Notations
For two strings $a, b$, we denote $\texttt{concat}(b)$ by $a + b$ (that is, if $a = \text{“Hello”}$, and $b = \text{“World”}$, then $a + b = \text{“HelloWorld”}$). For any natural number $n$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$.

Instructions  The lab-submission problem (marked ♠) is to be submitted via Moodle by 11:59PM on October 15th.

1 Introduction
The Central Board of Secondary Education (CBSE) has announced a new project – Academic BlockChain Document (ABCD) for providing transparent, tamper proof and paperless certificates (see here and other similar news reports). Inspired by this, our state DS-Pradesh would like to have a similar (maybe better?) solution for maintaining academic records – let’s call it ABCD++.

On a more serious note, the goal of this assignment is to illustrate the power of ideas developed in the previous lab-modules. Authenticated lists, authenticated sets play a crucial role in any cryptocurrency, but they have lots of other applications too, and this lab module will illustrate one such application. Additionally, we will also see some nice features of tree-based data structures.

Consider our hypothetical state DS-Pradesh. Every year, it has a number of students, whose scores need to be saved securely. Since we have already seen Merkle trees, we know that they provide a secure storage system, along with fast updates, and a mechanism to prove membership. We will use a combination of authenticated lists (module 2) and Merkle trees (module 3) to generate a secured storage system for all students’ scores. Additionally, if a student wishes to prove that he/she received a certain score in this exam, then he/she can provide a succinct proof, which can be verified efficiently.

In our system ABCD++, we will have an Authenticated List as the main structure, and each node in the list will store all students’ scores from a specific year. The students and their score data itself will be stored in the form of a Merkle tree, so you can assume that the number of students in any year is a power of 2.

1.1 Merkle Tree
The way the Merkle tree will be built is as follows - you will be given a list containing pairs of (student name, their score). We will assume for simplicity that the student id of each student is the index of its entry in this list. The students data will be stored in the leaf nodes of the tree, in the order of their student id. The val string of each leaf node will just contain the name of the student followed by an underscore and then his/her score ($<\text{name}>,<\text{score}>$). Apart from this, each TreeNode also has a maxleafval integer, which is maintained as the maximum score among all leaves of the subtree rooted at that TreeNode. For the leaf nodes, it is just the score of the student.
For all non-leaf nodes, the val string is the output of the CRF applied on the val strings of the two child nodes separated by a "#". For each node, you also need to maintain the maxleafval parameter and the numberLeaves parameter (it is the total number of leaves in the subtree rooted in that TreeNode).

1.2 Blockchain

The blockchain is an authenticated list (almost same as the one discussed in module 2). Each block in the blockchain contains pointers to the previous and next block, and strings value and dgst. In addition, every block also contains an integer year, representing the year of which the scores and student data is present. Also, each block has a Merkle tree, where as discussed above, the leaf nodes store all the students name and scores. The value of the block is calculated as follows - the Merkle tree corresponding to the block has a string val and an integer maxleafval associated with it. We concatenate the two attributes with an underscore in between, and get the value for that block. Thus value= <mtree.rootnode.val>_<mtree.rootnode.maxleafval>. Finally, the dgst for any block in the blockchain is computed by applying the CRF on the dgst of the previous node, and the value of that node (concatenated using #).

The DS-Pradesh State Board will only store the last block’s dgst on its website.

1.3 Digital Certificates

Each student receives a digital certificates together with his/her final score. A digital certificate is simply a sibling-coupled-path-to-root in the Merkle tree. This digital certificate can be used by the student to prove that he/she got score x in the year y. Here is how the student will prove authenticity of his/her score:

- the student provides the digital certificate (corresponding to year y). The digital certificate ties his/her score x to the val of the Merkle tree root.
- next, the student provides a sequence of tuples (dgst_y, value_y), (dgst_{y+1}, value_{y+1}), ..., (dgst_{curr}, value_{curr}), where dgst_j (resp. value_j) corresponds to the dgst (resp. value) for the block corresponding to year j. This sequence ties val to the last digest dgst_{curr}, and therefore it is not possible for someone to alter his/her scores (without breaking the CRF security).

Anyone can verify this proof by first checking the sibling-coupled-path-to-root, then checking that val is a substring of value_y (that is, the part before the ").), then checking that each dgst_j is correctly computed using dgst_{j-1} and value_j, and finally, checking that dgst_{curr} is the latest dgst (available on the DS-Pradesh State Board website).

One can even prove that he/she scored the maximum marks in a particular year. We can also augment the Merkle tree with additional attributes so that a student can prove that he/she was in the top 5%.

1.4 Updating Scores

The scores in the latest block can get updated (due to regrade requests). Suppose the i\textsuperscript{th} student’s score is updated. The update function first updates the val and maxleafval entries in the i\textsuperscript{th} leaf node. Next, it updates all the ancestors of this leaf node. Finally, it returns the val of the Merkle tree root.

2 Assignment Questions

You are given the following classes:

- public class Pair<A,B> : Objects of this class would be used to store each element in the sequence Sibling-Coupled Path to Root. The attributes of the class are:
  - public A First: the first element stored in the object.
• **public class TreeNode**: This class has the following attributes:
  
  - **public TreeNode parent**: the parent node
  - **public TreeNode left**: the left child node
  - **public TreeNode right**: the right child node
  - **public String val**: the value contained in this node
  - **public boolean isLeaf**: indicates whether the node is a leaf node or internal node
  - **public int numberLeaves**: the number of leaves in the sub-tree corresponding to this TreeNode.
  - **public int maxleafval**: the maximum score among all leaves of the subtree rooted at this TreeNode.

• **public class MerkleTree**: This class has the following attributes:

  - **public TreeNode rootnode**: pointer to the root node of the Merkle Tree
  - **public int numstudents**: number of students \( n \) whose records are stored in this Merkle tree.

  This class has the following methods (to be implemented):

  - **public String Build(List<Pair<String, int>> documents)**: Accepts an array of \( n \) documents, each of which is an (String, int) pair containing the name of a student (which is unique) and a non-negative integer corresponding to his/her score. It then builds a Merkle Tree using it, where the leaf node stores the string \(<\text{name}>, <\text{score}>\). Eg: If the name was "Pete" and score was 42, string in val will be “Pete_42”. Also, the maxleafval for that leaf node will be 42. For all other non-leaf nodes, val needs to be calculated as before, and maxleafval should be maintained as the maximum score among all leaves of the subtree rooted at that TreeNode. It returns the val corresponding to the root node.

  The method also sets numstudents = \( n \). You can assume \( n \) is a power of 2.

  - **public String UpdateDocument(int student_id, int newScore)**: Updating a student’s score given the student id\(^1\) and newScore will be similar as before. Recall that only the scores in the final block are allowed to be updated.

    First, go to the corresponding leaf node using the student_id and then change the val and maxleafval according to the new score. For all the ancestors of that leaf, the val and maxleafval need to change accordingly. Finally, return val of the rootnode.

• **public class Block**: This class has the following attributes:

  - **public Block previous**: pointer to previous block.
  - **public Block next**: pointer to next block.
  - **public String dgst**: a string representing the digest of this block.
  - **public MerkleTree mtree**: the Merkle Tree contained in the current Block of the BlockChain.
  - **public int year**: the year for which data is stored in this block.
  - **public String value**: it is the string defined as follows:

\[
\text{value} = \text{mtree.rootnode.val} \cdot \text{.} \cdot \text{mtree.rootnode.maxleafval} \]

\(^1\)Recall, the student id is the position of this student in the Merkle tree leaves
• public class BlockChain: This class has the following attributes:

- public static final String start_string: a string representing the starting digest for all objects of this class.
  This string should be equal to “LabModule5”.
- public Block lastblock: represents the last block present in the blockchain.
- public Block firstblock: represents the first block present in the blockchain.

The class has the following methods (to be implemented):

- public String InsertBlock(List<Pair<String, int>> Documents, int inputyear): first it creates a new Block with the year corresponding to inputyear and Merkle tree built using documents. It then adds this new block to the blockchain. Thereafter, it computes the digest using the value and the previous digest as follows:

\[
\text{u.digest} = \begin{cases} 
\text{CRF64.Fn(BlockChain.start_string + "#" + u.value)} & \text{if u.previous = null} \\
\text{CRF64.Fn(u.previous.digest + "#" + u.value)} & \text{if u.previous \neq null}
\end{cases}
\]

where CRF64 is an instance of the class CRF with outputsize = 64.
Finally, it returns the digest of the last Block of the updated BlockChain.

- public Pair<List<Pair<String, String>>, List<Pair<String, String>>> ProofOfScore (int student_id, int year): Accepts the id of student and returns a pair. The first element in the pair is Sibling-Coupled Path to Root of the corresponding Merkle tree node. The second element in the pair is the list of pair of values and digest (n.value, n.digest) of all nodes of the BlockChain, from the block corresponding to input year to the last block (both blocks included).

2.1 Exercises

1. Merkle Tree - Proof of membership and updates:
Implementing methods of class MerkleTree

(a) ♠ Build: This method accepts an array of n students data and builds a Merkle Tree using it.
- the index/id of a student in a particular year is the index of the list where the input data is received.
- the input contains pair of string and integers, the string is the student’s name and the corresponding integer is his/her score for that year.
- Each of these students data would be stored at the leaf nodes.
- Each of the non leaf nodes would store the CRF output after concatenating the strings stored of the two child nodes.
- At the end, the root of the tree should be stored at rootnode and n (length of documents) should be stored at numstudents.

(b) ♠ UpdateDocument: Takes input student_id ∈ [0, n – 1] and newScore and updates the corresponding student record.
- Start from N_{0,1} and traverse down the Merkle Tree until you reach \( N_{\log(n), \text{student_id}} \)
- Update the score part in the string \( \text{val}_{\log(n), \text{student_id}} \) with newScore. Also update the maxleafval of that node.
- Traverse back up from \( N_{\log(n), \text{student_id}} \) to the root \( N_{0,1} \) while updating \( \text{val}_{i,j} \) and maxleafval (if needed) for each ancestor node \( N_{i,j} \) encountered in the path.

\(^2\)Note the array indices of the input array are from 0 to n – 1.
• Return val of the rootnode as the updated summary.

2. BlockChain:
   Implementing methods of BlockChain:
   (a) InsertBlock: this method takes as input a list of student records and an integer inputyear.
      • It first creates a new object of class Block. It sets the pointer to previous and next blocks appropriately and sets the attribute year equal to the inputyear integer.
      • Then it builds a Merkle tree with the provided list and sets themtree attribute to it. The Merkletree rootnode is used to calculate the value of the Block as explained earlier. The value is then used to compute the dgst of the newly inserted Block using the dgst of the immediately preceding Block (or start_string, as explained earlier).
      • Finally this Block object is appended to the BlockChain and the dgst of the last Block of the updated BlockChain is returned.
   (b) ProofOfScore: It takes as input a student_id and an year. It then returns a pair to retrieve and verify the students score.
      • It iterates over the list to find the block corresponding to the input year.
      • It goes to the corresponding mtree node using the student_id.
      • It collects the Sibling-Coupled Path to Root of the corresponding Merkle tree node in the form of a list of pairs.
      • It collects the values and digests of all nodes, starting from the block corresponding to year, to the last block, in the form of a list of pairs.
      • Finally, it returns a pair of the two lists collected. This list can be used to know and authenticate the score of a student in a particular year.

3 Instructions
• Do not change the accessibility, names or signatures of the attributes and methods in the driver code. You may add your own attributes and methods to the Block, BlockChain and MerkleTree classes.
• The default constructor is used to instantiate objects of all the above classes. It is your responsibility to ensure appropriate initialization of the attributes of a newly created object.
• Submission instructions for lab-submission problem: You must submit BlockChain.java, Block.java and MerkleTree.java on Moodle. You must create a directory whose name is your Kerberos id, followed by “Module5” (for example, if your Kerberos id is “xyz120100”, then the folder name should be “xyz120100Module5”). The directory must contain MerkleTree.java, Block.java and BlockChain.java. Finally, compress this directory, and upload it on Moodle. The file name should be kerberosidModule5.zip (that is, xyz120100Module5.zip in the above example).

Acknowledgements
This is Version 1 of the document. Many thanks to the following students for identifying typos in previous version: Simran Malik, Tanish Singh Tak, Tushita Pandey, Vaibhav Misra, Somaditya Singh, Divyansh Mittal, Himasindhu Appili, Manas Jain, Prateek Mishra, Chirag, Aditya Mathur, Shivam Jain, Atif Anwar, Yash Pravin Shirke, Hemali Priyadarshi, Sachit Sachdeva, Dhruv Tyagi, Maitree Shandilya, Puurshottam Malviya, Yeruva Hitesh Reddy, Kushagra Rode, Srirabanti Dey, Adit Malhotra, Tanish Gupta, Abhinav Barnawal, Rishabh Dhiman, Vansh Kachchwal, Aditya Agrawal. If you find any errors, please send an email to kvenkata@iitd.ac.in and Akshay.Mattoo.me218@mech.iitd.ac.in (and participate in the ‘error-finding competition’).

3If you found an error and your name is not listed here, please send an email to kvenkata@iitd.ac.in.