Programming Assignment 1
Points: 500
Due: Oct 26, 11:59PM
Late Submission Due: Oct 27, 11:59PM (25% penalty)

Description of a programming assignment is not a linear narrative and may require multiple readings before things start to make sense. You are encouraged to start working on the assignment as soon as possible. If you start a day or two before the deadline, it is highly unlikely that you will be able to come with correct and efficient programs.

You are encouraged to consult instructor/Teaching Assistants for any questions/clarifications regarding the assignment. Your programs must be in Java, preferably Java 8.1. You should not use any external libraries. Any libraries that you import must start with `java.`

For this PA, you **may work in teams of 2**. It is your responsibility to find a team member. If you can not find a team member, then you must work on your own.

In this programming assignment you will

- Implement a Binary Search Tree that can store strings.
- Implement the war story from the text using various data structures—Arrays, BST, Hashing/Rollover Hashing.

You will design following classes:

- BinaryST
- WarWithArray
- WarWithBST
- WarWithHash
- WarWithRollHash

All your classes must be in the **default package** (even though it is not a good programming practice).

1 BinaryST

This class will implement a (variant of) binary search tree that can store a *multi-set* of `strings` and supports the queries described below. Recall that a string may appear more than once in a multi-set.

For the purpose of this PA, you may assume that all the strings consists of only **upper case alphabet characters**. I.e., none of the strings have numeric values, special characters, lower case letters, punctuation symbols etc. You **must** use the dictionary order when comparing strings: With A less than B less than C and so on. For example $AZDTF$ is smaller than BE.

This class will have following public methods and constructor.

`BinaryST()` Creates an empty Binary Search Tree.

`BinaryST(String[] s).` Creates a Binary Search Tree obtained by adding the elements **in the order they appear** in the array s. You **must not** balance the tree. If a string appears ℓ times in the array, your BST must store this information.

`distinctSize()` Returns the number of *distinct* strings stored in the Tree. If you have added "AB", "CD", "AB", then this method returns 2.

`size()` Returns the total number of elements stored in tree. If you have added "AB", "CD", "AB", then this method returns 3.

`height()` returns the current height of the tree. Height of a tree with a single node is **one**. Height of an empty tree is **zero**.

`add(String s).` Adds the string s to the BST. Even if s already appears in the tree, it must add s.

`search(String s)` Returns `true` is s appears in the tree; otherwise returns `false`.

`frequency(String s)` Returns the number of times s appears in the tree.

`remove(String s)` If s appears in the tree, then removes the string s from the tree and returns `true`. If s does not appear in the tree, then returns `false`. If s appears, more than once then remove only one occurrence.

`inOrder()` Returns an `array of Strings` obtained by doing an in-order traversal of the tree.

`preOrder()` Returns an `array of Strings` obtained by doing an pre-order traversal of the tree.

`rankOf(String s)` Returns number of strings that are smaller than s.

Runtime of all you methods must me $O(h)$ (except `inOrder` and `preOrder`), where h is the height of the tree.

2 Skiena's War Story

There is an unknown DNA sequence U about which you would like to find some information. You are given a set S of n strings, and each string has length exactly k such that S is the set of all

k-length substrings of U. Now your goal is to construct a set T such that T is the set of all $2k$-length substrings of U. Please see Section 3.9 for more on this problem. Note that U is not given as input. Input is S: A set of k-length substrings of U and output is T—set of all $2k$-length substrings of U.

2.1 WarWithArray

This class will contain a method to return T. This class uses array to store S. You must not use any other data structure to store S. This class will have following constructor and method.

`WarWithArray(String[] s, int k ).` s is the set of all k-length substrings of U.

`compute2k().` This method returns an arraylist of $2k$-length strings which is the set of all $2k$-length substrings of U. Return type of this method must be `ArrayList<String>` .

2.2 WarWithBST

This class will contain a method to return T. This class uses BST to store S. You must not use any other data structure to store S. This class will have following constructor and method.

`WarWithBST(String[] s, int k ).` s is the set of all k-length substrings of U. In the constructor, you **must create an instance of** `BinaryST(s)` so that it can be used in `compute2k()`.

`compute2k().` This method returns an arraylist of $2k$-length strings which is the set of all $2k$-length substrings of U. Return type of this method must be `ArrayList<String>` .

2.3 WarWithHash

This class will contain a method to return T. This class uses hash sets/hash tables to store S. You must not use any other data structure to store S. This class will have following constructor and method.

`WarWithBST(String[] s, int k ).` s is the set of all k-length substrings of U. In the constructor, you **must store** s **in a hash table/hash set** so that it can be used in `compute2k()`. You may use Java's inbuilt classes `HashSet` or `HashTable`

`compute2k().` This method returns an arraylist of $2k$-length strings which is the set of all $2k$-length substrings of U. Return type of this method must be `ArrayList<String>` .

2.4 WarWithRollHash

This class will also contain a method to compute T. Here you should use the idea of roll-over-hashing and hash tables to return T. This class must have following constructor and method.

`WarWithBST(String[] s, int k ).` s is the set of all k-length substrings of U.

`compute2k().` This method returns an arraylist of $2k$-length strings which is the set of all $2k$-length substrings of U. Return type of this method must be `ArrayList<String>` . Judicious use of hashing and roll-over-hashing will make this method run faster than the corresponding method from `WarWithHash`.

3 Report

Write a brief report that includes the following.

- Pseudo code of method `compute2k()` from
  - WarWithArray
  - WarWithBST
  - WarWithHash
  - WarWithRollHash

  In addition, analyze the asymptotic run times of your algorithms. Express the run time as a function of k and n, where n is the size of the set S.

4 Guidelines

For this PA, you **may work in teams of 2**. It is your responsibility to find a team member. If you can not find a team member, then you must work on your own.

You are not allowed to use any external libraries. Your code **must strictly adhere to the specifications**. The names of methods and classes must be exactly as specified. The return types of the methods must be exactly as specified. For each method/constructor the types and order of parameters must be exactly as specified. Otherwise, you will lose a significant portion points (even if your program is correct).

Your grade will depend on *adherence to specifications*, *correctness of the methods/programs*, and *efficiency of the methods/programs*, `quality/correctness of the report`. If your programs do not compile, you will receive **zero** credit.

5 What to Submit

Your submission must have following files.

- BinaryST
- WarWithArray
- WarWithBST
- WarWithHash
- WarWithRollHash
- report.pdf (must be in pdf format)
- ReadMe.txt (list the name(s) of team member(s)).

You must include any additional helper classed that you designed. Please include only source .java files, do not include any .class files. Please remember to use default package for all the java classes. Place all the files that need to be submitted in a folder (without any sub-folders) and `zip` that folder. Submit the `.zip` file. Only `zip` files please. Name of the folder must be `YourNetID-PA1.zip`. Please include all team members names as a JavaDoc comment in each of the Java files. **Only one submission per team please.**