COMP 2140 Assignment 5: Merkle Trees

Cuneyt Akcora

Due: Monday, March 6th, 2020 at 09:00 p.m.

Hand-in Instructions

Go to COMP2140 in UM Learn; then, under "Assessments" in the navigation bar at the top, click on "Assignments". You will find an assignment folder called "Assignment 5". Click the link and follow the instructions. Please note the following:

- Submit the following java files: Block.java, Blockchain.java,MerkleTree.java, PeerToPeerNetwork.java, Transaction.java, UtilityFunctions.java. The Blockchain.java file must contain the small report (in the comments at the end of the file). The Blockchain.java file must be renamed `A5<your last name><your first name>.java` (e.g., `A5AkcoraCuneyt.java`).
- Please do not submit anything else.
- We only accept homework submissions via UM Learn. Please DO NOT try to email your homework to the instructors or TAs or markers — it will not be accepted.
- We reserve the right to refuse to grade the homework or to deduct marks if you do not follow instructions.
- **Assignments become late immediately after the posted due date and time.** Late assignments will be accepted up to 49 hours after that time, at a penalty of 2% per hour or portion thereof. After 49 hours, an assignment is worth 0 marks and will no longer be accepted.

How to Get Help

Your course instructor is helpful: For our office hours and email addresses, see the course website on UM Learn (on the right side of the front page).

For email, please remember to put "[COMP2140]" in the subject and add a meaningful phrase after that to the subject, and to send from your UofM email account.

Course discussion groups on UM Learn: A discussion group for this assignment is available in the COMP 2140 course site on UM Learn (click on "Discussions" under "Communications"). Post questions and comments related to Assignment 1 there, and we will respond. We will also post questions related to the assignment that we receive by email, and answer them there. Please do not post solutions, not even snippets of solutions there, or anywhere else. **We expect you to read the assignment discussion group.**

Computer science help centre: The staff in the help centre can help you (but not give you assignment solutions!). See their website at http://www.cs.umanitoba.ca/undergraduate/computer-science-help-centre.php for location and hours. You can also email them at helpctr@cs.umanitoba.ca.

Programming Standards

When writing code for this course, **follow the programming standards**, available on this course's website on UM Learn. Failure to do so will result in the loss of marks.

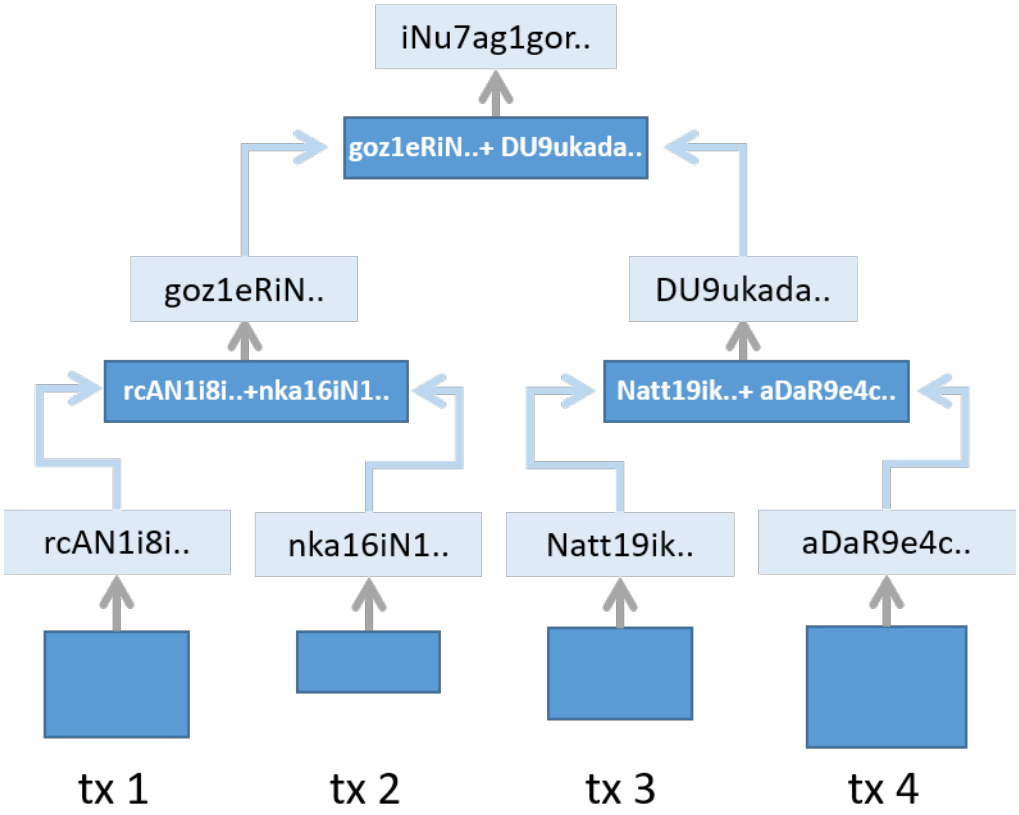

Figure 1: A Merkle tree

Question

Remember: You will need to read this assignment many times to understand all the details of the program you need to write.

Goal: The purpose of this assignment is to write a Java program that implements a MerkleTree to be used in a blockchain. Then you will write a brief report describing your results, in the comments of your program; see the end of this document.

Code you can use: You are permitted to use and modify the code your instructor gave you in class for the various sorting algorithms, and code from Labs. Of course, you must use the code we provided to you below. You are NOT permitted to use code from any other source, You are NOT permitted to show or give your code to any other student. Any other code you need for this assignment, you must write for yourself. We encourage you to write ALL the code for this assignment yourself, based on your understanding of the algorithms — you will learn the material much better if you write the code yourself.

What you need to know about Blockchain:

1. This assignment is self-contained. We do not expect you to know how Blockchain works. However, by the end of this assignment, if you read the code well enough, you will learn Blockchain in detail.

2. You will be surprised how this Blockchain technology, considered revolutionary, is built on such simple ideas that a beginner student can code. And later in life you can mention that you were writing blockchains in your second year of undergraduate.

3. A transaction is a transfer of coins from a set of addresses (i.e., senders) to another set of addresses

(i.e., receivers).

4. Each sender or receiver is an address, which is stored in a String variable (see `Transaction.java`). In real life, people can create and use many addresses. We cannot know the owner of an address by just looking at the string.

5. A Bitcoin transaction can have many senders and receivers. This is why transactions in Figure 1 have different shape sizes. For this assignment, we simplify the transaction model to only one sender and one receiver (see `Transaction.java`).

6. A transaction has sender, receiver and amount attributes. In Bitcoin, transactions are created by ordinary users and sent to a Peer to Peer network. Miners listen to the network, discover new transactions and create blocks out of them. In this assignment, we will not have a real Peer to Peer network. The `PeerToPeerNetwork.java` simulates this, and returns a random number of artificial transactions whenever someone calls the `collectNewTransactions()` function.

7. A miner is a user (anyone can choose to be a miner) who wants to create a block. The process of getting transactions, putting them into a block and solving the Proof-of-Work puzzle is called *mining a block*.

8. Proof-of-Work (see `mineTheBlock()` in `Blockchain.java`) involves creating a string from the blockHash of the previous block, topHash of the MerkleTree, and a long integer (called nonce). Once the SHA256 hash is applied to this string, a 256 bit integer is computed. If the integer is less than a predefined difficulty, the nonce is said to satisfy the difficulty. The block is said to be mined.

9. Any helper function that you need (e.g., to hash SHA256) is already given in the files.

What you should implement: Implement the following algorithms and methods (you can add any necessary `private` helper methods):

1. `buildFrom` **function in MerkleTree.java:** Concerns and steps:

   - Implement the algorithm that takes n transactions and creates a Merkle tree from them. A Merkle tree is a binary tree where leaf nodes are transactions, and interior nodes are transaction hashes (SHA256 algorithm). See figure 1.
   - In leaf nodes, we take a SHA256 of each transaction, and then concatenate these hashes in the next level, and take their hash again. For example, in the figure the hash "goz1erin..." is computed by SHA256(SHA256(tx1.toString())+sha256(tx2.toString())). This is applied until we end up with one top hash in the Merkle tree.
   - At every level (from bottom up), output how many hashes are computed at each level:
     - Merkle Tree, Bottom Up, Level: 0, number of hashes: 21
     - Merkle Tree, Bottom Up, Level: 1, number of hashes: 11
     - Merkle Tree, Bottom Up, Level: 2, number of hashes: 6
     - Merkle Tree, Bottom Up, Level: 3, number of hashes: 3
     - Merkle Tree, Bottom Up, Level: 4, number of hashes: 2
     - Merkle Tree, Bottom Up, Level: 5, number of hashes: 1
     - See the Assignment5Output.txt file for output details.
   - Note that by definition, SHA256(tx1.toString() +SHA256(tx2.toString()) is not equal to SHA256(tx2.toString()+ SHA256(tx1.toString()). When you are creating the Merkle tree, the transaction order is important. You should follow the transaction order defined in the block.

- A blockchain does not need to store the Merkle tree, it computes the Merkle tree just to find the top hash, which is "iNu7ag1gor..." in the figure. As a result your code can either i) implement the Merkle tree and store each interior node to reach the top hash, or ii) find the tophash and not store interior nodes.
- 1st solution: When implementing MerkleTree with nodes, child pointers need to point to parents, because you should build the tree from bottom up, starting from transactions.
- 2nd solution: If you want to use the second approach, be efficient. Code can be written in 20 lines.
- The current block hash is computed from SHA256(previousBlockHash+ topHash+nonce). This way, the block hash of the current block is linked to the block hash of the previous block. If anyone changes the previous block, its hash will change. We will call this event a corruption.

2. `validate()` **in** `Blockchain.java` **A method to validate the blockchain:**
   - The validation starts from the second block. The first block (genesis block) is mined by the creator of the Bitcoin: Satoshi Nakamoto.
   - Take the list of transactions stored in a block, and create a Merkle tree from them again (use the existing BuildFrom() function in `MerkleTree.java`) to find the top hash.
   - Link the previous block and nonce (follow the order given in mineTheBlock() function in `Blockchain.java`. Compute the Block hash and compare it to the block hash stored in the block. if they do not match (use string1.equals(string2)), call it a corruption, and return.
3. Optional 1: write code to update the difficulty every 2 weeks. For this, you can run the blockchain in a while(true) loop.
4. Optional 2: write code to update block reward every 4 years.
5. A log file for sample output will be shared on D2L. The output of your code must follow similar steps.
6. See the discussion forum on D2L for your questions.

Concerns. Please avoid magical numbers in your code. There should be only ONE return per method. See the programming standards (under content/course documents) file on UMLearn, and follow the suggestions. Assignments will be graded by considering these standards. Your code must not give any warnings or errors when compiled and run.

Report

Write a small report in comments at the end of your program.

Answer the following questions in terms of transaction count n. (use short sentences, ideally only one sentence):

1. What is the depth of the Merkle tree in a block.
2. If we want to detect corruption at the transaction level, how many comparisons should we make in the Merkle tree.
3. Is there a way to corrupt any part of the blockchain without being detected?