CS61065 Theory and Applications of Blockchain

Assignment 4: Hyperledger Fabric

Date: 20 October 2022

Submission Deadline: 30th October 2022 EOD (Hard Deadline)

You can create a group of two and solve this assignment. Only one member from each group should submit the assignment in Moodle. Clearly mention your group details in the submission.

In this assignment, we will get familiar with the basics of Hyperledger Fabric. We will learn how to bootstrap a fabric test network and connect to it. We will also learn how to write, deploy and execute chaincodes (smart contracts). We will then write client applications to invoke those chaincodes.

Submission Instructions

Make a folder named 'A4_ROLLNUMBER', inside which create two subfolders for 'PartA' , 'PartB', and 'PartC'. These PartA and PartB will be the folders having your smart contract and all other dependencies required. Part C will have a client application used to test the chaincode deployed in part B.
Compress the folder 'A4-ROLLNUMBER' and submit it on moodle.

Part A

Write a chaincode which stores student data on the ledger. Student data contains two fields

- Roll No. **(Has to be Unique)**
- Name

Chaincode has to support the following methods:

The signature of all the following three functions should be exactly as specified here.

- `CreateStudent(ctx contractapi.TransactionContextInterface, roll string, name string) error`
    - `It will return the error (if any)`
    - `This function will take 'roll' and 'name' as input.`

    - Then it checks(using **StudentExists()**) if this 'roll' is already present in the ledger or not.
        - If present, return 'already present' as an error.
        - Otherwise, add this 'roll' and 'name' entry on the ledger using **ctx.GetStub().PutState()** function.

- **StudentExists(ctx contractapi.TransactionContextInterface, roll string)(bool, error)**
    - This function will take 'roll' as input.
    - It will return two values,
        - boolean
        - error (if any)
    - It will return true if it already exists, false otherwise.
    - Then it uses **ctx.GetStub().GetState()** to check if it exists or not.

- **ReadStudent(ctx contractapi.TransactionContextInterface, roll string)(string, error)**
    - It will return two values,
        - the Student's name corresponding to 'roll'
        - error (if any)
    - This function will take 'roll' as input.
    - Then it uses **ctx.GetStub().GetState()** to access the Student's data corresponding to 'roll'.

- **ReadAllStudents(ctx contractapi.TransactionContextInterface)(string, error)**
    - It will return a list with each item having two values,
        - the Student's name corresponding to 'roll'
        - error (if any)
    - This function will not take any input.
    - Then it uses **ctx.GetStub().GetState()** to access the Students' data and returns the roll and name of all students.

Part B

Write a chaincode to create a **Binary Search Tree** as data stored on the ledger. Different peers will be able to access this BST with the help of chaincode invoke and query transactions. Because of the shared ledger where each transaction is committed through the consensus protocol, each peer will have a consistent view of the BST.

Binary Search Tree structs:

```
TreeNode struct {
       Val  int
       Left *TreeNode
       Right *TreeNode
}

MyBST struct {
       PrimaryKey string
       Root *TreeNode
}
```

The chaincode has to support the following methods:

The signature of first 5 functions (Insert, Delete, Preorder, Inorder, TreeHeight) should be exactly same as that specified in this assignment.

- `Insert(ctx contractapi.TransactionContextInterface, val int) error`
    - It will return the error (if any)
    - This function will take an integer value 'val' to be inserted into the Binary Search Tree(BST).
    - Then it checks if any BST already exists in the ledger or not (using **ReadMyBST()**).
        - If a BST exists, then insert the value 'val' in the BST (using **UpdateMyBST()**).
        - Otherwise, create a BST with the root as 'val', compute a unique key that will represent this tree entry in the ledger. Then add this tree (containing only the 'root') into the ledger using **ctx.GetStub().PutState()** function.

- `Delete(ctx contractapi.TransactionContextInterface, val int) error`
    - This function will take an integer value 'val' to be deleted from the Binary Search Tree(BST).
    - It checks if there a BST exists in the ledger or not (using **ReadMyBST()**).
        - If present, then delete the value 'val' from the BST(using **UpdateMyBST()**).
        - Otherwise, throw an error stating that the tree is not found.
        - See "Node Deletion Rules" specified later in the assignment.
    - If the 'val' does not exist in the BST, return error

- **Preorder(ctx contractapi.TransactionContextInterface) (string, error)**
    - It will return two values,
        - string
        - error (if any)
    - It uses the **ReadMyBST()** function to get the latest MyBST from the chain.
    - Returns blank string and error if no BST exists in the ledger.
    - It computes the preorder traversal of the BST (using **preorderTraversal()** function).
    - It will return this preorder traversal in the form of a string. Ex-> "3,4,2,5,6,32,21".
      Make sure the format of the string is the same as the example: integers separated with a single comma.

- **Inorder(ctx contractapi.TransactionContextInterface) (string, error)**
    - It will return two values,
        - string
        - error (if any)
    - It uses the **ReadMyBST()** function to get the latest MyBST from the chain.
    - Returns blank string and error if no BST exists in the ledger.
    - Then it computes the inorder traversal of the BST (using **inorderTraversal()** function).
    - It will return this inorder traversal in the form of a string. Ex-> "2,3,4,5,6,21,32".

- **TreeHeight(ctx contractapi.TransactionContextInterface) (string, error)**
    - It will return two values,
        - string
        - error (if any)
    - It uses the **ReadMyBST()** function to get the latest MyBST from the chain.
    - Returns "0" and error if no BST exists in the ledger.
    - Then it uses the **heightOfTree()** function to find the height of the BST.
    - It will return this height in the form of a string. Ex-> "23".
      A tree with only root has height = "1"

The following methods are helper functions that will not be used for evaluation:

- **MyBSTExists(ctx contractapi.TransactionContextInterface, key string) (bool, error)**
    - It will return two values,
        - boolean
        - error (if any)
    - It will return true if the key already exists, false otherwise.

    - This function will take ‘key’ as input.
    - Then it uses **ctx.GetStub().GetState()** to check if it exists or not.

- **ReadMyBST(ctx contractapi.TransactionContextInterface) (*MyBST, error)**
    - It will return two values,
        - BST entry in ledger in form of struct variable
        - error (if any)
    - It will return the BST entry stored on the ledger (if any)
    - This function may retrieve a MyBST entry from a ledger using either **ctx.GetStub().GetStateByRange()** or **ctx.GetStub().GetState()**.

- **UpdateMyBST(ctx contractapi.TransactionContextInterface, val int, bst *MyBST, operation int) error**
    - It will return the error (if any)
    - This function will take an integer value ‘operation’ as one argument, which can be either 0 or 1.
        - If 0, it means you need to insert the value ‘val’ in the tree which is represented by ‘bst.Root’ (using **InsertValue()**).
        - If 1, it means you need to delete the value ‘val’ from the tree which is represented by ‘bst.Root’ (using **DeleteValue()**).
        - On deleting a node, make sure to delete the entry from the ledger (using **ctx.GetStub().DelState(key)**) whenever required.
    - Then update the entry in the ledger with the root of the changed tree and use the same key ‘bst.PrimaryKey’ (thus replacing the old root with the new one).
    - Then add this updated tree (‘root’) and key ‘bst.PrimaryKey’ entry on the ledger using **ctx.GetStub().PutState()** function.

- **InsertValue()**
    - Use this function to insert the value at the correct place as per BST rules.
    - Do not insert duplicates. If the value is already present just skip the insertion.

- **DeleteValue()**
    - Use this function to delete the value from the BST.
    - If value is not present in the BST, inform it to the caller function using appropriate return values.
    - **Node Deletion Rules:**
        - If both of the node’s children are present, then replace this node’s value with the inorder successor of the node (min value of subtree rooted at its right child). Then delete that inorder successor.
        - If only one of the node’s children is present, replace this node’s value with the child. Delete the child.
        - If it has no child, just delete the node.

- **`preorderTraversal()`**
    - `Use this function to get the preorder of the BST.`
- **`inorderTraversal()`**
    - `Use this function to get the inorder of the BST.`
- **`heightOfTree()`**
    - `Use this function to find the height of the BST.`
    - `If the Tree is empty, return 0.`
    - `Only root, then return 1.`

Part C

Write a Fabric Application to test the **Binary Search Tree** contract deployed in part B. Consider it will be run assuming two peers (one per org) - peer0.org1. And peer0.org2.

Internally the application will execute the following broad steps:

- Initialize the following separately for each of the two peers:
    - Connection Configuration
    - CA
    - Admin identity
    - User Registration
    - Gateway
    - Contract Object

- Take inputs from stdin, execute transactions, and validate output as follows.
- Input Format: The input will start with one command token in a single line, followed by arguments in the following lines. The possible commands are:
    - INSERT - Inserts the integer argument into BST
        - Argument
    - DELETE - Deletes the integer argument from the BST
        - Argument
    - INORDER - Queries inorder traversal and prints the output.
    - PREORDER - Queries preorder traversal and prints the output.
    - TREEHEIGHT - Queries treeheight and prints the output.

- Example input:

    INSERT
    5

```
INSERT
2
INSERT
1
INSERT
3
INSERT
7
INORDER
PREORDER
```

- **The commands are executed through alternate peers. The first command (e.g. INSERT \n 5) is executed by peer0.org1. The second command (e.g. INPUT \n 2) is executed by peer0.org2. 6th command (e.g. INORDER) is executed by peer0.org2. And so on.**

- **During testing, sufficient sleep duration (2 seconds) will be given between two inputs to avoid conflicts.**

NOTE: Parts B and C will be evaluated on the basis of the output of the Part C application.