LOVELY PROFESSIONAL UNIVERSITY

Academic Task-3 (Artificial Intelligence)

School of Computer Science and Engineering Faculty of Technology & Sciences

Name of the faculty member: Mrs. Ankita Wadhawan

Course Code: INT 404 Course Title: Artificial Intelligence

Student Name : Manvir Kaur, Satyam, Manish Badsara, Brinderjit, Manish Kumar Roy
Student ID : 11604523, 11607168, 11804958, 11600805, 11604770
Roll Number : A-19, A-21, A-03, B-39, A-18
Section : K18UW
GitHub Link : https://github.com/ManishBadsara/11607168_K18UW_03_18_19_21_39

Cryptarithmetic Problem

Abstract:

A cryptarithmetic problem is a type of mathematical puzzle where numbers are replaced by letters and the goal is to find the mapping of letters to numbers. Cryptarithms can be solved as search problems or as constraint satisfaction problems. We can solve cryptarithms involving addition or subtraction as a constraint satisfaction problem.

Introduction:

A cryptarithmetic puzzle consists of an equation in which each letter stands for a different digit. The goal of the puzzle is to find the values represented by each letter such that the values also satisfy the given equation. Each letter represents a unique digit i.e. there is always a one-one mapping between the letters and the digits. Cryptarithms can be made very complicated by using longer words and operations like multiplication and division. Cryptarithms can be considered as a means of encryption but they are merely used to amuse the readers rather than for hiding important information. Cryptarithmetic puzzles can be solved as a search problem. They can be formulated using a search tree where each node represents the possible values of some of the variables of the problem and values of other variables are determined by going down the search tree.

This method will obviously take a long time because we will have to search for many possible combinations which on final substitution in the equation may not be the answers that we need. Another approach is one where we formulate the cryptarithmetic puzzle as a constraint satisfaction problem. Constraint satisfaction problems are a subset of search problems in which the states are defined using variables and the goals are defined using constraints. The main constraint of solving cryptarithms is that each letter should map to a unique digit. We know that the digits are – {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Taking into consideration the following example: ODD + ODD = EVEN For this example, we should return a set of 5 numbers mapping to each of the letters of the set - {O, D, E, V, N} respectively. We also know for instance that the leading letters i.e. O and E cannot map to the digit 0. In a constraint satisfaction problem, all such constraints are checked to be satisfied by the current variable assignments before proceeding to assign the next variable.

Literature Review:

Applying a search algorithm to solve a cryptarithmetic puzzle is a naïve solution and usually involves searching through all the possible combinations of values that can be assigned to the variable under consideration. These algorithms are guaranteed to find a solution but sometimes they might end up taking a long time to reach the solution. These algorithms cannot find a solution until all variables are assigned and hence, they expand a huge number of leaf nodes before finding a solution. But before applying any search algorithm on the problem, we formulate the cryptarithm as a constraint satisfaction problem.

A constraint satisfaction problem is defined by:

1. A set of variables and their associated domains.
2. A set of constraints that must be satisfied.

Thus, two other approaches which can be considered as slight modifications to the routine search algorithms and are used to solve constraint satisfaction problems are Backtracking and Forward Checking.

Backtracking Search:

A backtracking search involves assigning a value to a variable and then checking this assignment to ensure that it does not conflict with the assignments for the previous variables.

If a current assignment violates any previous assignment then the algorithm abandons this branch of the search and backtracks to the previous node to assign it a new value and then proceeds down another branch. The backtracking algorithm is better than general search algorithms because this tries to identify possible dead-ends that may arise due to the assignment of some value to a variable. This results in fewer leaf nodes being expanded in case the algorithm does come across some combination of values for some of the variables that may conflict with the assignments for other variables.

Forward Checking:

The forward checking algorithm is a look-ahead algorithm that ensures that the value assignment of a current variable does not violate the constraints of the previous as well as the future variables. This algorithm tries to detect situations beforehand in cases where failure is inevitable. In a forward checking algorithm, we keep track of the domain for each unassigned variable and once a variable is assigned a value, the domains of all the other unassigned variables are updated by removing the value that conflicts with the current assignment. Failure is detected when the domain of an unassigned variable becomes empty. This is detected when we update the domain of the unassigned variables, and if we come across such a situation it means that the value assigned to current variable is not a viable option. Then we backtrack and change the domains of the unassigned variables to the states they were in before the variable assignment occurred. Thus, forward checking allows us to prune those branches of the search tree which we would have explored had we used only the backtracking algorithm.

Proposed Methodology:

For any search problem, we can always try to find some heuristics which will help us to find the solution to a problem in an efficient manner. In the same way, we can also use some heuristics to improve the performance of the forward checking algorithm for solving constraint satisfaction problems.

There are two heuristics that are used with the forward checking algorithm. These heuristics are used to decide which variable to be chosen for assignment at each step of the algorithm. They are the minimum remaining values heuristic and the least constraining values heuristic.

1. **Minimum Remaining Values (MRV):** Using the minimum remaining values heuristic, the variable with the smallest domain is assigned first. This ensures that try to detect failures caused due to empty domains of variables in the earlier stages.

2. **Least Constraining Values (LCV):** On assigning a value to a variable, check if this variable assignment constrains the domains of any other variables.

Choose the value which constrains the other variable domains the least. A combination of minimum remaining values and least constraining values can be used with backtracking or forward checking algorithm to solve cryptarithms more efficiently. Another method which is generally used to solve constraint satisfaction problems is arc consistency. After assigning a value to a variable, draw arcs pointing from all other variables to the changed variable and delete conflicting values from domains at the tail. Repeat this process till you reach an assignment that satisfies the given problem. Applying arc consistency to cryptarithmetic puzzles is not usually preferred. For solving cryptarithmetic puzzles, we can formulate many constraints but implementing all these constraints in Python is not feasible. Prolog is a programming language which is better suitable for solving problems that involve reasoning and logic.

Project Description:

To solve a cryptarithmetic problem, there are two main constraints that we consider. The first constraint is that one letter should map to only one digit. The other constraint is that the first letter of each word should not be a zero. This is because when we add any two numbers, the leading digit can never be a zero for both the operands and the result. Since we know that the number of mappings will be equal to the number of unique letters present in the cryptarithm, we generate all the different permutations possible. As we are generating permutations, this automatically implies that we are following the constraint that each letter must correspond to only one digit. And after generating each permutation, we check to see that none of the leading letters is mapped to '0'. We filter out all the permutations that follow these two constraints and then evaluate each permutation to see if it satisfies the given mathematical equation. If the number of unique letters for the given puzzle is greater than 10, then such a puzzle is invalid because the constraint that there should exist only one-to-one mapping between the letters and digits.

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)

Example:

- Given a cryptarithmetic problem; i.e.,

```
  SEND
 +MORE
------
 MONEY
------
```

Below is the representation of the assignment of the digits to the alphabets.

S	9
E	5
N	6
D	7
M	1
O	0
R	8
y	2

Test Case:

The experiments were different cryptarithmetic problems given as string inputs to the “solve” function. The table given below shows the results for some experiments.

Input	Output
SEND + MORE = MONEY	9567 + 1085 = 10652
ODD + ODD = EVEN	655 + 655 = 1310 855 + 855 = 1710
BASE + BALL = GAMES	7483 + 7455 = 14938

Result and Discussion:

Cryptarithms are solved as a variant of Constraint Satisfaction Problems. The heuristics MRV (minimum remaining values) and LCV (least constrained values) can be used to improve the performance. Since the domain of the outputs and inputs are constrained it is better to use CSP rather than applying any search algorithms or simple brute force.

We expect to show that the algorithm can solve cryptarithm, the number of iterations it can be solved in and compare the performances of the algorithm without using any heuristics.

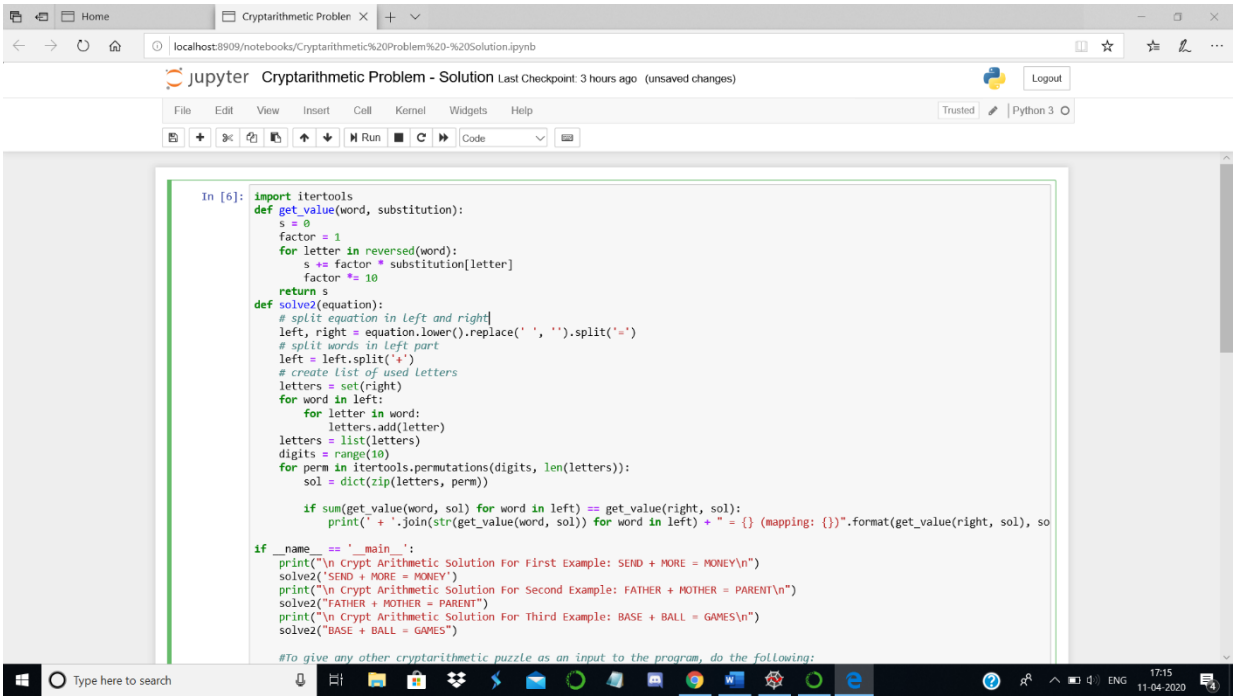
Conclusion:

Cryptarithmetic problems can be solved by applying search algorithms and some constraints on the variables. Programming a code completely based on constraints is not feasible in Python. Hence, we can implement only some of the constraints. To implement the cryptarithmetic puzzle purely based on constraints, Prolog is feasible and preferred.

Code Snippets:

Crypt Arithmetic Problem – Solution

GitHub Link: **https://github.com/ManishBadsara/11607168_K18UW_03_18_19_21_39**



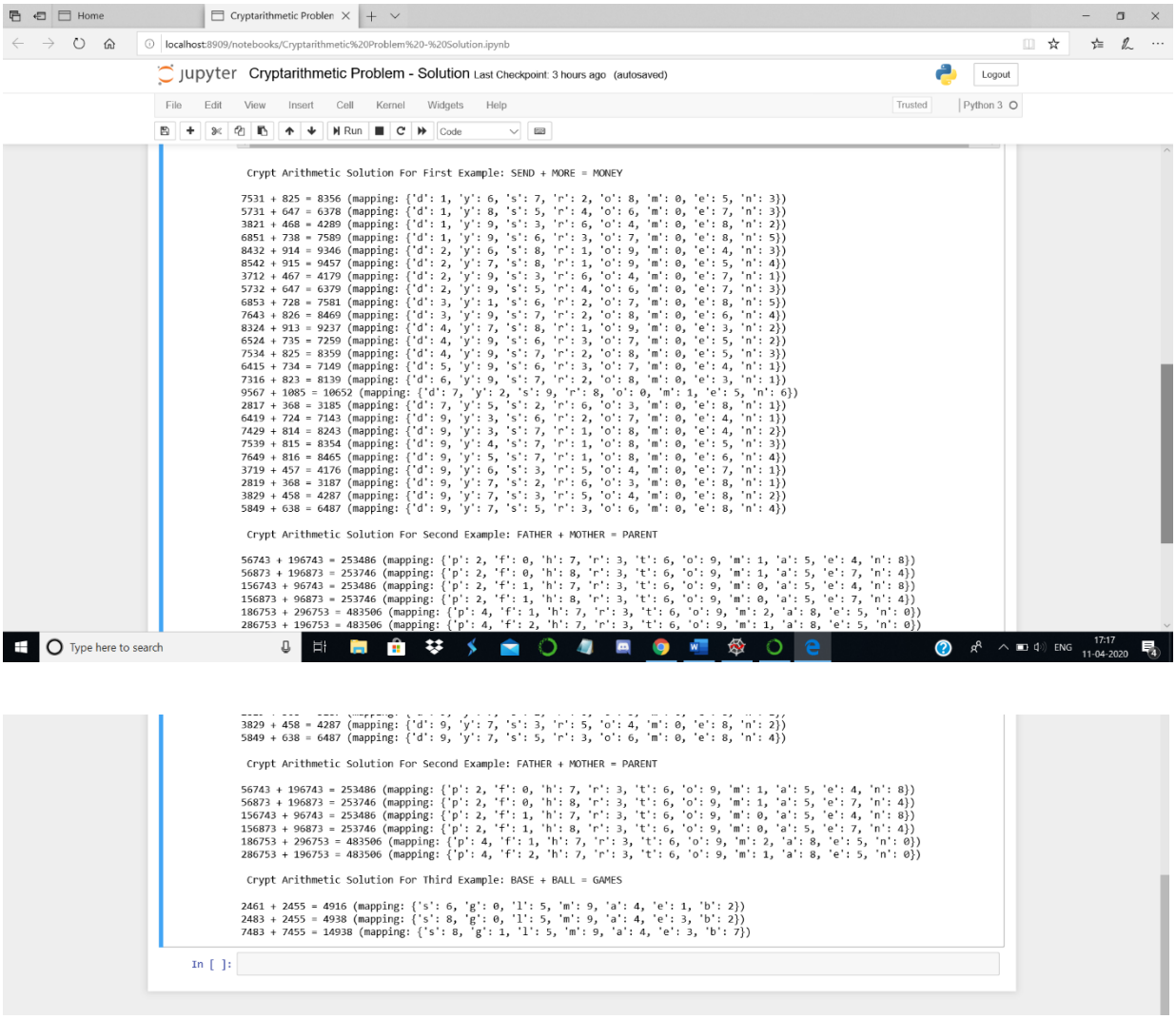
```
In [6]: import itertools
def get_value(word, substitution):
    s = 0
    factor = 1
    for letter in reversed(word):
        s += factor * substitution[letter]
        factor *= 10
    return s
def solve2(equation):
    # split equation in left and right
    left, right = equation.lower().replace(' ', '').split('=')
    # split words in left part
    left = left.split('+')
    # create list of used letters
    letters = set(right)
    for word in left:
        for letter in word:
            letters.add(letter)
    letters = list(letters)
    digits = range(10)
    for perm in itertools.permutations(digits, len(letters)):
        sol = dict(zip(letters, perm))

        if sum(get_value(word, sol) for word in left) == get_value(right, sol):
            print(" + '.join(str(get_value(word, sol)) for word in left) + " = {} (mapping: {})".format(get_value(right, sol), so

if __name__ == '__main__':
    print("\n Crypt Arithmetic Solution For First Example: SEND + MORE = MONEY\n")
    solve2('SEND + MORE = MONEY')
    print("\n Crypt Arithmetic Solution For Second Example: FATHER + MOTHER = PARENT\n")
    solve2("FATHER + MOTHER = PARENT")
    print("\n Crypt Arithmetic Solution For Third Example: BASE + BALL = GAMES\n")
    solve2("BASE + BALL = GAMES")

    #To give any other cryptarithmetic puzzle as an input to the program, do the following:
```

Output:



```
 Crypt Arithmetic Solution For First Example: SEND + MORE = MONEY

7531 + 825 = 8356 (mapping: {'d': 1, 'y': 6, 's': 7, 'r': 2, 'o': 8, 'm': 0, 'e': 5, 'n': 3})
5731 + 647 = 6378 (mapping: {'d': 1, 'y': 8, 's': 5, 'r': 4, 'o': 6, 'm': 0, 'e': 7, 'n': 3})
3821 + 468 = 4289 (mapping: {'d': 1, 'y': 9, 's': 3, 'r': 6, 'o': 4, 'm': 0, 'e': 8, 'n': 2})
6851 + 738 = 7589 (mapping: {'d': 1, 'y': 9, 's': 6, 'r': 3, 'o': 7, 'm': 0, 'e': 8, 'n': 5})
8432 + 914 = 9346 (mapping: {'d': 2, 'y': 6, 's': 8, 'r': 1, 'o': 9, 'm': 0, 'e': 4, 'n': 3})
8542 + 915 = 9457 (mapping: {'d': 2, 'y': 7, 's': 8, 'r': 1, 'o': 9, 'm': 0, 'e': 5, 'n': 4})
3712 + 467 = 4179 (mapping: {'d': 2, 'y': 9, 's': 3, 'r': 6, 'o': 4, 'm': 0, 'e': 7, 'n': 1})
5732 + 647 = 6379 (mapping: {'d': 2, 'y': 9, 's': 5, 'r': 4, 'o': 6, 'm': 0, 'e': 7, 'n': 3})
6853 + 728 = 7581 (mapping: {'d': 3, 'y': 1, 's': 6, 'r': 2, 'o': 7, 'm': 0, 'e': 8, 'n': 5})
7643 + 826 = 8469 (mapping: {'d': 3, 'y': 9, 's': 7, 'r': 2, 'o': 8, 'm': 0, 'e': 6, 'n': 4})
8324 + 913 = 9237 (mapping: {'d': 4, 'y': 7, 's': 8, 'r': 1, 'o': 9, 'm': 0, 'e': 3, 'n': 2})
6524 + 735 = 7259 (mapping: {'d': 4, 'y': 9, 's': 6, 'r': 3, 'o': 7, 'm': 0, 'e': 5, 'n': 2})
7534 + 825 = 8359 (mapping: {'d': 4, 'y': 9, 's': 7, 'r': 2, 'o': 8, 'm': 0, 'e': 5, 'n': 3})
6415 + 734 = 7149 (mapping: {'d': 5, 'y': 9, 's': 6, 'r': 3, 'o': 7, 'm': 0, 'e': 4, 'n': 1})
7316 + 823 = 8139 (mapping: {'d': 6, 'y': 9, 's': 7, 'r': 2, 'o': 8, 'm': 0, 'e': 3, 'n': 1})
9567 + 1085 = 10652 (mapping: {'d': 7, 'y': 2, 's': 9, 'r': 8, 'o': 0, 'm': 1, 'e': 5, 'n': 6})
2817 + 368 = 3185 (mapping: {'d': 7, 'y': 5, 's': 2, 'r': 6, 'o': 3, 'm': 0, 'e': 8, 'n': 1})
6419 + 724 = 7143 (mapping: {'d': 9, 'y': 3, 's': 6, 'r': 2, 'o': 7, 'm': 0, 'e': 4, 'n': 1})
7429 + 814 = 8243 (mapping: {'d': 9, 'y': 3, 's': 7, 'r': 1, 'o': 8, 'm': 0, 'e': 4, 'n': 2})
7539 + 815 = 8354 (mapping: {'d': 9, 'y': 4, 's': 7, 'r': 1, 'o': 8, 'm': 0, 'e': 5, 'n': 3})
7649 + 816 = 8465 (mapping: {'d': 9, 'y': 5, 's': 7, 'r': 1, 'o': 8, 'm': 0, 'e': 6, 'n': 4})
3719 + 457 = 4176 (mapping: {'d': 9, 'y': 6, 's': 3, 'r': 5, 'o': 4, 'm': 0, 'e': 7, 'n': 1})
2819 + 368 = 3187 (mapping: {'d': 9, 'y': 7, 's': 2, 'r': 6, 'o': 3, 'm': 0, 'e': 8, 'n': 1})
3829 + 458 = 4287 (mapping: {'d': 9, 'y': 7, 's': 3, 'r': 5, 'o': 4, 'm': 0, 'e': 8, 'n': 2})
5849 + 638 = 6487 (mapping: {'d': 9, 'y': 7, 's': 5, 'r': 3, 'o': 6, 'm': 0, 'e': 8, 'n': 4})

 Crypt Arithmetic Solution For Second Example: FATHER + MOTHER = PARENT

56743 + 196743 = 253486 (mapping: {'p': 2, 'f': 0, 'h': 7, 'r': 3, 't': 6, 'o': 9, 'm': 1, 'a': 5, 'e': 4, 'n': 8})
56873 + 196873 = 253746 (mapping: {'p': 2, 'f': 0, 'h': 8, 'r': 3, 't': 6, 'o': 9, 'm': 1, 'a': 5, 'e': 7, 'n': 4})
156743 + 96743 = 253486 (mapping: {'p': 2, 'f': 1, 'h': 7, 'r': 3, 't': 6, 'o': 9, 'm': 0, 'a': 5, 'e': 4, 'n': 8})
156873 + 96873 = 253746 (mapping: {'p': 2, 'f': 1, 'h': 8, 'r': 3, 't': 6, 'o': 9, 'm': 0, 'a': 5, 'e': 7, 'n': 4})
186753 + 296753 = 483506 (mapping: {'p': 4, 'f': 1, 'h': 7, 'r': 3, 't': 6, 'o': 9, 'm': 2, 'a': 8, 'e': 5, 'n': 0})
286753 + 196753 = 483506 (mapping: {'p': 4, 'f': 2, 'h': 7, 'r': 3, 't': 6, 'o': 9, 'm': 1, 'a': 8, 'e': 5, 'n': 0})
```



```
3829 + 458 = 4287 (mapping: {'d': 9, 'y': 7, 's': 3, 'r': 5, 'o': 4, 'm': 0, 'e': 8, 'n': 2})
5849 + 638 = 6487 (mapping: {'d': 9, 'y': 7, 's': 5, 'r': 3, 'o': 6, 'm': 0, 'e': 8, 'n': 4})

 Crypt Arithmetic Solution For Second Example: FATHER + MOTHER = PARENT

56743 + 196743 = 253486 (mapping: {'p': 2, 'f': 0, 'h': 7, 'r': 3, 't': 6, 'o': 9, 'm': 1, 'a': 5, 'e': 4, 'n': 8})
56873 + 196873 = 253746 (mapping: {'p': 2, 'f': 0, 'h': 8, 'r': 3, 't': 6, 'o': 9, 'm': 1, 'a': 5, 'e': 7, 'n': 4})
156743 + 96743 = 253486 (mapping: {'p': 2, 'f': 1, 'h': 7, 'r': 3, 't': 6, 'o': 9, 'm': 0, 'a': 5, 'e': 4, 'n': 8})
156873 + 96873 = 253746 (mapping: {'p': 2, 'f': 1, 'h': 8, 'r': 3, 't': 6, 'o': 9, 'm': 0, 'a': 5, 'e': 7, 'n': 4})
186753 + 296753 = 483506 (mapping: {'p': 4, 'f': 1, 'h': 7, 'r': 3, 't': 6, 'o': 9, 'm': 2, 'a': 8, 'e': 5, 'n': 0})
286753 + 196753 = 483506 (mapping: {'p': 4, 'f': 2, 'h': 7, 'r': 3, 't': 6, 'o': 9, 'm': 1, 'a': 8, 'e': 5, 'n': 0})

 Crypt Arithmetic Solution For Third Example: BASE + BALL = GAMES

2461 + 2455 = 4916 (mapping: {'s': 6, 'g': 0, 'l': 5, 'm': 9, 'a': 4, 'e': 1, 'b': 2})
2483 + 2455 = 4938 (mapping: {'s': 8, 'g': 0, 'l': 5, 'm': 9, 'a': 4, 'e': 3, 'b': 2})
7483 + 7455 = 14938 (mapping: {'s': 8, 'g': 1, 'l': 5, 'm': 9, 'a': 4, 'e': 3, 'b': 7})

In [ ]:
```