Writeup for third project of
CMSC 420: "Data Structures"
Section 0301, Spring 2018

Theme: Binary Patricia Tries

Handout date: Thursday, 04-05
On-time deadline: Thursday, 04-19, 12pm (noon)

Late deadline (30% penalty): Saturday, 04-21, 11:59pm (midnight)

1 Overview

In this programming project, you will implement **Binary Patricia Tries**. Binary Patricia Tries are Patricia Tries over the binary alphabet $\{0, 1\}$. The goal of the project is to construct and maintain Patricia tries with simple binary strings as input. All programming will be done in Java, and you will be graded automatically by the CS departments submit server. This write-up contains guidelines about what needs to be implemented as well as instructions on submitting your project.

2 Prerequisites

We expect that you have familiarity with both Tries and Patricia Tries. Review the class recordings and slides if necessary to make sure you are up to speed with these two excellent data structures.

3 Binary Patricia Tries

Your project involves implementing insertion, deletion, lookup and an inorder traversal for Binary Patricia tries, along with a few additional "accessor" methods. The lecture slides offer you a number of different ways that you can structure your inner node. One of these ways requires an integer, `splitInd` which essentially tells you the length of the common substring of **all string keys stored somewhere in the subtree rooted at the current node**. This method does **not** require a bit flag to tell us if the current node stores an actual String key that was inserted sometime in the past or whether it is a "splitter" node. Another method we have shown eschews that integer and instead stores only suffixes of the strings that "pass" through it. This latter method is more space-efficient when it comes to the stored strings, but it **does** require a bit flag in every node. The choice is yours! The API **does not care about how you structure the nodes**! It only wants you to be able to insert, search, delete, find, generate inorder traversal, and query about the longest string stored in the trie, its size (# stored keys) and whether it is empty.

4 Starter code & docs

Everything you need to get started is available in our common Git repository. You will need to fill-in the implementation of the class `BinaryPatriciaTrie`. The class comes with sufficient documentation that you will be able to find under the directory `doc`, so that you can have a full view of the functionality exposed by the class' `public` interface.

Some points about the implementation:

- The input type for a key is a `java.lang.String`. Specifically, in our case, it will be a binary string, i.e it will contain only **'0'**s and **'1'**s. Example binary strings are "0101010", "1101" and "0". Since the input type is known at compile-time, this class, unlike the ones that you dealt with in your first project, does **not have to be a generic!** Dont go searching for any unknown types `T` in the skeleton code we have provided for you! You wont find any :)
- Recall that it is **absolutely possible** that the trie may contain a **key** that is a **prefix** of **other keys**! In this case, the nodes that contain these "prefix keys" are **non-leaf nodes**. Do not assume that keys are located only in the leaves of the trie, like in B+-trees!
- For this project, we assume that there are **no duplicate keys** in your data structure. This means that, in our unit tests, **whenever we delete a key from your tree, we expect it to no longer be found in the tree**.
- We do **not** test you for the insertion, deletion or search of either the `null` reference **or** the empty string (`""`). However, please note that the method `getLongest` should return an empty string (`""`) if the trie is empty! This is also specified in the class' `JavaDoc`.

5 Submission / Grading

Credit in this project is defined by the number of submit server unit tests that you pass.

Projects in this class are different from your typical 131/2 projects in that **we do not maintain an Eclipse - accessed CVS repository** for you or us. This means that you can **no longer use the Eclipse Course Management Plugin** to submit your project on the submit server. This turns out to be a good thing, since it frees you up from the need to use Eclipse if you don't want to.

To submit your project, run the script `src/Archiver.java` as a **Java application** from your IDE (tested with Eclipse and IntelliJ in Mac and Windows). This will create a .zip file of your **entire project directory** at **the same filesystem level of that directory, without** including the hidden `git` directory `.git`. This directory can sometimes be very large and cause problems with uploads on `submit.cs`.

For example, if your project directory is under `/home/users/me/mycode/project1/`, this script will create the .zip archive `/home/users/me/ mycode/project1.zip`, which will contain `src`, `doc`, and any other directories that you may have, but will **not** contain the directory `.git`. After you have done this, upload the archive on the submit server as seen on figure 1.

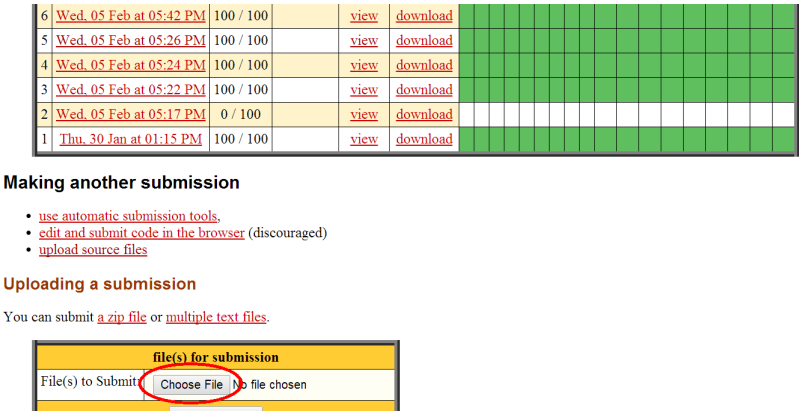

Figure 1: Uploading your project on the submit server.

All tests are release tests, and you can submit **up to 5 times** every 24 hours. **We urge you to unit-test your code thoroughly before submitting**: treat every token like a bar of gold that is not to be wasted! We will **not** share the source code of the unit tests with you, not even after the deadline for the project!

We maintain your **highest-scoring submission** for grading purposes. Finally, for the late deadline, we take 30% off your maximum possible score. This means that, if you submit late, passing all the unit tests will give you 70% of the total grade.

Finally, we should remind you that for the past few years the Software Similarity Detection System MoSS has been incorporated into the CS department's submit server. For n student submissions, it is **ridiculously easy** (literally a single click) for us to run MoSS against all $\binom{n}{2}$ pairs of submissions. MoSS is tuned towards higher than 50% Recall, which means that plagiarized submissions **will** be caught. We would much rather be spending time teaching you data structures and assisting you with your queries than going back and forth with the Honor Council; **help us help you**!