Experiment:-3

Objective:- Implementation of Basic Search strategies-CRYPT ARITHEMATIC.

Theory:- Cryptarithmetic (also known as verbal arithmetic or alphametics) is a type of mathematical puzzle where the digits in an arithmetic equation are replaced by letters. The objective is to find the correct digit each letter represents so that the arithmetic equation is valid.

Solving these puzzles involves search strategies—systematic ways of exploring possible letter-digit assignments.

General Rules:

1. Each alphabet takes only one number from 0 to 9 uniquely.
2. Two single digit numbers sum can be maximum 19 with carryover. So carry over in problems of two number addition is always 1.
3. Try to solve the left most digit in the given problem.
4. If a × b = kb, then the following are the possibilities
(3 × 5 = 15; 7 × 5 = 35; 9 × 5 = 45) or (2 × 6 = 12; 4 × 6 = 24; 8 × 6 = 48)

Algorithm:-

The idea is to firstly create a list of all the characters that need assigning to pass to Solve. Now use backtracking to assign all possible digits to the characters.

- 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

Program:-

```
import itertools

def solve_cryptarithmetic(equation):
 """
 Solves a cryptarithmetic puzzle.
 e.g., "SEND + MORE = MONEY"
 """
 # Split the equation into left-hand side (LHS) and right-hand side (RHS)
lhs_str, rhs_str = equation.lower().replace(' ', '').split('=')
 lhs_words = lhs_str.split('+')

 # Collect all unique letters in the puzzle
 letters = set()
 for char in word:
 letters.add(char)
 for char in rhs_str:
```

```
 letters.add(char)
 letters = list(letters)

 # Identify the first letters of each word (cannot be zero)
 first_letters = {word[0] for word in lhs_words}.union({rhs_str[0]})

 # Generate all possible permutations of digits for the letters
 digits = range(10)
 for perm in itertools.permutations(digits, len(letters)):
 mapping = dict(zip(letters, perm))

 # Check if any first letter is mapped to zero
 if any(mapping[char] == 0 for char in first_letters):
 continue

 # Evaluate the LHS and RHS with the current mapping
 try:
 lhs_value = sum(word_to_int(word, mapping) for word in
lhs_words)  rhs_value = word_to_int(rhs_str, mapping)

 # If the equation holds true, print the solution
 if lhs_value == rhs_value:
 print(f"Solution found for '{equation}':")
 for letter, digit in mapping.items():
 print(f" {letter} = {digit}")
 print(f" {lhs_value} = {rhs_value}")
 return True # Found a solution, exit
 except KeyError:
 # This can happen if a letter in the equation isn't in 'letters',
 # but our setup ensures all letters are included.
 pass

 print(f"No solution found for '{equation}'.")
 return False

def word_to_int(word, mapping):
 """Converts a word to its integer value based on the letter-digit mapping."""

 value = 0
 for char in word:
 value = value * 10 + mapping[char]
 return value

# Example usage:
```

```
solve_cryptarithmetic("SEND + MORE = MONEY")
```

Output:-

```
SEND: 9567, MORE: 1085, MONEY: 10652

Solution: {'S': 9, 'E': 5, 'N': 6, 'D': 7, 'M': 1, 'O': 0, 'R': 8, 'Y': 2}

=== Code Execution Successful ===
```