Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyse their time and space complexity

Non-recursive Fibonacci Program

def calculate\_fibonacci\_series(n):

    a, b = 0, 1

    step\_count = 0

    fibonacci\_series = []

    for i in range(n):

        step\_count += 1

        fibonacci\_series.append(a)

        a, b = b, a+b

    return fibonacci\_series

if \_\_name\_\_ == "\_\_main\_\_":

    # Input from the user

    n = int(input("Enter the number of terms in the Fibonacci series: "))

    # Calculate the Fibonacci series and step count

    fibonacci\_series= calculate\_fibonacci\_series(n)

    print(f"Fibonacci Series for the first {n} terms: {fibonacci\_series}")

Output



Recursive Fibonacci Program

def recur\_fibo(n):

if n <= 1:

return n

else:

return(recur\_fibo(n-1) + recur\_fibo(n-2))

if \_\_name\_\_ == "\_\_main\_\_":

nterms = int(input("How many terms? "))

if nterms <= 0:

print("Plese enter a positive integer")

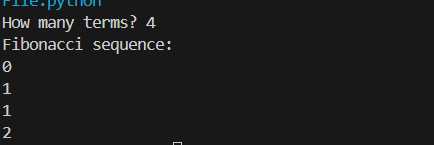
else:

print("Fibonacci sequence:")

for i in range(nterms):

print(recur\_fibo(i))

Output



Write a program to implement Huffman Encoding using a greedy strategy

class NodeTree(object):

    def \_\_init\_\_(self, left=None, right=None):

        self.left = left

        self.right = right

    def children(self):

        return (self.left, self.right)

    def nodes(self):

        return (self.left, self.right)

    def \_\_str\_\_(self):

        return '%s\_%s' % (self.left, self.right)

def huffman\_code\_tree(node, left=True, binString=''):

    if type(node) is str:

        return {node: binString}

    (l, r) = node.children()

    d = dict()

    d.update(huffman\_code\_tree(l, True, binString + '0'))

    d.update(huffman\_code\_tree(r, False, binString + '1'))

    return d

if \_\_name\_\_  == "\_\_main\_\_":

string = 'BCAADDDCCACACAC'

    freq = {}

    for c in string:

        if c in freq:

            freq[c] += 1

        else:

            freq[c] = 1

    freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

    nodes = freq

    while len(nodes) > 1:

        (key1, c1) = nodes[-1]

        (key2, c2) = nodes[-2]

        nodes = nodes[:-2]

        node = NodeTree(key1, key2)

        nodes.append((node, c1 + c2))

        nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

    huffmanCode = huffman\_code\_tree(nodes[0][0])

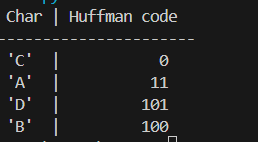
    print(' Char | Huffman code ')

    print('----------------------')

    for (char, frequency) in freq:

        print(' %-4r |%12s' % (char, huffmanCode[char]))

Output:



Write a program to solve a fractional Knapsack problem using a greedy method.

class Item:

    def \_\_init\_\_(self, value, weight):

        self.value = value

        self.weight = weight

def fractionalKnapsack(W, arr):

    arr.sort(key=lambda x: (x.value/x.weight), reverse=True)

    finalvalue = 0.0

    for item in arr:

        if item.weight <= W:

            W -= item.weight

            finalvalue += item.value

        else:

            finalvalue += item.value \* W / item.weight

            break

    return finalvalue

if \_\_name\_\_ == "\_\_main\_\_":

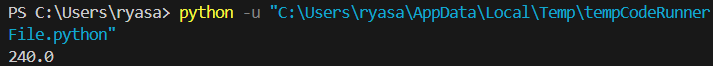
    W = 50

    arr = [Item(60, 10), Item(100, 20), Item(120, 30)]

    max\_val = fractionalKnapsack(W, arr)

    print(max\_val)

Output:



Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

#USING DYNAMIC PROGRAMMING

def knapsack\_dynamic\_programming(values, weights, capacity):

    n = len(values)

    dp = [[0] \* (capacity + 1) for \_ in range(n + 1)]

    for i in range(n + 1):

        for w in range(capacity + 1):

            if i == 0 or w == 0:

                dp[i][w] = 0

            elif weights[i - 1] <= w:

                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w])

            else:

                dp[i][w] = dp[i - 1][w]

    selected\_items = []

    i, w = n, capacity

    while i > 0 and w > 0:

        if dp[i][w] != dp[i - 1][w]:

            selected\_items.append(i - 1)

            w -= weights[i - 1]

        i -= 1

    selected\_items.reverse()

    return dp[n][capacity], selected\_items

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    values = [60, 100, 120]

    weights = [10, 20, 30]

    capacity = 50

    max\_value, selected\_items = knapsack\_dynamic\_programming(values, weights, capacity)

    print("Maximum value in the knapsack:", max\_value)

    print("Selected items:", selected\_items)

Output:



Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen’s matrix

class NQBacktracking:

    def \_\_init\_\_(self):

        self.ld = [0] \* 30

        self.rd = [0] \* 30

        self.cl = [0] \* 30

    def printSolution(self, board):

        print("\n\nN Queen Backtracking Solution:")

        for line in board:

            print(" ".join(map(str, line)))

    def solveNQUtil(self, board, col):

        if col >= N:

            return True

        for i in range(N):

            if (self.ld[i - col + N - 1] != 1 and

                self.rd[i + col] != 1) and self.cl[i] != 1:

                board[i][col] = 1

                self.ld[i - col + N - 1] = self.rd[i + col] = self.cl[i] = 1

                if self.solveNQUtil(board, col + 1):

                    return True

                board[i][col] = 0  # BACKTRACK

                self.ld[i - col + N - 1] = self.rd[i + col] = self.cl[i] = 0

        return False

    def solveNQ(self):

        board = [[0 for \_ in range(N)] for \_\_ in range(N)]

        if self.solveNQUtil(board, 0) == False:

            print("Solution does not exist")

            return False

        self.printSolution(board)

        return True

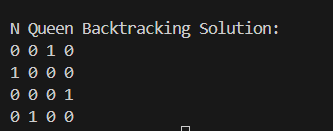
if \_\_name\_\_ == "\_\_main\_\_":

    N = 4

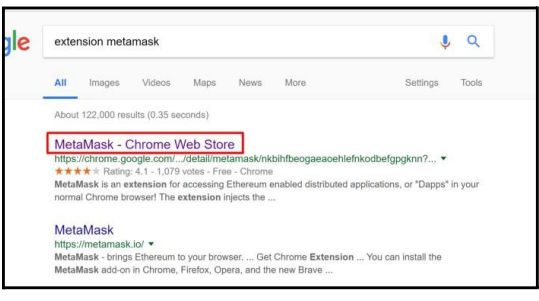
    NQBt = NQBacktracking()

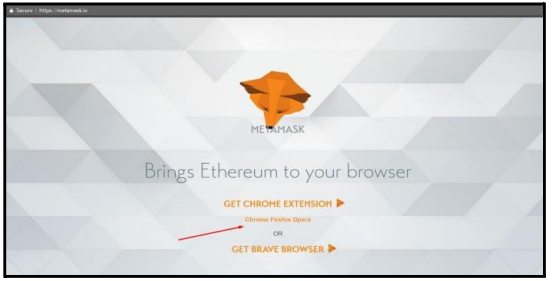
    NQBt.solveNQ()

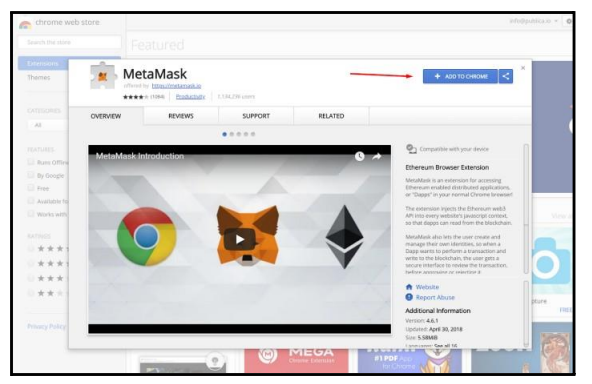
Output:

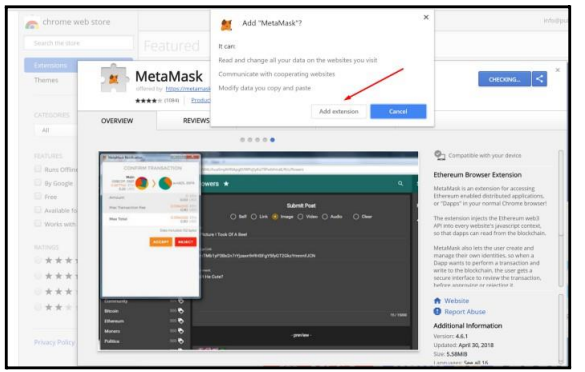


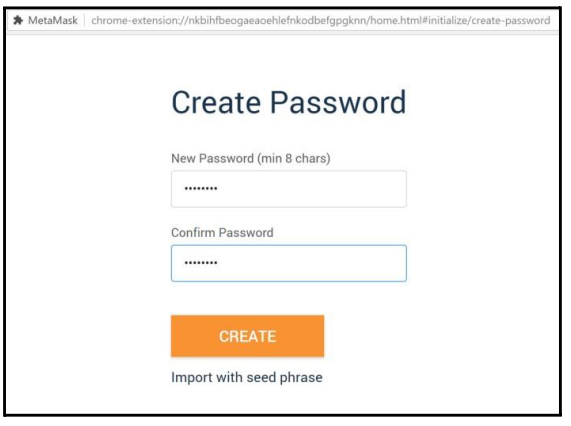
Installation of MetaMask and study spending Ether per transaction

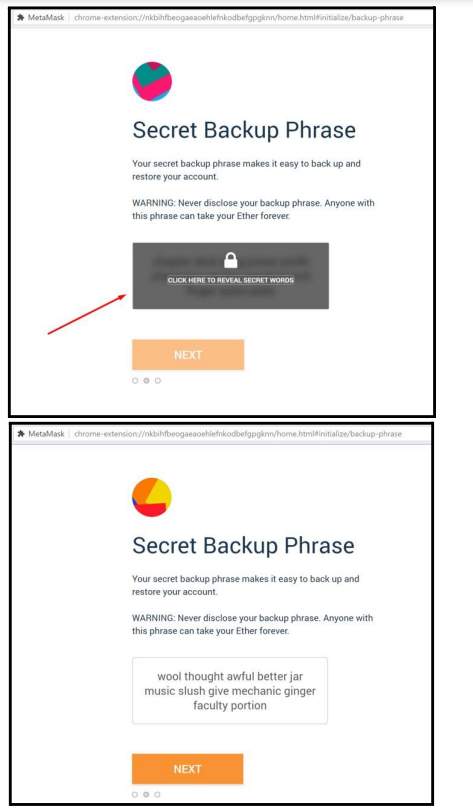
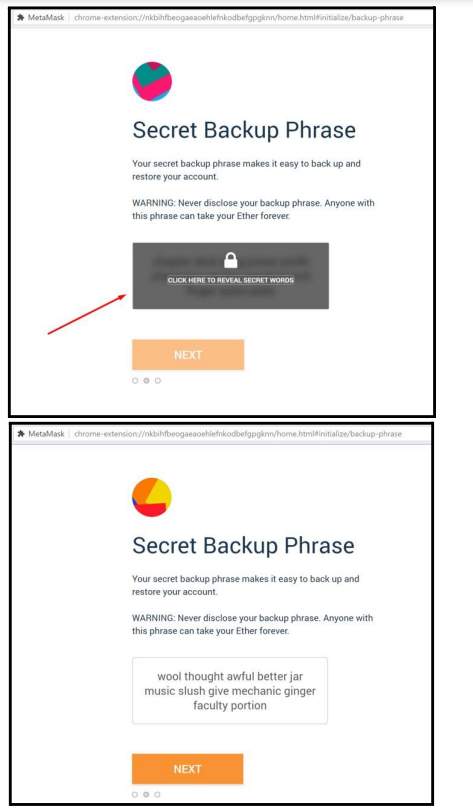


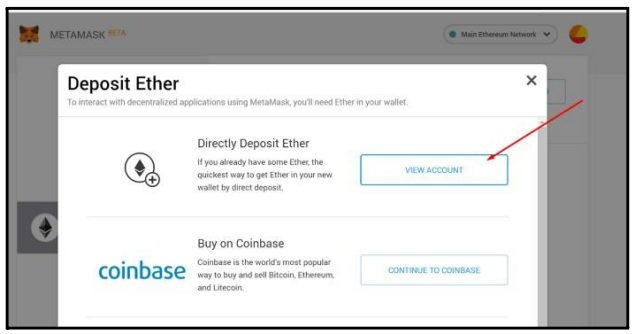


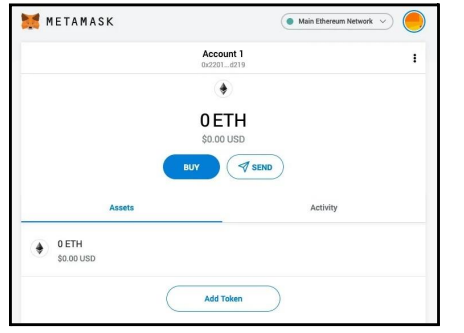


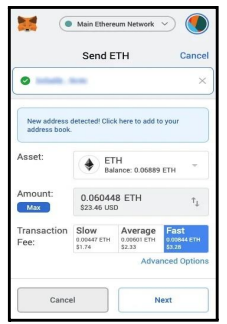




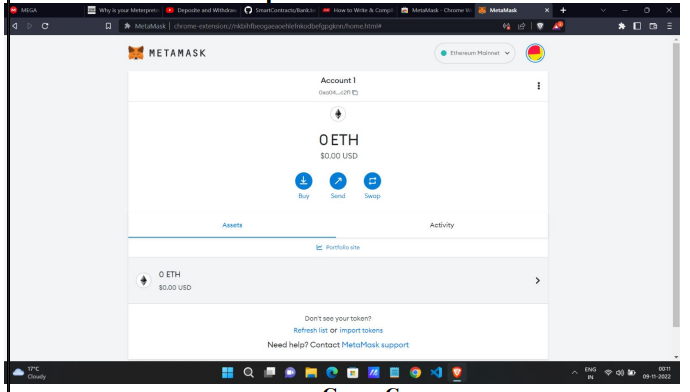








Create your own wallet using Metamask for crypto transactions



Write a smart contract on a test network, for Bank account of a customer for following operations: • Deposit money • Withdraw Money • Show balance

pragma solidity ^0.8.0;

contract BankAccount {

address public owner;

uint256 public balance;

event Deposit(address indexed account, uint256 amount);

event Withdrawal(address indexed account, uint256 amount);

constructor() {

owner = msg.sender;

}

modifier onlyOwner() {

require(msg.sender == owner, "Only the account owner can call this function");

\_;

}

function deposit() public payable {

balance += msg.value;

emit Deposit(msg.sender, msg.value);

}

function withdraw(uint256 amount) public onlyOwner {

require(amount <= balance, "Insufficient funds");

balance -= amount;

payable(msg.sender).transfer(amount);

emit Withdrawal(msg.sender, amount);

}

function getBalance() public view returns (uint256) {

return balance;}

}

Write a program in solidity to create Student data. and Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values

pragma solidity ^0.8.0;

contract StudentData {

// Structure to represent a student

struct Student {

string name;

uint256 age;

}

// Array to store multiple students

Student[] public students;

// Event to log student data addition

event StudentAdded(string name, uint256 age);

// Counter for the number of times Ether is sent

uint256 public etherSentCount;

// Fallback function to receive Ether and increment the counter

receive() external payable {

etherSentCount++;

}

// Function to add a new student

function addStudent(string memory name, uint256 age) public {

Student memory newStudent = Student(name, age);

students.push(newStudent);

emit StudentAdded(name, age);

}

// Function to get the count of students

function getStudentsCount() public view returns (uint256) {

return students.length;

}

// Function to get a specific student's details by index

function getStudent(uint256 index) public view returns (string memory, uint256) {

require(index < students.length, "Student index out of range");

Student memory student = students[index];

return (student.name, student.age);

}

}

