Project 2: Cryptarithmetic Problem

Participants:

Ke Zhang, Yiyang Zhou

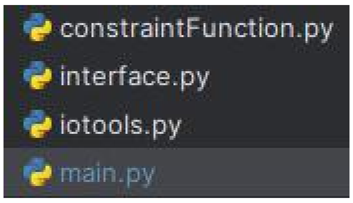
Run this project:

This project requires python3.

Use python [input file path] [output file puth] to run.

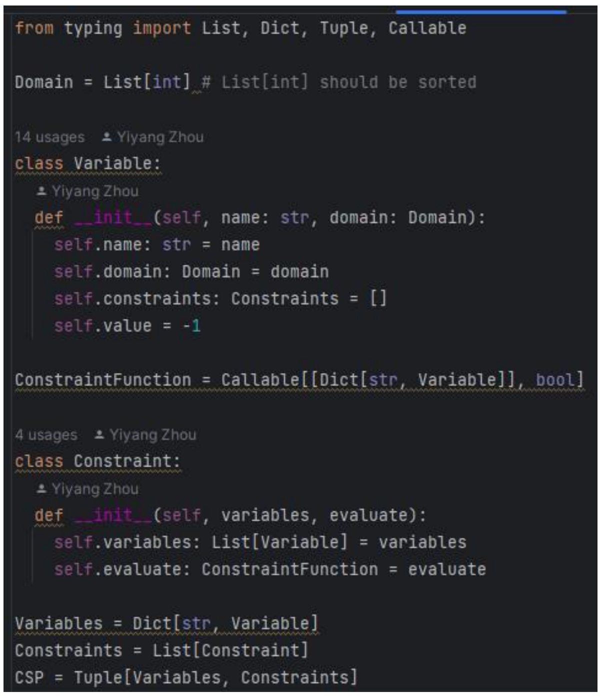
Source code:

The project consists of the following four parts:



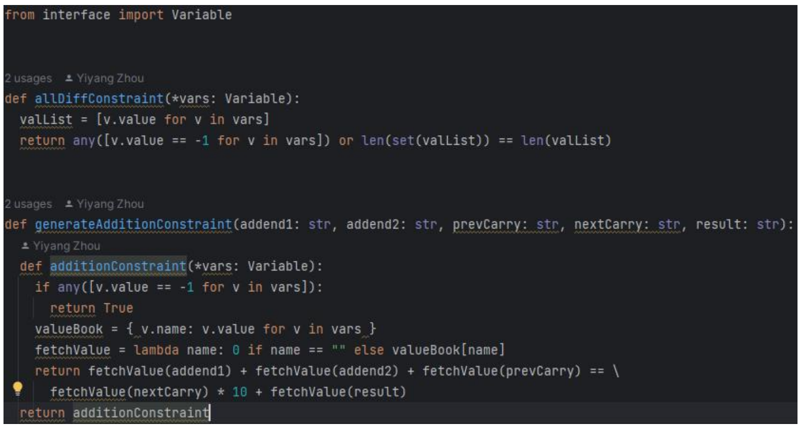
```
constraintFunction.py
interface.py
iotools.py
main.py
```

Firstly, **interface.py** defines the structures and relationships of Variable and Constraint.



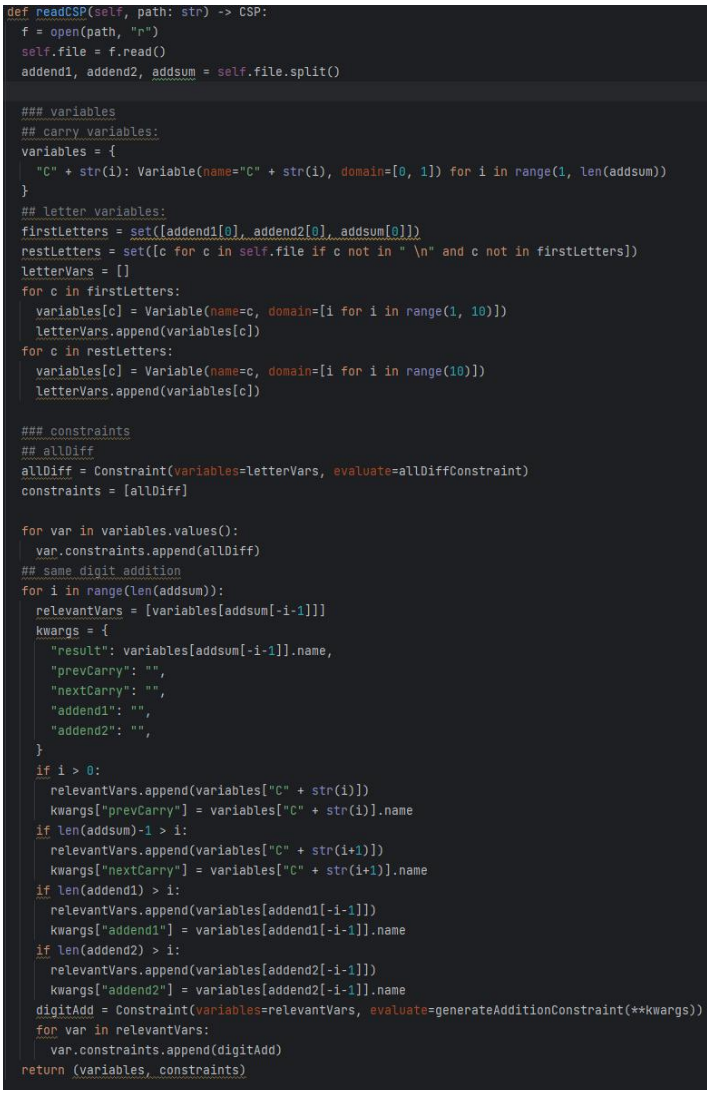
```
from typing import List, Dict, Tuple, Callable

Domain = List[int] # List[int] should be sorted

14 usages  Yiyang Zhou
class Variable:
    Yiyang Zhou
    def __init__(self, name: str, domain: Domain):
        self.name: str = name
        self.domain: Domain = domain
        self.constraints: Constraints = []
        self.value = -1

ConstraintFunction = Callable[[Dict[str, Variable]], bool]

4 usages  Yiyang Zhou
class Constraint:
    Yiyang Zhou
    def __init__(self, variables, evaluate):
        self.variables: List[Variable] = variables
        self.evaluate: ConstraintFunction = evaluate

Variables = Dict[str, Variable]
Constraints = List[Constraint]
CSP = Tuple[Variables, Constraints]
```

The Constraint function is stored in **constraintFunction.py**.



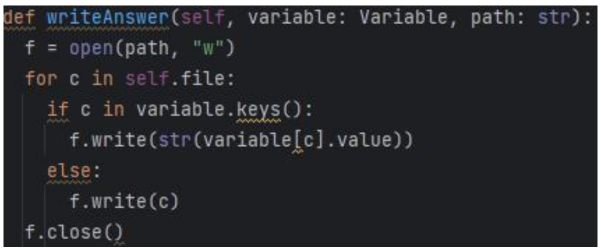
```python
from interface import Variable


2 usages  Yiyang Zhou
def allDiffConstraint(*vars: Variable):
  valList = [v.value for v in vars]
  return any([v.value == -1 for v in vars]) or len(set(valList)) == len(valList)


2 usages  Yiyang Zhou
def generateAdditionConstraint(addend1: str, addend2: str, prevCarry: str, nextCarry: str, result: str):
  Yiyang Zhou
  def additionConstraint(*vars: Variable):
    if any([v.value == -1 for v in vars]):
      return True
    valueBook = { v.name: v.value for v in vars }
    fetchValue = lambda name: 0 if name == "" else valueBook[name]
    return fetchValue(addend1) + fetchValue(addend2) + fetchValue(prevCarry) == \
      fetchValue(nextCarry) * 10 + fetchValue(result)
  return additionConstraint
```

iotools.py handles file reading and writing; reading extracts the contents of **input.txt** and processes it into the format required for CSP problems.



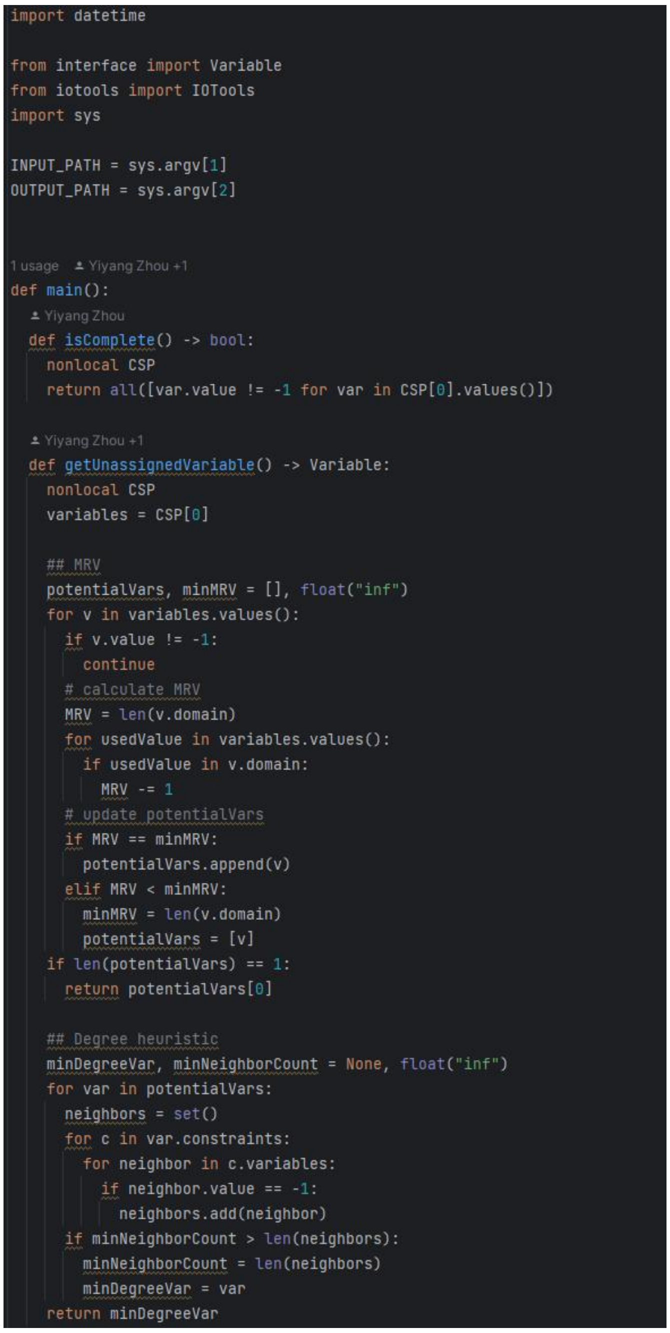
```python
def readCSP(self, path: str) -> CSP:
  f = open(path, "r")
  self.file = f.read()
  addend1, addend2, addsum = self.file.split()

  ### variables
  ## carry variables:
  variables = {
    "C" + str(i): Variable(name="C" + str(i), domain=[0, 1]) for i in range(1, len(addsum))
  }
  ## letter variables:
  firstLetters = set([addend1[0], addend2[0], addsum[0]])
  restLetters = set([c for c in self.file if c not in " \n" and c not in firstLetters])
  letterVars = []
  for c in firstLetters:
    variables[c] = Variable(name=c, domain=[i for i in range(1, 10)])
    letterVars.append(variables[c])
  for c in restLetters:
    variables[c] = Variable(name=c, domain=[i for i in range(10)])
    letterVars.append(variables[c])

  ### constraints
  ## allDiff
  allDiff = Constraint(variables=letterVars, evaluate=allDiffConstraint)
  constraints = [allDiff]

  for var in variables.values():
    var.constraints.append(allDiff)
  ## same digit addition
  for i in range(len(addsum)):
    relevantVars = [variables[addsum[-i-1]]]
    kwargs = {
      "result": variables[addsum[-i-1]].name,
      "prevCarry": "",
      "nextCarry": "",
      "addend1": "",
      "addend2": "",
    }
    if i > 0:
      relevantVars.append(variables["C" + str(i)])
      kwargs["prevCarry"] = variables["C" + str(i)].name
    if len(addsum)-1 > i:
      relevantVars.append(variables["C" + str(i+1)])
      kwargs["nextCarry"] = variables["C" + str(i+1)].name
    if len(addend1) > i:
      relevantVars.append(variables[addend1[-i-1]])
      kwargs["addend1"] = variables[addend1[-i-1]].name
    if len(addend2) > i:
      relevantVars.append(variables[addend2[-i-1]])
      kwargs["addend2"] = variables[addend2[-i-1]].name
    digitAdd = Constraint(variables=relevantVars, evaluate=generateAdditionConstraint(**kwargs))
    for var in relevantVars:
      var.constraints.append(digitAdd)
  return (variables, constraints)
```

The **writeAnswer** function outputs the results of the problem.

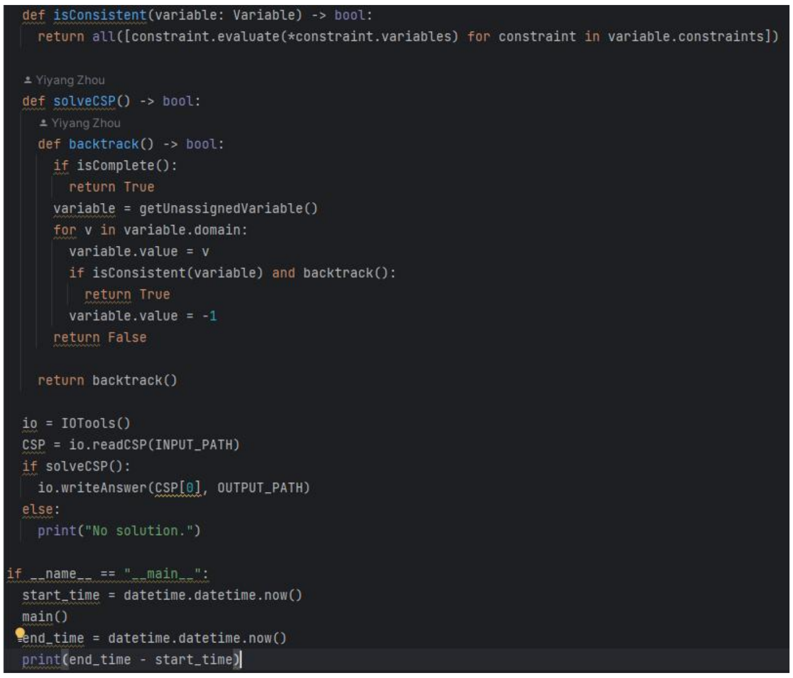


```python
def writeAnswer(self, variable: Variable, path: str):
  f = open(path, "w")
  for c in self.file:
    if c in variable.keys():
      f.write(str(variable[c].value))
    else:
      f.write(c)
  f.close()
```

Lastly, **main.py** contains functions for solving CSP problems and is responsible for handling the main logic.



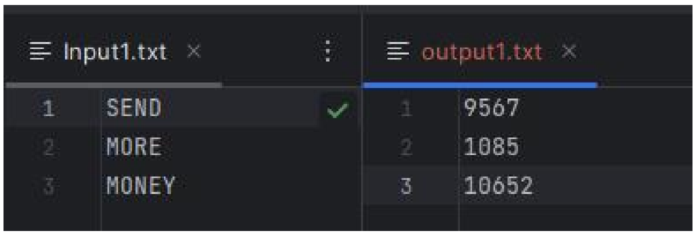
```python
import datetime

from interface import Variable
from iotools import IOTools
import sys

INPUT_PATH = sys.argv[1]
OUTPUT_PATH = sys.argv[2]


1 usage   Yiyang Zhou +1
def main():
  Yiyang Zhou
  def isComplete() -> bool:
    nonlocal CSP
    return all([var.value != -1 for var in CSP[0].values()])

  Yiyang Zhou +1
  def getUnassignedVariable() -> Variable:
    nonlocal CSP
    variables = CSP[0]

    ## MRV
    potentialVars, minMRV = [], float("inf")
    for v in variables.values():
      if v.value != -1:
        continue
      # calculate MRV
      MRV = len(v.domain)
      for usedValue in variables.values():
        if usedValue in v.domain:
          MRV -= 1
      # update potentialVars
      if MRV == minMRV:
        potentialVars.append(v)
      elif MRV < minMRV:
        minMRV = len(v.domain)
        potentialVars = [v]
    if len(potentialVars) == 1:
      return potentialVars[0]

    ## Degree heuristic
    minDegreeVar, minNeighborCount = None, float("inf")
    for var in potentialVars:
      neighbors = set()
      for c in var.constraints:
        for neighbor in c.variables:
          if neighbor.value == -1:
            neighbors.add(neighbor)
      if minNeighborCount > len(neighbors):
        minNeighborCount = len(neighbors)
        minDegreeVar = var
    return minDegreeVar
```



```python
  def isConsistent(variable: Variable) -> bool:
    return all([constraint.evaluate(*constraint.variables) for constraint in variable.constraints])

  Yiyang Zhou
  def solveCSP() -> bool:
    Yiyang Zhou
    def backtrack() -> bool:
      if isComplete():
        return True
      variable = getUnassignedVariable()
      for v in variable.domain:
        variable.value = v
        if isConsistent(variable) and backtrack():
          return True
        variable.value = -1
      return False

    return backtrack()

  io = IOTools()
  CSP = io.readCSP(INPUT_PATH)
  if solveCSP():
    io.writeAnswer(CSP[0], OUTPUT_PATH)
  else:
    print("No solution.")

if __name__ == "__main__":
  start_time = datetime.datetime.now()
  main()
  end_time = datetime.datetime.now()
  print(end_time - start_time)
```

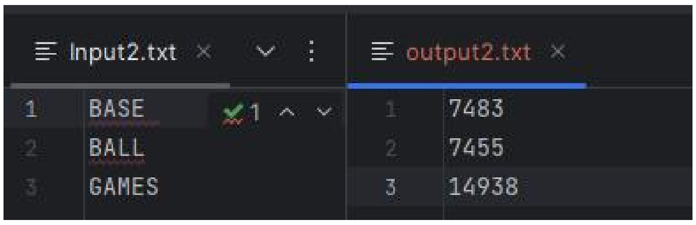
Outputs:

The output results for input1 and input2, as shown in the graphs, are output1 and output2.



```
SEND
MORE
MONEY
```

```
9567
1085
10652
```



```
BASE
BALL
GAMES
```

```
7483
7455
14938
```