CSC2001F 2019 Assignment 1

Instructions

The goal of this assignment is to compare the Binary Search Tree with a traditional unsorted array data structure, both implemented in Java, using real power consumption data.

Dataset

The attached file represents a time series of power usage for a suburban dwelling. The base data set and additional information can found at

https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption

Load the file into LibreOffice/OpenOffice/MS-Excel to see what the data looks like. Note that the first line consists of headings and needs to be ignored by your code. Every subsequent line represents a single data item/record.

In your application, you MUST write your own code to read in the Comma Separated Value (CSV) file. You may not use a CSV library. Your data structure items must each store the following **3** values extracted from the CSV file:

- "Date/time"
- "Global_active_power" (which we will call "Power")
- "Voltage"

Part 1

Write an application **PowerArrayApp** to read in the attached CSV file and store the data items within a traditional *unsorted* array (a single array of objects). There are **500** data items - you may use a fixed-size array or try to determine the size programmatically. Do not use a LinkedList, ArrayList or other advanced data structure.

Include the following methods in your code:

- *printDateTime (dateTime)* - to print out the **Date/time**, **Power** and **Voltage** values for the matching *dateTime* record; or "Date/time not found" if there is no match.
- *printAllDateTimes ()* - to print out the data as above, but for **all** date/time records, in any order

You should be able to invoke your application using a command line such as

```
java PowerArrayApp "16/12/2006/17:44:00"
```

to print details for the indicated Date/time string or

```
java PowerArrayApp
```

to print all date/time details. You may use quotes in your parameters or not - it is up to you. *Be sure to explain, in your report, how you specify the CSV input to your program.* The java command line does not have to look like the above, this is just to illustrate how you might call your application.

Test your application with **3** known date/time strings, one unknown date/time string and without any parameters. Use output redirection in Unix to save the output in each case to different files.

Part 2

Add additional code to your solution to Part 1 to discretely count the number of *comparison* operations (<, >, =) you are performing in the code. *Only count where you are comparing the value of date/time strings*. This is called **instrumentation**. There are 3 basic steps.

First, create a variable/object (e.g., opCount=0) somewhere in your code to track the counter; maybe use an instance variable in the data structure class.

Secondly, wherever there is an operation you want to count, increment the counter (opCount++). For example:

```
opCount++;   // instrumentation to count comparison

if (queryDateTime == theDateTime) // comparison op
...
```

Finally, report the value of the counter before the program terminates. Perhaps add a method to write the value to a file before the program terminates.

Test your application with **3** known date/time strings, one unknown date/time string and without any parameters. Take note of the operation count in each case.

Part 3

Write an application **PowerBSTApp** to perform the same tasks as Part 1, but using a Binary Search Tree (BST) instead of an array.

Your BST implementation can be created from scratch or re-used from anywhere (with appropriate code attribution in the comments i.e. credit the source – credit must also appear in your report). You may **NOT** replace the BST with a different data structure.

Once again, test your application with 3 date/time strings, one unknown date/time string and without any parameters. Use output redirection in Unix to save the output in each case to a different file.

Part 4

Instrument your **PowerBSTApp** code just as you did in Part 2.

Test your application with 3 date/time strings, one unknown date/time string and without any parameters. Take note of the operation count in each case.

Part 5

Conduct an experiment with **PowerArrayApp** and **PowerBSTApp** to demonstrate the speed difference when performing a search between the BST and a traditional array implementations.

You want to vary the size of the dataset (N) and measure the number of comparison operations in the best/average/worst case for every value of N (from 1-5000, in steps of 20). For each value of N:

- Create a subset of the sample data (you can use the Unix **head** command to extract the first N lines).
- Run both instrumented applications for every date/time string in the subset of the data file. Store all operation count values. You can simply iterate through the subset calling your program repeatedly. Or you could develop a more efficient means of testing – doing this will earn you extra marks if you explain this clearly in the report (see below).
- Determine the best case (minimum), worst case (maximum) and average of these count values.

It is recommended that you use Unix (or Python) scripts to automate this process. You can set up the output so that it consists of text formatted in a sensible way. This can be read by a graphing program which can process text-based input data (e.g. gnuplot, matplotlib (a python module)) or anything else that you find.

Report

Write a report (of up to 6 pages) that includes the following (with appropriate section headings):

- What your OO design is: what classes you created and how they interact.
- What the goal of the experiment is and how you executed the experiment.
- What test values you used in the trial runs (Part 2 and Part 4) and what the operations counts were in each case. Only show the first 10 and last 10 lines for the trial run where you invoke *printAllDateTimes ()*.
- What your final results for Part 5 are (use one or more graphs), showing best, average and worst cases for both applications. Discuss what the results mean.
- A statement of what you included in your application(s) that constitutes creativity - how you went beyond the basic requirements of the assignment.
- Summary statistics from your use of git to demonstrate usage. Print out the first 10 lines and last 10 lines from "git log" , with line numbers added. You can use a Unix command such as:

```
git log | (ln=1; while read l; do echo $ln\: $l; ln=$((ln+1)); done) | (head
-10; echo ...; tail -10)
```

Dev requirements

As a software developer, you are required to make appropriate use of the following tools:

- **git**, for source code management
- **javadoc**, for documentation generation
- **make**, for automation of compilation and documentation generation

Submission requirements

Submit a .tar.gz compressed archive containing:

- Makefile
- src/
    - all source code
- bin/
    - all class files
- doc/
    - javadoc output
- report.pdf

Your report must be in PDF format. **Do not submit the git repository.**

Marking Guidelines

Your assignment will be marked by tutors, using the following marking guide.

Artefact	*Aspect*	*Mark*
Report	Appropriate design of OOP and data structures	5
Report	Experiment description	5
	Trial test values and outputs (Part 2)	5
	Trial test values and outputs (Part 4)	5
	Results - tables and/or graphs	5
	Discussion of results	5
	Creativity	5
	Git usage log	2
Code	Looks reasonable and no obvious inefficiencies	5
Dev	Documentation - javadoc	5
	Makefile - compile, docs, clean targets	3