GROUP
003-395-456

REPORT DOCUMENT

Project 01 –
Cryptarithmetic Problem in AI

REPORT DOCUMENT

I. Introduction

1. Description

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.

2. Goal

- Level 1: Solve the addition equation or subtraction equation with 2 operands
- Level 2: Solve any kind of equation with multiple operands and single operator (+ -).
- Level 3: Solve any kind of equation with multiple operands and operators (+ -). We use the parentheses () are used to specify the order of operations.
- Level 4: Solve level 3 with the multiplication equation (*).

3. Specifications

Input: is a text file including a single line describing the equation. The valid letters are:

- Uppercase characters A-Z.
- Operators: +, -, *
- The parentheses ( )
- The equal sign =

Output: is a text file with lines that shows the decoded values for letters. The order of values follows the order of the letters in alphabetical order. There is no space between the digits.

4. Team

ID: 003-395-456

STT	Name	Student ID	Work

01	Nguyễn Hữu Đạt	19127003	Proposing algorithms, giving instructions, Coding the main parts of the processing program, Algorithm report
02	Phan Đức Hiển	19127395	Data preprocessing, Testing, Generate test case, Experiments report
03	Nguyễn Thanh Kiên	19127456	Data preprocessing, testing, Generate test case, Experiments report

II. Report

1. Environment

Python 3.x

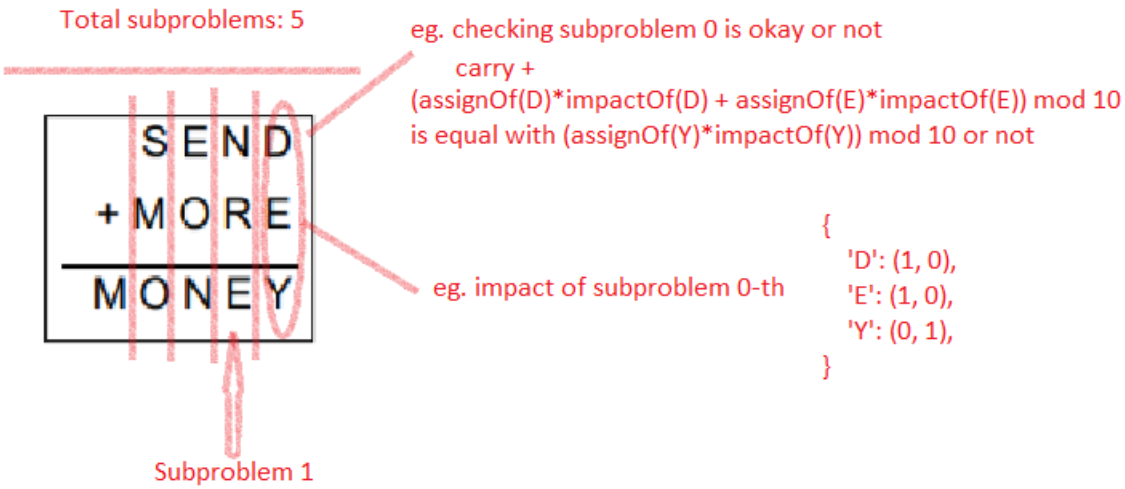
2. Algorithm

Using tree decomposition technique, combining dynamic programming with data preprocessing tricks. Apply a single algorithm to all 4 levels, dealing with levels 1-3 together and levels 4 in a separate section. The levels differ only in the data processing and extraction stage.

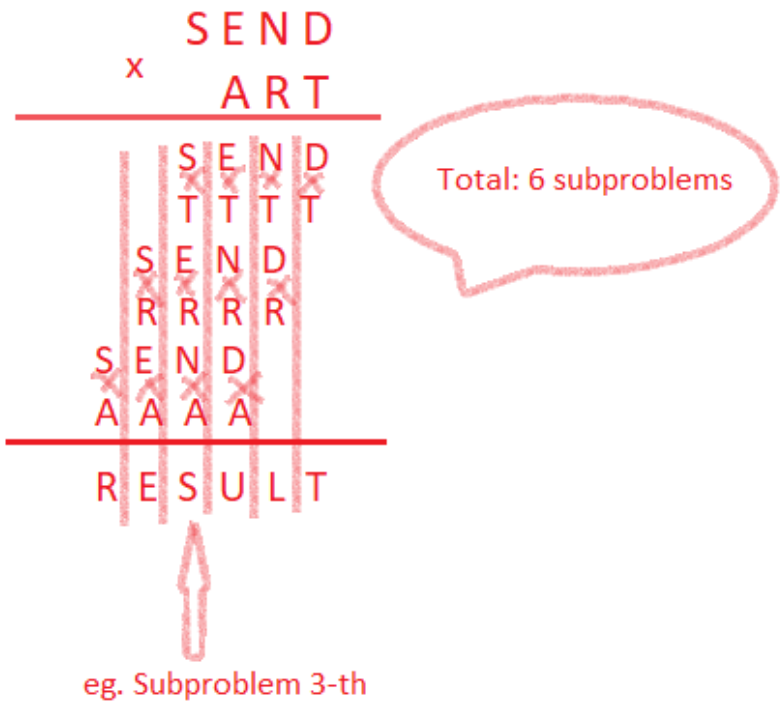
Algorithmic thought:

- We have a remark that if the input has a number of distinct letters greater than 10, there will be no solution, so we only deal with cases less than or equal to 10.
- Instead of solving the big problem, we divide the problem into subproblems, each of which corresponds to finding the result that satisfies a column of the operation.
- Then, we apply the idea of dynamic programming here, the big problem is only solved when the subproblems have found the results, we solve the subproblems in order from right to left. Let's say we are standing at the *i*-th subproblem, we will solve the *i*-th subproblem and use the generated state to solve the *i+1*-th subproblem, and so on. Hence, the big problem will be solved when the last subproblem is solved without any exception being raised.
- Regarding the brackets problem, we can easily preprocess it by removing the brackets and changing the sign of the corresponding operation inside the brackets depending on whether the preceding operator is plus or minus.

- What about the problem of multiple operands and multi-operators? This will also be very simple when we preprocess the data skillfully. For each subproblem as we mentioned above, we will store a list of (non-repeating) characters of that problem to be solved (characters in the corresponding column). The number of repetitions of characters in each subproblem will be stored in a dictionary with {key(character) : value(positive impact, negative impact)}



- As the illustration above, assuming the character under consideration belongs to an operator that is preceded by a minus sign or belongs to the resulting string, it will have a negative impact. And whether or not a subgoal is satisfied depends on whether the last digit of the negative impact is the same as the positive impact, and the rest will be the difference (which is the carry of that column, and returned for search along with the next subproblem).
- For level 4, the data implementation will be slightly different from level 1-3, but generally still use the above tricks. And we still apply the same algorithm to handle when going from multiplication to addition.



3. Experiments

4. Level 1	
Input1.txt	Input size: 25 Number of operands: 2 Longest operand: 8 {'A': 1, 'L': 7, 'P': 5, 'H': 3, 'B': 9, 'E': 0, 'T': 8, 'R': 6, 'S': 2, 'C': 4} [Done] exited with code=0 in 0.16 seconds
Input2.txt	Input size: 17 Number of operands: 2 Longest operand: 5 NO SOLUTION [Done] exited with code=0 in 0.135 seconds
Input3.txt	Input size: 13 Number of operands: 2 Longest operand: 4 {'W': 4, 'E': 6, 'L': 1, 'Y': 7, 'O': 8, 'U': 5, 'D': 3, 'N': 2} [Done] exited with code=0 in 0.158 seconds

Input4.txt	Input size: 302 Number of operands: 2 Longest operand: 100 {'A': 4, 'B': 1, 'C': 5, 'D': 2, 'E': 9, 'G': 3} [Done] exited with code=0 in 0.146 seconds
Input5.txt	Input size: 302 Number of operands: 2 Longest operand: 100 {'A': 9, 'B': 3, 'C': 4, 'D': 1, 'E': 5, 'G': 2} [Done] exited with code=0 in 0.13 seconds

Level 2	
Input1.txt	Input size: 171 Number of operands: 41 Longest operand: 5 {'S': 4, 'O': 0, 'M': 2, 'A': 7, 'N': 5, 'Y': 1, 'R': 9, 'E': 8, 'T': 6, 'H': 3} [Done] exited with code=0 in 0.252 seconds
Input2.txt	Input size: 31 Number of operands: 5 Longest operand: 5 {'E': 7, 'A': 2, 'R': 9, 'T': 4, 'H': 3, 'I': 8, 'F': 0, 'W': 5, 'N': 1, 'U': 6} [Done] exited with code=0 in 0.201 seconds
Input3.txt	Input size: 37575 Number of operands: 7314 Longest operand: 5 {'S': 4, 'O': 0, 'M': 2, 'A': 7, 'N': 5, 'Y': 1, 'R': 9, 'E': 8, 'T': 6, 'H': 3} [Done] exited with code=0 in 0.295 seconds
Input4.txt	Input size: 53291 Number of operands: 13161 Longest operand: 5 NO SOLUTION [Done] exited with code=0 in 11.396 seconds
Input5.txt	Input size: 23252837 Number of operands: 6202094 Longest operand: 12 {'A': 5, 'N': 9, 'C': 7, 'E': 4, 'L': 0, 'R': 2, 'T': 1, 'I': 6, 'G': 8, 'F': 3} [Done] exited with code=0 in 64.588 seconds

Level 3	
Input1.txt	Input size: 28 Number of operands: 5 Longest operand: 5 {'S': 7, 'E': 2, 'N': 6, 'D': 0, 'M': 5, 'O': 1, 'R': 8, 'Y': 3, 'I': 9, 'U': 4} [Done] exited with code=0 in 0.148 seconds
Input2.txt	Input size: 72 Number of operands: 15 Longest operand: 5 {'Y': 2, 'O': 7, 'U': 8, 'R': 3, 'E': 9, 'A': 0, 'H': 5, 'D': 1, 'N': 4, 'S': 6} [Done] exited with code=0 in 1.117 seconds
Input3.txt	Input size: 228378 Number of operands: 49621 Longest operand: 31 {'R': 1, 'T': 2, 'U': 4, 'Y': 3, 'F': 5, 'H': 7, 'G': 6, 'V': 8, 'N': 0, 'B': 9} [Done] exited with code=0 in 5.514 seconds
Input4.txt	Input size: 1420134 Number of operands: 85361 Longest operand: 53 NO SOLUTION [Done] exited with code=0 in 4.064 seconds
Input5.txt	Input size: 5708083 Number of operands: 1902690 Longest operand: 5 {'S': 7, 'E': 2, 'N': 6, 'D': 0, 'M': 5, 'O': 1, 'R': 8, 'Y': 3, 'I': 9, 'U': 4} [Done] exited with code=0 in 10.99 seconds

Level 4

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] python -u "c:\Users\datng\Desktop\AI-Project-01\sources\Level 4\main.py"
ABCD*DEF=AKFHNNG
Input size: 16
Number of operands: 2
Longest operand: 4
{'A': 0, 'B': 5, 'C': 7, 'D': 2, 'E': 4, 'F': 3, 'K': 1, 'H': 8, 'N': 9, 'G': 6}

[Done] exited with code=0 in 0.28 seconds
```

```
[Running] python -u "c:\Users\datng\Desktop\AI-Project-01\sources\Level 4\main.py"
ABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCD*DEFEFEFEFEFEFEF
Input size: 260
Number of operands: 2
Longest operand: 68
NO SOLUTION

[Done] exited with code=0 in 0.291 seconds
```

```
[Running] python -u "c:\Users\datng\Desktop\AI-Project-01\sources\Level 4\main.py"
ABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDFFFFFFFFFFFFFFFFFF
Input size: 1177
Number of operands: 2
Longest operand: 294
NO SOLUTION

[Done] exited with code=0 in 0.308 seconds
```

```
[Running] python -u "c:\Users\datng\Desktop\AI-Project-01\sources\Level 4\main.py"
ABCDEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
Input size: 197
Number of operands: 2
Longest operand: 97
{'A': 4, 'B': 3, 'C': 2, 'D': 0, 'E': 1}

[Done] exited with code=0 in 0.336 seconds
```