Experiment # 7: Implement a Cryptographic Arithmetic Problem

Aim/Objective:

Implement a Cryptographic Arithmetic Problem

Description:

The aim of this lab experiment is to implement Cryptographic arithmetic problems typically involve operations on large integers where the details of the numbers are obscured or hidden through encryption techniques. These problems are often used in cryptography for secure communication, digital signatures, and other cryptographic protocols.

Pre-Requisites:

a. To effectively understand and work with cryptographic arithmetic, especially in the context of cryptographic protocols like RSA, several prerequisites are essential. These prerequisites span mathematical concepts, computer science fundamentals, and specific knowledge related to cryptography.
b. Familiarity with programming concepts and basic knowledge of Python programminglanguage

Pre-Lab:

- Define Constraint Satisfaction Problem (CSP)?

A **Constraint Satisfaction Problem (CSP)** involves finding values for variables that satisfy a set of constraints. Each variable has a domain of possible values, and the goal is to assign a value to each variable such that all constraints are met.

- Give an example of a cryptographic arithmetic problem that can be modeled as a CSP.

A **cryptographic problem** such as finding the private key in RSA encryption can be modeled as a CSP. The variables would be the prime factors of a large number (public key), and the constraints would ensure that the correct factorization is found (such as the properties of modular arithmetic for decryption).

- What are the advantages of using CSPs in cryptography?

- **Structured Search**: CSPs offer a systematic way to explore potential solutions, reducing unnecessary calculations.
- **Efficient Solution Finding**: They allow for pruning of invalid paths early, improving computational efficiency.
- **Modeling Complex Problems**: CSPs can model the relationships between cryptographic variables, making it easier to define and solve security-related problems.

- How can CSPs contribute to the security and efficiency of cryptographic protocols?

CSPs can improve security by ensuring that cryptographic keys and operations satisfy certain conditions, making unauthorized access or attacks more difficult. They also contribute to efficiency by reducing the search space for key generation or algorithm execution, speeding up cryptographic processes.

In-Lab:

Implement a Cryptographic Arithmetic Problem to solve the following equation

C R O S S + R O A D S = D A N G E R

AIM: Students will be able to apply constraint satisfaction problem to solve Cryptographic Arithmetic Problem.

Description: Crypto-arithmetic problems, also known as verbal arithmetic or alphabetic puzzles, are a type of constraint satisfaction problem (CSP) where digits are assigned to letters to satisfy a given arithmetic equation.

Procedure/Program:

```
from itertools import permutations

def solve_crypto_arithmetic():
    letters = 'CROSSROADSANGER'
    unique_letters = ''.join(set(letters))

    if len(unique_letters) > 10:
        print("Too many unique letters for a solution!")
        return

    for perm in permutations(range(10), len(unique_letters)):
        mapping = dict(zip(unique_letters, perm))

        CROSS = 10000 * mapping['C'] + 1000 * mapping['R'] + 100 * mapping['O'] +
10 * mapping['S'] + mapping['S']
        ROADS = 10000 * mapping['R'] + 1000 * mapping['O'] + 100 * mapping['A'] +
10 * mapping['D'] + mapping['S']
        DANGER = 100000 * mapping['D'] + 10000 * mapping['A'] + 1000 *
mapping['N'] + 100 * mapping['G'] + 10 * mapping['E'] + mapping['R']

        if (CROSS + ROADS == DANGER and
            mapping['C'] != 0 and mapping['R'] != 0 and mapping['D'] != 0):
            print(f"SOLUTION FOUND:")
            print(f"CROSS = {CROSS}, ROADS = {ROADS}, DANGER = {DANGER}")
            print(f"Mapping: {mapping}")
            return

    print("No solution found!")
```

solve_crypto_arithmetic()

OUTPUT

SOLUTION FOUND:
CROSS = 96233, ROADS = 62513, DANGER = 158746
Mapping: {'D': 1, 'N': 8, 'C': 9, 'R': 6, 'A': 5, 'O': 2, 'S': 3, 'E': 4, 'G': 7}

Data and Results:

Data:

This problem involves assigning unique digits to letters to satisfy a cryptographic arithmetic equation.

Result:

The valid solution assigns digits to letters, ensuring the equation CROSS + ROADS = DANGER holds true.

Analysis and Inferences:

Analysis:

By testing all digit combinations, we ensured no violations of constraints such as leading zeros.

Inferences:

Constraint satisfaction algorithms can efficiently solve cryptographic puzzles by systematically exploring all possible valid solutions.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT] THANOS

VIVA-VOCE Questions (In-Lab):

1. What are the basic principles of cryptographic arithmetic, and how do they ensure data security?

Cryptographic arithmetic uses modular arithmetic, exponentiation, and prime numbers to encrypt and decrypt data. These operations ensure data security by making it hard to reverse the process without the correct key.

2. How do you implement basic cryptographic operations such as encryption and decryption using arithmetic techniques?

- **Encryption:** In symmetric encryption, the plaintext is transformed using a key. In asymmetric encryption, the plaintext is raised to an exponent modulo a prime.
- **Decryption:** For symmetric encryption, the same key is used. In RSA, the private key reverses the encryption process.

3. Explain the n-Queen problem and the role of forward-checking in solving it.

The n-Queen problem places `n` queens on an `n x n` board without conflicts. Forward-checking eliminates invalid moves early, improving efficiency and reducing backtracking.

4. What are the advantages of using forward-checking in constraint satisfaction problems like the n-Queen problem?

- Reduces backtracking.
- Increases efficiency by pruning invalid options.
- Maintains consistency with constraints early in the process.

5. How do you determine the efficiency of your cryptographic arithmetic implementation?

Efficiency is measured by time complexity (encryption/decryption time), space complexity (memory usage), security level (resistance to attacks), and performance benchmarks (throughput and latency).

Post-Lab:

Implement an n-Queen problem using forward-checking

Description: Implementing the n-Queens problem using forward-checking involves using constraint satisfaction techniques to find all possible placements of n queens on n×n chessboard such that no two queens threaten each other. Forward-checking is a technique that reduces the domain of variables (in this case, potential queen positions) by considering the constraints imposed by already placed queens.

Procedure/Program:

```
def is_safe(board, row, col, n):
    for i in range(row):
        if board[i] == col or \
           board[i] - i == col - row or \
           board[i] + i == col + row:
            return False
    return True

def forward_check(board, row, n):
    restricted_cols = set()
    restricted_diagonals1 = set()
    restricted_diagonals2 = set()

    for i in range(row):
        restricted_cols.add(board[i])
        restricted_diagonals1.add(i - board[i])
        restricted_diagonals2.add(i + board[i])

    return restricted_cols, restricted_diagonals1, restricted_diagonals2

def solve_n_queens(n):
    board = [-1] * n
    solutions = []

    def backtrack(row):
        if row == n:
            solutions.append(board[:])
            return

        restricted_cols, restricted_diagonals1, restricted_diagonals2 =
forward_check(board, row, n)
```

```
        for col in range(n):
            if col not in restricted_cols and \
                (row - col) not in restricted_diagonals1 and \
                (row + col) not in restricted_diagonals2:
                board[row] = col
                backtrack(row + 1)
                board[row] = -1

    backtrack(0)
    return solutions

def print_solution(solution, n):
    for row in solution:
        board = ['Q' if i == row else '.' for i in range(n)]
        print(" ".join(board))
    print("\n")

n = 4
solutions = solve_n_queens(n)
print(f"Total solutions for {n}-Queens: {len(solutions)}")
for solution in solutions:
    print_solution(solution, n)
```

OUTPUT

```
Total solutions for 4-Queens: 2
. Q . .
. . . Q
Q . . .
. . Q .


. . Q .
Q . . .
. . . Q
. Q . .
```

Data and Results:

Data:

This section represents the initial input for solving the n-Queens problem.

Result:

The algorithm successfully solves the n-Queens problem for all sizes.

Analysis and Inferences:

Analysis:

The backtracking with forward-checking significantly reduces search space and time.

Inferences:

Forward-checking optimizes the solution, improving efficiency and reducing computations.

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date