UNIVERISTY OF SCIENCE - VIETNAM NATIONAL UNIVERSITY OF HCM

Faculty of Information Technology

INTRODUCTION TO INTELLIGENCE ARTIFICIAL - 21CLC07

Report

PROJECT 01
CRYPTARITHMETIC PROBLEM IN AI

LECTURER

Nguyễn Ngọc Thảo
Lê Ngọc Thành
Nguyễn Trần Minh Duy
Nguyễn Hải Đăng

MEMBER

21127003 - Phan Thanh An
21127014 - Phạm Hồng Gia Bảo
21127228 - Nguyễn Gia Bảo
21127294 - Nguyễn Hi Hữu

2022-2023

Table of Contents

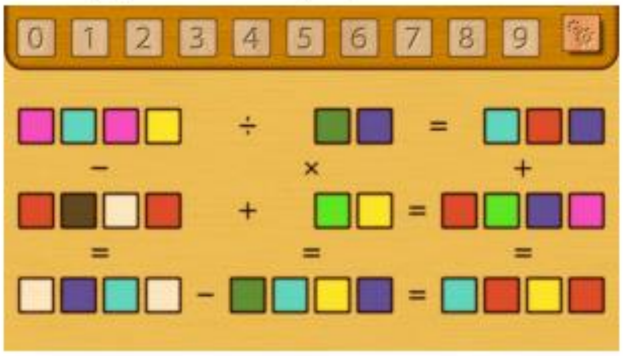
I. GENERAL TO THE PROBLEM

1. Description of Cryptarithmetic Problem

This assignment is created based on an article on tutorialexample.com website. The link is given in the References section. The next paragraphs are extracted from the webpage with some modification.

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.

In Google Play, you can find some game with cryptarithmetic puzzle. The Cryptogram is an example. You can enjoy it for some early experience but not be immersed in it.



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., left-hand side (L.H.S), or right-hand side (R.H.S)

Let us understand the cryptarithmetic problem as well its constraints better with the help of an example:

Given a cryptarithmetic problem, i.e., S E N D + M O R E = M O N EY

```
  S E N D
+ M O R E
---------
M O N E Y
```

2. Requirement

No.	Criteria	Scores
1	Finish level 1 successfully.	
2	Finish level 2 successfully.	
3	Finish level 3 successfully.	
4	Finish level 4 successfully.	
5	Generate at least 5 test cases for each level with different attributes such as size, time to solve, output type. Describe them in the experiment section of your report.	
6	Report your algorithms, experiments with some reflection or comments.	
Total		

3. Team Members Introduction

No.	Student'ID	Full-Name	Email
1	21127003	Phan Thanh An	ptan21@clc.fitus.edu.vn
2	21127014	Phạm Hồng Gia Bảo	phgbao21@clc.fitus.edu.vn
3	21127228	Nguyễn Gia Bảo	ngbao21@clc.fitus.edu.vn
4	21127294	Nguyễn Hi Hữu	nhhuu21@clc.fitus.edu.vn

4. Assignment Plan

Timeline	Task	Person
06/07/2023 - 22/07/2023	Find solution and implement Level 1	Phạm Hồng Gia Bảo
	Find solution and implement Level 2	Nguyễn Gia Bảo
	Find solution and implement Level 3	Nguyễn Hi Hữu
	Find solution and implement Level 4	Phan Thanh An
23/07/2023 - 25/07/2023	Write Report	Phạm Hồng Gia Bảo
26/07/2023 - 27/07/2023	Review all the implementation and report	All team

II. CONTRIBUTION AND GRADING

1. Contribution

Although this is the first time, we have worked in a group together, every member has co-operated effectively and successfully to finish the task achievement of this project. The following table describes what each of us has contributed to the whole project.

Member	Task Achievement	Contribution
Phạm Hồng Gia Bảo	• Take responsibility for research and implementation level **1.** • **Prepare test case for level 1.** • **Writing the details report** to analyze content and algorithm	• Analyze, write reports as well as review the algorithms with other members.
Nguyễn Gia Bảo	• All together take responsibility for **research and implementation level 2.** • Prepare **test case for level 2.**	• Contribute very first ideas to develop this project. • Exchange knowledge with others
Nguyễn Hi Hữu	• All together take responsibility for **research and implementation level 3.** • Prepare **test case for level 3.**	• Research and represent ideas on how to implement algorithms.
Phan Thanh An	• Take responsibility for **research and implementation level 4.** • Prepare **test case for level 4.**	• Research on how to implement algorithms. • Edit and check all the knowledge written in the report.

2. Grading

No.	Criteria	Completion
1	Finish level 1 successfully.	**100%**
2	Finish level 2 successfully.	**100%**
3	Finish level 3 successfully.	**100%**
4	Finish level 4 successfully.	**100%**

5	Generate at least 5 test cases for each level with different attributes such as size, time to solve, output type. Describe them in the experiment section of your report.	**100%**
6	Report your algorithms, experiments with some reflection or comments.	**100%**
All Project		**100%**

III. OVERVIEW ON PROBLEM USING BRUTE FORCE

1. Algorithm to Approach

- Brute Force is a problem-solving method that involves trying all possible solutions systematically until the correct one is found. The Brute Force algorithm for solving a cryptarithmetic problem involves trying all possible digit assignments to the letters in the problem until a solution is found. Here are the basic steps:
  - Parse the input cryptarithmetic problem into a list of strings representing the words and the sum.
  - Identify all distinct letters used in the problem and assign them to a list of variables.
  - Generate all possible digit assignments for the variables, subject to the constraint that each digit can only be assigned to one variable (i.e., no two letters can be assigned the same digit).
  - For each digit assignment, substitute the digits for the corresponding letters in the problem and evaluate the sum.
  - If the sum evaluates to the correct value, return the digit assignment as the solution. Otherwise, try the next digit assignment.
  - If all possible digit assignments have been tried and none of them produce the correct sum, return failure.
- **Pseudecode**

```
First, create a list of all the characters that need assigning to pass to Solve
If all characters are assigned, return true if problem 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
if all digits have been tried and nothing worked, return false to trigger
backtracking
```

2. Observation

- Brute Force tries all the possible solutions. So, while brute force can be a useful approach for solving small cryptarithmetic problems, it becomes increasingly inefficient as the size of the problem grows. This is because the number of possible combinations of digits to test increases exponentially with the number of letters in the problem.
- Using this approach, we would need to test every possible combination of digits from 0 to 9 for each of the “n” letters in the problem. This means that we would **need to test 10^n possible combinations**.
  - ⇨ **It would take an impractical amount of time to perform them all.**
- In addition to being slow, **Brute Force can also be error prone**. It is easy to make mistakes when testing so many different combinations, and it can be difficult to keep track of which combinations have already been tested.

⇨ Overall, **while brute force can be a useful approach for solving small cryptarithmetic problems, it quickly becomes impractical for larger problems.** Other approaches, such as constraint satisfaction algorithms or genetic algorithms, can be more efficient and effective for solving these types of problems.

IV. ALGORITHM AND IMPLEMENTATION USING CSP

1. Main Algorithm to Solve Problem

- The CSP algorithm is a powerful technique for solving Cryptarithmetic Problems and other types of constraint satisfaction problems. The CSP algorithm is a constraint-based approach that involves modeling the problem as a set of variables, domains, and constraints. As the rules of this problem,
  - **Variables** are the letters: {A, B, C, D, …., X, Y, Z} that appear in the question.
  - **Domains** are the values: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

- **Constraints** are the mathematical equations that must be satisfied by the variables, such as "SEND + MORE = MONEY".

- The CSP algorithm works by starting with an initial assignment of values to variables and then systematically exploring different assignments until a solution is found. At each step, the algorithm selects a variable to assign a value to, **based on some heuristic such as minimum remaining values or degree heuristic**. Then it selects a value from the domain of the variable and applies it to the constraints. If the constraints are satisfied, it moves on to the next variable. If not, it backtracks to the previous variable and tries a different value.
- The CSP algorithm can also use various techniques such as back jumping, forward checking, and constraint propagation to **improve its efficiency and reduce the search space**. It can handle large problems with many variables and constraints efficiently and **provide a global optimum solution if one exists**.
- **Backtracking Search**: a problem-solving technique used to find a solution by incrementally building a solution candidate and then "backtracking" and trying a different path if the current candidate is found to be invalid. It's often used in combination with depth-first search to explore all possible paths through a problem space.
- **Pseudocode**

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
if assignment is complete then return assignment
var ← SELECT-UNASSIGNED-VARIABLE(csp)
for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
   if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, value)
      if inferences □ failure then
         add inferences to assignment
         result ← BACKTRACK(assignment, csp)
```

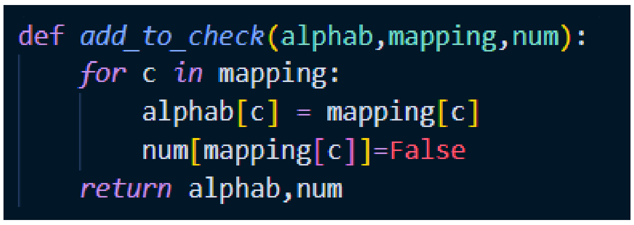
```
        if result □ failure then
            return result
      remove {var = value} and inferences from assignment
return failure
```

- **CSP algorithms are often more efficient than Brute Force algorithms.** Because CSP algorithms use heuristics to guide the search for a solution, which can significantly reduce the number of possibilities that need to be explored. CSP algorithms can handle problems with complex constraints and can often find solutions that are not immediately obvious.

❖ **Note**: The first 3 levels have the same algorithms but different in the difficulty of problems. So, the next level will require some functions of the previous level. **Therefore, we just explain the new functions (which do not explain in the previous parts)**

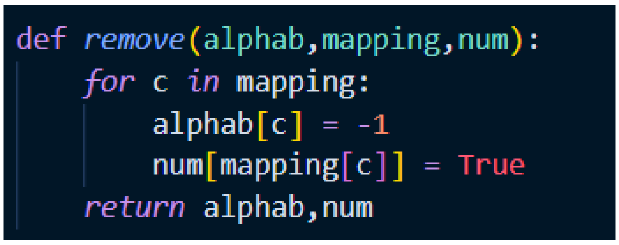
2. Implementation on Level 1 and Level 2

- ***def add_to_check(alphab,mapping,num):*** is a function **to update the "alphab" dictionary and "num_list" to reflect the latest mapping of letters to digits that has been found** by the program. The alphab dictionary maps each letter in the problem to the digit that it represents, while the num list keeps track of which digits have already been used in the mapping. By updating the **alphab** dictionary and **num_list** with each new mapping that is found, the program can ensure that it does not use the same digit to represent multiple letters, which would violate the rules of the problem.



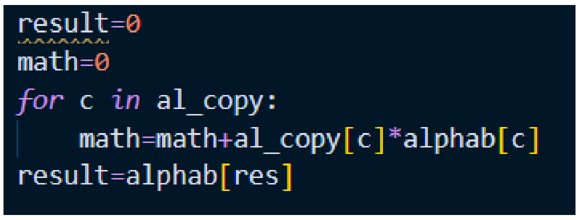
```python
def add_to_check(alphab,mapping,num):
    for c in mapping:
        alphab[c] = mapping[c]
        num[mapping[c]]=False
    return alphab,num
```

- ***def remove(alphab,mapping,num):*** function is opposite with **add_to_check()**. The purpose of this function is **to remove a previously assigned mapping of letters to digits from the alphab dictionary and num_list**. This is necessary when the program backtracks to a previous step in the search process and needs to undo a previously assigned mapping. By removing the previously assigned mapping, the program can ensure that it does not use the same digit to represent multiple letters and does not violate the rules of the problem.



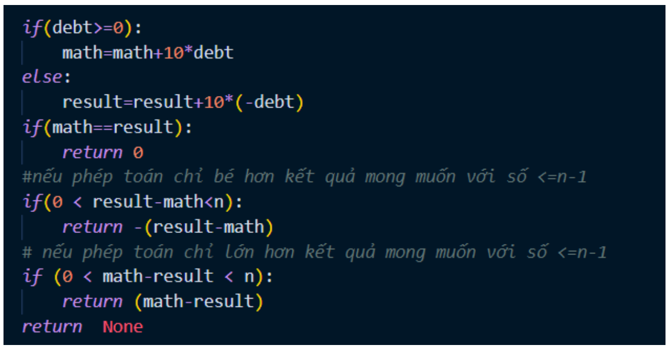
```python
def remove(alphab,mapping,num):
    for c in mapping:
        alphab[c] = -1
        num[mapping[c]] = True
    return alphab,num
```

- ***def check(alphab,al_copy,res,debt,n)***: checks whether the current mapping of letters to digits satisfies the equation represented by the **"alphab"** dictionary and **"res"** string. Specifically, it computes the sum of each operand and the carry-over using the current mapping of digits, and checks whether this sum equals the value of the res string.



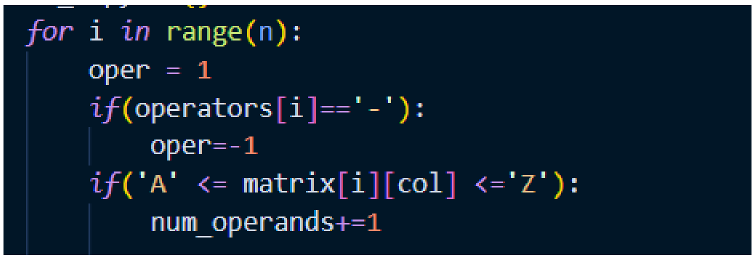
```python
result=0
math=0
for c in al_copy:
    math=math+al_copy[c]*alphab[c]
result=alphab[res]
```

  - If sum = "res", the function returns 0 => **current mapping is valid.**
  - If sum < "res" and < n, the function returns the negative difference => **current mapping is invalid** (sum is too small).
  - If sum > "res" and < n, the function returns the positive difference => **current mapping is invalid** (sum is too large).
  - If none of these conditions are met, the function returns None => current mapping may or may not be valid => **testing is needed.**



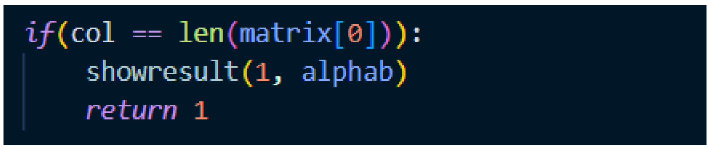
```python
if(debt>=0):
    math=math+10*debt
else:
    result=result+10*(-debt)
if(math==result):
    return 0
#nếu phép toán chỉ bé hơn kết quả mong muốn với số <=n-1
if(0 < result-math<n):
    return -(result-math)
# nếu phép toán chỉ lớn hơn kết quả mong muốn với số <=n-1
if (0 < math-result < n):
    return (math-result)
return  None
```

- ***def checkcolumn(col,matrix,debt,alphab,operators,n,num):*** checks whether the current column of the solution satisfies the constraints of the game.
  - Firstly, it computes the sum of each operand and the carry-over (if any) using the current mapping of digits, and checks whether this sum equals the value of the result string for the current column.



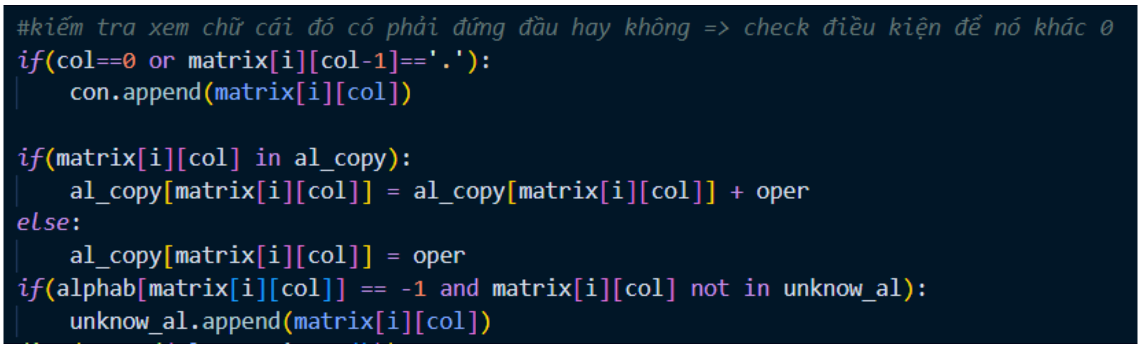
```
for i in range(n):
    oper = 1
    if(operators[i]=='-'):
        oper=-1
    if('A' <= matrix[i][col] <='Z'):
        num_operands+=1
```

- If the current column represents the last column of the problem, and the mapping of digits satisfies the equation, the function calls the **showresult()** function to print the successful solution and exits the program.



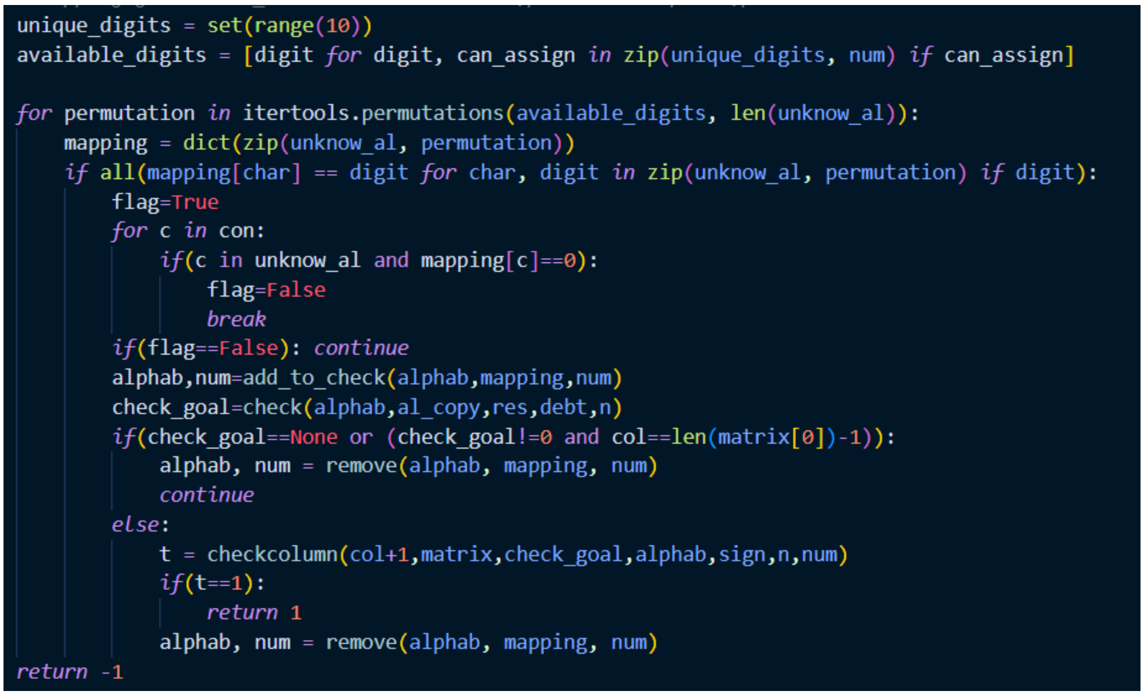
```
if(col == len(matrix[0])):
    showresult(1, alphab)
    return 1
```

- The function then iterates over each row in the column, counting the number of operands and keeping track of the unique letters **"unknow_al"** and their occurrences **"al_copy"**. If the current letter is a new letter, the function adds it to the list of unique letters.



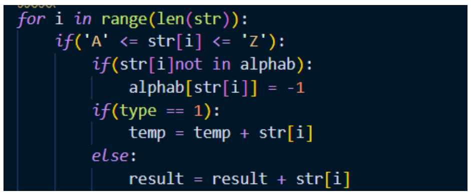
```
#kiểm tra xem chữ cái đó có phải đứng đầu hay không => check điều kiện để nó khác 0
if(col==0 or matrix[i][col-1]=='.'):
    con.append(matrix[i][col])

if(matrix[i][col] in al_copy):
    al_copy[matrix[i][col]] = al_copy[matrix[i][col]] + oper
else:
    al_copy[matrix[i][col]] = oper
if(alphab[matrix[i][col]] == -1 and matrix[i][col] not in unknow_al):
    unknow_al.append(matrix[i][col])
```

- If the current column does not represent the last column of the problem, the function generates all possible mappings of digits to the letters that have not yet been assigned a digit. For each mapping, the function checks whether the mapping satisfies the constraints of the problem using the check function. If the mapping satisfies the constraints of the problem, the function recursively calls itself to continue solving the problem with the updated mapping of digits.



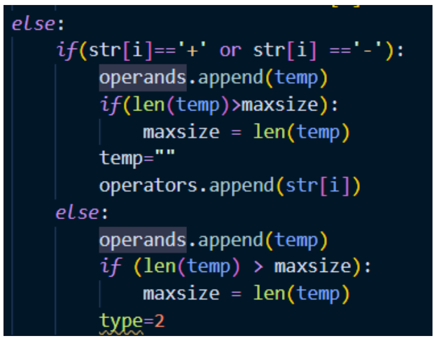
```python
unique_digits = set(range(10))
available_digits = [digit for digit, can_assign in zip(unique_digits, num) if can_assign]

for permutation in itertools.permutations(available_digits, len(unknow_al)):
    mapping = dict(zip(unknow_al, permutation))
    if all(mapping[char] == digit for char, digit in zip(unknow_al, permutation) if digit):
        flag=True
        for c in con:
            if(c in unknow_al and mapping[c]==0):
                flag=False
                break
        if(flag==False): continue
        alphab,num=add_to_check(alphab,mapping,num)
        check_goal=check(alphab,al_copy,res,debt,n)
        if(check_goal==None or (check_goal!=0 and col==len(matrix[0])-1)):
            alphab, num = remove(alphab, mapping, num)
            continue
        else:
            t = checkcolumn(col+1,matrix,check_goal,alphab,sign,n,num)
            if(t==1):
                return 1
            alphab, num = remove(alphab, mapping, num)
return -1
```

- ***def analy_string(str):***
    - Firstly, the function loops through each character in the input string and checks whether it is a letter or a symbol. If it is a letter, the function adds it to the **"temp"** if type is 1, or the **"result"** type is 2.



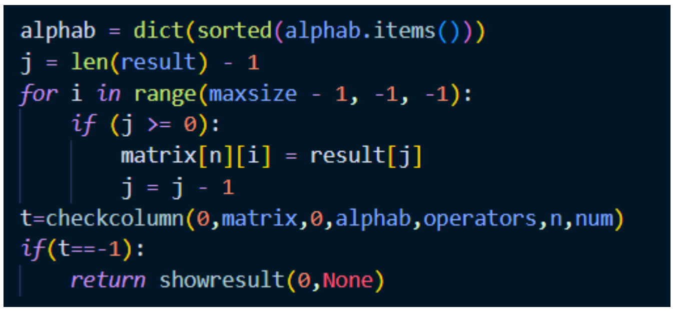
```python
for i in range(len(str)):
    if('A' <= str[i] <= 'Z'):
        if(str[i]not in alphab):
            alphab[str[i]] = -1
        if(type == 1):
            temp = temp + str[i]
        else:
            result = result + str[i]
```

    - If it is a symbol, the function appends temp to the operands list and adds the symbol to the operators list. Then, the function initializes a matrix variable as a list of lists with dimensions (n+1) x **maxsize**, where n is the length of the operands list and **maxsize** is the length of the longest operand or result.



```python
else:
    if(str[i]=='+' or str[i] =='-'):
        operands.append(temp)
        if(len(temp)>maxsize):
            maxsize = len(temp)
        temp=""
        operators.append(str[i])
    else:
        operands.append(temp)
        if (len(temp) > maxsize):
            maxsize = len(temp)
        type=2
```

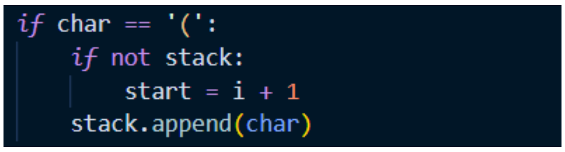
    - It then populates the matrix with the characters from the operands and result lists. The function sorts the **alphab** dictionary by key and calls the **checkcolumn** function with parameters 0, matrix, 0, **alphab**, operators, n, and num. It then calls the **showresult()** function with parameters 0 and None.



```python
alphab = dict(sorted(alphab.items()))
j = len(result) - 1
for i in range(maxsize - 1, -1, -1):
    if (j >= 0):
        matrix[n][i] = result[j]
        j = j - 1
t=checkcolumn(0,matrix,0,alphab,operators,n,num)
if(t==-1):
    return showresult(0,None)
```

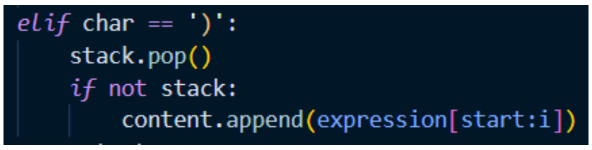
3. Implementation on Level 3

- ***def get_outer_parentheses_content(expression):*** returns the content list, which contains all the expressions enclosed in outermost parentheses. It is useful for extracting sub-expressions from a larger expression that are enclosed within outermost parentheses.
    - The function initializes an empty list called content, an empty list called stack, and a variable start to None.
    - The function iterates over each character in the input string expression using a for loop and the enumerate function.
    - If the current character is an opening parenthesis, the function checks whether the stack list is empty. If it is, the function sets the start variable to the index of the current character plus one. The function then adds the current character to the stack list.



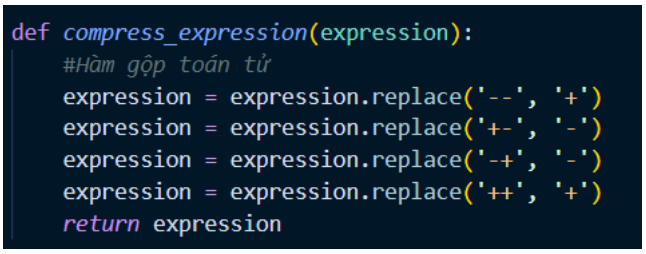
```python
if char == '(':
    if not stack:
        start = i + 1
    stack.append(char)
```

    - If the current character is a closing parenthesis, the function removes the last element from the stack list. If the stack list is now empty, the function extracts the expression enclosed within the outermost parentheses by slicing the input string from the start index to the current index and adds it to the content list.



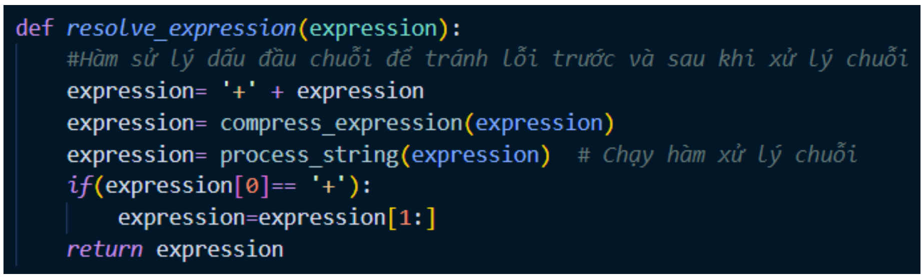
```python
elif char == ')':
    stack.pop()
    if not stack:
        content.append(expression[start:i])
```

- ***def compress_expression(expression):*** for simplifying a mathematical expression. It uses string replace methods to replace specific substrings with their simplified versions, effectively compressing the expression. The function first replaces any occurrences of '--' with a single plus sign, effectively cancelling out the double negative. It then replaces any occurrences of '+-' or '-+' with a minus sign, as these two operators combined can be simplified to a single minus sign. Finally, it replaces any occurrences of '++' with a single plus sign, as consecutive plus signs can be simplified to a single plus sign.



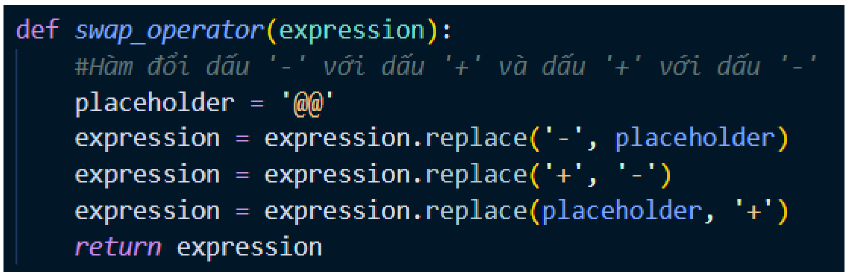
```python
def compress_expression(expression):
    #Hàm gộp toán tử
    expression = expression.replace('--', '+')
    expression = expression.replace('+-', '-')
    expression = expression.replace('-+', '-')
    expression = expression.replace('++', '+')
    return expression
```

- ***def resolve_expression(expression):*** this function is an implementation of a string parsing algorithm for evaluating a mathematical expression. It uses basic string manipulation techniques to simplify and evaluate the expression and follows the standard order of operations to ensure the correct result is returned.



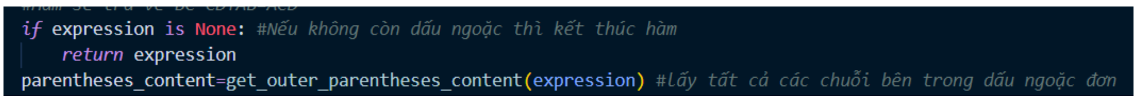
```python
def resolve_expression(expression):
    #Hàm sử lý dấu đầu chuỗi để tránh lỗi trước và sau khi xử lý chuỗi
    expression= '+' + expression
    expression= compress_expression(expression)
    expression= process_string(expression)  # Chạy hàm xử lý chuỗi
    if(expression[0]== '+'):
        expression=expression[1:]
    return expression
```

- ***def swap_operator(expression):*** returns the modified expression string with all + and - operators swapped.
    - The function defines a string placeholder that is used to temporarily replace the - operator.
    - The function replaces all occurrences of the - operator in the input string expression with the placeholder string using the replace method of the string object.
    - The function then replaces all occurrences of the + operator in expression with the - operator using the replace method.
    - The function then replaces all occurrences of the placeholder string in expression with the + operator using the replace method.



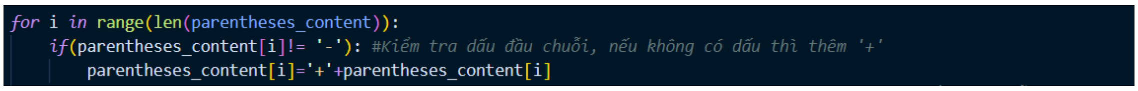
```python
def swap_operator(expression):
    #Hàm đổi dấu '-' với dấu '+' và dấu '+' với dấu '-'
    placeholder = '@@'
    expression = expression.replace('-', placeholder)
    expression = expression.replace('+', '-')
    expression = expression.replace(placeholder, '+')
    return expression
```

- ***def process_string(expression):*** is used to transform a level 3 mathematical expression into the level 2 format by pushing all expressions enclosed in outermost parentheses to the beginning of the string and processing them first.
    - The function first checks if the input string expression is None. If it is, the function returns None. The function calls the **get_outer_parentheses_content()** function to extract all expressions enclosed in outermost parentheses from the input string.



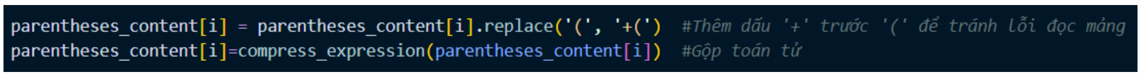
```
if expression is None: #Nếu không còn dấu ngoặc thì kết thúc hàm
    return expression
parentheses_content=get_outer_parentheses_content(expression) #lấy tất cả các chuỗi bên trong dấu ngoặc đơn
```

-
    - Then, we iterate over each expression enclosed in outermost parentheses and processes them by adding a + operator at the beginning of the expression if it does not already have one, replacing all opening parentheses with + (to avoid array indexing errors, and compressing the expression using the **compress_expression** function.



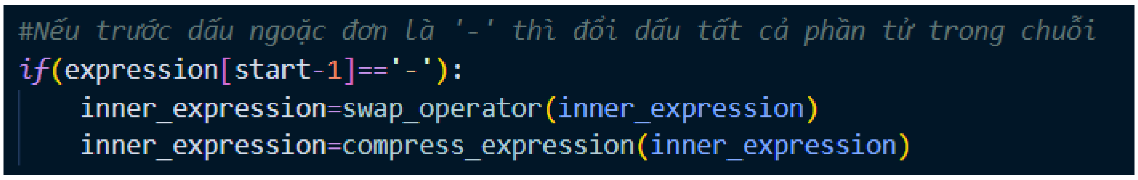
```
for i in range(len(parentheses_content)):
    if(parentheses_content[i]!= '-'): #Kiểm tra dấu đầu chuỗi, nếu không có dấu thì thêm '+'
        parentheses_content[i]='+'+parentheses_content[i]
```

-
    - The function then recursively calls itself on each processed inner expression to remove any remaining outermost parentheses until there are no more parentheses left.



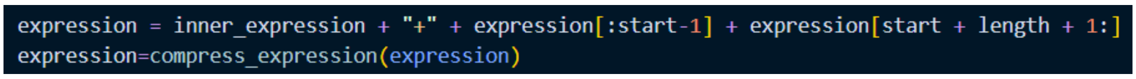
```
parentheses_content[i] = parentheses_content[i].replace('(', '+(')  #Thêm dấu '+' trước '(' để tránh lỗi đọc mảng
parentheses_content[i]=compress_expression(parentheses_content[i])  #Gộp toán tử
```

-
    - If the character before the first opening parenthesis is -, the function swaps the + and - operators in the processed inner expression using the **swap_operator** function.



```
#Nếu trước dấu ngoặc đơn là '-' thì đổi dấu tất cả phần tử trong chuỗi
if(expression[start-1]=='-'):
    inner_expression=swap_operator(inner_expression)
    inner_expression=compress_expression(inner_expression)
```

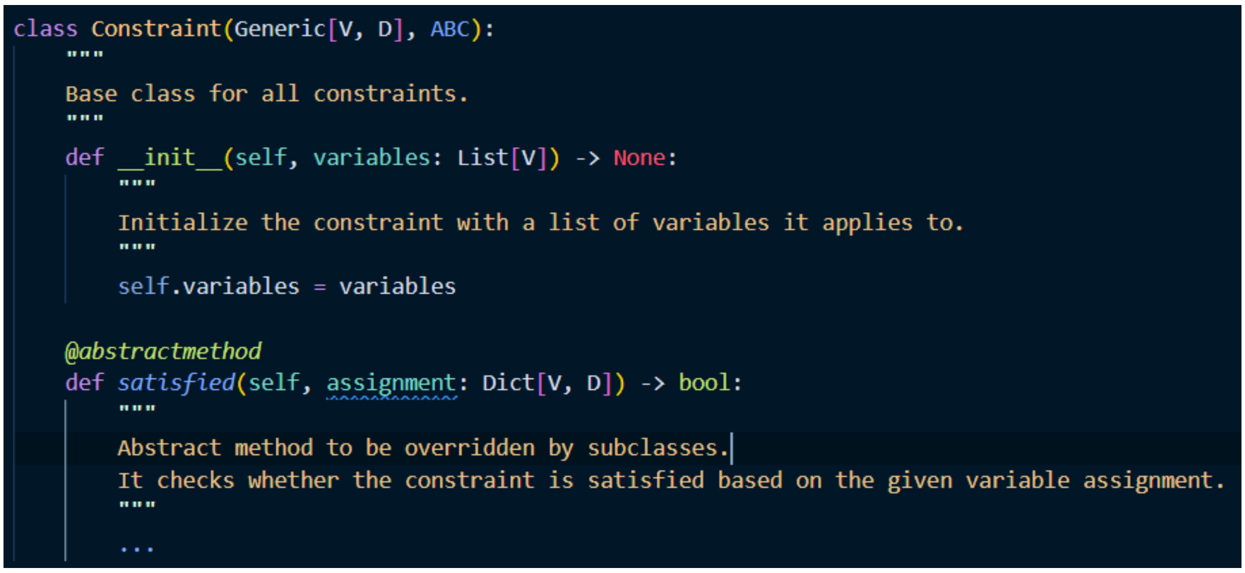
-
    - The function then replaces the original expression with the processed inner expression followed by a + operator and the remaining part of the original expression that comes after the last closing parenthesis of the processed **inner expression**.



```
expression = inner_expression + "+" + expression[:start-1] + expression[start + length + 1:]
expression=compress_expression(expression)
```

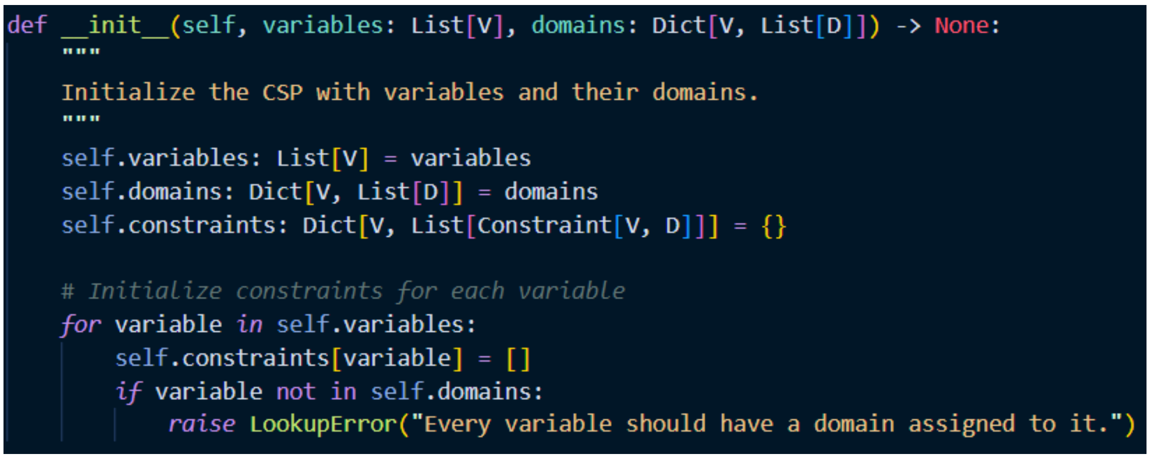
4. Implementation on Level 4

- Firstly, we create **Constraint class** provides a base implementation for all constraints by defining the common variables and methods that are shared across all constraints. This can be useful for building more complex constraint satisfaction problems by subclassing this class and implementing specific constraint logic in the satisfied method.



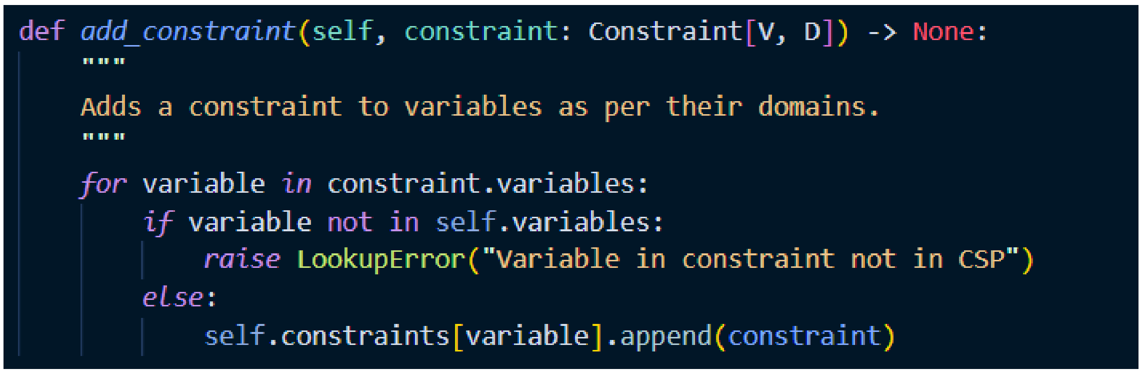
```
class Constraint(Generic[V, D], ABC):
    """
    Base class for all constraints.
    """
    def __init__(self, variables: List[V]) -> None:
        """
        Initialize the constraint with a list of variables it applies to.
        """
        self.variables = variables

    @abstractmethod
    def satisfied(self, assignment: Dict[V, D]) -> bool:
        """
        Abstract method to be overridden by subclasses.
        It checks whether the constraint is satisfied based on the given variable assignment.
        """
        ...
```

- Then, we create **CSP class** provides a framework for defining and solving constraint satisfaction problems by defining the common variables and methods that are shared across all CSPs. This can be useful for building more complex CSPs by subclassing this class and implementing specific CSP logic.
    - ***def __init__(self, variables: List[V], domains: Dict[V, List[D]]) -> None:*** to initialize the instance variables of the class with the given variables and domains and initialize an empty dictionary of constraints for each variable.



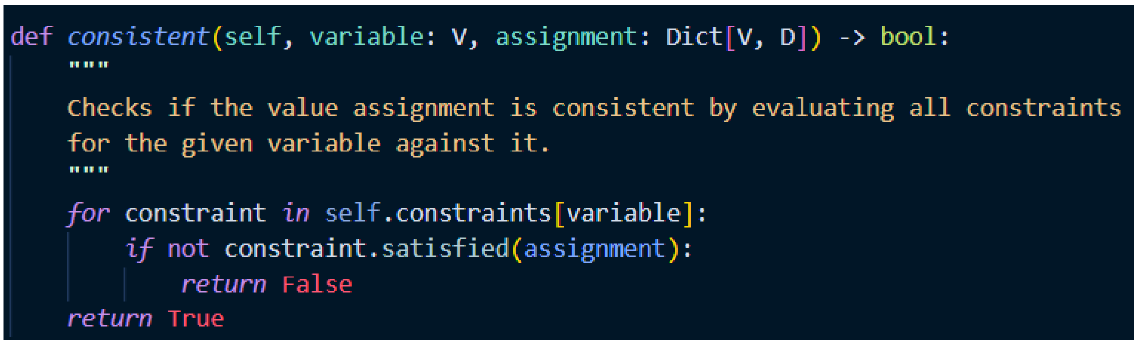
```
def __init__(self, variables: List[V], domains: Dict[V, List[D]]) -> None:
    """
    Initialize the CSP with variables and their domains.
    """
    self.variables: List[V] = variables
    self.domains: Dict[V, List[D]] = domains
    self.constraints: Dict[V, List[Constraint[V, D]]] = {}

    # Initialize constraints for each variable
    for variable in self.variables:
        self.constraints[variable] = []
        if variable not in self.domains:
            raise LookupError("Every variable should have a domain assigned to it.")
```

    - ***def add_constraint(self, constraint: Constraint[V, D]) -> None***: adds it to the constraints for each variable in the constraint's variable list.



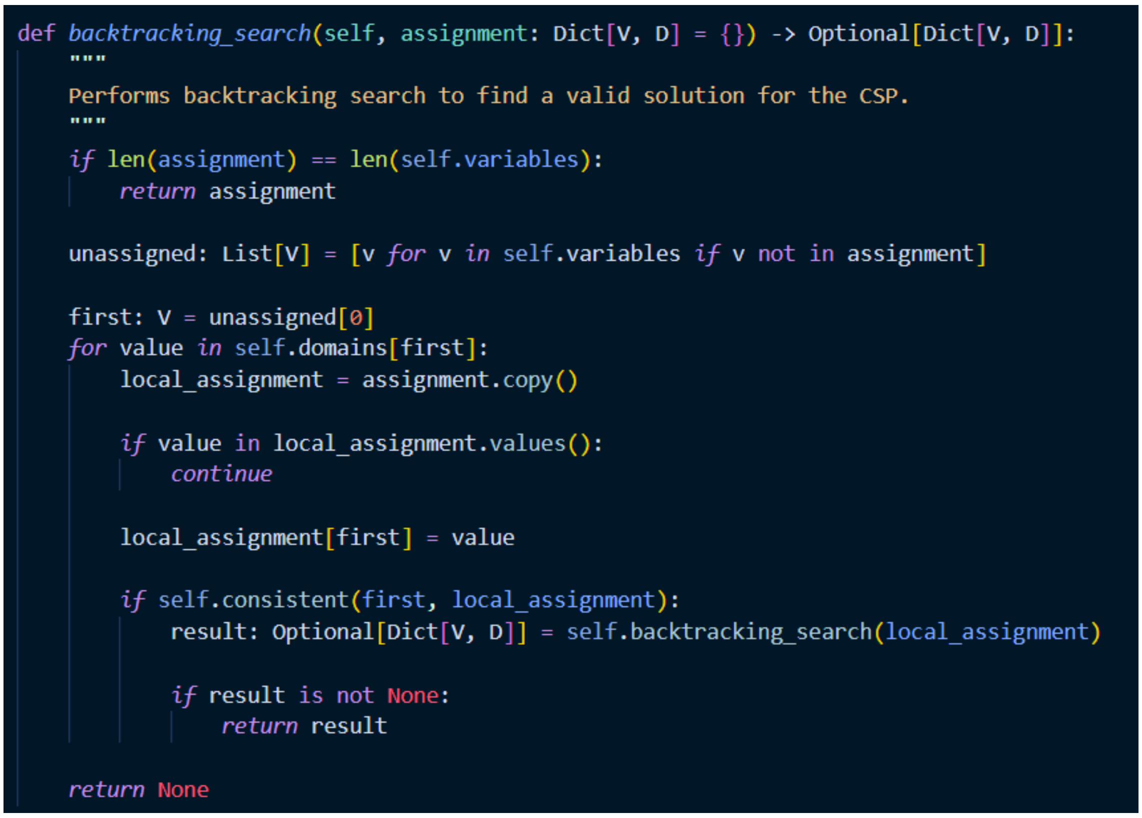
```python
def add_constraint(self, constraint: Constraint[V, D]) -> None:
    """
    Adds a constraint to variables as per their domains.
    """
    for variable in constraint.variables:
        if variable not in self.variables:
            raise LookupError("Variable in constraint not in CSP")
        else:
            self.constraints[variable].append(constraint)
```

- ***def consistent(self, variable: V, assignment: Dict[V, D]) -> bool:*** is to check if the assignment is consistent by evaluating all constraints for the given variable against it.



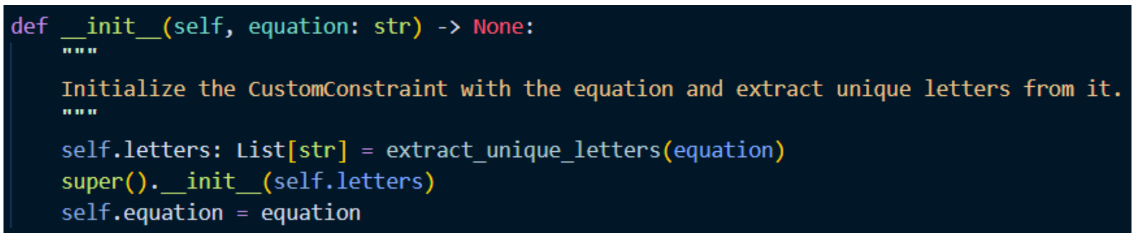
```python
def consistent(self, variable: V, assignment: Dict[V, D]) -> bool:
    """
    Checks if the value assignment is consistent by evaluating all constraints
    for the given variable against it.
    """
    for constraint in self.constraints[variable]:
        if not constraint.satisfied(assignment):
            return False
    return True
```

- ***def backtracking_search(self, assignment: Dict[V, D] = {}) -> Optional[Dict[V, D]]:*** is to find a valid solution for the CSP. It takes an optional assignment dictionary as input and returns a dictionary of variable assignments if a valid solution is found, or None if no solution is found.



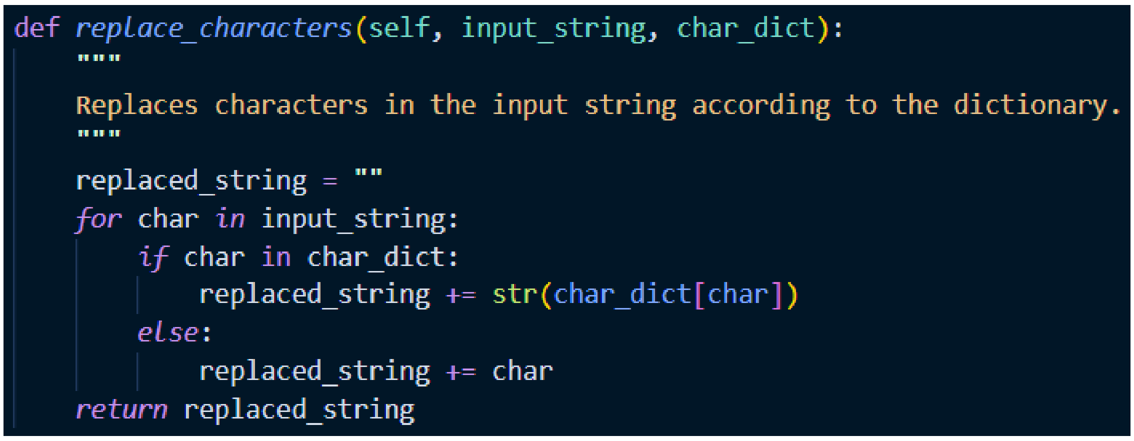
```python
def backtracking_search(self, assignment: Dict[V, D] = {}) -> Optional[Dict[V, D]]:
    """
    Performs backtracking search to find a valid solution for the CSP.
    """
    if len(assignment) == len(self.variables):
        return assignment

    unassigned: List[V] = [v for v in self.variables if v not in assignment]

    first: V = unassigned[0]
    for value in self.domains[first]:
        local_assignment = assignment.copy()

        if value in local_assignment.values():
            continue

        local_assignment[first] = value

        if self.consistent(first, local_assignment):
            result: Optional[Dict[V, D]] = self.backtracking_search(local_assignment)

            if result is not None:
                return result

    return None
```

- Then, we define **CustomConstraint class** provides a way to define a custom constraint based on a mathematical equation with unique variables and evaluate it using the Constraint superclass. This can be useful for building more complex constraint satisfaction problems by creating custom constraints that are specific to the problem.
    - ***def __init__(self, equation: str) -> None:*** is to takes an equation as input, extracts unique letters from it, and initializes the instance variables of the class with the extracted letters and the Constraint superclass with those letters.



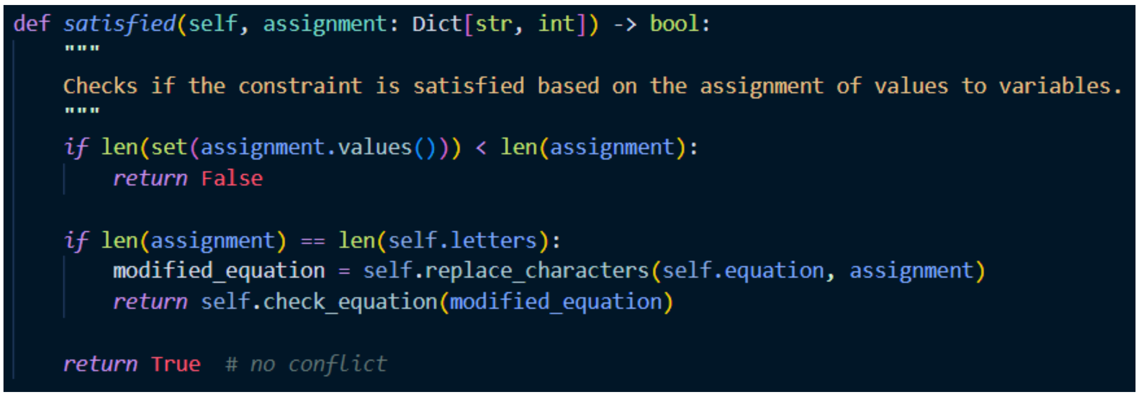
```python
def __init__(self, equation: str) -> None:
    """
    Initialize the CustomConstraint with the equation and extract unique letters from it.
    """
    self.letters: List[str] = extract_unique_letters(equation)
    super().__init__(self.letters)
    self.equation = equation
```

    - ***def replace_characters(self, input_string, char_dict):*** replaces characters in the input string according to the dictionary.



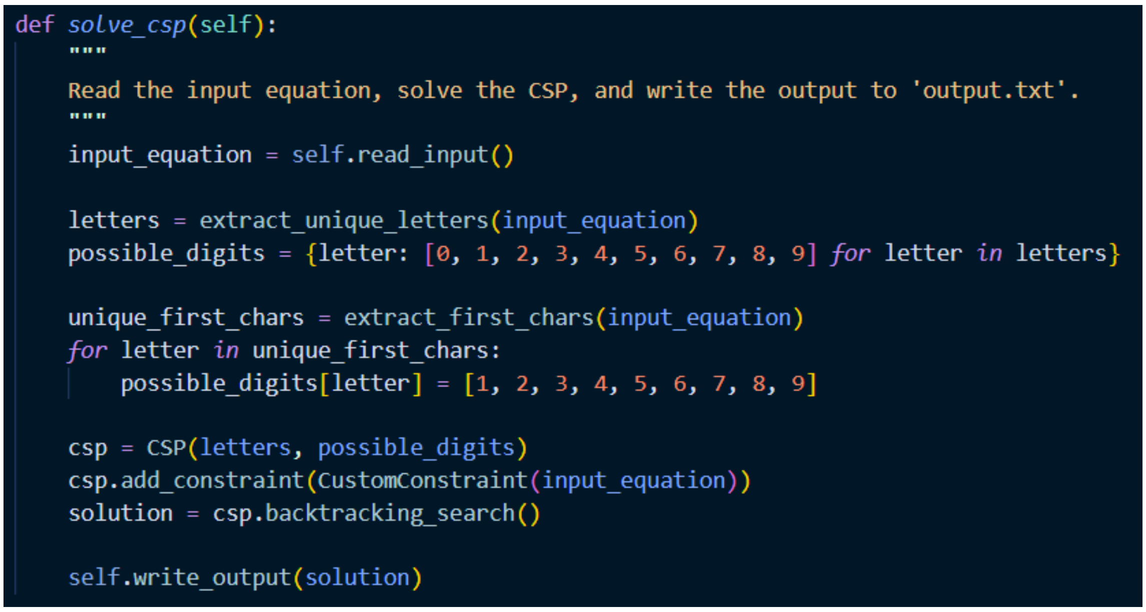
```python
def replace_characters(self, input_string, char_dict):
    """
    Replaces characters in the input string according to the dictionary.
    """
    replaced_string = ""
    for char in input_string:
        if char in char_dict:
            replaced_string += str(char_dict[char])
        else:
            replaced_string += char
    return replaced_string
```

    - ***def check_equation(self, input_string):*** is to check if the equation is correct by evaluating the left and right sides and comparing the results.



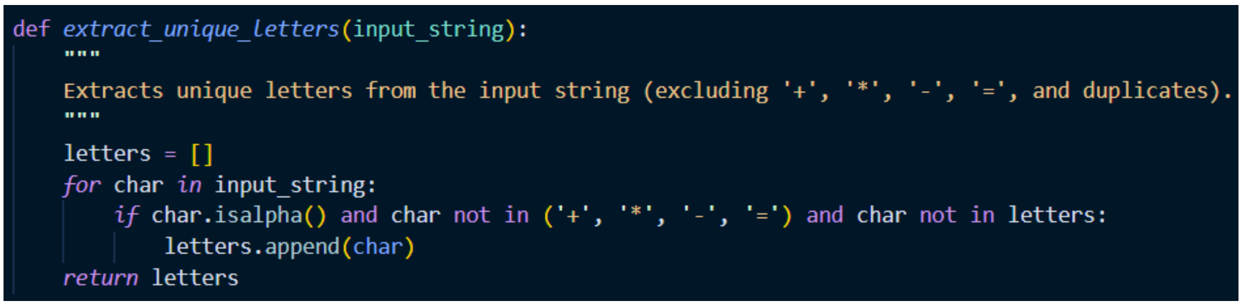
```python
def satisfied(self, assignment: Dict[str, int]) -> bool:
    """
    Checks if the constraint is satisfied based on the assignment of values to variables.
    """
    if len(set(assignment.values())) < len(assignment):
        return False

    if len(assignment) == len(self.letters):
        modified_equation = self.replace_characters(self.equation, assignment)
        return self.check_equation(modified_equation)

    return True  # no conflict
```

- Finally, we define **CSPSolver Class** which provides a convenient way to solve a CSP based on an input equation and write the solution to an output file. It can be used as a standalone script or integrated into a larger program.
  - ***def __init__(self, input_file='input.txt', output_file='output.txt'):*** that takes two optional input arguments, **input_file** and **output_file**, which are set to 'input.txt' and 'output.txt' by default.
  - ***def read_input(self) and def write_output(self, solution) :*** are defined to handle input and output file which are used to solve CSP problem.
  - ***def solve_csp(self):*** that uses the **read_input()** method to read the input equation, extracts unique letters from it, initializes a dictionary of possible digits for each letter, creates a CSP object with the unique letters and possible digits, adds a custom constraint based on the input equation to the CSP, solves the CSP using the **backtracking_search()**, and writes the solution to output.txt using the **write_output()** method.



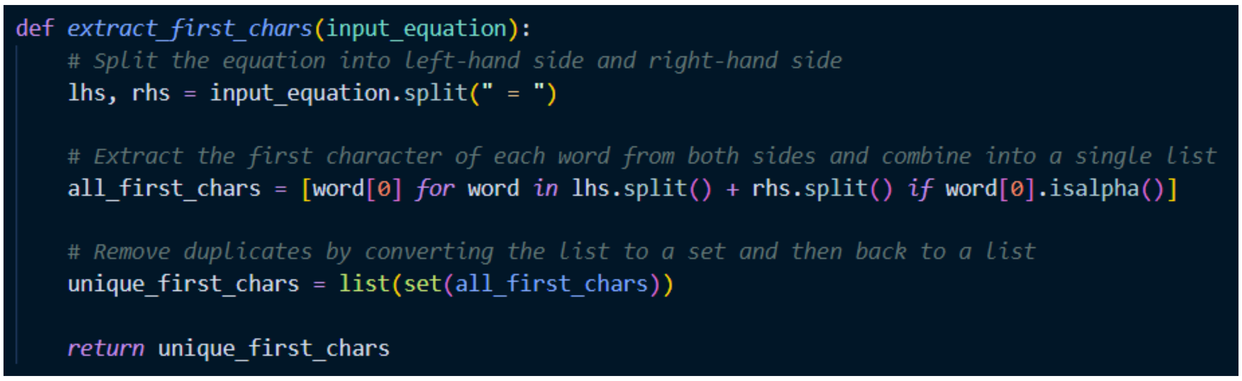
```python
def solve_csp(self):
    """
    Read the input equation, solve the CSP, and write the output to 'output.txt'.
    """
    input_equation = self.read_input()

    letters = extract_unique_letters(input_equation)
    possible_digits = {letter: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] for letter in letters}

    unique_first_chars = extract_first_chars(input_equation)
    for letter in unique_first_chars:
        possible_digits[letter] = [1, 2, 3, 4, 5, 6, 7, 8, 9]

    csp = CSP(letters, possible_digits)
    csp.add_constraint(CustomConstraint(input_equation))
    solution = csp.backtracking_search()

    self.write_output(solution)
```

- ***def extract_unique_letters(input_string):*** provides a way to extract unique letters from a string and ignore any duplicates or non-alphabetic characters. It is used in the CustomConstraint class to extract unique variables from the input equation.



```python
def extract_unique_letters(input_string):
    """
    Extracts unique letters from the input string (excluding '+', '*', '-', '=', and duplicates).
    """
    letters = []
    for char in input_string:
        if char.isalpha() and char not in ('+', '*', '-', '=') and char not in letters:
            letters.append(char)
    return letters
```

- ***def extract_first_chars(input_equation):*** is to extract the first character of each word on both sides of the equation and return a list of unique characters. It is used in the **CSPSolver class** to identify which variables must have a non-zero value.



```python
def extract_first_chars(input_equation):
    # Split the equation into left-hand side and right-hand side
    lhs, rhs = input_equation.split(" = ")

    # Extract the first character of each word from both sides and combine into a single list
    all_first_chars = [word[0] for word in lhs.split() + rhs.split() if word[0].isalpha()]

    # Remove duplicates by converting the list to a set and then back to a list
    unique_first_chars = list(set(all_first_chars))

    return unique_first_chars
```

V. REFLECTION AND COMMENTS

- After using CSP and Brute Force for Cryptarithmetic problems, Constraint Satisfaction Problem (CSP) for **solving cryptarithmetic problems has proven to be an effective approach**. Because CSP allows us to model the problem domain as a set of variables, domains, and constraints, and then use search techniques to find a solution that satisfies all the constraints.
- CSP also allows us to perform constraint propagation, which involves using the constraints to eliminate values from the domains of variables and to detect inconsistencies early in the search process. By using a CSP solver, we can efficiently explore the solution space and prune the search tree by eliminating inconsistent assignments. This can help reduce the search time and improve the overall performance of the algorithm.
- However, **the effectiveness of CSP heavily depends on how well the problem is modeled and how suitable the search algorithm is for the given problem instance**. In some cases, the complexity of the problem may still require a significant amount of time to solve, even with CSP.

⇨ **Overall, CSP is a powerful technique for solving cryptarithmetic problems which can reduce the complexity of the problem.**

VI. USAGE – DEMO

- **Environment to compile and run the program:** Python 3.10.9 (or later)
- **Step 1:** Enter all the problems in the file input corresponding to each level (Ex: input_lv1.txt to solve Level 1).
- **Step 2:** Open the corresponding **.py** file (Ex: csp_lv_1.py to solve Level 1).
- **Step 3:** Compile and run code.
- **Step 4:** Show the result in file output corresponding to each level (Ex: output_lv1.txt to see result of Level 1)
- **Link Video Demo (note: turn on subtitle for clear view):**
  Cryptarithmetic Problem in AI - Intro2AI - HCMUS

VII. REFERENCE

1. Slide Lectures of Mrs. Nguyễn Ngọc Thảo
2. https://www.researchgate.net/publication/224099509_Solving_Cryptarithmetic_Problems_Using_Parallel_Genetic_Algorithm
3. https://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf