SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6

BATCH-1

REGISTRATION NUMBER	**RA1911003011013**
NAME	**Shivank Arora**

B.Tech- CSE / CC, Third Year (Section: H2)

Faculty In charge: Dr. S. Prabakeran, B.Tech, M.E, PhD
Assistant Professor
School of Computing - Department of Networking and Communications

Year 2021-2022 / Even Semester

INDEX

Ex No	DATE	Title	Page No	Marks
1	07/02/22	**Implementation of constraint satisfaction problems** (Cryptarithmetic problem)		

Exercise: 3

Date : 1/02/22

Implementation of constraint satisfaction problems

Cryptarithmetic Problem

Problem Statement : The goal here is to assign each letter a digit from 0 to 9 so that the arithmetic works out correctly. The rules are that all occurrences of a letter must be assigned the same digit, and no digit can be assigned to more than one letter.

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)

make that choice and then recursively try to assign the rest of the characters
if recursion successful, return true
if !successful, unmake assignment and try another digit

Optimization technique : The algorithm above actually has a lot in common with the permutations algorithm, it pretty much just creates all arrangements of the mapping from characters to digits and tries each until one works or all have been successfully tried. For a large puzzle, this could take a while.
A smarter algorithm could take into account the structure of the puzzle and avoid going down dead-end paths. For example, if we assign the characters starting from the one's place and moving to the left, at each stage, we can verify the correctness of what we have so far before we continue onwards. This definitely complicates the code but leads to a tremendous improvement in efficiency, making it much more feasible to solve large puzzles.

- 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.

Tool : aws cloud9 and Python 3.9.0

Programming code :

```
import itertools

import pdb

def get_val(word, substitution):

    s = 0

    factor = 1

    for let in reversed(word):

        s += factor * substitution[let]

        factor *= 10

    return s

def solve(equation):
```

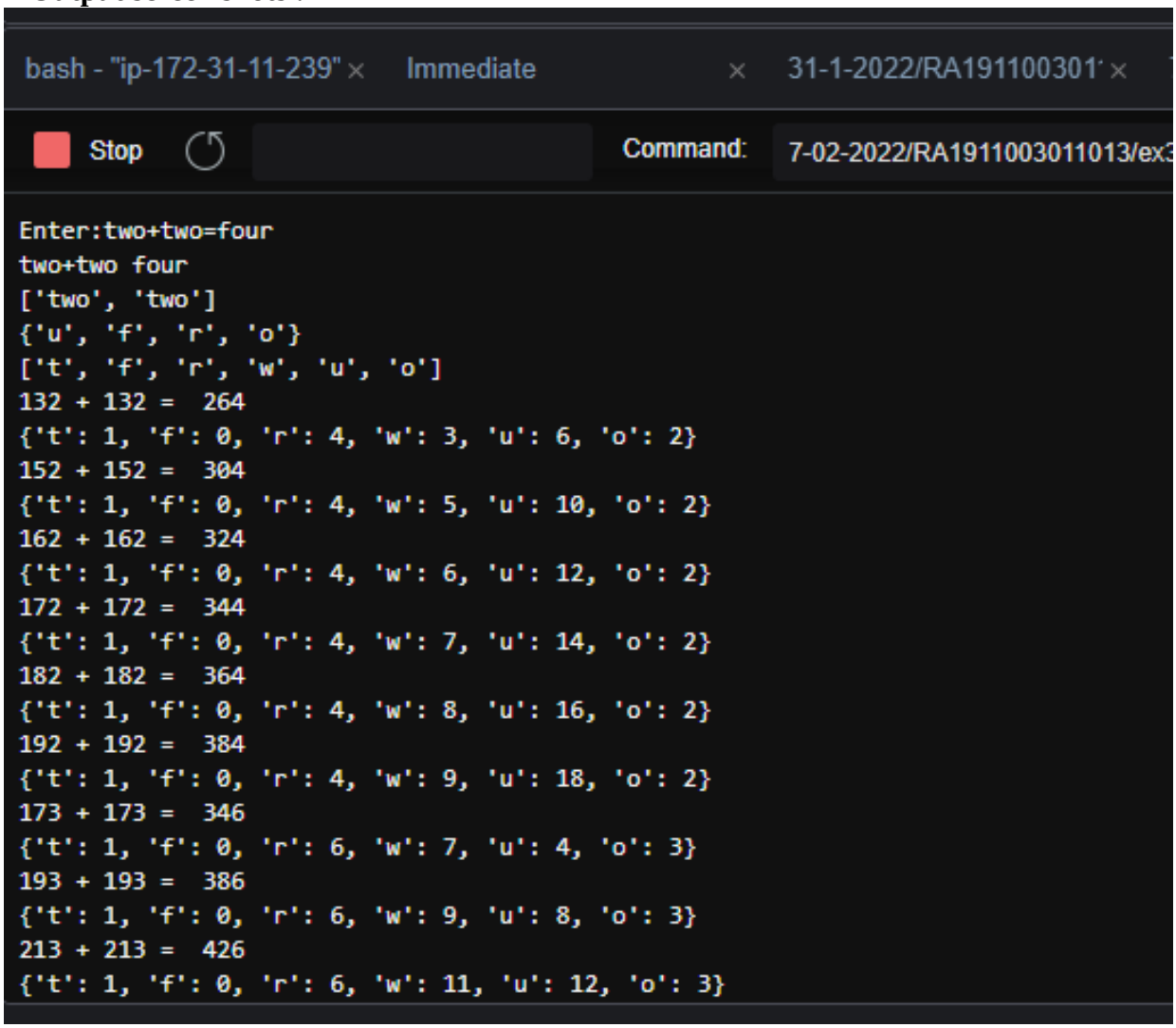
```
    l, r = equation.lower().replace(' ', '').split('=')
    print(l,r)
    l = l.split('+')
    print(l)
    lets = set(r)
    print(lets)
    for word in l:
        for let in word:
            lets.add(let)
    lets = list(lets)
    print(lets)
    digits = range(20)
    for perm in itertools.permutations(digits, len(lets)):
        sol = dict(zip(lets, perm))

        if sum(get_val(word, sol) for word in l) == get_val(r, sol):
            print(' + '.join(str(get_val(word, sol)) for word in l) + " = ",get_val(r, sol))



equation = input("Enter:")
solve(equation)
```

Output screen shots :



Result : Successfully solved the given constraint satisfaction problem.