Charles Nana Kwakye (r0879035) Cryptoarithmetic Puzzle solution

Live Streamlit App - https://charles-cryptographic-puzzle-solver.streamlit.app/

Brief Assignment: Use simpleai to solve cryptoarithmetic puzzles. The assignment objective is to develop a program that can solve cryptarithmetic puzzles. Cryptarithmetic puzzles are word problems that use numbers to represent letters, and the goal is to find the correct numerical values for the letters so that the equation in the puzzle is true. The program should allow the user to enter the cryptarithmetic puzzle as input, and the program will then generate the appropriate constraints and solve them to generate a solution. The program should also be well-commented and have a clear structure.

- Ref: Bard AI:
- Prompt 1: Go to my latest google doc called assignment and give me a summary or brief to use in my assignment report
- Prompt 2: Can you give some substantial amount of brief

- Import simpleai and use it to implement and backtracing
- Import regex and use to search for a pattern in a text since this solution requires a user input

In [ ]:
```
#import simple and CspProblem
from simpleai.search import CspProblem, backtrack
import re #import regex
```

Perform some processing on the user input

Expample of such processing are:

- Remove the spaces in the input
- Divide the equation into parts
- Find some parts in the equation
- Find the variables of the equation etc

Ref: Bing AI Prompt: Give me a regex code that finds words from an equation example SEND+MORE=MONEY should give me SEND,MORE,MONEY

In [ ]:
```
user_input = input("Enter a cryptographic puzzle")# Ask for a user input
user_input = user_input.replace(" ", "") #remove spaces in the input by r

equal_sign_index = user_input.index("=") # find the index of the equal to
```

```
factor = user_input[:equal_sign_index+1] #slice the left hand side of the
result = user_input[equal_sign_index+1:] # slice the right hand from the


all_words = re.findall("[a-zA-Z]+", user_input) # Use regex expression to

not_zero_letters = set([word[0] for word in all_words]) # Create a set of

variables = tuple(set(tuple(input_str for s in all_words for input_str in
```

Create a dictionary domain, this is a list of values that each variable can take. They should be selected based on the position of the characted in the equation

In [ ]:
```
domains = {} #Create an empty dictionary to store domains since we alread

# for each variable assign a list of range 1 to 10 if this variable is in
#  or else  if it is not present in the not_zero_letter set then assign a
for variable in variables:
    if variable in not_zero_letters:
        domains[variable] = list(range(1, 10))
    else:
        domains[variable] = list(range(0, 10))
```

Section to make a unique constraint i.e every value is supposed to be different

- Make a constraint function that would expect all different variable to have a different value

In [ ]:
```
def constraint_unique(variables, values):
    return len(values) == len(set(values))  # remove repeated values and
```

Section to generate dynamic constraint

This section is how I converted the user input would consist of alphabets and operators. I had to convert this into a mathematical expression that can be evaluated. The value of the result is needed to compare the left side of the equation to the right side over the "=" sign. To do this I had to go over these steps

- Loop through each character in the factor, which is the left side of the equal sign
- If the character is a letter, find its index in the tuple of variables and append the corresponding value to a string called processed_factor
- If the character is an operator, add the processed_factor to a list of integers called int_list_processed_factor and add the operator to a list of strings called str_list_processed_operations. Then reset the processed_factor to an empty string
- Loop through each character in the result, which is the right side of the equal sign

- If the character is a letter, find its index in the tuple of variables and append the corresponding value to a string called processed_result
- Convert the processed_result to an integer and store it in a variable called int_processed_result
- Initialize an empty string to store the expression
- Loop through the int_list_processed_factor and str_list_processed_operations lists and concatenate them into a string
- Evaluate the string as a mathematical expression and compare it with the int_processed_result
- Return True if they are equal, False otherwise

Ref: Bing AI. How to combine two list in python

- Prompt: I have two lists in python list_1= ["x","y","z"] and list_2 =["+","-"] give me a python code that can combine both into something like x +y-z

Ref: Bing AI.

- NB : After writing the code making the code. Writing the explanation was a little tiring so I asked bing to explain the steps of the code.
- Prompt: Can you explain the def constraint_dynamic_operation of this code in steps

In [ ]:
```
def constraint_dynamic_operation(variables, values):
    int_list_processed_factor = [] # create a list to hold each factor, t
    str_list_processed_operations =[] # create a list to hold the operati
    int_processed_result = int # create anothe list to hold only integers
    processed_factor = "" # a string variable to hold a string of factor(
    processed_result = "" # a string variable to hold the result
    for char in factor:
    # if the character is a letter, find its index in the tuple and appen
        if char.isalpha():
            processed_factor += str(values[variables.index(char)])
    #else convert if to an integer and add it to the processed factor
        else:
            int_list_processed_factor.append(int(processed_factor))
            #  if the character is an eqaul to sign, convert it to a stri
            if char != "=":
                str_list_processed_operations.append(str(char))
                # empty the processed factor string
            processed_factor = ""

    #loop over each character in the result, find matching value using th
    #Convert this to a string and append it to processed result
    for char in result:
        processed_result += str(values[variables.index(char)])
    int_processed_result = int(processed_result) # finally convert everyt

    # make an empty string variable,for the range of the length of the in
    string = ""
    for i in range(len(int_list_processed_factor)):
        string+= str(int_list_processed_factor[i])
        if i < len(str_list_processed_operations):
            string+= str_list_processed_operations[i]
```

```
    #use built in python function to evaluate this string and compare the
    return eval(string)== int_processed_result # return true if they are
```

Section to create the CspProblem object.

- The three major components needed are variables, domains, and constraints.
- The constraints are first created using the variables generated earlier on and the functions created
- The problem is created usign the the variables,domains which were all generated earlier and the just now created constraints

In [ ]:
```
#generate the constraint first and use it to create a csp problem
constraints = [
    (variables, constraint_unique),
    (variables, constraint_dynamic_operation)
]


problem = CspProblem(variables, domains, constraints)
```

Backtrack using cspProblem

The backtrack function is a recursive algorithm that tries to assign values to variables one by one, and backtracks if it encounters a conflict. The output of the backtrack function is a dictionary that maps each variable name to a value that satisfies the constraints, or None if no solution exists.

Ref : Bing AI Prompt: I have a dictionary example like {'N': 0, 'V': 3, 'O': 6, 'D': 5, 'E': 1}, I also have a string like this ODD+ODD=EVEN use each character to get the value from the diction so I should have something like this 655+755 = 1310 ni return. Give me a python code to do this.

In [ ]:
```
# create an output variable and call the backtract algorithm on the cspPr
output = backtrack(problem)

#create an empty string variable to hold the keys in the form of the equa
nums_output = ""

# for each variable in the user input, if the variable is a character use
#convert it to a string and add it to the nums_output
for char in user_input:
    if char.isalpha():
        nums_output+= str(output.get(char))
    else: # if its not an alphabet add it directly to the nums output
        nums_output+=char
print('\nSolutions:', output)
print(nums_output)
print(user_input)
```

```
Solutions: {'D': 5, 'E': 1, 'O': 6, 'N': 0, 'V': 3}
655+655=1310
ODD+ODD=EVEN
```

`This assignment is based on the work of Charles Nana Kwakye by the help of Bing AI and Bard AI .`