Blockchain Assignment - 1

This code simulates a simplified blockchain system for a supply chain management system that tracks transactions among manufacturers, distributors, and clients. Key components and actions include:

- **`MerkleTree` class**: Constructs a Merkle tree from transactions.

- **`Blockchain` class**: Manages blockchain operations, including registration, transaction creation, and block mining.

- Simulated functions for distributor and consumer confirmations.

The code registers participants, simulates transactions, mining, and confirmation processes. The users i.e. manufacturer, distributor and receiver can add a transaction to the blockchain. A transaction shows attributes like -

1. Path
2. Product
3. Product ID
4. Miner hash ID
5. Timestamp
6. Confirmation status of both sender and receiver
7. Dispatched and received status

- Initially we have taken a manufacturer, 3 distributors and 3 clients and registered them along with a security deposit. All transactions happening in real-time are being added to the blockchain. A transaction is added only after being verified by the consensus algorithm we have used i.e. PoET (Proof of Elapsed Time).

- Proof of Elapsed Time (PoET) is a consensus algorithm that relies on participants competing to solve a cryptographic puzzle with a wait time that is randomized and unique to each participant. The first to solve the puzzle gains the right to create a new block, providing a fair and energy-efficient method for block validation.

- After a transaction is added to the block, a QR code is generated for the same, which can be viewed in the folder. On scanning the QR, we can see the respective transaction details.

- We have kept in mind the real-life blockchain implementation wherein the mining keeps going on. To stop this, use the termination command (ctrl + c).

Blockchain and Merkle Tree Implementation

This document provides an explanation of the functions and classes used in the blockchain and Merkle Tree implementation code.

1. MerkleTree Class

__init__(self, transactions)

- Initializes a Merkle Tree with a list of transactions.
- Creates a Merkle Tree data structure based on the list of transactions.

build_tree(self)

- Builds the Merkle Tree using the list of transactions.
- Ensures that the number of transactions is even by duplicating the last transaction if necessary.
- Returns the root of the Merkle Tree.

hash_transaction(self, transaction)

- Computes the SHA-256 hash of a transaction.
- Returns the hash value.

2. Blockchain Class

__init__(self)

- Initializes the blockchain with empty data structures and sets some initial values.
- Creates the first block (genesis block) with a default previous hash and proof.

register_manufacturer(self, manufacturer_id, security_deposit)

- Registers a manufacturer with a unique ID and security deposit.
- Checks if a manufacturer is already registered.
- Prints the registration information.

register_distributor(self, distributor_id, security_deposit)

- Registers a distributor with a unique ID and security deposit.
- Stores distributor information in a dictionary.
- Prints the registration information.

register_client(self, client_id, security_deposit)

- Registers a client with a unique ID and security deposit.
- Stores client information in a dictionary.
- Prints the registration information.

new_block(self, previous_hash=None, miner=None)

- Creates a new block to add to the blockchain.
- Takes optional parameters for the previous block's hash and miner's ID.
- Returns the newly created block.

new_transaction(self, sender, recipient, product, price)

- Creates a new transaction to be included in the next block.
- Takes parameters for sender, recipient, product, and price.
- Returns the number of pending transactions.

hash(self, block)

- Computes the SHA-256 hash of a block.
- Returns the hash value.

stop_mining(self)

- Sets the **mine_flag** attribute to **False**, stopping the mining process.

mine(self, node_identifier)

- Simulates the mining process with a proof-of-elapsed-time (PoET) style algorithm.
- Mines a new block with a random sleep time to simulate PoET.
- Validates and includes pending transactions in the new block.
- Updates security deposits and prints mining information.

mine_pending_transactions(self, node_identifier)

- Mines a new block using the PoET-style algorithm.
- Includes pending transactions in the new block and clears the pending transactions list.
- Updates security deposits and prints mining information.

valid_proof(self, last_proof, proof, previous_hash, merkle_root)

- Checks the validity of a proof by hashing and comparing it.
- Returns **True** if the proof is valid; otherwise, returns **False**.

last_block

- Property that returns the last block in the blockchain.

3. Distributor and Consumer Confirmation Functions

distributor_confirm_dispatch(blockchain, index)

- Simulates a distributor confirming product dispatch for a transaction.
- Sets the **confirmed_by_distributor** and **dispatched** flags for the transaction.
- Prints a confirmation message.

consumer_confirm_reception(blockchain, index)

- Simulates a consumer confirming product reception for a transaction.
- Sets the **confirmed_by_consumer** and **received** flags for the transaction.
- Prints a confirmation message.

resolve_delivery_issues(self)

- If the distributor has dispatched the product, and the client has received it, but the client is denying it (The client is lying, but the distributor is not), this function removes 50 units of the client's security deposit.
- If the distributor has not dispatched the product, and the client has not received it (The client is not lying, but the distributor is), this function removes 50 units of the distributor's deposit.

The **features implemented** in this assignment are -

1. Registration of clients, distributors and a manufacturer along with their security deposits.

2. Verifying transactions using Proof of Elapsed Time (PoET) consensus algorithm.

3. Implementation of Merkle tree to calculate the hash of all transactions.

4. QR code generation to get the current status of each transaction.

5. Atomicity of transactions such that only one transaction takes place at a time.

6. Identifying the liar and penalizing his security deposit is handled by resolve_delivery_issues() function.

Submitted by -

2020A3PS0570H	Aniketh Purackattu Sabu
2020A3PS2213H	Chahat Gupta
2020A3PS0550H	G Saikanth
2020A7PS2087H	Kartikay Dhall
2021A3PS0809H	Ananya Goyal