IT486: Blockchains and Cryptocurrencies **Autumn 2022-23**
Homework 2 (Due via classroom: December 2, 23:50)

Instructions. This is an individual assignment. All work should be your own except for the referenced code for the hash function. Use *python* as the programming language.

1 Building Merkle hash trees

It is quite easy to implement the Merkle hash tree functionality using `openssl` tool. All you need is to organize the different calls to the corresponding hash function. Indeed, you only need two types of hash computations:

- Compute the hash of a document doc3.dat with something like

  ```
  $ openssl dgst -sha1 -in doc3.dat -binary > node0.3
  ```

- Aggregate two existing hashes node2.4, node2.5 into one single value node3.2 with

  ```
  $ cat node2.4 node2.5 | openssl dgst -sha1 -binary > node3.2
  ```

In the above examples the following labelling convention is taken:

- Documents to be hashed are named doc0.dat, doc1.dat, ..., doc(n-1).dat, assuming there are n documents in total
- Tree nodes at level i (level 0 corresponds to the leaves) are named nodei.j, where the hash contained in nodei.j is computed from the nodes node(i-1).(2j) and node(i-1).(2j+1)
- Node nodei.j is computed only if the node node(i-1).(2j) exists
- If node(i-1).(2j+1) does not exist, then it is removed from the hash computation

Each tree layer has about half of the nodes than the previous layer. The root node is obtained when a layer has only one node, and this is the hash value of the collection of documents.
You can add a new node to the tree by just putting it in docn.dat and updating the corresponding nodes (including the root). When n is an exact power of 2, then you will need to add a new layer into the tree with a new root, along with some new nodes from it to the new leave. For instance, if a fifth document doc4.dat is added to the Merkle tree of four documents, then the old root is node2.0 and the root of the new tree is node3.0, and the new nodes node2.1, node1.2 and node0.4 are added to the tree.

One interesting feature of Merkle hash trees is that the creator stores the whole tree but she only sends the root node. Then at some point she can convince anyone that a certain document (e.g., doc3.dat) is in the collection at a specific position (position 3, in the example) by sending a limited number of node values (the necessary nodes to allow the computation of all the intermediate hashes from the document itself to the root: node0.2 and node1.0 in the example).

2 What you need to do

In this assignment you will implement the tree building, the generation of the proof of membership, and the verification of the proof of membership.

2.1 Task 1

Start from a list of n arbitrary nodes, and name them doc0.dat, doc1.dat, ...

Write a python code that on input n it computes all hashes and stores them into the files named nodei.j for all suitable values of i and j. Remember that you would need to deal with non-existent nodes (e.g., node(i-1).(2j) exists but node(i-1).(2j+1) does not).

Create an tree.txt file with one line per node, containing in each line the values of i, j and the node hash (i.e., the contents of nodei.j in hexadecimal). The file can have a header (e.g., the first line) containing the hash algorithm used (e.g. sha1), the number of documents (decimal number), the depth of the tree (the number of layers including the leaves and the root, as a decimal number) and the root hash (the contents of the tree root in hexadecimal):

```
MerkleTree:sha1:12:5:3f786850e387550fdab836ed7e6dc881de23001b
0:0:89e6c98d92887913cadf06b2adb97f26cde4849b
0:1:2b66fd261ee5c6cfc8de7fa466bab600bcfe4f69
...
4:0:3f786850e387550fdab836ed7e6dc881de23001b
```

The first line is the public information of the Merkle tree, and the remaining lines give the private information that allows updating the tree and producing proofs of document membership.

2.2 Task 2

Write a python code that produces a proof of membership for a given document at position k. The proof consists of a list of nodes that allows the verifier to recompute the hashes in the path from the k-th leaf to the root and check that the root value is the same as the given one (the verifier was given in advance the public information of the tree). The proof will be the corresponding lines of the Merkle tree text file description, store the selected lines in another file proof.txt.

2.3 Task 3

Write a python code that verifies the proof given the public information of the Merkle tree (hash of root), the document (doc.dat) and the proof (text file with the necessary nodes: proof.txt).

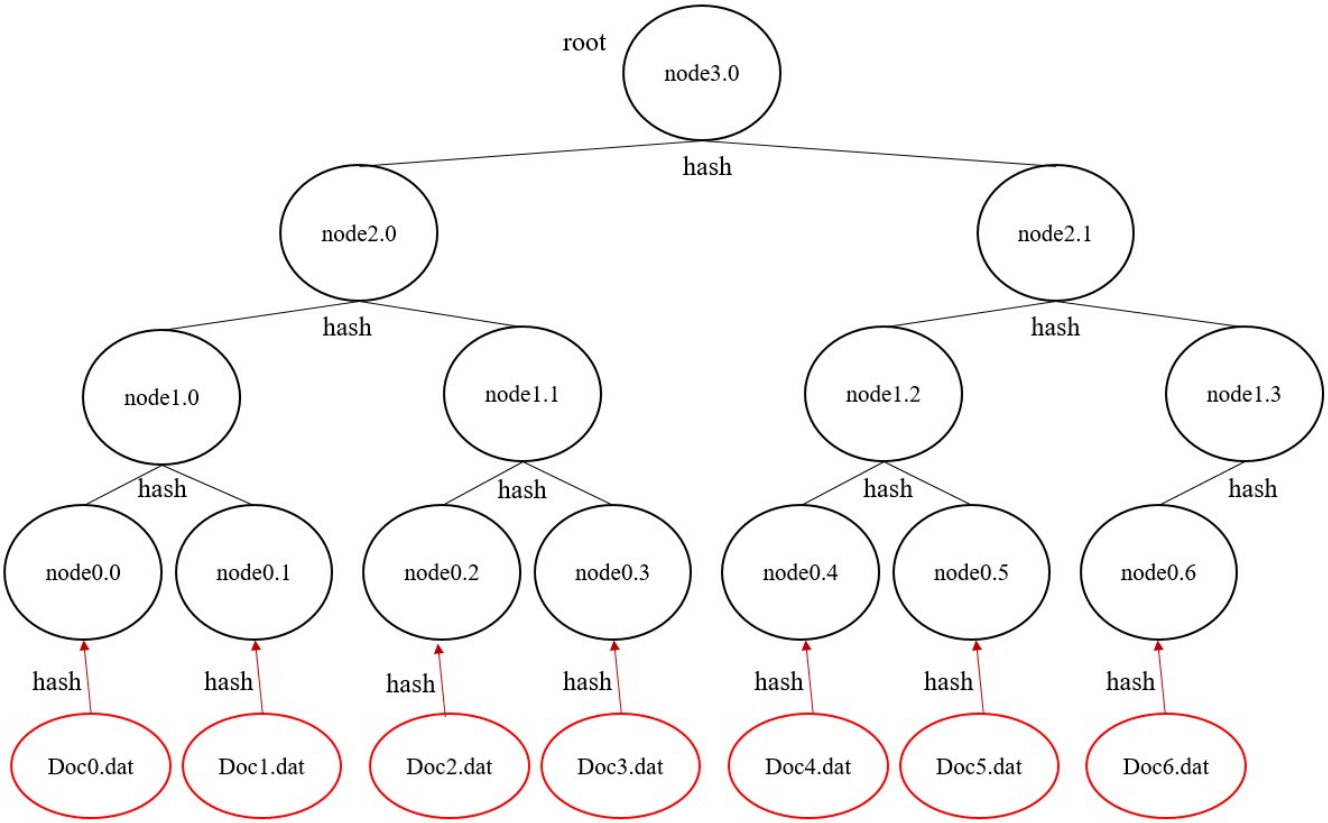
3 Example

If the input size is 7, then you create 7 documents with Doc0.dat naming convention and store them in a directory. The first program will take the directory name as input and will compute hash for each file. The program must also compute the hashes required by internal nodes of the three as shown in the figure. Finally, the program will compute hash of the Merkle root. All these information must be stored in a tree.txt file as per the description mentioned above.

The task-2.py will take the tree.txt file and a file name and its position as input, and produce a proof of membership. In the given Fig., if we give the correct tree.txt file and doc5.dat as inputs. The program should return hashes contained in node0.4, node1.3, node2.0, and store them in another txt file called proof.txt

```
0:4:c98d89e6928cadf0687913b2adb9cde4849b7f26
1:3:2261eb66fde5c6c6bab600bcfe4f69fc8de7fa46
2:0:87550fdab83c881de23001b3f7866ed7e6d850e3
```

In task-3.py, the program will take two inputs: proof.txt and the input document for which the proof was generated (in our example doc5.dat). The program will compute hash of the root from the given inputs and match it with the root hash given in the tree.txt file. If they match, the doc5.dat is a member of the tree otherwise not a member.



4 Submission

- You need to submit 3 python files: task1.py, task2.py, task3.py. Do not put your ID/Name in the filename itself, the first two lines in each file must contain your name and student id.
- The code should have proper comments about your thought process or idea that you are following in the next few lines of code (do not comment on each and every line)
- The program should be able to handle at least 100 files as input in a directory
- Task-2 should be able to run on any valid txt file produced by Task-1. It means that the output file (txt file) of Task-1 from student-1 should be acceptable in Task-2 solution of student-2 assuming they both have programmed Task-1 and Task-2 correctly.