**PRACTICAL 6**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name:** | Harsh Shah | **Semester:** | VII | **Division:** | 6 |
| **Roll No.:** | 21BCP359 | **Date:** | 05-09-24 | **Batch:** | G11 |
| **Aim:** | To Implement PoW Consensus Mechanism on your own Blockchain | | | | |

**Proof of Work (PoW)**

Proof of work (PoW) is a blockchain consensus mechanism that requires significant computing effort from a network of devices. The concept was adapted from digital tokens by Hal Finney in 2004 through the idea of "reusable proof of work" using the 160-bit secure hash algorithm 1 (SHA-1).

**Program**

import hashlib

import time

class Block:

def \_\_init\_\_(self, index, previous\_hash, data, nonce=0):

self.index = index

self.timestamp = time.time()

self.previous\_hash = previous\_hash

self.data = data

self.nonce = nonce

self.hash = self.calculate\_hash()

def calculate\_hash(self):

block\_string = f"{self.index}{self.timestamp}{self.previous\_hash}{self.data}{self.nonce}".encode()

return hashlib.sha256(block\_string).hexdigest()

def \_\_str\_\_(self):

return (

f"Block Index : {self.index}\n"

f"Timestamp : {time.ctime(self.timestamp)}\n"

f"Previous Hash : {self.previous\_hash}\n"

f"Hash : {self.hash}\n"

f"Data : {self.data}\n"

f"Nonce : {self.nonce}\n"

f"{'-'\*41}"

)

class Blockchain:

def \_\_init\_\_(self):

self.chain = [self.create\_genesis\_block()]

def create\_genesis\_block(self):

return Block(0, "0", "Genesis Block")

def get\_latest\_block(self):

return self.chain[-1]

def add\_block(self, data):

latest\_block = self.get\_latest\_block()

new\_block = Block(len(self.chain), latest\_block.hash, data)

new\_block = self.proof\_of\_work(new\_block)

self.chain.append(new\_block)

def proof\_of\_work(self, block, difficulty=4):

while block.hash[:difficulty] != "0" \* difficulty:

block.nonce += 1

block.hash = block.calculate\_hash()

return block

def is\_chain\_valid(self):

for i in range(1, len(self.chain)):

current\_block = self.chain[i]

previous\_block = self.chain[i - 1]

if current\_block.hash != current\_block.calculate\_hash():

return False

if current\_block.previous\_hash != previous\_block.hash:

return False

return True

def \_\_str\_\_(self):

return "\n".join(str(block) for block in self.chain)

blockchain = Blockchain()

while True:

data = input("Enter transaction data for the new block (or 'q' to quit): ")

if data.lower() == "q":

break

blockchain.add\_block(data)

print("\nBlock added successfully!")

print("\nFinal Blockchain:")

print(blockchain)

print("\nBlockchain is valid:", blockchain.is\_chain\_valid())

**Output**





