Programming assignment

Part A

In this part, your job will be to implement various forms of tree data structures and various related algorithms. You will create a binary search tree.

Overview 1) Your program should be set up to receive two command line arguments. These two arguments will specify two input files for your program. Both input files will match the format of starwars.txt. The two arguments will represent the following:

○ A data file with names to store.

○ A data file with names to search for. (We will be testing your program with our own test files. Make sure your program can handle input files of any length.)

2) Your program will store the data from the first file into various data structures and then read the data from the second file and search for each name in it in your previously built structures. For duplicates, report the first hit.

Binary search tree 1) Create a binary search tree class. Again, this will be of an explicit representation. 2) Again, insert all of the names in the first array into your search tree and report the average time it takes to do so. 3) Print out the contents of the tree with an inorder traversal. 4) Again, print the minimum and maximum values stored in the tree and output the time taken to find each. 5) Again, search for each name from the second array and report: ○ The depth of the node. (The root node has a depth of 0). ○ Whether the node is a leaf node or not. ○ The size of the subtree where the name was found. ○ The time it took to find the name. 6) Make sure to account for when names are not found. For duplicates, report the first hit

Part B (AVL Tree)

Each line of data\_main.csv represents one hard drive test. Create a class HDTestData whose instance can contain all the data from a single test.

It should contain:

• The hard drive’s serial number (which is actually a string)

• The hard drive’s model

• The hard drive’s capacity

• The hard drive’s total “powered on hours”

Implementation

1) Start off by reading in data from “data\_tiny.csv” and store them in an array of HDTestData objects

2) Create an AVLNode class that at least keeps track of leftChild, rightChild, height, and key. You may also need an attribute to store a pointer to the data itself.

3) Create an AVLTree class that implements insert, delete, and search. Each of which can be made up of multiple methods if needed. ○ Your trees (and nodes) must be implemented with generics. I.e. it can hold any kind of data type.

4) Create 4 AVLTrees that each sort their contents by a different column of the data file (namely, serial\_number, model, capacity\_bytes, and power\_on\_hours).

5) Print out each of the HDTestData objects and their node's height in each tree in breadth first order and depth first order. (Implemented using queues and stacks (or recursion), respectively.)

Testing

1) Read the data of "data\_main.csv" and store it in a new array of HDTestData objects.

2) Once you have read in the file, randomly choose n objects from the array.

3) Insert each of your chosen test data objects into each of your 4 trees. Time your inserts and print the average insertion time.

4) Select m random HDTestData objects from your array and find the first occurrence of each of its attributes in their respective trees. Print the full HDTestData information and the node’s height for each occurance you find.

5) Calculate and print the average time it takes to search in each tree.

6) Delete each object in your tree one by one. Time each deletion and print the average time.

7) Compare the time it takes to insert, search, and delete. The values for “n” should be as large as possible. But at least run your program for n = 10, 100, 250, 500, 1000, 10000, and 100000. Print trees (i.e., item A5 above) only for n=10, 100 and 250.

Note: This PA is time consuming start early.