A light bulb with magnifying glass

Description automatically generated

**Table of Contents**

[**Table of Contents** 2](#_Toc141354748)

[**I.** **GENERAL TO THE PROBLEM** 3](#_Toc141354749)

[1. Description of Cryptarithmetic Problem 3](#_Toc141354750)

[2. Requirement 4](#_Toc141354751)

[3. Team Members Introduction 4](#_Toc141354752)

[4. Assignment Plan 4](#_Toc141354753)

[**II.** **CONTRIBUTION AND GRADING** 5](#_Toc141354754)

[1. Contribution 5](#_Toc141354755)

[2. Grading 5](#_Toc141354756)

[**III.** **OVERVIEW ON PROBLEM USING BRUTE FORCE** 6](#_Toc141354757)

[1. Algorithm to Approach 6](#_Toc141354758)

[2. Observation 7](#_Toc141354759)

[**IV.** **ALGORITHM AND IMPLEMENTATION USING CSP** 7](#_Toc141354760)

[1. Main Algorithm to Solve Problem 7](#_Toc141354761)

[2. Implementation on Level 1 and Level 2 9](#_Toc141354762)

[3. Implementation on Level 3 13](#_Toc141354763)

[4. Implementation on Level 4 15](#_Toc141354764)

[**V.** **REFLECTION AND COMMENTS** 20](#_Toc141354765)

[**VI.** **USAGE – DEMO** 21](#_Toc141354766)

[**VII.** **REFERENCE** 21](#_Toc141354767)

1. **GENERAL TO THE PROBLEM**
2. **Description of Cryptarithmetic Problem**

**A screenshot of a computer game

Description automatically generated**

**A white text with black text

Description automatically generated**

1. **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** | |  |

1. **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 |

1. **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 |

1. **CONTRIBUTION AND GRADING**
2. **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. |

1. **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%** |

1. **OVERVIEW ON PROBLEM USING BRUTE FORCE**
2. **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*** |

1. **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.

1. **ALGORITHM AND IMPLEMENTATION USING CSP**
2. **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)**

1. **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.

A computer screen shot of a code

Description automatically generated

* ***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.

A blue background with red and blue text

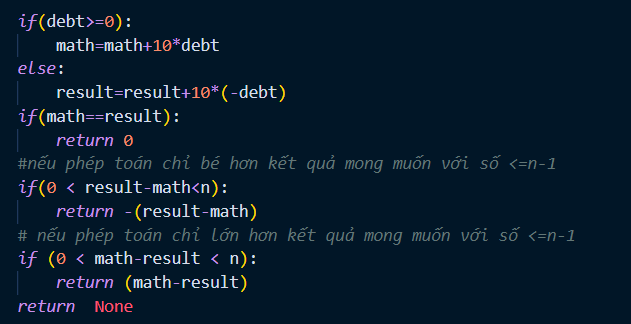
Description automatically generated

* ***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.

A computer code on a dark background

Description automatically generated

* + 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.**



* ***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.

A computer screen with numbers and symbols

Description automatically generated

* + 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.

A blue background with white text

Description automatically generated

* + 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.

A computer screen with text

Description automatically generated

* + 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.

A computer screen shot of text

Description automatically generated

* ***def analy\_string(str):*** 
  + **F**irstly,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.

A screen shot of a computer program

Description automatically generated

* + 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.

A computer screen shot of code

Description automatically generated

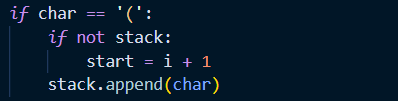
* + It then populates the matrix with the characters from the operands and result lists. The function sorts the **alphab** dictionary by key and calls the **checkcolumn** function with parameters 0, matrix, 0, **alphab**, operators, n, and num. It then calls the **showresult()** function with parameters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

A screen shot of a computer code

Description automatically generated

1. **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.



* + 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.

A blue screen with white text

Description automatically generated

* ***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.

A screen shot of a computer code

Description automatically generated

* ***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.

A screen shot of a computer code

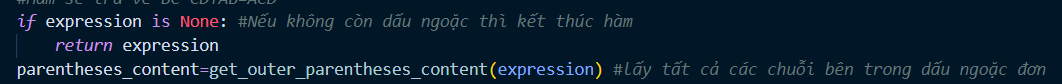
Description automatically generated

* ***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.

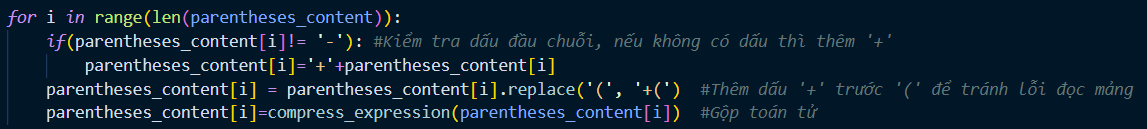
A computer screen shot of a computer code

Description automatically generated

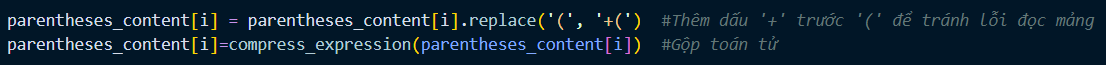
* ***def process\_string(expression):*** is used to transform a level 3 mathematical expression into the level 2 format by pushing all expressions enclosed in outermost parentheses to the beginning of the string and processing them first.
  + The function first checks if the input string expression is 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.



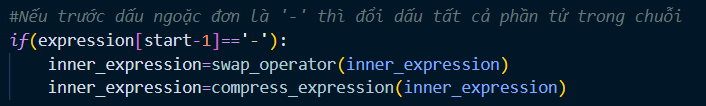
* + Then, we iterate over each expression enclosed in outermost parentheses and processes them by adding a + operator at the beginning of the expression if it does not already have one, replacing all opening parentheses with + ( to avoid array indexing errors, and compressing the expression using the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



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



* + 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.



* + 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**.



1. **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.

A screen shot of a computer code

Description automatically generated

* 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.

***A computer screen shot of text

Description automatically generated***

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

A screen shot of a computer program

Description automatically generated

* + ***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.

A screen shot of a computer code

Description automatically generated

* + ***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.

A computer screen shot of text

Description automatically generated

* 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.

A computer screen with text

Description automatically generated

* + ***def replace\_characters(self, input\_string, char\_dict):*** replaces characters in the input string according to the dictionary.

A computer screen with text

Description automatically generated

* + ***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.

A screen shot of a computer code

Description automatically generated

* 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.

A computer screen shot of a program code

Description automatically generated

* ***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.

A computer screen shot of text

Description automatically generated

* ***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.

A computer code with colorful text

Description automatically generated

1. **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.**

1. **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**](https://www.youtube.com/watch?v=wf8ES3GDsAQ&ab_channel=BaoPhamHongGia)

1. **REFERENCE**
2. Slide Lectures of Mrs. Nguyễn Ngọc Thảo
3. https://www.researchgate.net/publication/224099509\_Solving\_Cryptarithmetic\_Problems\_Using\_Parallel\_Genetic\_Algorithm
4. https://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf