S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR

Practical No. 04

Aim: Write a code to Implement Cryptarithmetic puzzle (SEND + MORE=MONEY) using the concept of constraint satisfaction concept.

Name of Student: Shrutika Pradeep Bagdi

Roll No.: CS22130

Semester/Year: V/III

Academic Session: 2023-2024

Date of Performance:

Date of Submission:

AIM: Write a code to Implement Cryptarithmetic puzzle (SEND + MORE=MONEY) using the concept of constraint satisfaction concept.

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- To be able to understand the concept of constraint satisfaction.
- To be able to acquire the puzzle solving capability.
- To develop the coding ability for cryptarithmetic puzzle.

THEORY:

Finding a solution that meets a set of constraints is the goal of constraint satisfaction problems (CSPs), a type of AI issue. Finding values for a group of variables that fulfill a set of restrictions or rules is the aim of constraint satisfaction problems. For tasks including resource allocation, planning, scheduling, and decision-making, CSPs are frequently employed in AI.

There are mainly three basic components in the constraint satisfaction problem: **Variables:** The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints. Boolean, integer, and categorical variables are just a few examples of the various types of variables Variables, for instance, could stand in for the many puzzle cells that need to be filled with numbers in a sudoku puzzle.
Domains: The range of potential values that a variable can have is represented by domains. Depending on the issue, a domain may be finite or limitless. For instance, in Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable representing a problem cell. **Constraints:** The guidelines that control how variables relate to one another are known as constraints. Constraints in a CSP define the ranges of possible values for variables. Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints. For instance, in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9. Constraint Satisfaction Problems (CSP) representation:

- The finite set of variables V_1, V_2, V_3 ……..V_n.
- Non-empty domain for every single variable D_1, D_2, D_3 ……..D_n.
- The finite set of constraints C_1, C_2 ……., Cm.
    - where each constraint C_i restricts the possible values for variables,
        - e.g., $V_1 \neq V_2$
- Each constraint C_i is a pair <scope, relation>
    - Example: <(V_1, V_2), V_1 not equal to V_2>
    - Scope = set of variables that participate in constraint.
    - Relation = list of valid variable value combinations.
- There might be a clear list of permitted combinations. Perhaps a relation that is abstract and that allows for membership testing and listing.

Constraint Satisfaction Problems (CSP) algorithms:

- The **backtracking algorithm** is a depth-first search algorithm that methodically investigates the search space of potential solutions up until a solution is discovered that satisfies all the restrictions. The method begins by choosing a variable and giving it a value before repeatedly attempting to give

values to the other variables. The method returns to the prior variable and tries a different value if at any time a variable cannot be given a value that fulfills the requirements. Once all assignments have been tried or a solution that satisfies all constraints has been discovered, the algorithm ends.

• The **forward-checking algorithm** is a variation of the backtracking algorithm that condenses the search space using a type of local consistency. For each unassigned variable, the method keeps a list of remaining values and applies local constraints to eliminate inconsistent values from these sets. The algorithm examines a variable's neighbors after it is given a value to see whether any of its remaining values become inconsistent and removes them from the sets if they do. The algorithm goes backward if, after forward checking, a variable has no more values.

• Algorithms for **propagating constraints** are a class that uses local consistency and inference to condense the search space. These algorithms operate by propagating restrictions between variables and removing inconsistent values from the variable domains using the information obtained.

CRYPTARITHMETIC PROBLEM

Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols. In cryptarithmetic problem, the digits (0-9) get substituted by some possible alphabets or symbols. The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct.

We can perform all the arithmetic operations on a given cryptarithmetic problem.

The rules or constraints on a cryptarithmetic problem are as follows:

• There should be a unique digit to be replaced with a unique alphabet.

• The result should satisfy the predefined arithmetic rules, i.e., 2+2 =4, nothing else.

• Digits should be from 0-9 only.

• There should be only one carry forward, while performing the addition operation on a problem.

• The problem can be solved from both sides, i.e., lefthand side (L.H.S), or righthand side (R.H.S)

Given an array of strings, **arr[]** of size **N** and a string **S**, the task is to find if it is possible to map integers value in the range **[0, 9]** to every alphabet that occurs in the strings, such that the sum obtained after summing the numbers formed by encoding all strings in the array is equal to the number formed by the string **S**.

Examples:

Input: *arr[][] = {"SEND", "MORE"}, S = "MONEY"*

Output: *Yes*

Explanation:

One of the possible ways is:

1. *Map the characters as the following, 'S'? 9, 'E'?5, 'N'?6, 'D'?7, 'M'?1, 'O'?0, 'R'?8, 'Y'?2.*
2. *Now, after encoding the strings "SEND", "MORE", and "MONEY", modifies to 9567, 1085 and 10652 respectively.*
3. *Thus, the sum of the values of "SEND" and "MORE" is equal to (9567+1085 = 10652), which is equal to the value of the string "MONEY".*

ALGORITHM:

• First, create a list of all the characters that need assigning to pass to Solve • If all characters are assigned, return true if puzzle is solved, false otherwise • Otherwise, consider the first unassigned character

- for (every possible choice among the digits not in use)
  - If all digits have been tried and nothing worked, return false to trigger backtracking

PSEUDOCODE, in this case, has more special cases, but the same general design

- Start by examining the rightmost digit of the topmost row, with a carry of 0
- If we are beyond the leftmost digit of the puzzle, return true if no carry, false otherwise
- If we are currently trying to assign a char in one of the addends If char already assigned, just recur on the row beneath this one, adding value into the sum If not assigned, then for (every possible choice among the digits not in use) make that choice and then on row beneath this one, if successful, return true if !successful, unmake assignment and try another digit return false if no assignment worked to trigger backtracking Else if trying to assign a char in the sum.
- If char assigned & matches correct, recur on next column to the left with carry, if success return true,
- If char assigned & doesn't match, return false
- If char unassigned & correct digit already used, return false
- If char unassigned & correct digit unused, assign it and recur on next column to left with carry, if success return true
- return false to trigger backtracking .

CODE:

```
def cryptoarithmatic():
    for S in range(1, 10):
        for E in range(10):
            for N in range(10):
                for D in range(10):
                    for M in range(1, 10):
                        for O in range(10):
                            for R in range(10):
                                for Y in range(10):
                                    # Ensure that all digits are distinct
                                    if len(set([S, E, N, D, M, O, R, Y])) == 8:
                                        send = 1000 * S + 100 * E + 10 * N + D
                                        more = 1000 * M + 100 * O + 10 * R + E
                                        money = 10000 * M + 1000 * O + 100 * N + 10 * E + Y

                                        if send + more == money:
                                            print(f"S = {S}, E = {E}, N = {N}, D = {D}")
                                            print(f"M = {M}, O = {O}, R = {R}, E = {E}")
                                            print(f"Y = {Y}")
                                            print(f"SEND = {send}")
                                            print(f"MORE = {more}")
                                            print(f"MONEY = {money}")

cryptoarithmatic()
```

INPUT & OUTPUT:

```
S = 9, E = 5, N = 6, D = 7
M = 1, O = 0, R = 8, E = 5
Y = 2
SEND = 9567
MORE = 1085
MONEY = 10652
```

CONCLUSION: Thus, I created a code for cryptarithmetic puzzle which gives us unique digit number for the above input.

DISCUSSION QUESTIONS:

1) How constraint satisfaction helps to solve cryptarithmetic puzzle?

2) Give the solution from right side take any example.

3) What are the problems solved by using CSP ?

REFERENCES:

- Constraint Satisfaction Problems in Artificial Intelligence - Javatpoint
- Solving Cryptarithmetic Puzzles - GeeksforGeeks
- Cryptarithmetic Problem in AI - TAE (tutorialandexample.com)
- Cryptarithmetic Puzzles | OR-Tools | Google for Developers
- Cryptarithm Solver - Online Cryptarithmetic Puzzle Calculator (dcode.fr)