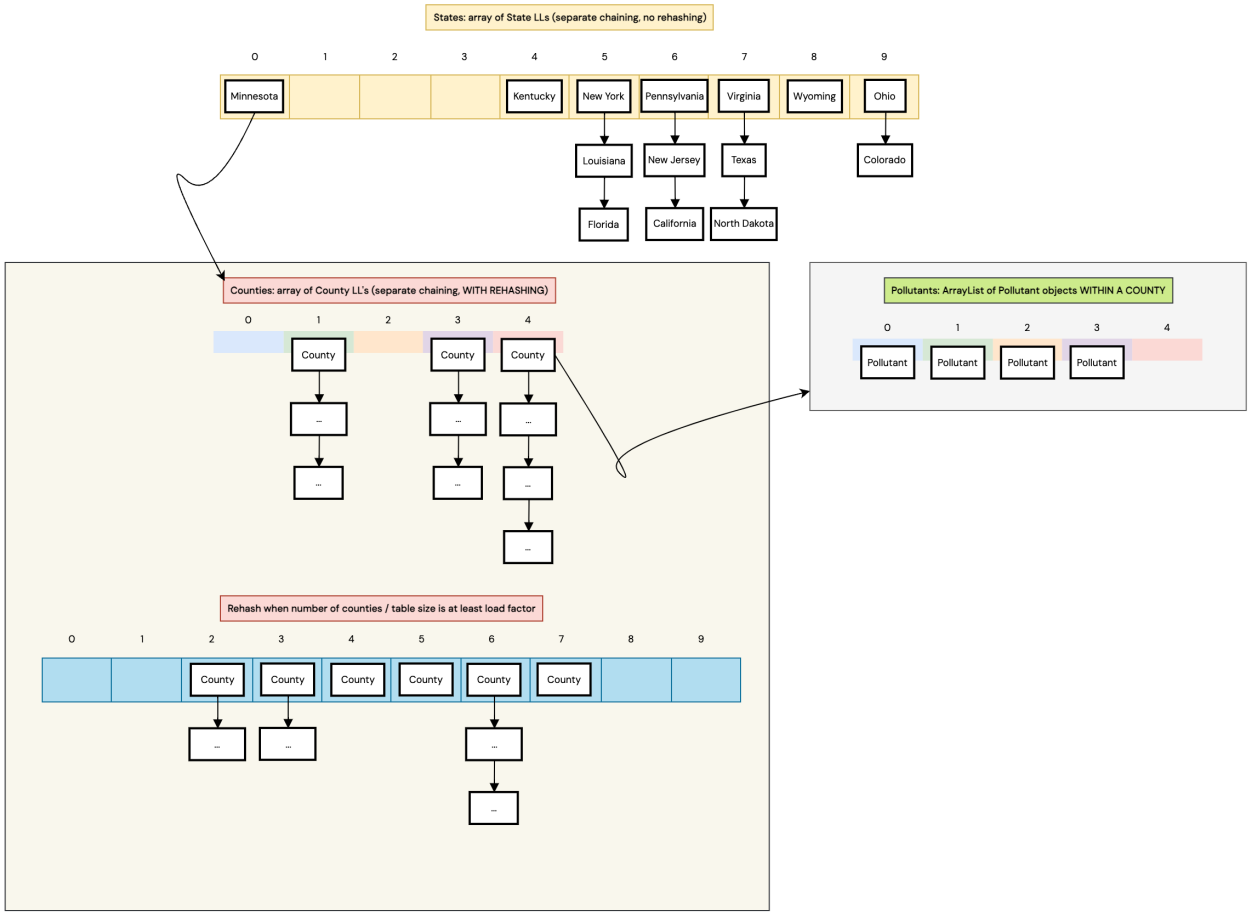
```
State Name,County Name,AQI,Latitude,Longitude,Pollutant Name,Color
California,Alameda,104,37.687526,-121.784217,Nitric oxide (NO),Orange
Colorado,Adams,168,39.838119,-104.94984,Carbon monoxide,Red
Florida,Broward,105,26.053889,-80.256944,Carbon monoxide,Orange
Kentucky,Boyd,157,38.45934,-82.64041,Sulfur dioxide,Red
Louisiana,Ascension,154,30.229653,-90.965628,Nitric oxide (NO),Red
Minnesota,Anoka,168,45.13768,-93.207615,Lead (TSP) LC,Red
New Jersey,Atlantic,195,39.464872,-74.448736,Sulfur dioxide,Red
New York,Bronx,228,40.816,-73.902,Sulfur dioxide,Purple
North Dakota,Billings,201,46.8943,-103.37853,Sulfur dioxide,Purple
Ohio,Allen,183,40.770944,-84.0539,Sulfur dioxide,Red
Pennsylvania,Adams,216,39.92002,-77.30968,Carbon monoxide,Purple
Texas,Bell,93,31.088002,-97.679734,Nitric oxide (NO),Yellow
Utah,Box Elder,67,41.945874,-112.233973,Nitric oxide (NO),Yellow
Virginia,Arlington,173,38.8577,-77.05922,Carbon monoxide,Red
Wyoming,Albany,132,41.32417,-105.61489,Sulfur dioxide,Orange
```

The Nested Hash Table

We will use a layered structure to store states, counties, and pollutants. There are no duplicate State, duplicate County, or duplicate Pollutant objects in the structure.

- The **states** instance variable refers to a separate-chaining hash table of State nodes (states[i] refers to the front of a list of states).
- Each state has **counties**, which are stored in a separate-chaining hash table of County nodes (counties[i] refers to the front of a list of counties) that uses rehashing.
- Each county has an ArrayList of **pollutants** that affect that county.

Refer to the following diagram, which illustrates how the linked structure works:



Lastly, colors represent hazard levels as shown below:

Daily AQI Color	Levels of Concern	Values of Index	Description of Air Quality
Green	Good	0 to 50	Air quality is satisfactory, and air pollution poses little or no risk.
Yellow	Moderate	51 to 100	Air quality is acceptable. However, there may be a risk for some people, particularly those who are unusually sensitive to air pollution.
Orange	Unhealthy for Sensitive Groups	101 to 150	Members of sensitive groups may experience health effects. The general public is less likely to be affected.
Red	Unhealthy	151 to 200	Some members of the general public may experience health effects; members of sensitive groups may experience more serious health effects.
Purple	Very Unhealthy	201 to 300	Health alert: The risk of health effects is increased for everyone.
Maroon	Hazardous	301 and higher	Health warning of emergency conditions: everyone is more likely to be affected.

buildTable – PROVIDED

This method reads the input file passed via parameters and for each line, calls **addState** and **addCountyAndPollutant**.

Do not change this method. You can call this method to test addState and addCountyAndPollutant as you implement each.

addState (inputLine)

The **inputLine** parameter contains the State Name that is used to create a State object.

Insert a single **State** object into the **states** hash table.

- No duplicates allowed.
- Use inputLine.split(",") to split the line into an array. The state name will be at index 0.
- Use Math.abs(*(State Name.hashCode())* as the key for the **states** hash table.
- Use *hash(key) = key % array length* as the hash function.
- If the state is already present, simply return the State object that is currently in the table. Otherwise, insert at the front of the list and return the newly created State object.

Note that the hash table **states** will not be resized. There are only 50 states in the US.

Submit AirQuality.java with this method completed under Early Submission for extra credit.

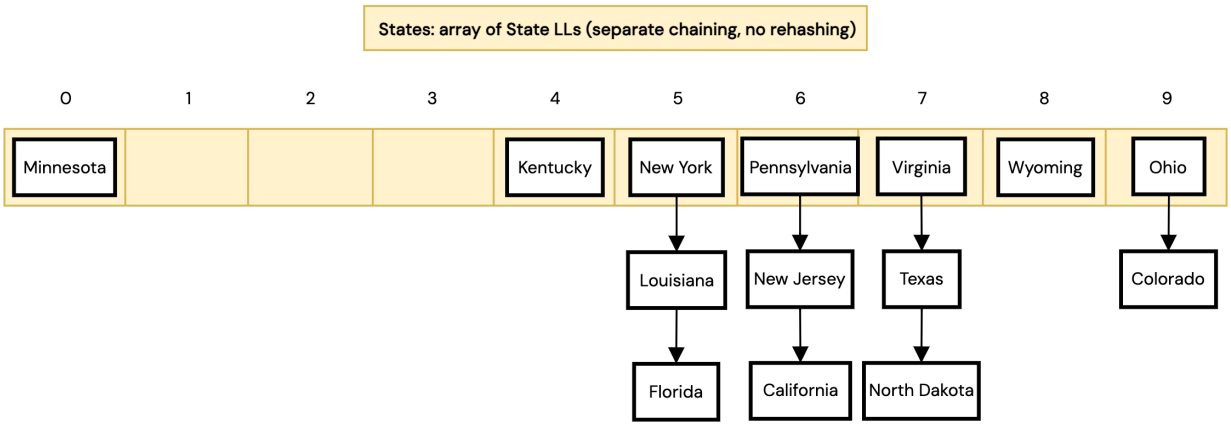
– USE the addState.csv file to test.

To test this method: Unlike previous assignments, you will be responsible for implementing Task.java files — each file will instantiate an AirQuality object, call the relevant methods, and call a print method to print out the table. **We have implemented AddState.java for you; you will need to implement other files to test.** We recommend that each task has its own class to test, although you are free to structure your test code however you see fit.

– Copy this file and use it as a template for other methods.

Alternatively, use JUnit tests to test specific assertions: the *AirQualityTest.java* test class found under the "test" directory in the project folder. The test for this method has been provided for you; you'll need to write other tests. If you've installed TestRunner for Java, you can run this test to evaluate your method's code.

For other methods, you will need to write each test yourself, as well as expand the test input file data.



checkCountiesHTLoadFactor (State state)

This method returns true if the parameter **State counties** hash table requires resizing (rehashing).

Note that the State object contains:

- the total number of counties within the sate.
- the size of the **counties** table.
- the load factor.

To test this method, implement this method, rehash, and addCountyAndPollutant and compare the entire hash table. It's helpful to trace through your code manually.

rehash (State state)

This method resizes (rehashes) the State's **counties** hash table by doubling its size.

- Note: to find the chain where a **county** will be inserted into the doubled size array you can use, Math.abs(**countyName.hashCode()) % new table size**
- Insert items to the front of its new index in the new table, going index-by-index, state-by-state for order (exhaust all states in an index before moving on to the next one).
- Remember to update the State's counties hash table with the doubled size array.

To test this method, implement this method, checkCountiesHTLoadFactor above, and addCountyAndPollutant and compare the entire hash table.

addCountyAndPollutant (State state, String inputLine)

The **inputLine** parameter contains the County Name, Latitude, Longitude, that are used to create a County object, as well as Pollutants (which have a name, AQI and Color).

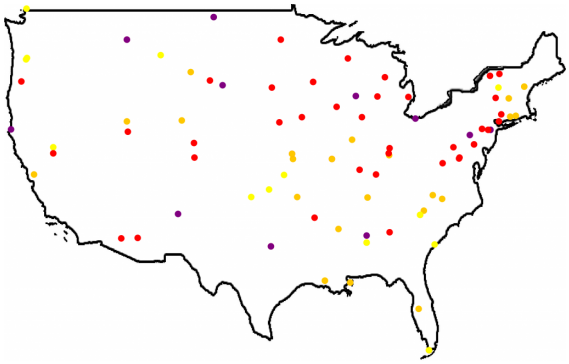
- No duplicates allowed.
- Use inputLine.split(",") to split the line into an array. The county name will be at index 1, AQI (as an int) at index 2, latitude (as a double) at index 3, longitude (as a double) at index 4, pollutant at index 5, and color at token 6.
- Use Math.abs(*County Name.hashCode()*) as the key into the **counties** hash table.
- Use *hash(key) = key % array length* as the hash function.
- Insert a single **County** object into the **State** parameter **counties** hash table, if a County with County Name is not already present.
  - States have an addCounty method.
- Then, check the State's **counties** hash table load factor after inserting. The hash table may need to be resized (implement rehashing in the **rehash()** method).
  - Implement **checkCountiesHTLoadFactor()** to perform this check.
- Then use Pollutant Name to search the County **pollutants** ArrayList.
  - If a Pollutant is found, update its AQI and color.
  - Otherwise use the Pollutant Name, AQI and color to create a Pollutant object and insert it into the County pollutants ArrayList.

To test this method, create an AddCountyAndPollutant.java file and call buildTable() on an instance of AirQuality to test. Use the printEntireTable() method from AddState.java to print the whole table. **USE AddState.java as a template.**

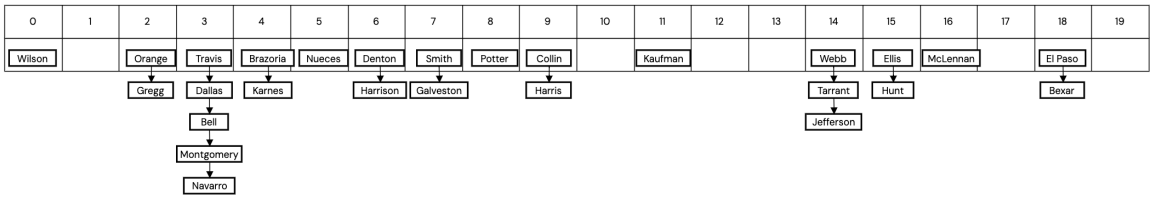
Alternatively, you can create JUnit cases to test specific assertions.
– Where should the state be located?
– We use a separate-chaining table with rehashing to store counties within a state. Given this, check the list in the corresponding index of counties to see if the county is contained in the correct position.

While this is not a method to test, we provide a visualizer so you can see air quality levels of approximate locations. Run MapView.java and enter any input file. This is the result for pollutedCounties.csv (locations on this map are approximate and not exact).

DO NOT USE the map to test your program or see if your output is incorrect or correct, as locations may be off from their actual ones in the US.



USE rehashCounties.csv to test this method.



Here is the expected output for a small subset of Texas using the above input file; note that there are 20 indices in total:

```
State Index 7
Texas:
  County Index 0
   -> Wilson:
    {Nitric oxide (NO), 36, Green}
  County Index 1: EMPTY
  County Index 2
   -> Orange:
    {Nitric oxide (NO), 101, Orange}
    {Sulfur dioxide, 101, Orange}
   -> Gregg:
    {Sulfur dioxide, 93, Yellow}
    {Nitric oxide (NO), 93, Yellow}
  County Index 3
   -> Travis:
    {Sulfur dioxide, 122, Orange}
    {Nitric oxide (NO), 122, Orange}
    {Carbon monoxide, 122, Orange}
   -> Dallas:
    {Carbon monoxide, 172, Red}
    {Sulfur dioxide, 172, Red}
    {Nitric oxide (NO), 172, Red}
   -> Bell:
    {Nitric oxide (NO), 93, Yellow}
   -> Montgomery:
    {Nitric oxide (NO), 122, Orange}
   -> Navarro:
    {Sulfur dioxide, 80, Yellow}
    {Nitric oxide (NO), 80, Yellow}
  County Index 4
   -> Brazoria:
    {Nitric oxide (NO), 136, Orange}
    {Sulfur dioxide, 136, Orange}
   -> Karnes:
    {Nitric oxide (NO), 29, Green}
  County Index 5
   -> Nueces:
    {Sulfur dioxide, 101, Orange}
```

setStatesAQIStats ()

For each State object in the **states** hash table:

- compute the **average** AQI.
- find the county with the **highest** AQI.
- find the county with the **lowest** AQI.

Remember that each county may contain many Pollutants and each Pollutant has its own AQI. Use the appropriate setter methods to update the State object with these values. Call buildTree first in your test code, and implement all previous methods. USE pollutedCounties.csv to test this method. Here is the expected output for a small sample of this input file:

```
State Index 0
Minnesota: AvgAQI= 168.67, HighestAQI County= Dakota, LowestAQI County= Olmsted
Maryland: AvgAQI= 167.00, HighestAQI County= Washington, LowestAQI County= Washington
Maine: AvgAQI= 118.00, HighestAQI County= Aroostook, LowestAQI County= Cumberland
Iowa: AvgAQI= 162.00, HighestAQI County= Polk, LowestAQI County= Polk
Arizona: AvgAQI= 157.50, HighestAQI County= Maricopa, LowestAQI County= Gila

State Index 1
Nebraska: AvgAQI= 169.00, HighestAQI County= Douglas, LowestAQI County= Douglas
Illinois: AvgAQI= 145.00, HighestAQI County= Jo Daviess, LowestAQI County= Champaign
Hawaii: AvgAQI= 112.33, HighestAQI County= Kauai, LowestAQI County= Hawaii

State Index 2
Puerto Rico: AvgAQI= 106.00, HighestAQI County= Ponce, LowestAQI County= Catano
Utah: AvgAQI= 147.50, HighestAQI County= SaltLake, LowestAQI County= Weber
Tennessee: AvgAQI= 135.00, HighestAQI County= Blount, LowestAQI County= Davidson
Oklahoma: AvgAQI= 97.50, HighestAQI County= Dewey, LowestAQI County= Kay
North Carolina: AvgAQI= 126.33, HighestAQI County= Mecklenburg, LowestAQI County= Rockingham
Georgia: AvgAQI= 166.00, HighestAQI County= Fulton, LowestAQI County= Fulton
```

meetsThreshold (String stateName, String pollutantName, int AQIThreshold)

This method is expected to return an ArrayList<County> where each County:

- belongs to stateName, and
- contains a Pollutant with pollutantName where the AQI value is greater or equal than AQIThreshold.

To test this method, call buildTree() and setStatesAQIStats(). You can use rehashCounties.csv (only two states, but many counties) to test this file, OR pollutedCounties.csv to work with multiple states and a smaller number of counties.

– In your test code, iterate through the returned ArrayList and print out the counties (ex: county name) in that ArrayList.

When we check in Texas for "Carbon monoxide" levels with a threshold of 143 (using rehashCounties.csv), we get the following counties in order:

Dallas
Harris
Tarrant
El Paso
Bexar

Implementation Notes

- YOU MAY only update the methods with the WRITE YOUR CODE HERE line.
- **COMMENT all print statements you have written from AirQuality.java**
- DO NOT add any instance variables to the **AirQuality** class.
- DO NOT add any public methods to the **AirQuality** class.
- DO NOT add/rename the project or package statements.
- DO NOT change the class **AirQuality** name.
- YOU MAY add private methods to the **AirQuality** class.
- YOU MAY use any of the libraries provided in the zip file.
- **DO NOT** use System.exit()

VSCode Extensions

You can install VSCode extension packs for Java. Take a look at this tutorial. We suggest:

- Extension Pack for Java
- Project Manager for Java
- Debugger for Java
- Test Runner for Java

Importing VSCode Project

1. Download ElectionAnalysis.zip from Autolab Attachments.
2. Unzip the file by double clicking.
3. Open VSCode
   - Import the folder to a workspace through **File** > **Open**

Executing and Debugging

- You can run your program through VSCode or you can use the Terminal to compile and execute. We suggest running through VSCode because it will give you the option to debug.
- How to debug your code
- If you choose the Terminal:
  - first navigate to **AirQuality** directory/folder (the one that directly contains the src, lib and bin folders).
    - to compile: **javac -d bin src/quality/*.java**
    - to execute: **java -cp bin quality.AddState**
    - **IMPLEMENT classes to test other methods, using AddState as a template.**
    - **OPEN the AirQuality folder in VSCode, not another folder that contains an inner AirQuality folder (ex: AirQuality(2) -> AirQuality, CS112 Code -> AirQuality).**

Before submission

COMMENT all printing statements you have written from AirQuality.java

Collaboration policy. Read our collaboration policy here.

Submitting the assignment. Submit *AirQuality.java* separately via the web submission system called Autolab. To do this, click the *Assignments* link from the course website; click the *Submit* link for that assignment.

Getting help

If anything is unclear, don't hesitate to drop by office hours or post a question on Piazza.

- Find instructors office hours here. Come to office hours if you have specific issues regarding your solution.
- Find tutors office hours on Canvas -> Tutoring
- Find head TAs office hours here
- In addition to office hours we have the CSL (Coding and Social Lounge) in Hill 252, a community space staffed with lab assistants which are undergraduate students further along the CS major to answer general questions.

By Anna Lu and Srimathi Vadivel