1. Algoritma *Brute Force*
   a) Menerima input berupa *list of strings*, lalu menggabungkan semua elemen *string* dalam list menjadi satu *string* utuh dan disimpan dalam sebuah variabel.
   b) Melakukan operasi *set*, sehingga setiap huruf yang ada pada *string* sebelumnya hanya muncul satu kali, lalu menyimpan set dari string tersebut dan panjang set ke dalam suatu variabel.
   c) Melakukan pengecekan terhadap jumlah huruf yang tersimpan dalam variabel untuk menentukan hal yang akan dilakukan berikutnya. Program hanya akan lanjut jika jumlah huruf lebih kecil atau sama dengan 10.
   d) Membangkitkan permutasi dari *sequence of number* (0 – 9) sejumlah huruf yang ada. Sebagai contoh jumlah huruf ada 5, maka akan dibangkitkan 5 angka ( cth : 12345 ) dengan syarat semua angka harus berbeda ( cth yang tidak diperbolehkan : 11233, karena 1 dan 3 mempunyai kembaran). Banyaknya angka yang akan dibangkitkan mengikuti rumus permutas ${}^{n}P_{x}$ ( dengan 'n' adalah *sequence of number* dan 'x' adalah jumlah huruf berbeda yang ada).
   e) Membentuk sebuah *dictionary* ( <key : value> ) dengan key adalah huruf yang tersimpan pada *set of string* dan *value* adalah permutasi angka yang dibangkitkan pada langkah d. ( metode *zip* pada python).
      Sebagai contoh:
      listOfString = ('A', 'T', 'M', 'P', 'N', 'U', 'I', 'S', 'Y')
      value = (0, 1, 2, 3, 4, 5, 6, 7, 8)

      word_dict = {'A': 0, 'T': 1, 'M': 2, 'P': 3, 'N': 4, 'U': 5, 'I': 6, 'S': 7, 'Y': 8}

   f) Tahap pengecekan:
      Telah didefinisikan sebelumnya fungsi untuk menghitung nilai dari string pada *list of string* yang didapat dari input menggunakan *dictionary* yang dibentuk pada tahap e.

      Inisialisasi *placeholder* untuk menyimpan nilai string ( tipe integer ) berupa list, dan variable lain bertipe integer untuk melakukan *checking* (<operand_result>) .

      Iterasi dilakukan dari elemen pertama *list of string* sampai elemen ke n-1, dimana setiap elemen yang dilewati akan dihitung dan disimpan penjumlahan nilainya ke variabel (<*operand_result*>). Selain itu setiap nilai string akan disimpan ke dalam *placeholder* yang telah diinisialisasi sebelumnya.
      (Tahap iterasi sampai ke elemen n-1 ini cukup penting, karena membuat program fleksibel untuk menerima sebanyak-banyaknya operand asalkan jumlah huruf berbeda tidak lebih dari 10).

      Tahap pengecekan terakhir adalah membandingkan (<*operand_result*>) dengan nilai string dari elemen akhir *list of string.*

      Jika pengecekan sama, maka nilai string elemen akhir juga akan dimasukkan dalam *placeholder*. Jika hasil berbeda program akan mengulang dari tahap d.

      Program akan berulang hingga ${}^{n}P_{x}$ ( dengan 'n' adalah *sequence of number* dan 'x' adalah jumlah huruf berbeda yang ada) pengulangan jika tidak ditemukan hasil hingga akhir.

FUNGSI UNTUK MENGHITUNG *VALUE* DARI STRING BERDASARKAN *DICTIONARY*

```python
def word_val(word, dict):
    result = 0
    k = 1
    reversed_word = word[::-1]
    for x in range(len(reversed_word)):
        result += dict[reversed_word[x]] * k
        k *= 10
    return result
```

FUNGSI CRYPTARITHM SOLVER

```python
def cryptarithm_solver(list_of_strings):
    # Menggunakan metode join untuk menggabungkan semua kata dalam list_of_strings
    joined_list_of_strings = ''.join(list_of_strings)
    # Membuat menjadi set, sehingga assignment value ke key dalam dictionary tidak ada yang double
    set_list_of_strings = set(joined_list_of_strings)
    finalList = tuple(set_list_of_strings)
    # Membuat kamus dengan key dari set(list_kata) dan value berupa angka dari permutasi dengan metode 'zip'
    digits = list(range(10)) # Angka yang bisa digunakan adalah 0 - 9
    sub_try = 0 # Menghitung jumlah percobaan
    n = len(finalList)
    if(n > 10):
        return " "
    else:
        pass
    for possibility in permutations(digits,n):
        word_dict = dict(zip(finalList,possibility)) # Membentuk <key:value> dengan key adalah huruf dan value adalah angka
        # RESTRIKSI 0 untuk awal kata operand dan result
        count0 = 0
        for i in range(len(list_of_strings)):
            if(word_dict[list_of_strings[i][0]] == 0) :
                count0 += 1
        if count0 != 0:
            continue
        # LOLOS RESTRIKSI 0
        else:
            operand_result = 0
            # Untuk save hasil dari perhitungan
            solutions = []
            # Iterate sampai 1 elemen sebelum terakhir, karena dipakai sebagai result
            for i in range(len(list_of_strings)-1):
                operand_result += word_val(list_of_strings[i],word_dict)
                solutions.append(word_val(list_of_strings[i],word_dict))

            if (operand_result == word_val(list_of_strings[-1], word_dict)):
                solutions.append(word_val(list_of_strings[-1], word_dict))
                return solutions, sub_try
        sub_try += 1
    return "Tidak ditemukan solusi"
```

KUMPULAN SCREENSHOT

```
NUMBER
NUMBER+
--------
PUZZLE

Solusi dari persamaan adalah :
201689
201689+
-----
403378
Dibutuhkan 21.852446399999998 detik dan 2106529 percobaan substitusi
```

```
Pastikan mengisi persamaan di problems.txt, dan meletakkan file di tempat yang sama dengan source code Python
Perhitungan akan dimulai, ketik 'Y' jika siap 'X' untuk keluar: Y
TILES
PUZZLES+
-------
PICTURE

Solusi dari persamaan adalah :
91542
3077542+
-----
3169084
Dibutuhkan 23.5629837 detik dan 1760961 percobaan substitusi
```

```
Perhitungan akan dimulai, ketik 'Y' jika siap 'X' untuk keluar: y
CLOCK
TICK
TOCK+
-------
PLANET

Solusi dari persamaan adalah :
90892
6592
6892+
-----
104376
Dibutuhkan 14.3760412 detik dan 1103779 percobaan substitusi
```

```
Pastikan mengisi persamaan di problems.txt, dan meletakkan file di tempat yang sama dengan source code Python
Perhitungan akan dimulai, ketik 'Y' jika siap 'X' untuk keluar: Y
COCA
COLA+
------
OASIS

Solusi dari persamaan adalah :
8186
8106+
-----
16292
Dibutuhkan 1.1497688000000001 detik dan 118260 percobaan substitusi
```

```
Pastikan mengisi persamaan di problems.txt, dan meletakkan file di tempat yang sama dengan source code Python
Perhitungan akan dimulai, ketik 'Y' jika siap 'X' untuk keluar: y
HERE
SHE+
-----
COMES

Solusi dari persamaan adalah :
9454
894+
-----
10348
Dibutuhkan 3.3946378 detik dan 378037 percobaan substitusi
```

```
Perhitungan akan dimulai, ketik 'Y' jika siap 'X' untuk keluar: y
DOUBLE
DOUBLE
TOIL +
-------
TROUBLE

Solusi dari persamaan adalah :
798064
798064
1936 +
-----
1598064
Dibutuhkan 27.9424034999999998 detik dan 2076611 percobaan substitusi
```

```
NO
GUN
NO+
-------
HUNT

Solusi dari persamaan adalah :
87
908
87 +
-----
1082
Dibutuhkan 0.15433000000000074 detik dan 11425 percobaan substitusi
```

```
Pastikan mengisi persamaan di problems.txt, dan meletakkan file di tempat yang sama dengan source code Python
Perhitungan akan dimulai, ketik 'Y' jika siap 'X' untuk keluar: y
THREE
THREE
TWO
TWO
ONE+
-------
ELEVEN

Solusi dari persamaan adalah :
84611
84611
803
803
391 +
-----
171219
Dibutuhkan 29.5060078 detik dan 1976833 percobaan substitusi
```

```
Pastikan mengisi persamaan di problems.txt, dan meletakkan file di tempat yang sama dengan source code Python
Perhitungan akan dimulai, ketik 'Y' jika siap 'X' untuk keluar: y
CROSS
ROADS+
------
DANGER

Solusi dari persamaan adalah :
96233
62513 +
-----
158746
Dibutuhkan 3.6070567999999996 detik dan 236966 percobaan substitusi
```

```
Pastikan mengisi persamaan di problems.txt, dan meletakkan file di tempat yang sama dengan source code Python
Perhitungan akan dimulai, ketik 'Y' jika siap 'X' untuk keluar: Y
MEMO
FROM+
------
HOMER

Solusi dari persamaan adalah :
8485
7358 +
-----
15843
Dibutuhkan 0.6865797999999999 detik dan 77377 percobaan substitusi
```