CS210 with Dr. Basit Qureshi **Term Project 2** **Weight 10%** **Due: End of Tuesday December 6, 2022**

The Green Computing Project

[Source: Lancaster University Center for Ecology & Hydrology, Last accessed 11 Nov 2022]

The goal for Green Computing is to reduce the carbon foot-print by increasing the energy-efficiency and reducing e-waste. This means using environment friendly energy sources producing less waste and promoting sustainability. A lot of work has been done in developing sustainable energy sources such as wind-farming, solar-energy etc, resulting in green energy utilization harnessed from these sources. However, a majority of energy generation is through "brown energy sources" which means burning fossil fuels to generate electricity. This obviously has negative impact on the environment.

If you think computers are green, think again! Computers have Central Processing Units (CPUs) and Game Processing Units (GPUs) in addition to other components responsible for high energy consumption. Business servers, Gaming machines and other peripherals consume large amounts of energy for resource-demanding applications. If you use the Internet applications such as Google Gmail, YouTube, Amazon EC2, Zoom, etc, chances are that your applications runs on remote servers housed in large Data Centers forming the cloud. The cloud is a collection of many computer systems/servers that work together in Data Centers. A data center may consist of thousands of computers interconnected to work as one large computer. These Data centers, the so-called "Farms of servers" consume Giga-watts of energy, adding a lot of carbon to the environment. You can learn about technologies created and initiatives taken by IT giants such as Microsoft, Google, Amazon etc that reduce energy consumption in their data centers.

It is estimated that out of $250 billion per year spent on powering computers worldwide only about 15% of that power is spent computing, the rest is wasted idling (i.e. consumed by computers which are not in use but still turned ON) [Source: EnergyStar]. There are many ways this problem can be tackled. One of the solutions to address this is writing optimized algorithms that require less energy to run on computers and systems.

In this project we are going to write a program using two-data structures, Binary Search Trees and Hash tables. The programs implement the classical word-count application, where you read a bunch of text file(s) in a directory and count how many times a word appears (frequency of words) in the files. You will conduct an empirical evaluation of the run-time of both these programs to see which one performs better and consumes less energy. We will be using datasets composed of e-books available as text files on Gutenberg website [Source: Gutenberg] of varying sizes. The results would be documented in a report providing conclusive evidence that we can effectively write better programs that are good for the environment.

1. Project API

Your project implements five classes namely `NodeBST`, `BST`, `NodeHT`, `HT` and `Solution`. The classes `NodeBST`, `BST` implement a Binary Search Tree Data Structure. The classes `NodeHT`, `HT` implement the Hash Table with separate Chaining method for collision resolution. The class provides the main method that triggers the appropriate methods. Your program takes input using command line arguments and reads text files from a directory.

The following provides an API for your project.

NodeBST	Description

String key;
int frequency;
NodeBST left;
NodeBST right;
	// a unique string keyword read from file;
	// Frequency for the word stored in key;
	// Node left
	// Node right

NodeBST ()
NodeBST (,,,)
 	
String toString()
	// default constructor
	//override constructor as necessary. You can edit the params as needed.
	
	//returns a String with the contents of the Node as follows:
	Key, Frequency
	
	//You can add additional methods setter/getters etc

The following is the API for the BST class. This implements a Binary Search Tree consisting of Nodes from the Node class.

BST	Description
NodeBST Root; int size;	// Anchors the Root of the Tree // Maintains the size of the Tree

BST();
 	
insert(String key)
 	
int search(String key)
 	
void remove(String key)
 	
void printAll()
	// default constructor
	
	// inserts the key in the tree if it is non-existent. Otherwise, the key/value (frequency) is updated/incremented.
	
	//searches for the key in the BST, returns the frequency or -1 (if not found).
	
	//Searches for the key in the BST and removes the node.
	
	//Prints ALL the keys with their frequencies. Essentially calls NodeBST.toString()

We suggest that you modify/edit the BST implementation provided at:
https://github.com/basit388/cs210/tree/master/Trees/src/BST

The following is the API for the NodeHT class which stores a Hash table node.

NodeHT	Description

String key;
int count;
NodeHT next;
	// a unique string keyword read from file;
	// Frequency for the word stored in key;
	// NodeHT next node pointer

NodeHT()
 	
String toString()
	// default constructor
	
	//returns a String with the contents of the NodeHT as follows:
	Key, Frequency
	
	//You can add additional methods setter/getters etc

The following is the API for the HT class which implements a Hash Table with chaining method.

HT	Description
Node HT [] Table; int size;	// an array of size 997 of type NodeHT. // Stores the size (number of nodes)

`NodeHT next;`	`// NodeHT next node pointer`

`HT()`
 	
`int HashCode(String key, int HashFunctionNo)`
 	
`void insert(String key)`
 	
`int Search(String key)`
 	
`Remove(String key)`
 	
`void printAll()`
	`// default constructor. Initializes the Array Table and sets size to 0.`
	
	`//Generates a HashCode for the provided string using Hash function Number:`
	`1. f(x) = x % 997`
	`2. f(x) = x % 463 [OPTIONAL]`
	`3. f(x) = x % 107 [OPTIONAL]`
	
	`//inserts a new node at the beginning if key is non-existent; Otherwise, the key/value (frequency) is updated/incremented.`
	
	`//searches for the key in the HT, returns the frequency or -1 (if not found).`
	
	`//Searches for the key in the HT and removes the node.`
	
	`//Prints ALL the keys with their frequencies. Essentially calls NodeHT.toString()`

All students are expected to implement Hash function 1. Implementation hash functions 2 and 3 are optional.

The following is the API for the Solution class. This class implements the main method, makes appropriate data Input/Output calls and implements data structures and all necessary method calls.

Solution	Description
`//Attributes NA`	`Not applicable`

`main(//arguments)`	`Implements the main method; reads data from console/files, processes information and make appropriate calls as necessary.`
	
	`You may add other methods as necessary`

You can write your program using an IDE of your choice.

2. Running your program

The program runs from console with the following parameters:

Usage:

```
java -jar [Solution] [1 for BST] [Directory containing files] [Search keywords]
```

or

```
java -jar [Solution] [2 for HT] [Directory containing files] [Hash Function No] [Search keywords]
```

Sample Input with BST and relevant output:

```
Solution 2 /mydir the hello
*****
Start time: 290182692
Started reading files from directory: mydir
Loaded all files in BST
End time: 292785317
Time taken: 3022.609 milliseconds
*****
*****
Start time: 292785319
Searching key: hello
End time: 292785322
Time taken: 2.103 milliseconds
the 2837
*****
*****
Start time: 292785326
Searching key: the
End time: 292785329
Time taken: 1.819 milliseconds
hello 137
*****
```

Sample Input with HT and relevant output:

```
Solution 2 /mydir 1 the hello
*****
Start time: 290182692
Started reading files from directory: mydir
Using Hash function: 1
Loaded all files in HT
End time: 292785317
Time taken: 3037.609 milliseconds
*****
*****
Start time: 292785319
Searching key: the
End time: 292785322
Time taken: 2.103 milliseconds
the 2837
*****
*****
Start time: 292785326
Searching key: hello
End time: 292785329
Time taken: 1.819 milliseconds
hello 137
*****
```

3. Generating and Collecting Data

Step 1. Generate Dataset

To generate data from this empirical study, select 5 to 20 books from the Gutenberg repository available at this page and download these as text format (Plain text UTF-8). Describe these files here

FileName	Book Name	Book URL	File Size (Bytes)

Now make 3 datasets as follows

1. Make a directory ds1 and copy the one file in this directory
2. Make a directory ds2 and copy the 3-5 files in this directory
3. Make a directory ds3 and copy all of the files in this directory

Step 2. Provide details about your computer

Write details about your computer

Hardware	Example	Your input here
Type of PC - Model	Apple / Wintel	
CPU	Intel i7	
# of cores	8	
Max core frequency	2.2 Ghz	
RAM	16 GB	
Operating System	macOS Mojave / Windows etc	

Used Memory	12.3GB	
Remaining Memory	About 4GB	
CPU usage	1%	
Disk type	HDD/SSD	
Disk capacity	?	
Virtual Memory used	Windows; Mac	
Power Supply rating (P)	Watts?	

For mac: Check “About this mac”

For windows: Right click on “My Computer” Properties

3. Run your program and collect data

Run the program for the text files according to the following, and complete the observation table 1. This needs to be completed individually. It is important that you record the data correctly.

Data Structure	Dataset	Loading/reading time
BST	Ds1	
BST	Ds2	
BST	Ds3	
HT	Ds1	
HT	Ds2	
HT	Ds3	

Search time for queries on DS1

Query	BST (time in milli seconds)	HT (time in milli seconds) with HashFunction1	HT (time in milli seconds) with HashFunction2	HT (time in milli seconds) with HashFunction3
be				
and				
sea				
the				
four				
baby				
age				
toward				
property				
shoulder				
	Total:	Total:		

Search time for queries on DS2

Query	BST (time in milli seconds)	HT (time in milli seconds) with HashFunction1	HT (time in milli seconds) with HashFunction2	HT (time in milli seconds) with HashFunction3
be				
and				
sea				
the				
four				
baby				

age				
toward				
property				
shoulder				
	Total:	Total:		

Search time for queries on DS3

Query	BST (time in milli seconds)	HT (time in milli seconds) with HashFunction1	HT (time in milli seconds) with HashFunction2	HT (time in milli seconds) with HashFunction3
be				
and				
sea				
the				
four				
baby				
age				
toward				
property				
shoulder				
	Total:	Total:		

4. Report your results

Write a short report covering the following. Students are expected to use the IEEE Conference Template to write their report.

I. Introduction: In this section provide some background about Big Oh run-time analysis methodology. Explain why you would like to evaluate the run-time of BST and HT algorithms. Briefly write about the contribution of your work.

II. Experimental Evaluation and Results: Conduct the experiments as stated above.

III. Run time Analysis and discussion: Write a few paragraphs supported by visualization (graphs, diagrams etc) to analyze and discuss your results. Answering the following can help write a good quality report:

Which data structure works faster for the smallest dataset? Why? Consider the Big-Oh runtime for both insert algorithms to provide your answer.
Which data structure works faster for the largest dataset? Why? Consider the Big-Oh runtime for both insert algorithms to provide your answer.
For DS1, which keyword search query resulted in the largest time? Argue why was it largest for BST and/or HT?
For DS2, which keyword search query resulted in the largest time? Argue why was it largest for BST and/or HT?
For DS3, which keyword search query resulted in the largest time? Argue why was it largest for BST and/or HT?
Determine the keyword at the root of the BST when you run ds1. What time would it take to search this keyword? Why is it fast?
Compare your results with those of your group mate. For any of these data structures and algorithm, did you notice the impact of the computers processing speed? Why or why not?
Compare your results with those of your group mate. Are the results similar for the same keyword(s) you searched in ds1, ds2 and ds3? Why or why not?

IV. Compute the power usage on your computer during this experimentation

The energy E is measured in kilowatt-hours (kWh) per day. This is equal to the power P in watts (W) times number of usage hours per day *t* divided by 1000 watts per kilowatt, given as:

$$E = \frac{P * t}{1000}$$

Where E the energy, P is the power, t is the time in hours. Compute the total energy cost of running this experimentation for BST and HT.

Note: Be sure to convert seconds to hours before computing E.

Data Structure	Total time (hours)	Energy Consumption (kWh)
BST – ds1		
BST – ds2		
BST – ds3		
HT – ds1		
HT – ds2		
HT – ds3		

V. Conclusions

Write a few words drawing conclusions from this experimental study. Based on this evaluation, write about the best data-structure and search algorithm that performs better and consume less energy.

Evaluation

You are allowed to work as a group with maximum 2 members in a group. Your work's evaluation would be based on Code inspection and successful execution of test cases. The instructor reserves the right to determine the scores of each test case.

Upload 4 files to LMS as follows:

- YourID.jar
- Yourids_code.zip (contains all IDE project files… java, jar etc; **no class files please**)
- Yourids_Report.docx (or pdf file containing your report)
- Yourids_Data.docx (File containing evaluation data.)

Code Inspection:
The code would be inspected by the instructor. The instructor would determine the score to be given for code inspection. Generally, a readable code (indentation, clear scope definition) is required. For this project, there are no limitations on time and memory usage.

Instructor reserves the right to use appropriate tools to detect plagiarism. If the similarity of your submission is more than 30%, you will be awarded a ZERO in the project.

Submission Dead-Line:

The submission deadline is final. Late Submissions will be awarded ZERO points.

Important Notes:

- It is the student's responsibility to check/test/verify/debug the code before submission.
- It is the student's responsibility to check/test/verify all submitted work (including jar files)

- It is the student's responsibility to verify that all files have been uploaded to the LMS.
- After an assignment/project has been graded, re-submission with an intention to improve an assignments scores will not be allowed.
- After the assignment/project has been graded, the instructor will post test-cases used for grading on the website.
- The Instructor has the right to share project execution reports that may have been auto-generated on the course website.

Useful resources

Tutorials on using command line arguments

Reading all files in a folder

Grading Rubrics

The grading for this project is relative (Not absolute), click on the link to watch/read more about relative grading. To assess your work, we will use the following grading rubrics:

Item	Good	Fair	Poor
Overall Report quality	A comprehensive report providing a thorough introduction, along with appropriate references.	Fairly written report providing an introduction with some references	Minimal work or no submission.
Downloaded and generated datasets	A good variation of files selected for datasets	Only 5 files selected	Less than 5 files selected
Recording experimental data	Complete experimental documentation populating all tables	Not all experimental work is completed	Only a few experiments completed (or no submission)
Visual	Used, graphs charts etc. to highlight the results	Few charts/graphs added	No visual representation of data
Discussion of results	A thorough run-time analysis of data structures and algorithms used in the study with appropriate references. Detailed discussion analyzing the results obtained from the experimental study. Use of Graphs, plots, charts etc in a meaningful way.	Listed answers to the provided questions only. Few visual aids shown. Material copied from external resource with minimal effort in writing the report.	Minimal or no work submitted. Material copied from external resource without due diligence (plagiarism).
Use of IEEE Template	Used the IEEE Conference paper format.	Not used IEEE template, but report is well formatted.	Poor formatting / no work submitted.