Homework 3
Due **Monday Sep 21 at 11:59pm EDT**: on coursesite

CSE 297: Fall 2020

1 General Description

This assignment is the first part of a multi-part project to build some components of a simplified blockchain. We'll omit details of transactions or accounts that exist in real blockchains and instead store a collection of strings in the each block. If you design your code well, you can upgrade from strings later on, but we won't do that in this assignment. This assignment focuses on the Merkle/Patricia-tree structure. Future assignments will cover creation of a block, adding them to a chain, validation of a block, and lookup of specific information by a light node.

The goal of this assignment is to construct a Merkle/Patricia tree over a given set of strings from an input file. Since this you are building a Patricia tree, the strings will need to be sorted by your code (we are not assuming the input file is sorted). Below, I shall describe node format, explain how to get a Java implementation of SHA-256, and provide a few more specifics. Although my discussion here is in terms of Java, you are free to use other languages, but you will have to provide me a simple "turnkey" way to run your code, collect source code for Moss, and comply with my specified input and output formats. I shall be using the sunlab as my test environment so that we have a common, standard platform. You may develop on your own machines, but in the end, you'l need to test in the sunlab. Obviously, this means that you are restricted to use programming languages supported in the default sunlab environment.

The project will be done in groups (see Section 5). The groups will persist for future assignments.

I realize that coding experience among students in this course varies and that, due to many students working remotely, team management may be more challenging. Thus, I shall be generous in extending deadlines, but note that my listed deadline for this and future assignments is designed to keep you on a good overall pace. Don't take advantage of my covid-based flexibility unless you really need it.

2 Details

2.1 Input

You shall prompt for a file name containing the input. The format of the input file is plain text (i.e. a .txt format file). We plan to give all test files a .txt extension but your code should be robust to other file names. The input file will contain strings: one string per line. The string on each line may contain spaces, tabs, and special characters (but you do not need to worry about escaped newline characters). Strings will have an upper bound of 100 characters (not counting the newline character at the end of each input line as that is not part of the input string). It is up to you to choose fixed length or variable length strings, and your language choice may have some impact on your decision.

As noted, the input will need to be sorted. It is fine to use any built-in sort utility you want.

There will be no test data provided since it is easy for you to generate your own test data.

2.2 Tree Structure

Nodes are of two types:

- **leaf nodes**: Two components:
  1. a string
  2. the SHA-256 hash of the string
- **nonleaf nodes**: For nonleaf nodes, we encode not only the hashes and children, but also the Patricia-tree labels on the edges to children.
  1. left-child: a node identifier in a format that fits your chosen language; for Java this is an object-reference
  2. label for left-child edge: a string. Since this string is a prefix of 100-character strings, it will generally be a lot less than 100 characters. You may waste all that space or go with variable-length nodes as you prefer.
  3. the SHA-256 hash of the SHA-256 hashes of the two children
  4. label for the right-child edge
  5. right-child

The tree itself is referenced using the node identifier of the root.

Your design of the tree structure beyond this level of detail is up to you. But a few words of warning are in order here. If you are using Java, object-references are a good idea since that will allow you to use Java serialization (covered in the Liang text used in CSE 017 even though this topic not covered in CSE 017) to write your tree out to a file and then read it back in. While that may not seem necessary for this assignment, you will need to be able to do that to write blocks out to a file later. If you are coding in C and using pointers, note that those pointers point to memory locations, and won't be meaningful if you write them to disk.

2.3 SHA-256

You are free to use existing implementations of SHA-256. Credit your source in a comment (if it is not obvious as it would be in Java). If you are using Java, you will find a SHA-256 implementation in the MessageDigest class. Import "java.security.MessageDigest" and instantiate a MessageDIgest object as follows:

```
MessageDigest messd = MessageDigest.getInstance("SHA-256");
```

More details can be found online, for example `https://www.geeksforgeeks.org/sha-256-hash-in-java/`

For other languages, you will need to either write your own SHA-256 function or find one online. If you use an online source, be sure to cite that source in a comment at the start of your code.

2.4 Printing the tree

To evaluate this assignment in a standardized way, we'll need code that provides a standard way of specifying the input file and the print output file. Your code must prompt for the file name (which will be X.txt for some X) and then produce an output file whose name is derived from the input file name by placing ".out" before the ".txt" as in X.out.txt. This output file must also be plain text.

Note that this print output is distinct from serialization of your language-specific tree representation to a file for future input to other code. The only thing that inputs the print output is a human.

This output must be in the format specified below for ease of evaluation.

- The print output of every node, including leaves, begins with a human readable node print-ID (not some address out of a pointer or object-ID). These print-IDs are constructed by labeling the root as 1, the left child of the root as 2, the right child of the root as 3, etc. More generally, if a nonroot node is the left child of a node with printID p, its printID is $2p$ and, if it is the right child, its printID is $2p + 1$. (This corresponds to the array representation of a tree used in CSE 340.)

- Nodes are to be output in printID order, with two blank lines separating each node in the output.
- A node is to be printed by first printing its printID on its own line. Then each field in printed in the order listed above in Section 2.2. Node contents are to be printed one field per line, with child pointers represented by their printIDs, edge labels represented by the corresponding string, and hashes by the 256-bit hash shown as 32 hex digits, using lower case for the letters in the hex value.
- There are to be NO blank lines within a node's print output

3 Submitting the Code

The code is to be submitted in a single zip file named team.zip, where "team" is the name you registered for your team (see Section 5). The unzipped file needs to have at the top level the following:

- A file named "run", which should have execute permission and runs your code. Most likely this file will be a bash shell script that invokes your sunlab-compiled executable. Note that it is necessary that final compilation and final testing be done in the sunlab so that we do not have version or machine/OS compatibility issues in testing your code. Your code will be run on a standard-issue student account with no changes to any environment variables.
- A file named "recompile", which will recompile your code. It needs to have execute permission. This can work anyway you wish (e.g. a makefile).
- A file named "allcode.X", where X is the default file extension for your language (e.g., ".c" or ".java"). This one file needs to contain ALL your code. It does not have to compile properly if you are actually storing your code in multiple files. The sole purpose of this file is for sending code to Moss for similarity testing. Just "cat" all your code up into one file so we do not have to write a script to search out the code in each student's directory structure for each possible language.
- A directory named "code" containing all other code or data you may need to make your run and recompile files work.

4 Honor Code and Getting Help

Your team's code for the Merkle/Particia tree must be that of your team and no one else. No code sharing is permitted between teams. While you may review websites for educational purposes (such as how to use built-in SHA-256 functions), you must not copy any actual code (except for, possibly, a SHA-256 implementation) from any source other than your own team. Any violations detected in the grading process are subject to being submitted to the university disciplinary process.

However, you are free to seek help in the use of the sunlab environment to construct the simple "run" and "recompile" bash scripts, to seek help in file transfer to/from the sunlab, and to seek help in the use of Linux utilities. (My use of the word "simple" is intended to rule out such things as actually sharing a fully coded Merkle/Patricia tree as a bash script.)

Be careful to avoid being an unintentional honor-code violator. If you store your code in a publicly readable location, your code may be copied without your knowledge. If the copier claims innocence you may be found partially guilty of violating the honor code. Therefore, make sure your file protection in the sunlab is set properly. If you use github or some other cloud service, be sure that your repository is private. (You may make it public after the course is over if you wish.)

5 Teams

I would like teams to consist of 3 or 4 students. Because of the various challenges in arranging collaboration (location, time zone, etc.) I shall let you form your own teams. See Piazza for details on the location of a shared Google Sheet in which you can enter your team's name and members. Feel free to post to Piazza to advertise for team members.

I realize that team member skills may vary and that grading by team can be somewhat unfair. I shall seek out input from each student individually on that student's view of teammate contribution at the end of the course. If issues arise during the course (such as a team member suffering a lengthy illness or simply not producing), please let me know privately via email so we can work out an appropriate path forward. I shall expect each team member to be fluent in the entire codebase, so although you will divide coding responsibilities, you should do full-team code reviews.

These teams will persist for the entire course because the code from this assignment will provide some of the input to the next and subsequent ones. With that in mind, be particularly careful about backing up your files and applying good version-control discipline.