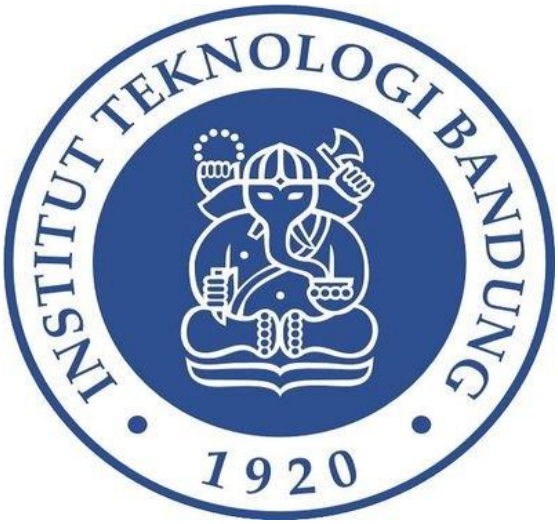
Tugas Kecil 1 IF2211 Strategi Algoritma
Penyelesaian *Cryptarithmetic* dengan Algoritma *Brute Force*



Disusun Oleh:

Richard Rivaldo
13519185
Kelas 04

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika

INSTITUT TEKNOLOGI BANDUNG
TAHUN AJARAN 2020/2021

1. Domain Permasalahan

Cryptarithmetic adalah sebuah permainan yang menyelesaikan sebuah permasalahan operasi matematika tetapi angka diganti dengan huruf. Permainan ini diselesaikan dengan mencari angka yang merepresentasikan tiap huruf yang ada di dalam operan. Adapun satu angka hanya bisa merepresentasikan satu huruf saja dan hasil akhir dari tiap substitusi angka tersebut tidak boleh diawali oleh angka 0.

Selain itu, operasi yang digunakan adalah operasi penjumlahan saja. Jumlah minimal dari operan yang digunakan dalam setiap permainan adalah minimal 2 buah. Jumlah huruf yang ada di dalam operan adalah maksimal 10 buah. Permainan ini dimulai dengan menerima input file berisi operasi keseluruhan dan diselesaikan dengan menggunakan Algoritma *Brute Force* dan menampilkan semua solusi permainan, lamanya program berjalan (tidak termasuk pembacaan file), dan jumlah percobaan yang dilakukan oleh program.

Contoh:

```
  S E N D
+ M O R E
---------
M O N E Y
```

Solusinya adalah:

```
  9 5 6 7
+ 1 0 8 5
---------
1 0 6 5 2
```

Jadi, S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2

Contoh-contoh *cryptrithmetic* dengan solusinya:

```
  J U N E        7 9 2 4
+ J U L Y      + 7 9 0 6
---------      ---------
A P R I L      1 5 8 3 0
```

```
    F O R T Y          2 9 7 8 6
        T E N              8 5 0
+       T E N      +       8 5 0
-------------      ---------------
S I X T Y          3 1 4 8 6
```

2. *Source Code* Program

Program dibuat secara modular dalam satu file dengan menggunakan bahasa pemrograman Python. Berikut merupakan *screenshot* dari *source code* program *Cryptarithmetic.py*.

```python
# Library
import time

# Function to open and process file contents into a list of strings
def read_file(filename):
    file = open(filename, "r")

    # Global variable to count program run time
    # Excluding the amount of time needed to read the file
    global startTime
    startTime = time.time()

    # Empty list to contain processed letters
    letters = []
    for line in file:
        # Remove excess spaces and newlines
        line = line.replace("\n", "")
        line = line.replace(" ", "")

        # Remove non-letter chars
        line = "".join([chars for chars in line if chars.isalpha()])

        # Append processed file lines into the list
        if line != "":
            letters.append(line)
    return(letters)

# Function to find and map all distinct alphabets in the equation
def map_distinct(letters):
    # Initiate empty lists to contain processed results
    words = [] # List of distinct chars
    mapped = [] # List of list of every indexes of each char

    # Find distinct chars and append it to words
    for word in letters:
        for i in range(len(word)):
            if(word[i] not in words):
                words.append(word[i])

    # Join distinct chars into a string
    distinct = "".join(words)

    # Map every char in each words into indexes based on distinct string
    for letter in letters:
        mapped.append([distinct.find(char) for char in letter])
    return(mapped)
```

Potongan *Source Code* I

```python
# Function to find spaces of possible numbers
def permute(targetList):
    # Initiate empty list to contain permutation results
    resultList = []

    # Recursive basis: empty target
    if(len(targetList) == 0):
        return(resultList)
    # Recursive basis: target length = 1
    elif(len(targetList) == 1):
        return([targetList])
    # Recurrents
    else:
        for idx in range(len(targetList)):
            for comp in permute(targetList[:idx] + targetList[(idx + 1):]):
                resultList.append([targetList[idx]] + comp)
        return(resultList)

# Function to match every possibilities into the words
def matching_to_numbers(target, spaces):
    # Join every chars into a single string and change its type into integer
    match = "".join(str(spaces[chars]) for chars in target)
    return(int(match))

# Brute Force approach to try finding every possible solutions of the cryptarithmetic
def search(list_of_idx):
    # Initiate empty list to contain temporary and final solutions of the equation
    solution_list = []
    final = []

    # Contains the numbers of test done to find the solutions
    count = 0

    # Find possible solutions
    for possibles in permute(list('1234567890')):
        # Sum all operands after matched and no leading zeros
        totalOperand = 0
        for idxs in list_of_idx[:-1]:
            # Handle leading zeros
            if(len(str(matching_to_numbers(idxs, possibles))) == len(idxs)):
                totalOperand += matching_to_numbers(idxs, possibles)

        # The result of the equation
        if(len(str(matching_to_numbers(list_of_idx[-1], possibles))) == len(list_of_idx[-1])):
            results = matching_to_numbers(list_of_idx[-1], possibles)
        else:
            results = -1

        # Check if the results is true
        if(totalOperand == results):
            # Append the solutions of each matching numbers
            for idx in (list_of_idx):
                match = matching_to_numbers(idx, possibles)
                solution_list.append(match)

        # Partition the list into a final solution list
        eqLength = len(list_of_idx)
        for i in range(0, len(solution_list), len(list_of_idx)):
            temp = solution_list[i: (i + eqLength)]

            # Handle non-unique solution
            if(temp not in final):
                final.append(temp)

        # Increment the tests count
        count += 1
    return(final, count)
```

Potongan *Source Code* II

```python
# Cryptarithmetic procedure to run all processes needed
def cryptarithmetic(filename):
    # Read the file and process it
    letters = read_file(filename)

    # Map each distinct chars
    list_of_idx = map_distinct(letters)

    # Searching for solutions
    sols, count = search(list_of_idx)

    # Width for right alignment
    width = len(letters[-1])

    # Output original file
    print("Cryptarithmetic:")
    for i in range(len(letters[:-2])):
        print(letters[i].rjust(width))
    print(letters[-2].rjust(width), "+")
    print("-" * (width + 2))
    print(letters[-1])
    print()

    # Output solutions
    if(len(sols) == 0):
        print("No solutions found!")
    else:
        solsOf = 1
        for solution in sols:
            print("Solution " + str(solsOf) + ":")
            for j in range(len(solution[:-2])):
                print(str(solution[j]).rjust(width))
            print(str(solution[-2]).rjust(width), "+")
            print("-" * (width + 2))
            print(solution[-1])
            solsOf += 1
            print()
        print()

    # Run Time
    totalTime = time.time() - startTime
    print("Total Duration : ", totalTime, "seconds")

    # Total tests
    print("Total Testcases: ", count, "testcases")

cryptarithmetic("filename.txt")
```

Potongan *Source Code* III

3. Deskripsi Pendekatan Algoritma dan Program *Cryptarithmetic*

Seperti yang telah dijelaskan sebelumnya, jenis algoritma yang digunakan di dalam program adalah Algoritma *Brute Force*. Algoritma *Brute Force* adalah algoritma yang memiliki pendekatan yang bersifat lempang atau *straightforward*, atau dalam kata lain berusaha memecahkan persoalan melalui definisi atau konsep yang melingkupi persoalan, atau dengan melalui *trial and error* seperti yang ada di dalam algoritma ini. Dalam program *cryptarithmetic* di atas, terdapat beberapa fungsi yang menjadi jembatan dari proses penerimaan input hingga penampilan output.

Fungsi pertama, yaitu *read_file* menerima argumen berupa nama file yang ingin diproses, atau dalam kata lain adalah file yang berisi *puzzle cryptarithmetic*. Ketika file telah berhasil dibaca oleh program, perhitungan waktu eksekusi program akan dimulai sehingga pembacaan file tidak termasuk ke dalam waktu total eksekusi program. Setelah itu, program akan melakukan pemrosesan terhadap setiap baris file tersebut dengan menghilangkan karakter-karakter yang bukan alfabet. Sebagai hasil akhir, program mengembalikan sebuah *list* berisi semua kata yang ada di dalam file input.

Fungsi kedua, yaitu *map_distinct*, menerima *list* yang dihasilkan dari fungsi *read_file* sebagai argumennya. Dari setiap kata yang ada di dalam *list* tersebut, fungsi ini akan mencari huruf-huruf yang unik dan menggabungkan huruf tersebut menjadi sebuah *string*. Untuk setiap kata tersebut pula, Fungsi ini akan melakukan pemetaan setiap alfabet yang ada di dalamnya berdasarkan indeks keberadaan huruf tersebut di *string* tadi. Nantinya, program akan mengembalikan *list* yang juga berisi daftar indeks untuk setiap huruf dalam semua kata tersebut.

Dua fungsi lain, yaitu *permute* dan *matching_to_numbers* merupakan fungsi bantuan dalam mencari solusi dari permainan. Fungsi *permute* adalah fungsi yang digunakan untuk mencari semua susunan angka yang mungkin dari susunan angka 0-9, atau secara umum dari *list* yang diterima. Fungsi ini bekerja dengan mengiterasi dan menggabungkan komponen *list* untuk menyusun kemungkinan yang ada dan kemudian memasukkannya ke sebuah *list* baru untuk menampung semua kemungkinan yang akan dikembalikan nantinya.

Kemudian, fungsi *matching_to_numbers* akan menggunakan pemetaan yang telah dihasilkan sebelumnya untuk melakukan substitusi yang benar sehingga tiap huruf memiliki subsitusi angka yang sama dan memudahkan perhitungan. Fungsi terakhir *search* adalah fungsi yang menerima *list* pemetaan tersebut dan kemudian mencari solusi dari permainan *cryptarithmetic* tersebut. Pendekatan *Brute Force* yang digunakan adalah untuk setiap kemungkinan yang dihasilkan oleh *permute*, akan digunakan fungsi *matching_to_numbers* untuk melakukan substitusi operan dan hasil dari operasi *cryptarithmetic*.

Sebelum melakukan substitusi tersebut, trik program untuk mengatasi kemungkinan *leading zero* adalah dengan mengecek panjang kata dan jumlah digit angka (panjang sebagai *string*) yang ada. Tentunya, jika panjang kedua *string* berbeda, pasti ada sebuah *leading zero* yang nantinya membuat hasil operasi menjadi tidak valid. Hal yang sama

dilakukan juga terhadap hasil dari operasi permainan, sehingga baik operan maupun hasil operasi tidak diawali oleh 0.

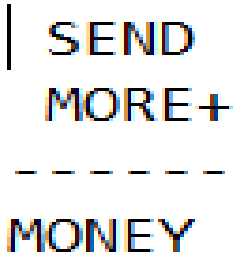
Setelah dilakukan substitusi yang valid, maka fungsi kemudian akan mengevaluasi jumlah dari tiap operan sama atau tidak dengan besar hasil operasi. Jika jumlah yang dihasilkan sama, fungsi tersebut akan memasukkan tiap angka hasil substitusi setiap komponen operasi ke dalam *list* untuk menjaga setiap solusi yang mungkin. Fungsi juga akan melakukan partisi *list* solusi tersebut sesuai dengan jumlah komponen yang ada di dalam operasi, sehingga nantinya akan dihasilkan sebuah *list* yang terdiri dari *list* solusi-solusi operasi. Selain mengembalikan *list* tersebut, fungsi juga akan mengembalikan jumlah percobaan yang dilakukan oleh program ketika melakukan pencarian solusi.

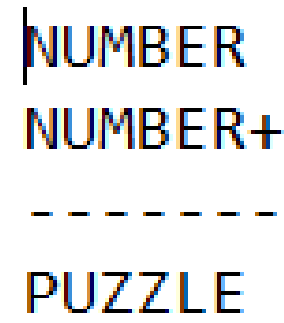
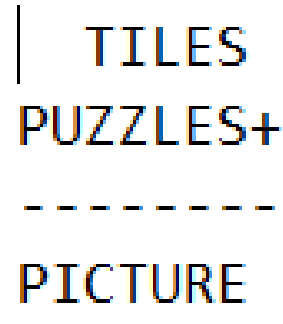
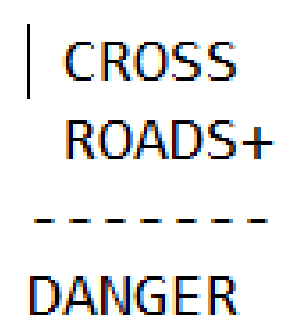
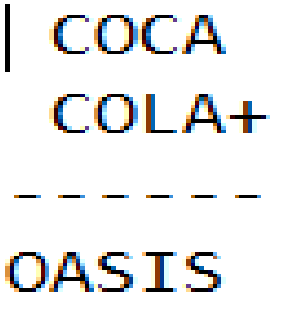
Prosedur *cryptarithmetic* merupakan prosedur yang memanggil dan menggabungkan semua fungsi yang telah disebutkan. Selain itu, prosedur ini juga akan melakukan pemrosesan output dan menampilkan semua solusi yang tersedia dengan melakukan iterasi terhadap isi dari *list* solusi. Prosedur juga melakukan *print* rata kanan setiap komponen operasi dan menyesuaikan dengan panjang kata hasil operasi yang sudah pasti merupakan kata paling panjang dalam permainan.

Selain itu, program juga mengeluarkan output berupa jumlah percobaan dalam mencari solusi serta waktu eksekusi yang dibutuhkan oleh program. Program dapat dijalankan secara keseluruhan dengan memanggil prosedur *cryptarithmetic* dan memberikan argumen berupa nama file yang mengandung operasi *cryptarithmetic* yang ingin dipecahkan. Jika ternyata operasi yang diminta tidak masuk akal, misalnya ada komponen operan yang memiliki jumlah digit yang lebih banyak dibandingkan hasil operasi, program akan memberikan pesan terkait dengan tidak adanya solusi permainan yang mungkin.

4. Hasil Eksekusi *Testcases*

Eksekusi *testcases* berikut dilakukan di dalam *environment* IDE, sehingga hanya perlu dilakukan *run* dengan mengganti file input sebagai argumen prosedur *cryptarithmetic*.

Deskripsi	Input File	Output *Cryptarithmetic*
2 Operan		Cryptarithmetic: SEND MORE + ------- MONEY Solution 1: 9567 1085 + ------- 10652 Total Duration : 95.32806754112244 seconds Total Testcases: 3628800 testcases

		Cryptarithmetic: NUMBER NUMBER + -------- PUZZLE Solution 1: 201689 201689 + -------- 403378 Total Duration : 110.48347854614258 seconds Total Testcases: 3628800 testcases
		Cryptarithmetic: TILES PUZZLES + --------- PICTURE Solution 1: 91542 3077542 + --------- 3169084 Total Duration : 107.733971118927 seconds Total Testcases: 3628800 testcases
		Cryptarithmetic: CROSS ROADS + -------- DANGER Solution 1: 96233 62513 + -------- 158746 Total Duration : 102.60493612289429 seconds Total Testcases: 3628800 testcases
		Cryptarithmetic: COCA COLA + ------- OASIS Solution 1: 8186 8106 + ------- 16292 Total Duration : 105.96040463447571 seconds Total Testcases: 3628800 testcases

<table>
<tr>
<td></td>
<td>
<pre>
 MEMO
  FROM+

HOMER
</pre>
</td>
<td>
<pre>
Cryptarithmetic:
 MEMO
 FROM +

HOMER

Solution 1:
 8485
 7358 +

15843

Total Duration : 113.59180402755737 seconds
Total Testcases: 3628800 testcases
</pre>
</td>
</tr>
<tr>
<td rowspan="3">>2 operan</td>
<td>
<pre>
 DOUBLE
  DOUBLE
    TOIL+

TROUBLE
</pre>
</td>
<td>
<pre>
Cryptarithmetic:
 DOUBLE
 DOUBLE
   TOIL +

TROUBLE

Solution 1:
 798064
 798064
   1936 +

1598064

Total Duration : 133.3045139312744 seconds
Total Testcases: 3628800 testcases
</pre>
</td>
</tr>
<tr>
<td>
<pre>
 SEVEN
    SIX
  SEVEN+

TWENTY
</pre>
</td>
<td>
<pre>
Cryptarithmetic:
 SEVEN
   SIX
 SEVEN +

TWENTY

Solution 1:
 68782
   650
 68782 +

138214

Total Duration : 91.14366269111633 seconds
Total Testcases: 3628800 testcases
</pre>
</td>
</tr>
<tr>
<td>
<pre>
 CLOCK
   TICK
   TOCK+

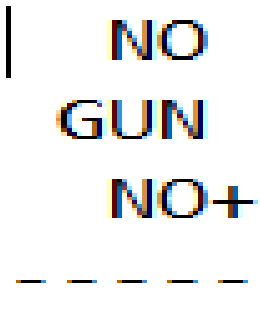
PLANET
</pre>
</td>
<td>
<pre>
Cryptarithmetic:
 CLOCK
  TICK
  TOCK +

PLANET

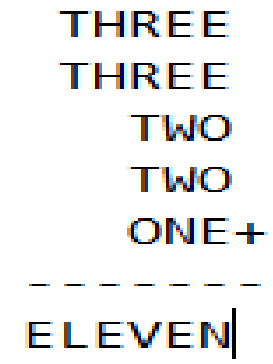
Solution 1:
 90892
  6592
  6892 +

104376

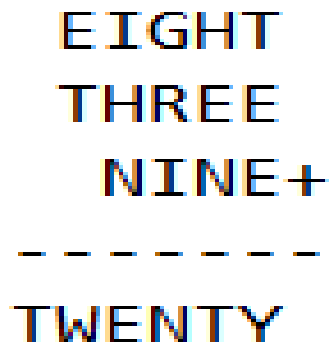
Total Duration : 75.72860646247864 seconds
Total Testcases: 3628800 testcases
</pre>
</td>
</tr>
</table>

<table>
<tr><td rowspan="2"></td><td> 
</td><td>

```
Cryptarithmetic:
  NO
 GUN
  NO +
------
HUNT

Solution 1:
  87
 908
  87 +
------
1082


Total Duration :  70.45764303207397 seconds
Total Testcases:  3628800 testcases
```

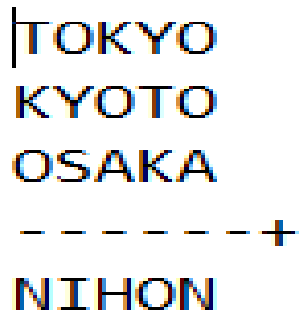
</td></tr>
<tr><td> 
</td><td>

```
Cryptarithmetic:
 THREE
 THREE
   TWO
   TWO
   ONE +
--------
ELEVEN

Solution 1:
 84611
 84611
   803
   803
   391 +
--------
171219


Total Duration :  101.19150066375732 seconds
Total Testcases:  3628800 testcases
```

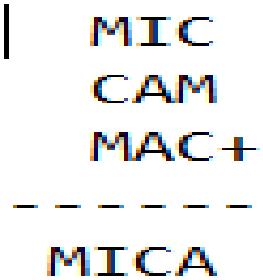
</td></tr>
<tr><td>Beberapa Solusi</td><td> 
</td><td>

```
Cryptarithmetic:
 EIGHT
 THREE
  NINE +
--------
TWENTY

Solution 1:
 85291
 19488
  3538 +
--------
108317

Solution 2:
 85491
 19288
  3538 +
--------
108317


Total Duration :  124.24072170257568 seconds
Total Testcases:  3628800 testcases
```

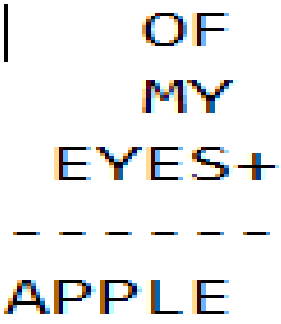
</td></tr>
</table>

<table>
<tr>
<td rowspan="3">Tanpa Solusi</td>
<td> 
</td>
<td>

```
Cryptarithmetic:
TOKYO
KYOTO
OSAKA +
-------
NIHON

Solution 1:
14204
20414
43929 +
-------
78547

Solution 2:
51241
24151
13626 +
-------
89018


Total Duration :  125.40285515785217 seconds
Total Testcases:  3628800 testcases
```

</td>
</tr>
<tr>
<td> 
</td>
<td>

```
Cryptarithmetic:
 MIC
 CAM
 MAC +
------
MICA

No solutions found!
Total Duration :  115.99383521080017 seconds
Total Testcases:  3628800 testcases
```

</td>
</tr>
<tr>
<td> 
</td>
<td>

```
Cryptarithmetic:
   OF
   MY
 EYES +
-------
APPLE

No solutions found!
Total Duration :  69.60939359664917 seconds
Total Testcases:  3628800 testcases
```

</td>
</tr>
</table>

5. Analisis Hasil Eksekusi

Berdasarkan hasil eksekusi yang telah dilakukan, program berhasil memecahkan beberapa kategori permasalahan. Program berhasil memecahkan permasalahan yang melibatkan dua buah operan dan juga *puzzle* yang memiliki lebih dari dua buah operan. Selain itu, program berhasil mencari semua solusi yang bisa dihasilkan dari suatu permasalahan. Dari *testcases* yang dilakukan, ada juga *puzzle* yang tidak memiliki penyelesaian dan program berhasil mengirimkan pesan yang sesuai.

Menurut analisis waktu eksekusi program, maka dapat diketahui bahwa dari 15 *testcases* tersebut diperlukan waktu rata-rata sekitar 90 sampai dengan 100 detik. Dari *testcases* tersebut, dapat diketahui bahwa lamanya waktu pemrosesan lebih bergantung kepada kemiripan susunan huruf-huruf dan panjang kata yang ada di dalam operasi, bukan jumlah dari operan dalam *puzzle cryptarithmetic.*

Hal ini dapat kita lihat misalnya pada *puzzle* `NO GUN NO HUNT` dan `CLOCK TICK TOCK PLANET`. Pada *puzzle* pertama, kata yang ada di dalam operasi sangat pendek (maksimal 4 karakter) disertai dengan adanya duplikat kata `NO`. Pada *puzzle* kedua, Meskipun kata yang ada di dalam operasi panjang, namun huruf yang sama di dalam kata-kata tersebut memiliki susunan yang mirip. Hal inilah yang memudahkan perhitungan dan substitusi sehingga program berjalan lebih cepat sehingga waktu eksekusi menjadi lebih cepat.

Adapun untuk semua *testcases* tersebut, jumlah percobaan yang dilakukan untuk mencari semua solusi persoalan adalah sama, yaitu 3.628.800 buah percobaan. Artinya, untuk setiap input file yang dimasukkan ke dalam program, program akan tetap mencoba semua kemungkinan yang mungkin dari hasil permutasi yang dilakukan oleh fungsi *permute*. Jumlah yang dihasilkan ini sesuai dengan jumlah yang dihasilkan oleh permutasi tersebut jika dihitung secara matematis, yaitu 10! percobaan.

6. Kode Program

Berikut merupakan alamat yang dapat diakses untuk mengunduh kode program *Cryptarithmetic* beserta file *testcases* yang digunakan dalam laporan.

Google Drive: *https://drive.google.com/drive/folders/17CFeRdQiCWMSlmSJMzQQbwM-I2qqBPBO?usp=sharing*.

Perlu diperhatikan bahwa untuk langsung menjalankan program, file input *testcases* perlu dikeluarkan terlebih dahulu dan harus berada di direktori yang sama dengan kode program, atau dengan menuliskan secara lengkap direktori file di dalam program.

7. Simpulan dan Refleksi

Berdasarkan penjelasan yang telah diberikan, program yang dibuat dengan bahasa Python ini berhasil memecahkan permasalahan *Cryptarithmetic* yang diberikan dalam input file, baik yang memiliki 2 operan, lebih dari 2 operan, lebih dari satu solusi, ataupun yang tidak memiliki solusi sama sekali. Selain itu, program memiliki waktu rata-rata 90 sampai dengan 100 detik untuk menyelesaikan permasalahan tersebut serta melakukan percobaan sebanyak 3.628.800 kali.

Tentunya, Algoritma *Brute Force* yang dibuat masih mungkin untuk lebih disederhanakan dan direfaktorisasi sehingga bisa memiliki tingkat efisiensi dan efektivitas yang lebih baik. Lebih dari itu, juga masih bisa diterapkan konsep heuristik di dalam algoritma tersebut sehingga dapat diperoleh ruang kemungkinan yang menjadi lebih kecil dan program bisa berjalan dengan lebih cepat.

8. Sumber

Basic Mathematics.com. *Cryptarithms* Diakses pada 21 Januari 2021 pukul 18.21 WIB melalui *https://www.basic-mathematics.com/cryptarithms.html*.

Cryptarithms.com. Diakses pada 22 Januari 2021 pukul 17.35 WIB melalui *http://www.cryptarithms.com/default.asp?pg=1*.

DCode. *Cryptarithm Solver*. Diakses pada 22 Januari 2021 pukul 18.32 WIB melalui *https://www.dcode.fr/cryptarithm-solver*.

GeeksforGeeks. Diakses pada 21 Januari 2021 pukul 21.42 WIB melalui *https://www.geeksforgeeks.org/*.

Stack Overflow. Diakses pada 21 Januari 2021 pukul 19.42 WIB melalui *https://stackoverflow.com/*.

Teknomo, Kardi. *Cryptarithm Problems.* Diakses pada 21 Januari 2021 pukul 20.32 WIB melalui *https://people.revoledu.com/kardi/tutorial/CryptArithmetic/index.html*.

9. Lampiran

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