KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Programavimo kalbų teorija (P175B124)

Laboratorinių darbų ataskaita

Atliko:
IFF-6/14 gr. studentas
Valdas Germanauskas
2019 m. vasario 24 d.
Priėmė:
Doc. Aštrys Kirvaitis

KAUNAS 2019

TURINYS

1. Python arba Ruby (L1)

1.1. Darbo užduotis

947 - Master Mind Helper
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=11&page=show_problem&problem=888

Input

The first line of input will contain a single integer N, indicating the number of test cases that follow ($1 \leq N \leq 30$).

Then follow exactly N lines, each containing one test case consisting of three parts separated by single spaces. Each case starts with a valid guess, in a form of a string of digits (remember that the colors are coded with digits 1 to 9). This string can have from 2 to 5 digits. Then follows the feedback received in the form of two integer numbers: the first one represents the number of correct colors on correct places, and the second one represents the number of correct colors on wrong places.

Output

For each test case you must output a line containing a single integer representing the number of possible secret codes that would give that feedback for that particular guess. Remember that there are always 9 different colors and that the size of the secret code must be equal to the size of the guess.

1.2. Programos tekstas

```
# L1 Python uzduotis Valdas Germanauskas IFF-6/14
#
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=11&page=show_problem&problem=888

import math
fileName = "data.txt"


class Guess:
    # test condition object
    def __init__(self, guess, black, white):
        self.guess = guess
        self.black = black
        self.white = white


class Code:
    def __init__(self, guessLength):
        self.combinations = []
        self.guessLength = guessLength

    # generates all possible combinations
    def generateCombinations(self):
        start = 0
        for power in range(self.guessLength):
            start += math.pow(10, power)
        end = math.pow(10, self.guessLength)

        for x in range(int(start), int(end)):
            flag = True
            s = list(str(x))
            for symbol in s:
                if(symbol == "0"):
                    flag = False
            if(flag == False):
                continue
            self.combinations.append(s)

        return self.combinations

    # prints number of possible secret codes
    def evaluate(self, guessCode):
        possibleCodes = 0
        # check every combination againts tested one
        # get number or black and white pegs for each combination
        for i in range(self.countCombinations()):
            black = self.checkBlack(guessCode, self.combinations[i])
            white = self.checkWhite(guessCode, self.combinations[i]) - black
            # if number of black and white pegs overlap with test condition
            # possible secret code count increases
```

```
            if(black == int(guessCode.black) and white == int(guessCode.white)):
                possibleCodes += 1
        #print(guessCode.guess, guessCode.black, guessCode.white, possibleCodes)
        print(possibleCodes)

    # returns number of black pegs (correct color and correct position)
    def checkBlack(self, guessCode, correctCode):
        black = 0
        for x in range(self.guessLength):
            if(guessCode.guess[x] == correctCode[x]):
                black += 1
        return black

    # returns number of white pegs (correct color wrong position)
    def checkWhite(self, guessCode, correctCode):
        white = 0
        tempCorrect = guessCode.guess.copy()

        for i in range(self.guessLength):
            for j in range(self.guessLength):
                if(tempCorrect[j] == correctCode[i]):
                    white += 1
                    tempCorrect[j] = 0
                    j = self.guessLength+1
                    break
        return white

    # returns number of all combinations
    def countCombinations(self):
        defaultString = "0"*self.guessLength
        count = 0
        for x in self.combinations:
            if(x != defaultString):
                count += 1

        return count


class Executor:
    def __init__(self):
        self.count = 0  # number of test cases
        self.guessList = []  # test cases list
        self.readData()  # read test data from file

        # for each test case conduct tests
        for i in range(self.count):
            self.code = Code(len(self.guessList[i].guess))
            self.code.generateCombinations()
            self.code.evaluate(self.guessList[i])

    # construct object from line string
    def parseLine(self, line):
        self.guess = line.split()
```

```
        return Guess(list(self.guess[0]), self.guess[1], self.guess[2])

    # read test data
    def readData(self):
        file = open(fileName, "r")
        self.count = int(file.readline())
        for x in range(self.count):
            line = file.readline()
            self.guessList.append(self.parseLine(line))


# execute program
execute = Executor()
```

1.3. Pradiniai duomenys ir rezultatai

Sample Input

```
5
1234 2 2
111 1 0
567 0 1
91543 5 0
91543 0 5
```

Sample Output

```
6
192
234
1
44
```