Mark Bekker

Mb6386

Cryptarithmetic Solver Instructions

To run the code simply open “solver.py” in any IDE that can run Python code. Then in the code underneath “if __name__ == “__main__” you will find where you can modify the input file name which is assigned to the variable “FILE_NAME” (line 129). Simply modify this variable to the name of your file and then run the code.

If there is a valid solution to the puzzle, an output file will be generated in the same directory as the solver.py file and the name of the output file is printed to the console. If a solution to the puzzle does not exist, the console will print “No solution exists.”

Output file solutions:

For Input1.txt:

9567

1085

10652

For Input2.txt:

7483

7455

14938

Code for solver.py:

```
#Class for Constraint Satisfaction Problem
class CSP:
    def __init__(self, letters, domains):
        self.letters = letters  #letters to be constrained
        self.domains = domains  #domain of each variable
        self.constraints = {}

        for letter in self.letters:
            self.constraints[letter] = []
```

```
    #Add constraint
    def addConstraint(self, constraint):
        for letter in constraint.letters:
            self.constraints[letter].append(constraint)

    #Check if the value assignment is consistent by checking all constraints
    def checkConstraints(self, variable, assignment):
        for constraint in self.constraints[variable]:
            if not constraint.satisfied(assignment):
                return False
        return True

    #BackTracking search implementation
    def backtrackingSearch(self, assignment = {}):
        # assignment is complete if every variable is assigned (our base case)
        if len(assignment) == len(self.letters):
            return assignment

        #Get all variables in the CSP but not in the assignment
        unassigned = []
        for variable in self.letters:
            if variable not in assignment:
                unassigned.append(variable)


        #Get the letter with the minimum remaining values in domain
        min_values_letter = """
        minimum_remaining_values = 10
        for letter in unassigned:
            if len(self.domains[letter]) <= minimum_remaining_values:
                minimum_remaining_values = len(self.domains[letter])
                min_values_letter = letter

        #Get every possible domain value of the minimum remainint value letter
        for value in self.domains[min_values_letter]:
            local_assignment = assignment.copy()
            local_assignment[min_values_letter] = value
            # if we're still consistent, we recurse (continue)
            if self.checkConstraints(min_values_letter, local_assignment):
                result = self.backtrackingSearch(local_assignment)
                #If result is not found we backtrack and recurse
                if result is not None:
                    return result
        return None

#Class for Cryptarithmetic Solver
class CryptarithmeticSolver:
    def __init__(self, letters, words):
```

```
        self.letters = letters
        self.words = words

    #Check if conditions are satisfied
    def satisfied(self, assignment):
        # if there are duplicate values then it's not a solution
        if len(set(assignment.values())) < len(assignment):
            return False
        first = []
        second = []
        solution = []

        # if all variables have been assigned, check if it adds correctly
        if len(assignment) == len(self.letters):
            for letter in self.words[0]:
                first.append(assignment[letter])
            for letter in self.words[1]:
                second.append(assignment[letter])
            for letter in self.words[2]:
                solution.append(assignment[letter])

            #Assign the numbers to the correct places
            first_num = first[0] * 1000 + first[1] * 100 + first[2] * 10 + first[3]
            second_num = second[0] * 1000 + second[1] * 100 + second[2] * 10 + second[3]
            solution_num = solution[0] * 10000 + solution[1] * 1000 + solution[2] * 100 + solution[3] * 10 +
solution[4]
            return first_num + second_num == solution_num
        return True

#Return first, second, and third word (sum) from input file
def getWordsFromFile(filename):
    file = open(filename, "r")
    first = file.readline().strip()
    second = file.readline().strip()
    solution = file.readline().strip()
    file.close()
    return first, second, solution

#Return all unique letters from the three words of the input
def getLetters(words):
    letters = []
    for word in words:
        for letter in word:
            if letter.upper() not in letters:
                letters.append(letter.upper())
    return letters

#Lower the domains of some of the letter variable values
```

```python
def initialAnalysis(values, words):
    # Remove 0 from the range of potential values for letters in the first position of the words
    values[words[0][0]] = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    values[words[1][0]] = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    # Set the the value of the first letter of the third word to 1 by default as it can not be anything else
    values[words[2][0]] = [1]
    return values

#Write the solution to a new file with the name of the old file + Output.txt
def writeFile(filename, answer, words):
    output_filename = f"{filename.split('.')[0]} Output.txt"
    file = open(output_filename, "w")

    for word in words:
        for letter in word:
            file.write(str(answer[letter]))
        file.write('\n')
    file.close()
    return output_filename

#Run the code on user-defined input
if __name__ == "__main__":
    # Name of input file goes here:
    FILE_NAME = "Input3.txt"
    words = getWordsFromFile(FILE_NAME)
    letters = getLetters(words)
    values = {}
    for letter in letters:
        values[letter] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

    values = initialAnalysis(values, words)
    csp = CSP(letters, values)
    solver = CryptarithmeticSolver(letters, words)
    csp.addConstraint(solver)
    answer = csp.backtrackingSearch()

    #Check if solution exists, if not inform user, if it does then write a file with the solution
    if answer is None:
        print("No solution exists")
    else:
        output_filename = writeFile(FILE_NAME, answer, words)
        print(f"Output file produced as {output_filename}")
```