CSC2001F 2020 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 a real-world application to read in and provide access to Cape Town's load shedding data.

Dataset

The attached file is the official load shedding area and time listing for Cape Town from http://resource.capetown.gov.za/documentcentre/Documents/Procedures%2C%20guidelines%20and%20regulations/Load_Shedding_All_Areas_Schedule_and_Map.pdf

For your convenience, this file has been converted from PDF to text and then cleaned to remove headings and other non-data cells. The remaining data has been pre-processed (by the attached script) to add a load-shedding stage, day of month and start of time period to the beginning of each line. The "clean.final" file is what you should use - it has 2976 entries.

Study the data carefully. The pre-processing is non-trivial but something you can do yourself as well.

In your application, you MUST write your own code to read in the text file. Your data structure items must each store 2 values for each entry, such as the ones below. The key is a concatenation of the stage, day and time, separated by underscores.

- 2_5_20 (Stage 2, 5th day of the month, 20h00 start of time period)
- 1, 5, 3, 6 (List of areas that will be load-shed)

Part 1

Write an application **LSArrayApp** to read in the attached text file and store the data items within a traditional array (a single array of objects). There are 2976 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:

- *printAreas (stage, day, startTime)* - to print out the list of areas for the first matching combination of the supplied parameters that is found; or "Areas not found" if there is no match.
- *printAllAreas ()* - to print out the areas for every stage, day and start time, in any order.

You should be able to invoke your application using commands such as

```
java LSArrayApp "2" "13" "04"
```

to print details for Stage 2, 13th of the month, starting time of 04:00, or

```
java LSArrayApp
```

to print all load shedding details. You may use quotes in your parameters or not - it is up to you.

Test your application with 3 known parameter combinations that work, 3 invalid parameters (errors in different parameters) 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 keys. 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
if (queryString == theKey)
...
```

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

Test your application with 3 known parameter combinations that work, 3 invalid parameters (errors in different parameters) and without any parameters. Take note of the operation count in each case.

Part 3

Write an application **LSBSTApp** 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. You may NOT replace the BST with a different data structure.

Once again, test your application with 3 known parameter combinations that work, 3 invalid parameters (errors in different parameters) and without any parameters. Use output redirection in Unix to save the output in each case to different files.

Part 4

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

Test your application with 3 known parameter combinations that work, 3 invalid parameters (errors in different parameters) and without any parameters. Take note of the operation count in each case.

Part 5

Conduct an experiment with **LSArrayApp** and **LSBSTApp** to demonstrate the speed difference for searching between a BST and a traditional array.

You want to vary the size of the dataset (n) and measure the number of comparison operations in the best/average/worst case for different values of n. Use 10 values of n that are spaced approximately equally apart (n=297, 593, ..., 2976). For each value of n:

- Create a subset of n entries from the sample data (preferably use a random subset of lines).
- Run both instrumented applications for every one of the n combinations of parameters corresponding to the subset of the data file. Store all operation count values.
- Determine the minimum (best case), maximum (worst case) and average of these count values.

It is recommended that you use Unix or Python scripts to automate this process.

Report

Write a report (of up to 6 pages) that includes the following:

- 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 *printAllAreas ()*.
- What your final results 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=0; 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	10
Report	Experiment description	10
	Trial test values and outputs (Part 2)	10
	Trial test values and outputs (Part 4)	10
	Results - tables and/or graphs	10
	Discussion of results	10
	Creativity	10
	Git usage log	5
Code	Looks reasonable and no obvious inefficiencies	10
Dev	Documentation - javadoc	10
	Makefile - make, docs, and clean targets	5