

**M.Sc C.S - II**

**SEM III**

**Journal**

|  |  |
| --- | --- |
| **Roll No.** | 4701 |
| **Name** | Abhang Mane |
| **Subject** | Blockchain & Cryptocurrency |



**CERTIFICATE**

This is here to certify that **Mr.** **Abhang Sanjay Mane** Seat Number **4701** of M.Sc-II Computer Science, has satisfactorily completed the required number of Practicals prescribed by Thakur College Of Science And Commerce during the academic year 2023 – 2024.

Date:

Place: Kandivali,Mumbai

**Mr.Rahul Dhuru**

**Teacher In-Charge Head of Department**

**External Examiner**

**INDEX**

| **PrNo** | **Aim** | **Date** | **Sign** |
| --- | --- | --- | --- |
| 1 | Directed Acyclic Graph | 27/06/23 |  |
| 2 | Demonstrate Cryptocurrency Transaction Process | 04/07/23 |  |
| 3 | Implement Public Key Crypto System | 11/07/23 |  |
| 4 | To Solve Mining Puzzle | 17/07/23 |  |
| 5 | Explore Smart Contract Construction | 25/07/23 |  |
| 6 | Implement Naive Blockchain Construction | 22/08/23 |  |
| 7 | Implement Helium Wallet Construction | 12/09/23 |  |

**Practical 1**

**Aim:**Implement Directed Acyclic Graph.

**Theory:**

**1. Overview of Directed Acyclic Graphs (DAGs):**

A Directed Acyclic Graph (DAG) is a graph structure consisting of nodes (vertices) and directed edges (arcs) connecting these nodes. The key characteristic of a DAG is its acyclic nature, meaning there are no closed loops or cycles within the graph. Each edge in the DAG has a direction, indicating a one-way relationship between nodes. This acyclic property makes DAGs suitable for modeling various real-world systems, including dependency resolution, scheduling, and, most notably, in blockchain technology.

**2. Blockchain and DAGs:**

Blockchain technology, which gained prominence through cryptocurrencies like Bitcoin, initially employed a linear structure known as a blockchain. However, scalability and transaction confirmation times became significant challenges for traditional blockchains. To address these issues, some blockchain platforms, like IOTA and Nano, adopted DAGs as an alternative data structure. Here's how DAGs are applied in blockchain:

**2.1 Transaction Validation:**

In a traditional blockchain, transactions are grouped into blocks, and these blocks are added linearly to the chain. In contrast, DAG-based blockchains allow transactions to be verified asynchronously and added to the ledger independently. Each transaction in a DAG references one or more previous transactions, forming a directed acyclic structure. This enables parallel transaction validation and reduces confirmation times.

**2.2 Scalability:**

DAG-based blockchains offer inherent scalability benefits. As more transactions occur, the network can process them in parallel, increasing the overall transaction throughput. In traditional blockchains, network congestion often leads to delays and higher fees, while DAGs are designed to mitigate these issues.

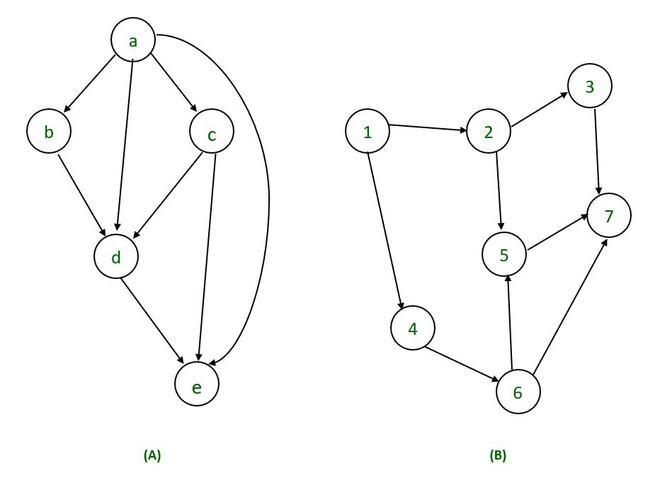
**2.3 Consensus Mechanisms:**

DAG-based blockchains typically use novel consensus mechanisms, such as the Tangle in IOTA or the Block Lattice in Nano. These mechanisms leverage the DAG structure to achieve consensus without the need for energy-intensive proof-of-work (PoW) or proof-of-stake (PoS) algorithms. In the Tangle, for instance, a transaction confirms two previous transactions, creating a web of trust.

**2.4 Security and DAGs:**

Security is a critical consideration in any blockchain system. DAGs, despite their benefits, present unique security challenges. Attackers can potentially create conflicting transactions within the DAG structure. However, mechanisms like cumulative weight and tip selection algorithms are used to mitigate these threats and maintain the integrity of the ledger.

**Example Diagram:**



**Source Code:**

class dag

{

constructor(id)

{

// Set value of node key

this.id = id;

this.next = null;

}

}

class Vertices

{

constructor(data)

{

this.data = data;

this.next = null;

this.last = null;

}

}

class Graph

{

constructor(size)

{

// Set value

this.size = size;

this.node = Array(size).fill(null);

this.setData();

}

// Set initial node value

setData()

{

if (this.size <= 0)

{

console.log("\nEmpty Graph");

}

else

{

for (var index = 0; index < this.size; index++)

{

// Set initial node value

this.node[index] = new Vertices(index);

}

}

}

connection(start, last)

{

// Safe connection

var edge = new dag(last);

if (this.node[start].next == null)

{

this.node[start].next = edge;

}

else

{

// Add edge at the end

this.node[start].last.next = edge;

}

// Get last edge

this.node[start].last = edge;

}

// Handling the request of adding new edge

addEdge(start, last)

{

if (start >= 0 && start < this.size &&

last >= 0 && last < this.size)

{

this.connection(start, last);

}

else

{

// When invalid nodes

console.log("\nSomething Wrong Here");

}

}

printGraph()

{

if (this.size > 0)

{

// Print graph dag Node value

for (var index = 0; index < this.size; ++index)

{

process.stdout.write("\nAdjacency list of vertex " + index + " :");

var edge = this.node[index].next;

while (edge != null)

{

// Display graph node

process.stdout.write(" " + this.node[edge.id].data);

// Visit to next edge

edge = edge.next;

}

}

}

}

// Find indegree of each nodes of a given graph

// Find the incoming edges of each node

findIndegree(indegree)

{

if (this.size <= 0)

{

return;

}

var edge = null;

for (var i = 0; i < this.size; ++i)

{

edge = this.node[i].next;

while (edge != null)

{

// Increase indegree of node

indegree[edge.id]++;

// Visit to next edge

edge = edge.next;

}

}

}

findSequence(indegree, visit, index, result)

{

if (index == this.size)

{

// Display result

process.stdout.write("\n");

var j = 0;

while (j < this.size)

{

process.stdout.write(" " + result[j]);

j++;

}

return;

}

var edge = null;

var i = 0;

while (i < this.size)

{

if (indegree[i] == 0 && visit[i] == false)

{

visit[i] = true;

result[index] = i;

// Get node edge

edge = this.node[i].next;

// Reduce indegree

while (edge != null)

{

indegree[edge.id]--;

// Visit to next edge

edge = edge.next;

}

this.findSequence(indegree,visit,index + 1,result);

visit[i] = false;

edge = this.node[i].next;

// Increase indegree

while (edge != null)

{

indegree[edge.id]++;

edge = edge.next;

}

}

i++;

}

}

topologicalSort()

{

if (this.size <= 0)

{

return;

}

// Use to track node

var visit = Array(this.size).fill(false);

// Store indegree of node edges

var indegree = Array(this.size).fill(0);

// Store result of topological sort

var result = Array(this.size).fill(-1);

// Find indegree of each node in graph

this.findIndegree(indegree);

process.stdout.write("\n");

this.findSequence(indegree, visit, 0, result);

}

}

function main()

{

process.stdout.write("\n\*\*\*\*\*Directed Acyclic Graph\*\*\*\*\*");

// 7 implies the number of nodes in graph

var g = new Graph(7);

// Connect node with an edge

// First and second parameter indicate node index

g.addEdge(1, 2);

g.addEdge(1, 4);

g.addEdge(2, 5);

g.addEdge(2, 3);

g.addEdge(3, 7);

g.addEdge(4, 6);

g.addEdge(5, 7);

g.addEdge(6, 5);

g.addEdge(6, 7);

// Print graph element

g.printGraph();

g.topologicalSort();

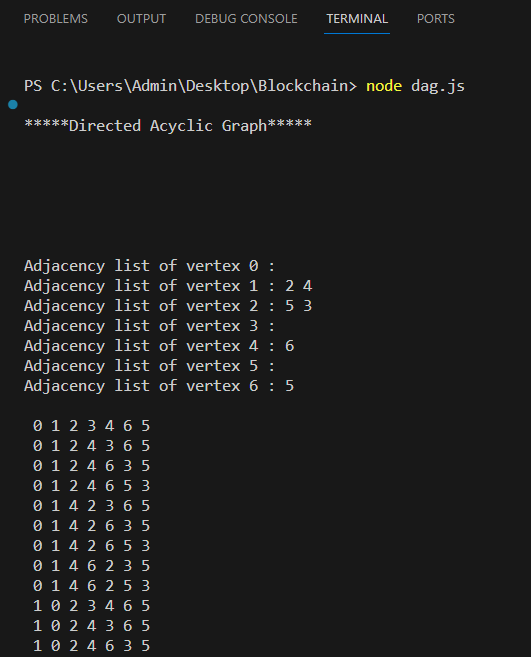
}

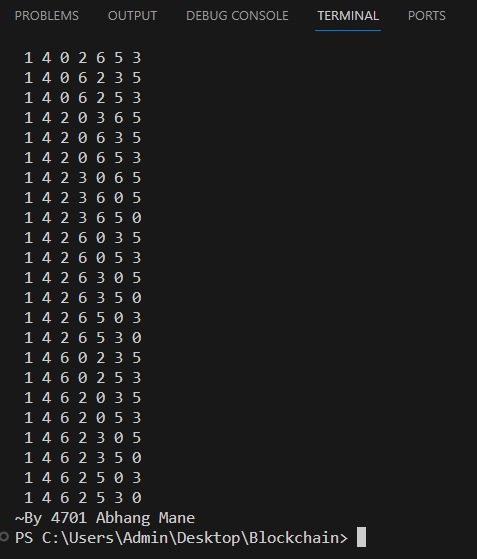
// Start program execution

main();

process.stdout.write("\n~By 4701 Abhang Mane");

**Output:**





**Conclusion:**Hence Directed Acyclic Graph was created and implemented successfully.

**Practical 2**

**Aim:**Demonstrate Cryptocurrency Transaction Process

**Theory:**

**Cryptocurrency transactions** are the fundamental operations within a blockchain-based digital currency system. These transactions enable the transfer of cryptocurrency units (e.g., Bitcoin, Ethereum) from one participant to another, and they play a pivotal role in the functioning of the entire cryptocurrency ecosystem. Below is an overview of the cryptocurrency transaction process:

**1. Transaction Initiation:**

A cryptocurrency transaction begins when a user, referred to as the sender or the payer, decides to transfer a certain amount of cryptocurrency to another user, known as the recipient or the payee. To initiate a transaction, the sender must have access to a digital wallet, a software application that stores their cryptocurrency holdings and facilitates transactions.

**2. Transaction Details:**

The sender specifies the details of the transaction, including the recipient's cryptocurrency address (a unique alphanumeric identifier), the amount of cryptocurrency to be sent, and potentially a transaction fee. Cryptocurrency addresses are essential for identifying the destination of funds, ensuring the accuracy of the transaction.

**3. Transaction Broadcast:**

After generating the digital signature, the sender broadcasts the transaction to the cryptocurrency network. This entails sending the transaction details and signature to a network of nodes (computers) that maintain the blockchain ledger. The transaction is then added to a pool of pending transactions waiting for confirmation.

**4. Transaction Verification:**

Miners or validators within the cryptocurrency network are responsible for verifying transactions. They collect pending transactions, verify the digital signatures, ensure that the sender has sufficient funds to complete the transaction, and check for any double-spending attempts (a critical security concern).

**5. Inclusion in a Block:**

Validated transactions are grouped together into blocks. In proof-of-work (PoW) cryptocurrencies like Bitcoin, miners compete to solve complex mathematical puzzles, and the first one to solve it gets the right to add a new block to the blockchain. The block contains a set of confirmed transactions, including the one initiated by the sender.

**6. Consensus and Confirmation:**

Cryptocurrency transactions are considered secure after a certain number of confirmations. The number of required confirmations varies between cryptocurrencies and is determined by the consensus algorithm. For instance, Bitcoin typically requires six confirmations for a transaction to be considered irreversible and secure.

**7. Transaction Completion:**

Once the required number of confirmations is achieved, the recipient's wallet balance is updated to reflect the received cryptocurrency. At this point, the transaction is considered complete, and the recipient can use the newly acquired cryptocurrency as they see fit.

**8. Transaction Fee:**

Transaction fees are typically paid by the sender to incentivize miners to prioritize their transaction. Higher fees may lead to faster confirmation times, while lower fees may result in longer processing times, especially during periods of high network congestion.

**Source Code:**

const SHA256 = require('crypto-js/sha256')

class Block{

constructor(index , timestamps , data , previousHash = ''){

this.index = index;

this.timestamps = timestamps;

this.data = data;

this.previousHash = previousHash;

this.hash = this.calculateHash();

this.nounce = 0;

}

calculateHash(){

return SHA256(this.index + this.previousHash + this.timestamps + JSON.stringify(this.data) + this.nounce).toString();

}

mineBlock(difficulty){

while(this.hash.substring(0,difficulty) !== Array(difficulty + 1).join("0")){

this.nounce++;

this.hash = this.calculateHash();

}

console.log("Block Mined :" + this.hash);

}

}

class mining{

constructor(){

this.chain = [this.createGenesisisBlock()];

this.difficulty = 4;

}

createGenesisisBlock(){

return new Block(0, "09/06/2023", "Genesis Block","0");

}

getLatestBlock(){

return this.chain[this.chain.length - 1]

}

addBlock(newBlock){

newBlock.previousHash = this.getLatestBlock().hash;

newBlock.mineBlock(this.difficulty);

this.chain.push(newBlock);

}

}

let AMCoin = new mining();

console.log("\*\*\*\*\*Cryptocurrency Transaction Process\*\*\*\*\*");

console.log("Mining Block 1...");

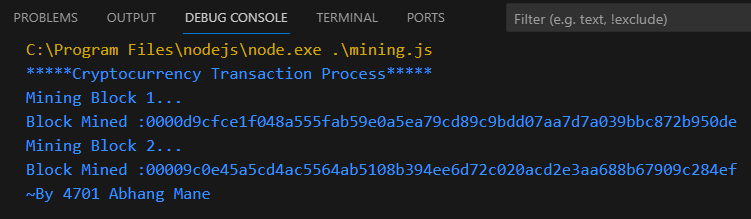
AMCoin.addBlock(new Block(1,"08/08/2023",{amount : 6}));

console.log("Mining Block 2...");

AMCoin.addBlock(new Block(2,"09/09/2023",{amount : 10}));

console.log("~By 4701 Abhang Mane");

**Output:**



**Conclusion:**Hence Cryptocurrency transaction process was created demonstrated where two blocks were mined successfully.

**Practical 3**

**Aim:**Implement Public Key Crypto System

**Theory:**

Public key cryptography, also known as asymmetric cryptography, is a fundamental component of blockchain technology. It plays a crucial role in securing transactions, ensuring the integrity of data, and enabling the unique features of blockchain networks. Here's a concise explanation of public key cryptography in the context of blockchain:

**1. Key Pair Generation:**

Public key cryptography relies on the generation of a key pair: a public key and a private key. The public key is openly shared with others, while the private key is kept secret. In blockchain, each participant, whether it's an individual user or a node on the network, possesses their own key pair.

**2. Digital Signatures:**

When a user initiates a transaction on the blockchain, they use their private key to create a digital signature for that transaction. This digital signature is a unique, cryptographic representation of the transaction data and the user's private key. It serves as proof of ownership and authorization.

**3. Transaction Verification:**

Once a transaction is broadcasted to the blockchain network, it's propagated to various nodes for verification. Using the sender's public key and the transaction data, nodes can verify the authenticity of the digital signature by decrypting it with the sender's public key. If the decryption process produces a valid signature, the transaction is considered genuine.

**4. Security and Trust:**

Public key cryptography provides a high level of security and trust in blockchain networks. Even though public keys are openly accessible, only the corresponding private key holder can generate the correct digital signature. This ensures that transactions cannot be tampered with or falsely created.

**5. Anonymity and Privacy:**

Public key cryptography also enables a level of anonymity and privacy in blockchain transactions. While transactions are recorded on the public ledger, users are identified by their public keys rather than their real-world identities. This pseudonymous nature helps protect user privacy.

**6. Address Generation:**

In addition to public keys, blockchain systems often use addresses as a user-friendly representation. Addresses are derived from public keys through a series of cryptographic operations. They are what users share when they want to receive cryptocurrency or tokens.

**7. Key Management:**

The secure management of private keys is paramount in blockchain. Losing access to a private key means losing control over associated assets permanently. Various wallet solutions and best practices are employed to safeguard private keys.

**8. Role in Consensus:**

Public key cryptography also plays a role in consensus mechanisms. For example, in proof-of-stake (PoS) and delegated proof-of-stake (DPoS) blockchains, validators are chosen based on their ownership of cryptocurrency and the ability to prove ownership through digital signatures.

**Source Code:**

const crypto=require("crypto");

const{publicKey,privateKey}=crypto.generateKeyPairSync("rsa",{modulusLength:2048, });

const data="very sensitive data";

const encryptedData=crypto.publicEncrypt(

    {

        key:publicKey,

        padding:crypto.constants.RSA\_PKSC1\_QAEP\_PADDING,

        qaephash:"sha256",

    },

    Buffer.from(data)

);

console.log("\*\*\*\*\*Public Key Crypto System\*\*\*\*\*");

console.log("Encrypted Data:",encryptedData.toString("base64"));

const decryptedData=crypto.privateDecrypt(

    {

        key:privateKey,

        padding:crypto.constants.RSA\_PKSC1\_QAEP\_PADDING,

        qaephash:"sha256",

    },

    encryptedData

);

console.log("\nDecrypted Data:",decryptedData.toString());

const verifiableData="This Needs To Be Verified!";

const signature=crypto.sign("sha256",Buffer.from(verifiableData),{

    key:privateKey,

    padding:crypto.constants.RSA\_PKSC1\_QAEP\_PADDING,

});

console.log(signature.toString("base64"));

const isVerified=crypto.verify(

    "sha256",

    Buffer.from(verifiableData),

    {

        key:publicKey,

        padding:crypto.constants.RSA\_PKSC1\_QAEP\_PADDING,

    },

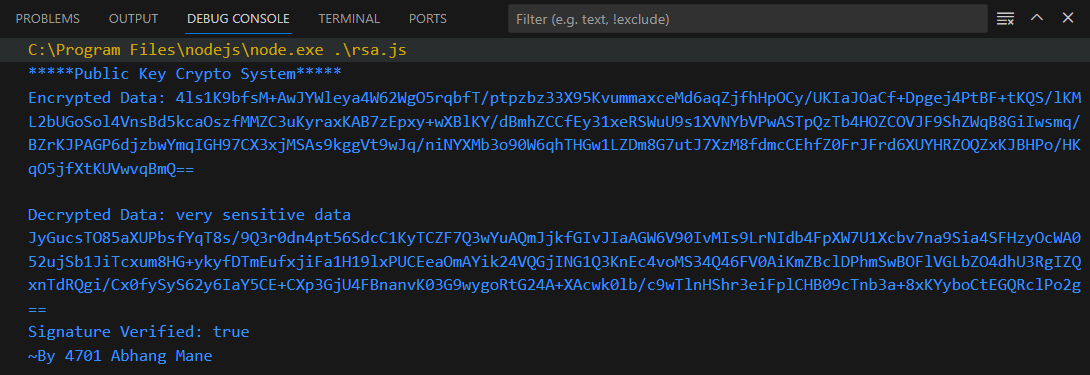
    signature

);

console.log("Signature Verified:",isVerified)

console.log("~By 4701 Abhang Mane");

**Output:**



**Conclusion:**Hence Public Key Crypto System in blockchain was created and implemented successfully.

**Practical 4**

**Aim:**To solve mining puzzle

**Theory:**

The mining puzzle, often referred to as the Proof of Work (PoW) mechanism, is a fundamental concept in blockchain technology. It plays a central role in achieving consensus, securing the network, and facilitating the creation of new blocks. Here's a concise explanation of how the mining puzzle is solved in blockchain:

**1. Block Creation and Transactions:**

In a blockchain network, transactions are grouped together into blocks. Miners are responsible for creating these blocks by selecting a set of pending transactions from the network's mempool. The block also includes a reference to the previous block, forming a linked chain.

**2. The Mining Puzzle (Proof of Work):**

To create a new block, miners must solve a computationally intensive mathematical puzzle. This puzzle is designed to be difficult to solve but easy to verify once a solution is found. It involves repeatedly hashing the block's header (which includes the transaction data and a nonce) using a cryptographic hash function, such as SHA-256 in Bitcoin.

**3. The Nonce:**

The nonce is a crucial component of the mining puzzle. It is a 32-bit integer that miners can adjust. Miners repeatedly change the nonce value and recompute the hash of the block header until they find a hash that meets certain criteria. This criteria typically involves producing a hash value that starts with a specific number of leading zeros (the "target").

**4. Proof of Work Difficulty:**

The difficulty of the mining puzzle is adjusted by the network at regular intervals (usually every 2016 blocks in Bitcoin). The goal is to maintain a consistent block generation rate, typically around 10 minutes in Bitcoin. If blocks are being generated too quickly, the difficulty increases, making the puzzle harder to solve. Conversely, if blocks are slow, the difficulty decreases.

**5. Competition among Miners:**

Miners compete to find a valid nonce and solve the puzzle. This competition is a race, as the first miner to discover a valid nonce gets the right to add the new block to the blockchain. They broadcast their solution to the network for verification.

**6. Verification and Consensus:**

Once a miner discovers a valid nonce and broadcasts the solution, other nodes in the network quickly verify the solution's correctness by recomputing the hash. If it meets the target criteria and is valid, the new block is added to the blockchain, and the miner is rewarded with a block reward (in the form of newly created cryptocurrency coins) and transaction fees.

**7. Security and Immutability:**

The mining puzzle, while resource-intensive, provides security to the blockchain network. To compromise the network, an attacker would need to control more computational power (hashrate) than the combined power of all honest participants, making it extremely secure. Additionally, once a block is added to the blockchain, it becomes extremely difficult to alter any information within it, ensuring immutability.

**Source Code:**

const SHA26 = require('crypto-js/sha256')

class Transaction {

constructor(fromAddress , toAddress , amount){

this.fromAddress = fromAddress;

this.toAddress = toAddress;

this.amount = amount;

}

}class Block{

constructor(timestamps , transaction , previousHash = ''){

this.timestamps = timestamps;

this.transaction = transaction;

this.previousHash = previousHash;

this.hash = this.calculateHash();

this.nounce = 0;

}

calculateHash(){

return SHA26(this.index + this.previousHash + this.timestamps +

JSON.stringify(this.data) + this.nounce).toString();

}

mineBlock(difficulty){

while(this.hash.substring(0,difficulty) !== Array(difficulty + 1).join("0")){

this.nounce++;

this.hash = this.calculateHash();

}

console.log("Block mined :" + this.hash);

}

}

class Blockchain{

constructor(){

this.chain = [this.createGenesisisBlock()];

this.difficulty = 2;

this.pendingTransaction = [];

this.miningReward = 100;

}

createGenesisisBlock(){

return new Block("05/09/2022", "Genesis Block","0");

}getLatestBlock(){

return this.chain[this.chain.length - 1]

}

minePendingTransactions(miningRewardAdress){

let block = new Block(Date.now() , this.pendingTransaction);

block.mineBlock(this.difficulty);

console.log("Block successfully mined !");

this.chain.push(block);

this.pendingTransaction = [

new Transaction(null,miningRewardAdress,this.miningReward),17];

}

creatTransaction(transaction){

this.pendingTransaction.push(transaction);

}

getBalanceOfAddress(address){

let balance = 0;

for(const block of this.chain){

for(const trans of block.transaction){

if(trans.fromAddress === address){

balance -= trans.amount;

}

if(trans.toAddress == address){

balance +=trans.amount;

}

}

}

return balance;

} }

console.log("\n\*\*\*\*\*Solving Mining Puzzles\*\*\*\*\*")

let AMCoin = new Blockchain();

AMCoin.creatTransaction(new Transaction('address1', 'address2',100));

AMCoin.creatTransaction(new Transaction('address2', 'address1',50));

console.log("\n Starting The Miner...");

AMCoin.minePendingTransactions('Anonymous\'s-address');console.log('\nBalance of Anonymous is' ,AMCoin.getBalanceOfAddress('Anonymous\'s address'));

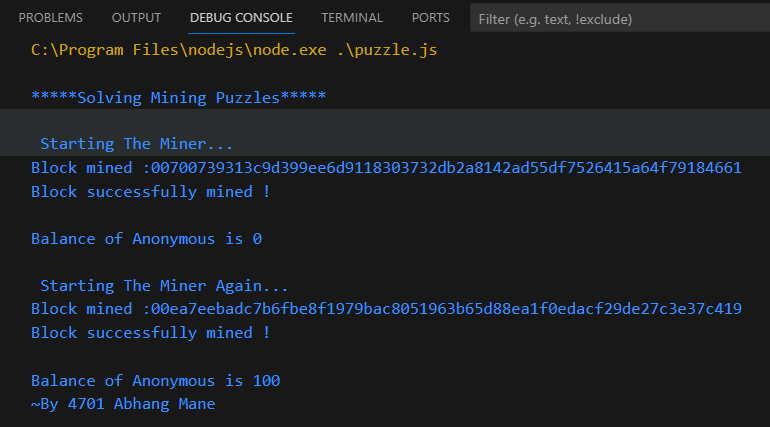
console.log("\n Starting The Miner Again...");

AMCoin.minePendingTransactions('Anonymous\'s-address');

console.log('\nBalance of Anonymous is',AMCoin.getBalanceOfAddress('Anonymous\'s-address'));

console.log("~By 4701 Abhang Mane");

**Output:**



**Conclusion:**Hence mining puzzle was mined and solved successfully for anonymous user.

**Practical 5**

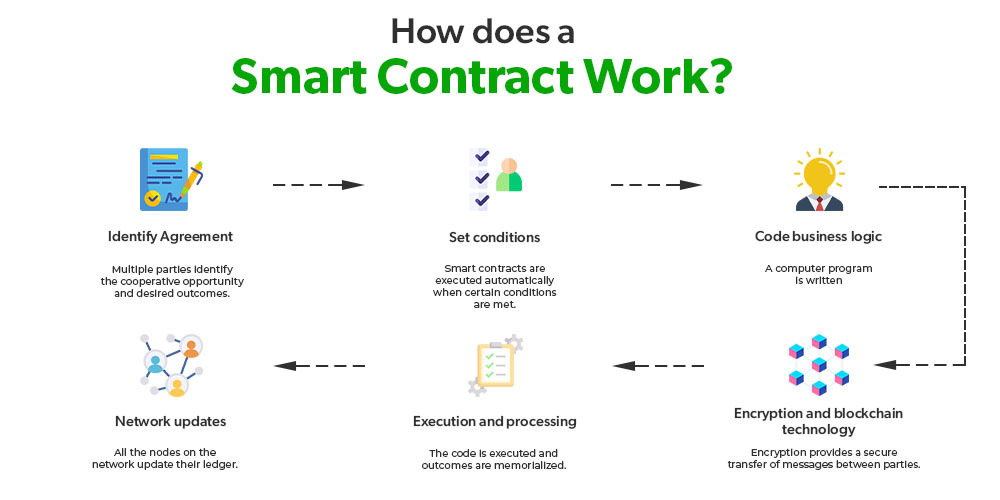
**Aim:**Explore Smart Contracts(Case Study)

**What is Smart Contracts?**

A smart contract is a self-executing contract with the terms of the agreement directly written into code. These contracts run on blockchain technology, and their primary purpose is to facilitate, verify, or enforce the negotiation or performance of a contract

Here are the five main points about smart contracts:

* Digital and Self-executing: Smart contracts are digital contracts with self-executing capabilities, automating contract enforcement without the need for intermediaries.
* Blockchain Technology: They run on blockchain platforms, ensuring security, transparency, and immutability.
* Code-Based: Smart contracts are written in code, specifying contract terms and actions to be taken when conditions are met.
* Decentralization: They operate on a decentralized network of nodes, eliminating single points of control and making them resistant to tampering.
* Real-World Applications: Smart contracts have diverse applications, including DeFi, supply chain management, voting systems, and more, transforming various industries.



1. Identify Agreement: Parties involved identify the terms and conditions of the agreement they want to automate using a smart contract.
2. Set Conditions: Clear and specific conditions are defined, which, when met, trigger actions within the smart contract.
3. Code Business Logic: Developers write the code for the smart contract, specifying the conditions, actions, and rules of the agreement using programming languages like Solidity.
4. Encryption and Blockchain Technology: The smart contract code is encrypted and deployed onto a blockchain network, such as Ethereum. Blockchain technology provides the necessary infrastructure for secure and transparent execution.
5. Execution and Processing: The smart contract autonomously executes its code when predefined conditions are satisfied. It performs actions like transferring cryptocurrency or updating data on the blockchain.

These steps ensure that the smart contract operates seamlessly, enforcing the agreement without the need for intermediaries while benefiting from the security and transparency of blockchain technology. Updates to the smart contract may require changes to the code and a new deployment on the blockchain.

**Real-World Scenario: Decentralized Finance (DeFi) Lending and Borrowing**

In the world of decentralized finance (DeFi), smart contracts play a central role in lending and borrowing cryptocurrencies. Here's a scenario illustrating how this works:

Lender-Borrower Interaction:

* Agreement: A lender and borrower agree on the terms of a cryptocurrency loan. This typically includes the loan amount, interest rate, collateral, and the loan duration.
* Smart Contract Creation: A smart contract is created on a DeFi lending platform like Compound or Aave. The terms of the loan are encoded into the smart contract.
* Collateralization: The borrower provides cryptocurrency as collateral, which is locked in the smart contract. The smart contract verifies the value of the collateral using real-time price feeds from oracles.
* Loan Disbursement: Once the collateral is verified and meets the required value, the smart contract automatically disburses the agreed-upon loan amount in cryptocurrency to the borrower's wallet.

Interest and Repayment:

* Interest Accrual: The smart contract calculates and accrues interest on the outstanding loan balance according to the predefined interest rate and duration.
* Repayment: As the borrower makes repayments (including both principal and interest) in cryptocurrency, the smart contract automatically updates the loan balance.

Liquidation:

* Liquidation Threshold: If the value of the collateral falls below a certain threshold (to ensure the lender's security), the smart contract may trigger liquidation.
* Liquidation Process: The smart contract automatically liquidates a portion of the borrower's collateral to cover the outstanding loan and any accrued interest. The lender receives their principal and interest, and the borrower gets any remaining collateral back.

Smart Contract Autonomy:

* No Intermediaries: Throughout this process, there are no intermediaries involved. The smart contract autonomously manages the loan, collateral, interest calculations, and disbursements based on the predefined terms.

This scenario illustrates how DeFi lending and borrowing platforms leverage smart contracts to enable peer-to-peer cryptocurrency loans, providing participants with transparency, efficiency, and security while eliminating the need for traditional financial institutions.

**Advantages of Smart Contracts:**

* Automation: Smart contracts automate contract execution, reducing the need for intermediaries and manual processes.
* Trust: They enhance trust by executing agreements based on predefined, tamper-proof rules.
* Efficiency: Transactions are processed faster, reducing costs and delays associated with traditional contracts.
* Transparency: All contract actions and data are recorded on the blockchain, providing transparency to all involved parties.
* Security: Smart contracts are highly secure due to blockchain encryption and immutability.

**Disadvantages of Smart Contracts:**

* Code Vulnerabilities: Smart contracts are susceptible to coding errors, which can lead to vulnerabilities and security breaches.
* Legal Recognition: Legal systems in many jurisdictions may not fully recognize or regulate smart contracts, leading to legal uncertainties.
* Scalability: As more transactions occur on a blockchain network, scalability issues can arise, affecting the efficiency and cost-effectiveness of smart contracts.

Here are some other use cases of smart contracts:

* Supply Chain Management: They improve transparency and traceability by automating and verifying the flow of goods and data within supply chains.
* Real Estate Transactions: Smart contracts streamline property sales and rental agreements, automating payments and transfer of ownership.
* Voting Systems: Enhance the security and transparency of elections by recording votes on a blockchain, ensuring accuracy and preventing fraud.
* Insurance Claims: Automate claims processing and payouts based on predefined conditions, reducing paperwork and delays.

**Conclusion:**Hence Case Study on Smart Contracts was implemented successfully.

**Practical 6**

**Aim:**ImplementNaive Blockchain Construction

**Theory:**

A naive blockchain construction, while simplified compared to more advanced implementations, serves as a foundational model for understanding the core principles of blockchain technology. In this simplified version, we'll outline the essential components and processes involved:

**1. Data Structure:**

A naive blockchain is a distributed ledger that consists of a linear chain of blocks, where each block contains a set of transactions. In its simplest form, a block includes a timestamp, a reference to the previous block (often referred to as the "parent" or "previous hash"), and a list of transactions.

**2. Transaction Processing:**

Participants in the network initiate transactions by broadcasting them to their peers. These transactions are collected and bundled into a candidate block. Each transaction typically includes sender and recipient addresses, transaction amount, and a digital signature to ensure authenticity.

**3. Block Creation:**

Miners (participants who perform the role of creating new blocks) compete to solve a computational puzzle. In the naive construction, this puzzle is often a straightforward cryptographic task, like finding a nonce that, when hashed with the block's data, results in a hash with specific leading zeros. The first miner to solve this puzzle creates a new block.

**4. Consensus Mechanism:**

In a naive blockchain, the consensus mechanism relies on the "longest chain" rule. Nodes accept the longest valid chain as the legitimate one. This rule assumes that the majority of participants are honest and acting in the network's best interest.

**5. Block Verification:**

Once a miner discovers a valid block, they broadcast it to the network. Other nodes verify the block's validity by checking its cryptographic integrity (e.g., hashing), confirming that transactions within the block are legitimate, and ensuring it adheres to protocol rules (e.g., the correct previous hash reference).

**6. Chain Extension:**

Upon successful verification, the new block is appended to the existing blockchain. This extension maintains the chronological order of blocks and ensures the integrity of the entire transaction history.

**7. Incentives:**

Incentives are essential to encourage miners to participate in block creation. In the naive model, miners are rewarded with a fixed number of cryptocurrency tokens (e.g., Bitcoin) and transaction fees for adding a new block to the chain.

**8. Security:**

The security of a naive blockchain relies on the computational difficulty of the mining puzzle and the assumption that the majority of miners are honest. An attacker would need to control over 50% of the network's computational power to manipulate the chain.

**9. Limitations:**

A naive blockchain construction simplifies many aspects for educational purposes but lacks some features found in real-world blockchain systems. It may not address issues like scalability, smart contracts, or complex consensus algorithms.

**Source Code:**

const SHA256 = require('crypto-js/sha256')

class Block{

constructor(index , timestamps , data , previousHash = ''){

this.index = index;

this.timestamps = timestamps;

this.data = data;

this.previousHash = previousHash;

this.hash = this.calculateHash();

}

calculateHash(){

return SHA256(this.index + this.previousHash + this.timestamps +

JSON.stringify(this.data)).toString();

}

}

class Blockchain{

constructor(){

this.chain = [this.createGenesisisBlock()];

}

createGenesisisBlock(){

return new Block(0, "01/01/2017", "Genesis Block","0");

}

getLatestBlock(){

return this.chain[this.chain.length - 1]

}

addBlock(newBlock){

newBlock.previousHash = this.getLatestBlock().hash;

newBlock.hash = newBlock.calculateHash();

this.chain.push(newBlock);}

ischainValid(){

for(let i = 1; i < this.chain.length; i++){

const currentBlock = this.chain[i];

const previousBlock = this.chain[i-1];

if(currentBlock.hash !== currentBlock.calculateHash()){

return false;

}

if(currentBlock.previousHash !== previousBlock.hash){

return false;

}

}

return true;

}

}

let AMCoin = new Blockchain();

AMCoin.addBlock(new Block(1,"08/08/2023",{amount : 2}));

AMCoin.addBlock(new Block(2,"09/09/2023",{amount : 8}));

console.log("\*\*\*\*\*Naive Blockchain Construction\*\*\*\*\*");

console.log("Is Block chain Valid ? " + AMCoin.ischainValid());

console.log("Block Chain Created :\n",JSON.stringify(AMCoin, null , 2));

4

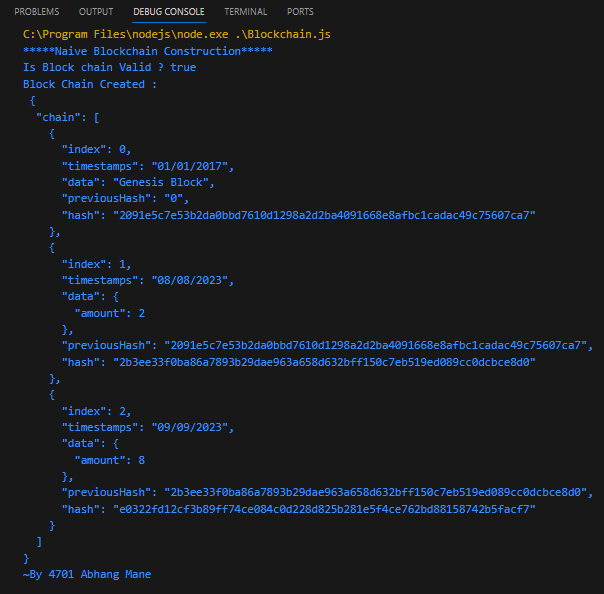
console.log("~By 4701 Abhang Mane");

//AMCoin.chain[1].data = { amount: 100};

// tampering the data to check block chain Validity

//console.log("is Block chain Valid ? " + AMCoin.ischainValid());

**Output:**



**Conclusion:**Hence valid blockchain was created and implemented successfully with 3 indexes.

**Practical 7**

**Aim:**Implement Helium Wallet

**Theory:**

A Helium wallet in the context of blockchain is designed for managing and interacting with the Helium network, which is a decentralized wireless network built to support the Internet of Things (IoT) and various wireless devices. Here's a concise explanation of what a Helium wallet is and its significance:

**Helium Wallet: Managing IoT Connectivity**

A Helium wallet is a software application that allows users to manage their interactions with the Helium network. The Helium network is unique because it leverages blockchain technology to create a decentralized, low-power, and low-cost wireless network for IoT devices. Here's a breakdown of key aspects related to a Helium wallet:

**1. IoT Connectivity:**

The Helium network is designed to provide connectivity for a wide range of IoT devices, such as sensors, trackers, and smart devices. These devices can communicate with the network using Helium's LongFi technology, which combines LoRaWAN and Helium's blockchain to create an efficient and secure wireless network.

**2. Earning HNT Tokens:**

One of the primary functions of a Helium wallet is to facilitate the earning of HNT (Helium Network Token) tokens. In the Helium network, participants, often referred to as "Hotspot operators," provide wireless coverage by running Hotspots (specialized devices). These operators are rewarded with HNT tokens for validating and transmitting IoT device data to the network. A Helium wallet is used to manage and monitor these earnings.

**3. Security and Identity:**

A Helium wallet also plays a role in securing the identity of Hotspot operators and IoT devices. It uses cryptographic key pairs to authenticate users and devices on the network. Private keys are used to sign transactions and interactions with the network, ensuring the integrity and security of data transmissions.

**4. Staking and Consensus:**

The Helium network uses a unique consensus mechanism known as Proof-of-Coverage (PoC) to validate Hotspot coverage and performance. Hotspot operators must stake HNT tokens to participate in the network and earn rewards. A Helium wallet is used to manage these staking operations, including the delegation of staking authority.

**5. Decentralized Marketplace:**

Helium also features a decentralized marketplace called the "Data Credits" marketplace. Users can purchase Data Credits with HNT tokens through their Helium wallet, which can then be used to pay for IoT device data transmission on the network.

**6. Monitoring and Analytics:**

Helium wallets often provide users with tools for monitoring the performance of their Hotspots, including data on coverage, rewards earned, and network activity. This data allows users to optimize their Hotspot placement and performance.

**7. Mobile and Hardware Wallets:**

Helium wallets are available in various forms, including mobile apps and hardware wallets. Mobile wallets offer convenience for managing Hotspots and HNT tokens on the go, while hardware wallets provide enhanced security by storing private keys offline.

**Source Code:**

const crypto = require('crypto');

class HeliumWallet {

  constructor() {

    this.privateKey = this.generatePrivateKey();

    this.publicKey = this.generatePublicKey(this.privateKey);

    this.address = this.generateAddress(this.publicKey);

    this.balance = 350;

    this.transactions = [];

  }

  generatePrivateKey() {

    return crypto.randomBytes(32).toString('hex'); // Generate a random private key

  }

  generatePublicKey(privateKey) {

    // In a real implementation, you would derive the public key from the private key using elliptic curve cryptography.

    // This is a simplified example.

    const hash = crypto.createHash('sha256').update(privateKey).digest('hex');

    return hash;

  }

  generateAddress(publicKey) {

    // In a real implementation, you would create a Helium-specific address format.

    // This is a simplified example.

    return publicKey.substring(0, 10); // Just use the first 10 characters of the public key.

  }

  sendTokens(receiverAddress, amount) {

    // In a real implementation, you would create and sign a Helium transaction.

    // This is a simplified example.

    if (amount <= this.balance) {

      this.balance -= amount;

      this.transactions.push({ sender: this.address, receiver: receiverAddress, amount });

      return true;

    } else {

      return false;

    }

  }

  getBalance() {

    return this.balance;

  }

  getTransactionHistory() {

    return this.transactions;

  }

}

// Example usage:

console.log("\n\*\*\*\*\*Helium Wallet\*\*\*\*\*");

const myWallet = new HeliumWallet();

console.log('Address:', myWallet.address);

console.log('Balance:', myWallet.getBalance());

myWallet.sendTokens('receiver\_address', 17);

console.log('New Balance:', myWallet.getBalance());

console.log('Transaction History:', myWallet.getTransactionHistory());

console.log("\n~ By 4701 Abhang Mane");

**Output:**



**Conclusion:**Hence Helium wallet was created and implemented successfully with one transaction where balance was 350 and after transaction new balance became 333.