CSL7490: Introduction to Blockchain
Assignment - 1

Description of the code and assumptions made:
General Transaction Life cycle:

1. The user creates a transaction using a wallet and signs the transaction using a private key and then sends it to the memory pool.
2. The transaction is saved in the memory pool (transactions.csv).
3. Miner selected at random connected to the user fetches the transaction (at max 3) and adds that to the block after digital signature verification.
4. Miner starts mining that is solving a proof of work and creates the block, with header and transactions and transaction hash as the body.
5. Miner sends the block to other miners for verification (both of block and transactions), if the block is verified a confirmation is generated and after 6 conformations (recommended confirmations to stop double spending attack) block is added to the blockchain.

Following are the features implemented in the blockchain:

Blockchain Network:

- There are 10 miner nodes having different port numbers from 5001 to 5010. Similarly, there are 10 user nodes having other port numbers from 5011 to 5025.
- Each miner is connected to the other miner (if available) and each miner is connected to at least two users who are selected randomly from the pool of available users that is users who are currently online.
- This is implemented **using connect_node()** function in blockchain.py file.

Wallet creation:

- Each node whether miner or user uses the function **get_wallet()** that is provided in the class Blockchain and User which uses **ecdsa-based** public key and private key generation.
- The function uses **SECP256k1** elliptic curve for which the value of **a = 0 and b = 7** which yields a curve $\mathbf{y^2 = x^3 + 7}$ where the general format is $\mathbf{y^2 = x^3 + ax + 7}$.
- Then two keys are generated first the signing key and the other verifying key which are the private key and public key respectively.
- Then these keys are stored in the wallet of each user and miner.
- The wallet of the user and miner does not differ but the basic difference between user and miner is its functionalities that the user can only create a transaction whereas a miner verifies the transaction and creates a block and add the block to the blockchain.

Address of each user:

- The wallet contains the public key and private key of the user.
- The public key of the user is known to everyone and is shared with everyone.

- The hash of the public key is used as the address of the user.
- For this purpose public key is stored with the data of nodes. When a node is created a function get_key_pair is used and a mapping from the user's URL to its hash of public key is made.
- For each transaction, the user only needs to enter the URL of the receiver, and the hash of the public key of the receiver is automatically mapped with the entered URL.

Digital signature:

- In the previous step, a wallet is created and it contains the public key and private key.
- For each transaction done by the user, the transaction is signed using the user's private key.
- The Sign is created using the **private_key.sign(message)** function which takes a message and creates a signature and then the message with the signature is sent to the miner.
- The message is the hash of the transaction.
- On the miner side the **public_key.verify(message, sign)** method is used that verifies the signature and the authenticity of the message. The public keys are already there with the other nodes.

Header and body:

- The header contains information like index, timestamp, proof of work, previous hash, transactions, and Merkle root.
- Transactions are stored using their hash values in UTXO (unspent transaction) format in the body.
- Merkle root is created using the hash of each transaction.
- Algorithm for Merkle root creation:

```
-> H = List of the hash of all transactions
-> M = Merkle root
-> while length(H) > 1:
        if length(H) %2 == 1 :
            H.append(H[-1]) // making length of H even
        j = 0 // appending new hashes in H from starting
        for i in range(length(H)-1):
            H[j] = (hash(H[i],H[i+1]))
            j = j + 1
            i = i + 2
        index_to_delete = i - j  // index for deleting old hashes
        del H[-index_to_del:] // deleting old hashes
-> M = H[0]
```

- In this way, the Merkle root is created.
- This is implemented in **create_merkle_root()** function in the Blockchain class.

UTXO Format:

- It consists of **in** and **out** fields where in fields contain the hash of the transaction received by the sender of the current transaction and out contain the receiver of the current transaction.
- There are mainly three conditions that need to be checked.
    a. If the amount of the received transactions is less than the current performing transaction then the transaction should not be done.
    b. If the input of current is already used once then it should not be used to prevent **double-spending**.
    c. If the amount is more than two separate transactions are created one to the receiver and the other to the sender with the remaining amount.

General Instructions for running the program.

There are three files **nodes_data.py**, **Blockchain.py**, and **user.py.**

Steps:

1. Run all the files in the flask environment as discussed in the lecture by Lokendra Sir.
2. First, run **nodes_data.py** to create a CSV file with the data of nodes. Use the request **/create_nodes** with the **GET** method. **Remember to change the localhost variable in the nodes_data.py file as your localhost address if you are running without wifi it should be http://127.0.0.1 else it should be wifi address like http://172.31.50.86.**
3. Then run user.py for each user create a new file and copy and paste the content of the file **remember to change the port number in each file.** Port numbers lie in the range of **5011 to 5025** for users.
4. After changing the port numbers for respective users use **/get_key_pair** request with the **GET** method to get the public and private keys.
5. Use **/add_transaction** request with the **POST** method to add a transaction from a user. The format of the transaction is like this.

    ```
    {
        'receiver': '',
        'amount': ''
    }
    ```

    Where the receiver is the URL of the receiver like we need to send 10 BTC from URL **http://172.31.50.86/5011** to **http://172.31.50.86/5012** then use /add_transaction in URL **http://172.31.50.86/5011** with receiver **http://172.31.50.86/5012** as and amount as "10".
6. After creating the transaction the transaction is added to the **transactions.csv** file which acts as a memory pool.
7. Then use Blockchain.py and for each miner use a separate file with the different port numbers which lie in the range 5001 to 5010.

8. In Blockchain.py first use **/get_key_pair** request with the **GET** method to get the public and private keys.
9. Then use the **/connect_node** request with the **GET** method to get the connected nodes.
10. Then use **/add_transactions** request with the **GET** method to get the transactions done by connected users. This function adds transactions to a block after verifying the signature.
11. Then use **/print_transactions** request with the **GET** method to print the transactions that are in the current block.
12. Then use the **/mine_block** request with the **GET** method to create a block with the current transactions.
13. Then use the **/get_chain** request with the **GET** method to get detailed information about the blockchain.
14. Use the **/replace_chain** request with the **GET** method to check the validity of the new block and add that to the existing chain.