**A\*STAR**

import heapq

def heu(a,b):

return abs(a[0]-b[0])+abs(a[1]-b[1])

def astar(grid,start,goal):

open\_list=[(0,start)]

come\_from={}

g\_score={start:0}

while open\_list:

\_,current=heapq.heappop(open\_list)

if current== goal:

path=[]

while current in come\_from:

path.append(current)

current=come\_from[current]

return [start]+path[::-1]

for dx,dy in([0,1],[0,-1],[-1,0],[1,0]):

neb=(current[0]+dx,current[1]+dy)

if 0<=neb[0]<len(grid) and 0<=neb[1]<len(grid[0]) and grid[neb[0]][neb[1]]==0:

tentitative\_g=g\_score[current]+1

if neb is not g\_score or tentitative\_g< g\_score[neb]:

g\_score[neb]=tentitative\_g

f=tentitative\_g+heu(neb,goal)

heapq.heappush(open\_list,(f,neb))

come\_from[neb]=current

return None

grid=[

[0,0,1,0],

[0,1,0,1],

[0,0,0,0],

[1,1,1,0]

]

start,goal=(0,0),(3,3)

path=astar(grid,start,goal)

print("path found:",path if path else "not found")

**AND XOR**

s="hello world "

print("original\tAND 127 \tXOR 127")

for p in s:

and\_res=ord(p) & 127

xor\_res=ord(p)^127

print(f"{p}({ord(p)})\t\t {and\_res}\t\t {xor\_res}")

**CHATBOT**

import re

def chatbot\_response(user\_input):

user\_input = user\_input.lower().strip()

responses = {

r"(hello|hi|hey)": "Hello! Welcome to our Customer Support. How can I help you today?",

r"(how are you)": "I'm doing great! How can I assist you?",

r"(help)": """Here are the services I can help you with:

1. Track Order

2. Cancel Order

3. Return Product

4. Offers and Discounts

5. Payment Options

6. Talk to Support

Type the number or the service name to continue.""",

r"(1|track order)": "Please provide your Order ID to track your order.",

r"(2|cancel order)": "To cancel your order, please go to 'My Orders' section or type your Order ID here.",

r"(3|return product)": "You can return the product within 7 days. Type your Order ID to initiate return.",

r"(4|offers|discounts)": "Use code SAVE10 for 10% off on your first order!",

r"(5|payment |payment methods)": "We accept UPI, credit/debit cards, and wallets.",

r"(6|support|talk to support|customer care)": "You can reach our customer care at 1800-111-222.",

r"(bye|exit|thank you)": "Thank you for using our support service. Have a great day!"

}

for pattern, response in responses.items():

if re.search(pattern, user\_input):

return response

return "I'm sorry, I didn't understand that. Can you please rephrase?"

# Chatbot interaction loop

print("Bot: Hello! I'm your virtual assistant. Type 'help' to see what I can do. Type 'exit' to leave.")

while True:

user\_input = input("You: ")

if user\_input.lower() in ["exit", "bye"]:

print("Bot: Thank you! Goodbye!!!")

break

print("Bot:", chatbot\_response(user\_input))

**DES**

from Crypto.Cipher import DES

from Crypto.Util.Padding import pad, unpad

def des\_encrypt(message, key):

cipher = DES.new(key, DES.MODE\_CBC)

encrypted = cipher.encrypt(pad(message.encode(), DES.block\_size))

return cipher.iv + encrypted

def des\_decrypt(encrypted, key):

iv = encrypted[:DES.block\_size]

cipher = DES.new(key, DES.MODE\_CBC, iv=iv)

decrypted = unpad(cipher.decrypt(encrypted[DES.block\_size:]), DES.block\_size)

return decrypted.decode()

# User input

key\_input = input("Enter 8-character key for DES encryption: ").encode('utf-8')[:8]

message = input("Enter message to encrypt: ")

# Encryption

encrypted = des\_encrypt(message, key\_input)

decrypted = des\_decrypt(encrypted, key\_input)

# Print the output

print(f"\nOriginal text: {message}")

print(f"Encrypted text (in hex): {encrypted.hex()}")

print(f"Decrypted text: {decrypted}")

**DFS+BFS**

from collections import deque

def dfs(visited, graph, node):

if node not in visited:

print(node, end=" ")

visited.add(node)

for neighbour in graph.get(node, []):

dfs(visited, graph, neighbour)

def bfs(visited, graph, start):

queue = deque([start])

visited.add(start)

while queue:

node = queue.popleft()

print(node, end=" ")

for neighbour in graph.get(node, []):

if neighbour not in visited:

visited.add(neighbour)

queue.append(neighbour)

def main():

graph = {}

n = int(input("Enter number of nodes: "))

for i in range(1, n+1):

edges = int(input(f"Enter number of edges for node {i}: "))

graph[i] = []

for j in range(edges):

dest = int(input(f"Enter edge {j+1} for node {i}: "))

graph[i].append(dest)

if dest not in graph:

graph[dest] = []

print("The following is DFS:")

dfs(set(), graph, 1)

print("\nThe following is BFS:")

bfs(set(), graph, 1)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**ENCRY\_DECRY**

def encrypt\_message(msg, key):

cipher = [''] \* key

for col in range(key):

pointer = col

while pointer < len(msg):

cipher[col] += msg[pointer]

pointer += key

return ''.join(cipher)

def decrypt\_message(cipher, key):

num\_rows = len(cipher) // key

if len(cipher) % key != 0:

num\_rows += 1

plain = [''] \* len(cipher)

index = 0

for col in range(key):

pointer = col

while pointer < len(cipher):

plain[pointer] = cipher[index]

index += 1

pointer += key

return ''.join(plain)

message = input("Enter message: ")

key = int(input("Enter key: "))

encrypted = encrypt\_message(message, key)

print("Encrypted:", encrypted)

decrypted = decrypt\_message(encrypted, key)

print("Decrypted:", decrypted)

**GREEDY**

def sel(arr):

n=len(arr)

for i in range(n):

min\_index=i

for j in range(i+1,n):

if arr[j]<arr[min\_index]:

min\_index=j

arr[i],arr[min\_index]=arr[min\_index],arr[i]

input=input("nu")

arr=list(map(int,input.split()))

print("or: ",arr)

sel(arr)

print("so ",arr)

**HELLMAN**

<!DOCTYPE html>

<html>

<head>

<title>Diffie-Hellman Key Exchange</title>

<style>

body { font-family: Arial; padding: 20px; }

input, button { padding: 6px; margin: 6px 0; }

</style>

</head>

<body>

<h2>Diffie-Hellman Key Exchange</h2>

<label>Prime (p):</label><br>

<input type="number" id="prime" value="23"><br>

<label>Base (g):</label><br>

<input type="number" id="base" value="5"><br>

<label>Your Secret Key (Alice - a):</label><br>

<input type="number" id="aliceSecret" value="6"><br>

<button onclick="performDH()">Exchange Keys</button>

<h3>Results</h3>

<p><strong>Alice's Public Key:</strong> <span id="alicePub"></span></p>

<p><strong>Bob's Secret Key (hidden):</strong> <span id="bobSecret"></span></p>

<p><strong>Bob's Public Key:</strong> <span id="bobPub"></span></p>

<p><strong>Shared Secret (Alice):</strong> <span id="aliceShared"></span></p>

<p><strong>Shared Secret (Bob):</strong> <span id="bobShared"></span></p>

<script>

function modPow(base, exponent, modulus) {

if (modulus === 1) return 0;

let result = 1;

base = base % modulus;

while (exponent > 0) {

if (exponent % 2 === 1) result = (result \* base) % modulus;

exponent = Math.floor(exponent / 2);

base = (base \* base) % modulus;

}

return result;

}

function performDH() {

const p = parseInt(document.getElementById('prime').value);

const g = parseInt(document.getElementById('base').value);

const a = parseInt(document.getElementById('aliceSecret').value);

const A = modPow(g, a, p); // Alice's public key

const b = Math.floor(Math.random() \* (p - 2)) + 2; // Bob's secret key (2 ≤ b < p)

const B = modPow(g, b, p); // Bob's public key

const sharedAlice = modPow(B, a, p); // Shared secret at Alice

const sharedBob = modPow(A, b, p); // Shared secret at Bob

document.getElementById('alicePub').innerText = A;

document.getElementById('bobSecret').innerText = b; // Show for demonstration only

document.getElementById('bobPub').innerText = B;

document.getElementById('aliceShared').innerText = sharedAlice;

document.getElementById('bobShared').innerText = sharedBob;

}

</script>

</body>

</html>

**MEDICAL\_ES**

def evaluate\_patient():

print("=== Expert System: Patient Health Evaluation ===")

name = input("Enter Patient Name: ")

age = int(input("Enter Age: "))

print("rate 5:low, 1:high")

symptoms = ["Fever", "Cough", "Headache", "Fatigue", "Shortness of Breath", "Sore Throat"]

ratings = [int(input(f"Rate {symptom} (1-5): ")) for symptom in symptoms]

avg\_score = sum(ratings) / len(ratings)

if avg\_score >= 4.5:

health\_status, recommendation = "Excellent", "No significant health concerns. Maintain a healthy lifestyle."

elif avg\_score >= 3.5:

health\_status, recommendation = "Good", "Minor symptoms. Rest and stay hydrated."

elif avg\_score >= 2.5:

health\_status, recommendation = "Moderate", "Symptoms suggest possible illness. Seek medical attention."

else:

health\_status, recommendation = "Critical", "Severe symptoms detected. Immediate medical consultation required."

print(f"\nPatient Name: {name}, Age: {age}")

print(f"Health Status: {health\_status}, Average Score: {avg\_score:.2f}/5")

print(f"Recommendation: {recommendation}")

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

evaluate\_patient()

**N\_QUEEN\_CSP**

def is\_safe(board, x, y, n):

for i in range(x):

if board[i][y] or (y - x + i >= 0 and board[i][y - x + i]) or (y + x - i < n and board[i][y + x - i]):

return False

return True

def solve(board, x, n):

if x == n:

return True

for y in range(n):

if is\_safe(board, x, y, n):

board[x][y] = 1

if solve(board, x + 1, n):

return True

board[x][y] = 0

return False

def main():

n = int(input("Enter number of Queens: "))

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

if solve(board, 0, n):

for row in board:

print(\*row)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**RSA**

import random

from sympy import isprime

# Function to generate a prime number of given bit size

def generate\_prime(bits):

while True:

num = random.getrandbits(bits)

if isprime(num):

return num

# Greatest Common Divisor

def gcd(a, b):

while b != 0:

a, b = b, a % b

return a

# Modular inverse using Extended Euclidean Algorithm

def mod\_inverse(e, phi):

for d in range(3, phi, 2):

if (d \* e) % phi == 1:

return d

return None

# Function to generate public and private keys

def generate\_keys(bits=16):

p = generate\_prime(bits)

q = generate\_prime(bits)

while p == q:

q = generate\_prime(bits)

n = p \* q

phi = (p - 1) \* (q - 1)

e = 3

while gcd(e, phi) != 1:

e += 2

d = mod\_inverse(e, phi)

return (e, n), (d, n)

# Encrypt function

def encrypt(plaintext, public\_key):

e, n = public\_key

cipher = [pow(ord(char), e, n) for char in plaintext]

return cipher

# Decrypt function

def decrypt(ciphertext, private\_key):

d, n = private\_key

plaintext = ''.join([chr(pow(char, d, n)) for char in ciphertext])

return plaintext

# Main code

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

print("=== RSA Encryption and Decryption ===\n")

# Generate keys

public\_key, private\_key = generate\_keys(bits=16)

print("Public Key :", public\_key)

print("Private Key :", private\_key)

# User input

message = input("\nEnter message to encrypt: ")

# Encrypt the message

encrypted\_msg = encrypt(message, public\_key)

print("Encrypted Message :", encrypted\_msg)

# Decrypt the message

decrypted\_msg = decrypt(encrypted\_msg, private\_key)

print("Decrypted Message :", decrypted\_msg)