IF2211 Strategi Algoritma

Tugas Kecil 1

Penyelesaian *Cryptarithmetic* dengan Algoritma *Brute Force*

Oleh:

Raffi Zulvian Muzhaffar
13519003

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2021

Daftar Isi

Algoritma *Brute Force*

Salah satu cara termudah untuk menyelesaikan persoalan *cryptarithmetic* adalah dengan algoritma *brute force*. Namun menggunakan algoritma *brute force* memiliki konsekuensi program yang dibuat akan berjalan dengan sangat lambat.

Dalam menyelesaikan Tugas Kecil 1 ini, penulis membuat sebuah program sederhana yang ditulis menggunakan bahasa Python yang menerapkan sebuah algoritma *brute force* dengan langkah-langkah seperti berikut:

1. Asosiasikan tiap huruf dengan sebuah angka dari 0 sampai 9, dengan setiap huruf memiliki pasangan angka yang berbeda,
2. Substitusikan posisi setiap huruf dengan angka yang bersesuaian dan cek apakah bilangan-bilangan yang terbentuk sesuai dengan aturan (tidak diawali angka 0 dan hasil operasi penjumlahannya sesuai),
3. Jika pasangan huruf-angka belum memenuhi aturan yang berlaku maka ganti angka pada huruf terakhir (paling kanan/posisi satuan) dengan menambahkan 1, namun saat hasil penjumlahannya 10 jadikan angkanya 0 dan tambahkan 1 pada huruf sebelah kirinya (seperti penjumlahan biasa),
4. Ulangi langkah 1 sampai 3 sampai menemukan solusi yang sesuai atau sudah tidak ada lagi kemungkinan pasangan huruf-angka yang bisa dicapai.

Kode Sumber Program

Secara keseluruhan program ini terdiri dari import modul, pendefinisian kelas, pendefinisian fungsi, dan bagian utama. Untuk menunjang keberjalanan program ini maka digunakan tiga buah *library* eksternal. *Library* yang digunakan adalah:

1. *sys*, untuk menampilkan *output* serta membaca soal saat program di-*compile* jadi .exe,
2. *pathlib*, untuk menentukan *path* program dan file soal,
3. *time*, untuk menentukan waktu eksekusi program.

Program ini di-*compile* menjadi sebuah file *executable* menggunakan *library pyinstaller*. Tidak ada satu pun penggunaan *library* yang ditujukan untuk membantu proses utama program ini.

Bagian utama dari program ini akan memanggil *method* pada kelas yang telah didefinisikan untuk menyelesaikan persoalan *cryptarithmetic* dengan algoritma *brute force*. Penyelesaian ini diwujudkan dengan sebuah *method* yang bernama *calculate*. Kode program utuh adalah sebagai berikut:

cryptarithmetic.py

```
import sys
from pathlib import Path
from time import time
```

```
class CryptarithmeticSolver:
    """
    Menyelesaikan persoalan Cryptarithmetic dengan algoritma brute force.

    Parameters
    ----------
    operand : list
        Daftar string kata yang berperan sebagai operan.

    answer : list
        Daftar string kata yang berperan sebagai hasil.

    Attributes
    ----------
    operand : list
        Daftar string kata yang berperan sebagai operan.

    answer : list
        Daftar string kata yang berperan sebagai hasil.

    mapping : dict
        Pemetaan yang menunjukan asosiasi huruf-angka yang bersesuaian.

    num_opr : list
        Daftar integer yang merupakan hasil substitusi operan dengan angka yang
        bersesuaian.

    num_ans : int
        Integer yang merupakan hasil substitusi hasil dengan angka yang
        bersesuaian.

    tries_count : int
        Menyimpan jumlah uji coba/percobaan untuk mencari pemetaan huruf-angka
        yang tepat.

    """
```

```
    def __init__(self, operand, answer):
        self.operand = operand
        self.answer = answer
        self.mapping = {}
        self.num_opr = []
        self.num_ans = 0
        self.tries_count = 0

    def calculate(self) -> bool:
        """
        Bagian utama dari penerapan algoritma brute force. Melakukan iterasi dan
        langkah langkah untuk menyelesaikan persoalan cryptarithmetic.

        """

        is_found = False

        # Melakukan pemetaan awal pada huruf-angka
        self._init_mapping()
        total_letters = len(self.mapping.values())

        # Mengulagi proses sampai ditemukan solusi yang sesuai
        while not is_found and list(self.mapping.values()).count(9) <
total_letters:

            # Menyubstitusikan huruf-huruf pada operan dan hasil serta
            # melakukan pengecekan apakah percobaan saat ini sesuai aturan
            guess = self._subtitute()
            if not guess:

                # Melakukan inkremen pada huruf paling kanan dalam pemetaan
                self._incr()
                self.tries_count += 1
                self.num_opr = []
                self.num_ans = 0
                continue
```

```
            # Mengecek hasil penjumlahan operan apakah sesuai dengan hasil
            is_found = self._evaluate()
            if not is_found:
                self.tries_count += 1
                self.num_opr = []
                self.num_ans = 0
                self._incr()

        return is_found

    def show_problem(self):
        """
        Mencetak persoalan yang terdiri dari operan-operan, garis
        batas serta hasil kelayar dengan format yang ditentukan.

        """

        print('\nPROBLEM', '=======', sep='\n')

        # Mencetak operan-operan
        for i in range(len(self.operand)):
            space = len(self.answer[0]) - len(self.operand[i])
            if len(self.operand) - 1 - i == 0:
                print(' '*space, self.operand[i], '+', sep='')
            else:
                print(' '*space, self.operand[i], sep='')

        # Mencetak garis batas serta hasil
        print('-' * len(self.answer[0]))
        print(self.answer[0])

    def show_solution(self):
        """
        Mencetak solusi persoalan dengan format yang ditentukan.

        """
```

```
        opr = self.num_opr
        ans = self.num_ans

        # Mencetak operan-operan
        for i in range(len(opr)):
            space = len(str(ans)) - len(str(opr[i]))
            if len(opr) - 1 - i == 0:
                print(' '*space, opr[i], '+', sep='')
            else:
                print(' '*space, opr[i], ' '*14, sep='')

        # Menyetak garis batas serta hasil
        print('-' * len(str(ans)))
        print(ans)

    def _init_mapping(self):
        """
        Menginisialisasi pemetaan awal antara huruf dengan angka dari 1 sampai 9.

        """

        i = 0

        # Menentukan huruf mana yang akan ditempatkan di kiri
        if len(self.operand[0]) == len(self.answer[0]):
            all_words = self.operand + self.answer
        else:
            all_words = self.answer + self.operand

        # Mengasosiasikan tiap huruf dengan sebuah angka dari 1 sampai 9
        for word in all_words:
            for letter in word:
                if letter not in self.mapping.keys():
                    self.mapping[letter] = i
                    i += 1
```

```
    def _subtitute(self) -> [None, tuple]:
        """
        Menyubstitusikan tiap huruf dengan angka-angka yang bersesuaian pada
        pemetaan saat fungsi ini dipanggil.

        """

        # Mengganti huruf-huruf pada hasil dengan angka yang bersesuaian
        answer_num = ''
        for letter in self.answer[0]:
            answer_num += str(self.mapping[letter])

            # Mengembalikan None jika huruf pertama pada kata diawali angka 0
            if answer_num[0] == '0':
                return None
        self.num_ans = int(answer_num)

        # Mengganti huruf-huruf pada operan dengan angka yang bersesuaian
        for i in range(len(self.operand)):
            word = self.operand[i]
            operand_num = ''
            for letter in word:
                operand_num += str(self.mapping[letter])

                # Mengembalikan None jika huruf pertama pada kata diawali angka 0
                if operand_num[0] == '0':
                    return None
            self.num_opr.append(int(operand_num))

        return (self.num_opr, self.num_ans)

    def _incr(self):
        """
        Membuat komposisi pemetaan huruf-angka baru.

        """
```

```
        i = -1
        is_finish = False
        key = list(self.mapping.keys())

        while not is_finish and i >= -1 * len(key):
            curr_val = self.mapping[key[i]]
            next_val = curr_val + 1

            # Menjadikan 0 dan menambahkan 1 pada huruf di kiri jika hasil
            # inkremen adalah 10
            if next_val == 10:
                next_val = 0
                self.mapping[key[i]] = next_val
                i -= 1

            # Menambahkan 1 sampai menjadi angka unik pertama setelah angka
            # sebelumnya
            else:
                while self._check_is_contain(next_val, self.mapping) and next_val
< 9:
                    next_val += 1
                self.mapping[key[i]] = next_val
                is_finish = True

    def _evaluate(self) -> bool:
        """
        Melakukan evaluasi pada operan dan hasil yang sudah diganti dengan angka.
        Evaluasi terdiri dari pengecekan agar tidak ada angka ganda dan juga
        pengecekan hasil operasi penjumlahan operannya apakah sesuai dengan
        hasil.

        """

        is_unique = True
        guess_answer = sum(self.num_opr)
```

```
        # Mengecek apabila ada angka ganda
        for val in list(self.mapping.values()):
            if list(self.mapping.values()).count(val) > 1:
                is_unique = False
                break

        return (guess_answer == self.num_ans) and is_unique

    def _check_is_contain(self, val, mapping) -> bool:
        """
        Mengecek apakah suatu angka sudah terdapat pada pemetaan huruf-angka.

        """

        return val in mapping.values()

def resource_path(relative_path) -> Path:
    """
    Mendapatkan absolute path file yang dituju, berfungsi untuk file .py
    dan untuk file .exe hasil PyInstaller.

    """

    try:
        # PyInstaller membuat folder sementara dan menyimpan di _MEIPASS
        base_path = sys._MEIPASS
    except Exception:
        base_path = Path('.').absolute()

    return Path(base_path).joinpath(relative_path)


def read_file(file_name) -> tuple:
    """
    Membaca dan memeroses file yang bebrisi persoalan Cryptarithmetic.

    """
```

```
    operand = []
    answer = []

    rel_path = Path("test").joinpath(file_name)
    full_path = resource_path(rel_path)

    with open(full_path) as f:

        # Membaca tiap baris pada file persoalan
        lines = [line.strip() for line in f.read().splitlines()]

        # Membuang baris yang tidak dibutuhkan
        lines.pop(-2)

        # Memasukkan operan dan hasil pada list masing-masing
        answer.append(lines.pop(-1))
        lines[-1] = lines[-1].strip('+')
        operand.extend(lines)

        f.close()

    return (operand, answer)


if __name__ == "__main__":
    problem_list = [
        "problem01.txt", "problem02.txt", "problem03.txt", "problem04.txt",
        "problem05.txt", "problem06.txt", "problem07.txt", "problem08.txt",
        "problem09.txt", "problem10.txt"
    ]

    selected_problem = int(
        input("\nChoose cryptarithmetic problem to solve (1-10): ")
    ) - 1

    operand, answer = read_file(problem_list[selected_problem])
```

```
    solver = CryptarithmeticSolver(operand, answer)
    solver.show_problem()

    print('\nSOLUTION', '========', sep='\n')
    initial_time = time()
    print("Calculating...", end='\r')
    result = solver.calculate()
    sys.stdout.flush()
    if result:
        solver.show_solution()
    else:
        print("Sorry, no solutions found!")
    final_time = time()

    print("\nEXECUTION TIME: ", end='')
    print(final_time - initial_time, 's', end='')

    print("\nTOTAL TRIES: ", end='')
    print("{:,}".format(solver.tries_count))
```

Tangkapan Layar

Persoalan 1

```
C:\Users\raffi\OneDrive\Documents\Raffi\Kuliah\Semester 4\Strategi Algoritma\Tucil - Cryptarithmetic>py cryptarithmetic.py

Choose cryptarithmetic problem to solve (1-10): 1

PROBLEM
=======
NUMBER
NUMBER+
------
PUZZLE

SOLUTION
========
201689
201689+
------
403378

EXECUTION TIME: 81.2608277797699 s
TOTAL TRIES: 10,153,602
```

Persoalan 2

```
C:\Users\raffi\OneDrive\Documents\Raffi\Kuliah\Semester 4\Strategi Algoritma\Tucil - Cryptarithmetic>py cryptarithmetic.py

Choose cryptarithmetic problem to solve (1-10): 2

PROBLEM
=======
  TILES
PUZZLES+
-------
PICTURE

SOLUTION
========
  91542
3077542+
-------
3169084

EXECUTION TIME: 492.63696908950806 s
TOTAL TRIES: 64,826,679
```

Persoalan 3

```
C:\Users\raffi\OneDrive\Documents\Raffi\Kuliah\Semester 4\Strategi Algoritma\Tucil - Cryptarithmetic>py cryptarithmetic.py

Choose cryptarithmetic problem to solve (1-10): 3

PROBLEM
=======
 CLOCK
  TICK
  TOCK+
------
PLANET

SOLUTION
========
 90892
  6592
  6892+
------
104376

EXECUTION TIME: 114.7926127910614 s
TOTAL TRIES: 34,492,985
```

Persoalan 4

```
C:\Users\raffi\OneDrive\Documents\Raffi\Kuliah\Semester 4\Strategi Algoritma\Tucil - Cryptarithmetic>py cryptarithmetic.py

Choose cryptarithmetic problem to solve (1-10): 4

PROBLEM
=======
 COCA
 COLA+
-----
OASIS

SOLUTION
========
 8186
 8106+
-----
16292

EXECUTION TIME: 0.20592665672302246 s
TOTAL TRIES: 48,883
```

Persoalan 5

```
C:\Users\raffi\OneDrive\Documents\Raffi\Kuliah\Semester 4\Strategi Algoritma\Tucil - Cryptarithmetic>py cryptarithmetic.py

Choose cryptarithmetic problem to solve (1-10): 5

PROBLEM
=======
 HERE
  SHE+
-----
COMES

SOLUTION
========
 9454
  894+
-----
10348

EXECUTION TIME: 0.5545563697814941 s
TOTAL TRIES: 230,933
```

Persoalan 6

```
C:\Users\raffi\OneDrive\Documents\Raffi\Kuliah\Semester 4\Strategi Algoritma\Tucil - Cryptarithmetic>py cryptarithmetic.py

Choose cryptarithmetic problem to solve (1-10): 6

PROBLEM
=======
 DOUBLE
 DOUBLE
   TOIL+
-------
TROUBLE

SOLUTION
========
 798064
 798064
   1936+
-------
1598064

EXECUTION TIME: 43.989623069763184 s
TOTAL TRIES: 8,515,976
```

Persoalan 7

```
C:\Users\raffi\OneDrive\Documents\Raffi\Kuliah\Semester 4\Strategi Algoritma\Tucil - Cryptarithmetic>py cryptarithmetic.py

Choose cryptarithmetic problem to solve (1-10): 7

PROBLEM
=======
  NO
 GUN
  NO+
----
HUNT

SOLUTION
========
  87
 908
  87+
----
1082

EXECUTION TIME: 0.11124253273010254 s
TOTAL TRIES: 38,436
```

Persoalan 8

```
C:\Users\raffi\OneDrive\Documents\Raffi\Kuliah\Semester 4\Strategi Algoritma\Tucil - Cryptarithmetic>py cryptarithmetic.py

Choose cryptarithmetic problem to solve (1-10): 8

PROBLEM
=======
 THREE
 THREE
   TWO
   TWO
   ONE+
------
ELEVEN

SOLUTION
========
 84611
 84611
   803
   803
   391+
------
171219

EXECUTION TIME: 52.32016563415527 s
TOTAL TRIES: 8,875,526
```

Persoalan 9

```
C:\Users\raffi\OneDrive\Documents\Raffi\Kuliah\Semester 4\Strategi Algoritma\Tucil - Cryptarithmetic>py cryptarithmetic.py

Choose cryptarithmetic problem to solve (1-10): 9

PROBLEM
=======
 CROSS
 ROADS+
------
DANGER

SOLUTION
========
 96233
 62513+
------
158746

EXECUTION TIME: 36.57952165603638 s
TOTAL TRIES: 8,467,590
```

Persoalan 10

```
C:\Users\raffi\OneDrive\Documents\Raffi\Kuliah\Semester 4\Strategi Algoritma\Tucil - Cryptarithmetic>py cryptarithmetic.py

Choose cryptarithmetic problem to solve (1-10): 10

PROBLEM
=======
 MEMO
 FROM+
-----
HOMER

SOLUTION
========
 8485
 7358+
-----
15843

EXECUTION TIME: 0.189178466796875 s
TOTAL TRIES: 47,762
```

Tautan Kode Program

Untuk membaca kode sumber program yang lebih lengkap dan lebih mudah dibaca silakan untuk mengaksesnya di *repository* GitHub yang dapat dituju menggunakan alamat:

https://github.com/raffizulvian/cryptarithmetic

Alamat di atas baru dapat diakses mulai pukul 13.00 WIB pada tanggal 27 Januari 2021 untuk memastikan kode sumber program tetap rahasia sampai waktu tersebut.

Tabel Pengecekan Spesifikasi

Poin	Ya	Tidak
Program berhasil dikompilasi tanpa kesalahan (*no syntax error*)	✓	
Program berhasil *running*	✓	
Program dapat membaca file masukan dan menuliskan luaran	✓	
Solusi *cryptarithmetic* hanya benar untuk persoalan *cryptarithmetic* dengan dua buah *operand*		✓
Solusi *cryptarithmetic* benar untuk persoalan *cryptarithmetic* untuk lebih dari dua buah *operand*	✓	