Implementasi Algoritma Backtracking Dalam Penyelesaian Cryptarithmetic Puzzle

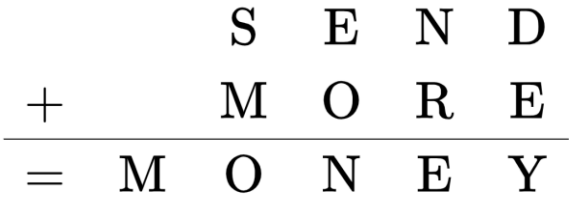
Mohamad Falah Sutawindaya / 13518102
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518102@std.stei.itb.ac.id

***Abstract*—Cryptarithmetic adalah tipe game matematika yang terdiri dari persamaan matematis di antara angka yang belum di ketahui yang digitnya direpresentasikan dengan huruf, Tujuan akhirnya adalah mengetahui nilai setiap hurufnya,. Persamaan matematis pada game melibatkan operasi dasar aritmatis seperti penjumlahan, perkalian, atau pembagian. Makalah ini adalah percobaan untuk membuat penyelesaiannya dengan algoritma backtracking.**

Keywords—Cryptarithmetic; backtracking; algorithm; solver

I. PENDAHULUAN

Cryptarithmetic yang juga di kenal dengan alphametics, cryptarithm, penjumlahan kata, ataupun verbal arithmetic. Cryptarithmetic puzzle adalah permainan yang cukup tua, dan penemunya tidak di ketahui, nama *cryptarithm* di sebut pertama kali oleh Simon Vatriquant pada Mei 1931 pada majalah belgia bertemakan *recreational mathematics*, dan di translasikan sebagai *cryparithmethic* oleh Maurice Kraitchik pada tahun 1942. Dan pada tahun 1955 J. A. H. Hunter mengenalkan kata *alphametic* untuk penjelasannya mengenai persoalan *cryptarithms.* Tipe-tipe dari *cryptarithms* termasuk juga mengenai alphametic, digimetic, dan pembagian kerangka atau *skeletal division*. Permainan dengan tujuan akhir untuk menentukan nilai dari setiap karakternya ini, dapat dikembangkan juga untuk menggunakan *non-alphabetic* karakter. Salah satu contoh klasik masalahnya di publikasikan oleh Henry Dudeney pada Juli 1984, Strand Magazine.



Gambar 1. Permasalahan klasik cryptarithmetic (Sumber: https://mindyourdecisions.com/ blog/2018/09/06/send-more-money-a-great-puzzle/)

Dengan solusi akhir puzzle tersebut adalah O = 0, M = 1, Y = 2, E = 5, N = 6, D = 7, R = 8, and S = 9.

Secara tradisional peraturannya adalah bahwa setiap huruf harus merepresentasikan digit yang berbeda dan seperti pada notasi aritmetik biasa digit huruf terdepan tidak boleh bernilai nol, dikarenakan teka-teki yang baik seharusnya memiliki solusi yang unik dan huruf-hurufnya mengikuti pernyataan tersebut, seperti contoh diatas.

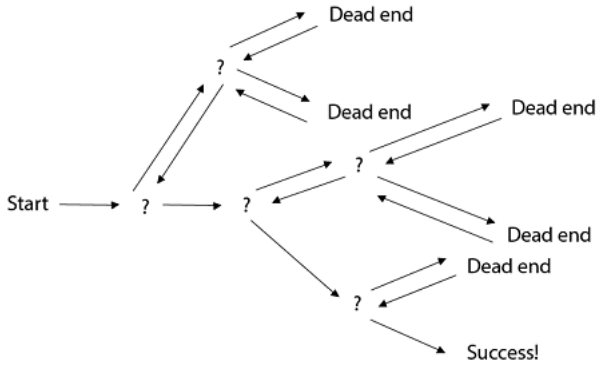
Secara umum peraturan permainannya dapat di rumuskan dengan tujuan akhir bahwa setiap hurufnya harus memiliki nilai digit diantara digit 0 hingga 9 sehingga persamaan aritmatika yang di persoalkan dapat terbukti benar, dan untuk semua kemunculan huruf yang sama memiliki nilai yang sama, dan tidak ada digit yang adalah nilai lebih dari satu huruf.

II. LANDASAN TEORI

A. Definisi Algoritma Backtracking

Algoritma backtracking adalah algoritma yang biasa digunakan untuk menemukan beberapa atau semua solusi yang mungkin dalam pemecahan masalah komputasinya. Algoritma backtracking memecahkan masalah dengan melibatkan pembangunan kandidat solusi secara bertahap, dan menyimpan setiap solusi yang memenuhi masalah, dan menghilangkan beberapa kandidat dengan segera karena kandidat tersebut tidak mungkin mengarah pada solusi, maka dari itu algoritma ini disebut dengan runut-balik.

Ide mengenai backtracking adalah membangun solusi perlangkahnya dengan menggunakan metode rekursif, dan ketika kandidat yang dibangun tidak dapat memenuhi solusi yang di inginkan, maka berhentikan proses solusi tersebut dan kembali ke langkah yang sebelumnya sebelum solusi tersebut muncul dan menghitung kemungkinan lain kandidat solusi. Karena itu, backtracking dianggap sebagai peningkatan dari metode brute-force yang sudah ada seperti exhaustive search yang membuat setiap solusi yang mungkin, tanpa memangkas kandidat yang tidak akan mengarah pada solusi yang memungkinkan. Solusi di pangkas melalu fungsi pembatas yang di definisikan._



Gambar 2. Ilustrasi pohon ruang status algoritma backtracking (sumber: https://static.javatpoint.com/tutorial/daa/images/backtracking-introduction.png)

Secara umum algoritma runut-balik memiliki sifat-sifat berikut:

1. Ruang Solusi

   Ruang solusi menyatakan kumpulan solusi suatu masalah. Ruang solusi direpresentasikan sebagai vektor dengan n-tuple buah bilangan bulat, dan setiap solusi dapat dinyatakan dengan 0 dan 1. Ruang vektor solusi:

   $$X = (x_1 , x_2 , \dots , x_n), x_i \in S_i$$

2. Fungsi Pembangkit

   Fungsi pembangkit dinyatakan dengan T(k), yang membangkitkan x_k atau solusi potensial.

3. Fungsi Pembatas

   Fungsi pembatas dinyatakan dengan $B(x_i,x_2,\dots, x_k)$, untuk setiap solusi potensial yang di bangkitkan. Fungsi pembatas di gunakan untuk setiap pembangkitan fungsi pembangkit untuk menentukan setiap solusi potensial mengarah ke solusi yang benar. Sehingga untuk setiap x_{k+1} solusi potensial akan di hilangkan pada ruang solusi sehingga tidak lagi di pertimbangkan dalam pencarian solusi selanjutnya. Predikat bentuk dari fungsi pembatas juga bisa di nyatakan sebagai nilai pengembalian yang dapat berupa nilai *boolean*.

B. Kelebihan Algoritma Backtracking

Algoritma backtracking sangat baik untuk eksekusi permasalahan yang berkaitan dengan *constraint*, sehingga solusi yang dihasilkan adalah *optimal global.* Ukuran kode biasanya tidak besar, dan relatif lebih mudah untuk di implementasikan.

C. Kekurangan Algoritma Backtracking

Algoritma backtracking menggunakan metode *multiple function calls* yang lebih boros atau mahal secara ukuran memori. Tidak efisien ketika banyak percabangan dari satu status ke yang lain, dan juga di butuhkan secara keseluruhan memori yang besar karena setiap status fungsi harus di simpan dalam sistem *stack*.

D. Prinsip Pencarian Solusi Algoritma Backtracking

Backtracking merupakan algoritma yang berdasarkan DFS pada pencarian solusinya. Pembentukan pencarian solusi pada pohon ruang status di bangkitkan secara dinamis. Secara umum langkah pencarian setiap solusinya dapat dirumuskan sebagai berikut:

a. Pencarian solusi yang dicari akan membentuk lintasan dari akar ke daun. Simpul-simpul yang dibentuk dinamakan simpul hidup. Sedangkan simpul hidup yang sedang diperluas atau dibentuk dinamakan simpul ekspansi. Aturan penomoran simpul dari atas ke bawah adalah sesuai dengan urutan pembentukannya.

b. Setiap saat simpul hidup atau ekspansi simpul, lintasan yang terbangun akan bertambah ukurannya. Jika suatu lintasan yang dibangkitkan tidak mengarah ke solusi yang di inginkan, maka simpul ekspan tersebut di hilangkan dari ruang solusi sehingga menjadi simpul mati. Fungsi yang digunakan untuk membunuh simpul ekspan adalah dengan mengimplementasikan fungsi pembatas atau bounding function. Simpul yang sudah mati tidak akan pernah diperluas lagi untuk mencari solusinya.

c. Setiap saat simpul hidup atau ekspansi simpul, lintasan yang terbangun akan bertambah ukurannya. Jika suatu lintasan yang dibangkitkan tidak mengarah ke solusi yang di inginkan, maka simpul ekspan tersebut di hilangkan dari ruang solusi sehingga menjadi simpul mati. Fungsi yang digunakan untuk membunuh simpul ekspan adalah dengan mengimplementasikan fungsi pembatas atau bounding function. Simpul yang sudah mati tidak akan pernah diperluas lagi untuk mencari solusinya.

d. Pencarian akan berhenti ketika sudah mendapatkan solusi atau tidak ada lagi simpul hidup untuk dapat dilakukan runut-balik.

E. Skema Umum Algoritma Backtracking

Pada umumnya algoritma runut-balik di implementasikan dalam dua versi, versi rekursif dan versi iteratif. Penggunaan skema rekursif dinilai lebih tepat dikarenakan pencarian solusi dalam pembangitan node nya lebih sesuai model ideal backtracking, walaupun belum tentu lebih baik, karena penggunaan memori ketika menggunakan pemanggilan fungsi secara terus menerus membebani ukurannya pada *stack* sistem. Dan berikut di sajikan skema ilustrasi algoritma runut-balik tersebut dalam versi iteratif dan rekursif.

```
procedure RunutBalikR(input k:integer) { versi rekusif }
{Mencari semua solusi persoalan dengan metode runut-balik; skema rekursif
 Masukan: k, yaitu indeks komponen vektor solusi, x[k]
 Keluaran: solusi x = (x[1], x[2], ..., x[n])}

Algoritma:
    for tiap x[k] yang belum dicoba sedemikian sehingga
    (x[k] ← T(k)) and B(x[1], x[2], ... ,x[k])=  true do
        if (x[1], x[2], ... ,x[k]) then
            CetakSolusi(x)
        endif
        RunutBalikR(k+1) { tentukan nilai untuk x[k+1]}
    endfor
```

```
procedure RunutBalikI(input n:integer) { versi iteratif }
{Mencari semua solusi persoalan dengan metode runut-balik; skema iteratif.
 Masukan: n, yaitu panjang vektor solusi
 Keluaran: solusi x = (x[1], x[2], ..., x[n])}

Delarasi:
    k : integer

Algoritma:
    k ← 1
    while k > 0 do
        if (x[k] belum dicoba sedemikian sehingga x[k] ← T(k)) and
          (B(x[1], x[2], ... ,x[k])= true)
        then
            if (x[1],x[2],...,x[k]) adalah lintasan dari akar ke daun
            then
                CetakSolusi(x)
            endif
            k ← k+1 {indeks anggota tupple berikutnya}
        else {x[1], x[2],..., x[k] tidak mengarah ke simpul solusi }
            xk ← k-1 {runut-balik ke anggota tupple sebelumnya}
    endif
    endwhile
    { k = 0 }
```

Pseudocode Algoritma Backtracking
(sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Algoritma-Runut-balik-(2020).pdf)

F. Penyelesaian Cryptarithmetic Dengan Backtracking

Karakterisasi penyelesaian cryptarithmetic menggunakan pendekatan backtracking di ilustrasikan dengan pendeskripsian pseudocode sebagai berikut:

1. Periksa huruf paling kanan dari baris pertama, dan beri nilai 0.
2. Jika posisi pemeriksaan terkini selain digit paling kiri (pemeriksaan belum sampai ujung terkini) mengembalikan true jika belum memiliki nilai dan false bila sudah.
3. Jika ketika memberi nilai pada karakter yang sedang pada operasi penjumlah / aritmatika, dan karakter tersebut telah memiliki nilai, lakukan pemberian nilai ke hasil karakter digit penjumlahannya.
4. Jika pada langkah ketiga karakter tersebut belum memiliki nilai maka untuk setiap kemungkinan digit yang belum terpakai gunakan nilai tersebut, dan jika sukses (kombinasi digit dan karakter memungkinkan) return true, dan jika tidak sukses return false, untuk melakukan backtrack pada pencarian nilai digit yang sudah sesuai.
5. Jika pada pemberian nilai pada karakter di sisi yang di jumlah, bila pemberian nilai benar ulangi pada kolom penjumlahan berikutnya.
6. Jika sudah diberi nilai dan kombinasi nilai tidak cocok (tidak konsisten sesuai aturan) kembalikan nilai false.
7. Jika belum diberi nilai dan digit telah terpakai semua kembalikan nilai false.
8. Jika belum diberi nilai dan digit masih ada belum yang terpakai, beri nilai dan lakukan pada kolom aritmatik selanjutnya, kembalikan true bila sukses dan false untuk melakukan backtrack.

III. IMPLEMENTASI ALGORITMA BACKTRACKING TERHADAP PENYELESAIAN CRYPTARITHMETIC PUZZLE

Penyelesaiannya adalah memberi nilai setiap karakter dengan digit 0 sampai 9, sehingga persamaan aritmatisnya konsisten. Untuk itu langkah pertamanya adalah dengan membuat fungsi permutasi untuk memeriksa digit kombinasi pada setiap langkah pemberian nilai tetap konsisten pada aturan permainan. Kemudian membuat fungsi untuk memeriksa operasi aritmatika dua karakter yang bersesuaian. Untuk kemudahan agar tidak melakukan pengecekan berulang pada digit yang sudah dipakai, nilai-nilai digit disimpan dalam array dinamis sehingga pemberian nilai pada digit tidak memerlukan pengecekan digit yang terpakai.

Struktur data penyelesaian masalah di representasikan dengan sebuah simpul dengan character dan nilai digitnya, direpresentasikan pada bahasa pemrograman sebagai class atau struct.

```
# Implementasi Cryparithmatic Puzzle Solver

# inisiasi array dinamis
use=[]

# digit yang sah, 0-9
for digitval in range(10):
        use.append(False)


# use untuk menyatakan keterpakaian angka dari 0-9


# class Node merepresentasikan karakter-karakter aritmatis
```

```
class Node:
   def __init__(self, char, digitval):
       self.char = char
       self.digit = digitval
```

Fungsi check dibuat untuk memeriksa nilai kombinasi pada posisi kolom terkini bahwa pemberian nilai pada karakter pada kolom tersebut memiliki nilai tetap konsisten pada nilai digit karakter yang sudah ada dan operasi aritmatis karakter pada kolom terkini benar. Fungsi check berisi parameter ruang solusi yang telah di hasilkan dari simpul ekspan dan menerima karakter pada kolom terkini.

Perulangan paling luar untuk memeriksa setiap barisnya, perulangan kedua untuk memeriksa nilai digit dari kanan, dan loop paling dalam untuk mendapatkan nilai dari karakter yang sudah pernah diberi nilainya. Variable count adalah jumlah karakter unik dari persoalan yang diberikan (karakter baris 1, 2, dan 3).

Fungsi check mengembalikan nilai true untuk dapat membangkitkan simpul ekspan solusi selanjutnya, dan mengembalikan nilai false untuk dapat memanggil backtracking. Dapat dikatakan bahwa fungsi check adalah *bounding function* atau fungsi pembatas.

```
def check(NodeArr,count,char1,char2,char3):
# inisiasi variabel
    iterVal=1
    val1=0
    val2=0
    val3=0
    val=[val1,val2,val3]
    char=[char1,char2,char3]

    for valIdx in range(0,3):
       for i in range(len(char[valIdx]), 0):
          for j in range(0, count):
             if (NodeArr[j].char == char1):
                       jIdx=j
                       break

          var[valIdx]+=(iterVal*NodeArr[jIdx]
          .digit)
          iterVal*=10
          iterVal=1
    if (val3==(val1+val2)):
          return True
    return False
```

Fungsi permutasi adalah fungsi rekursif yang memanggil fungsi check untuk setiap kemungkinan kombinasi bilangan dengan memanfaatkan ketersediaan bilangan pada variable *use.* Untuk setiap kemungkinan kombinasi yang selesai (*base case*) yang artinya adalah pemeriksaan kombinasi bilangan sudah mencapai bilangan terakhir, kemudian memeriksanya, bila mengembalikan *true*, maka kombinasi yang sedang di periksa merupakan solusi yang selesai.

Fungsi permutasi secara bertahap memeriksa untuk *n* kombinasi terendah terlebih dahulu, dan melakukan pemeriksaan rekursif untuk kombinasi *n+1* hingga jumlah karakter unik tersedia. Bila gagal dalam memeriksa untuk suatu *n* maka fungsi permutasi akan melakukan backtrack untuk memeriksa kemungkinan kombinasi yang lain.

```
def permutasi(NodeArr,count,n,char1,char2,char3):
   for i in range(10):
     if not(use[i]):
       NodeArr[n].digit = i
       if (n==count-1):
         if check(NodeArr,count,char1,char2,char3):
            solusi=''
            for j in range(count):
              solusi+=NodeArr[j].char+':'+
NodeArr[j].digit
            print(solusi)
            return True
          return False
      else:
        use[i] = True
        if
(permutasi(NodeArr,count,n+1,char1,char2,char3))
           return True
# digit tersebut dikatakan masih belum dipakai
        use[i] = False
        return False
```

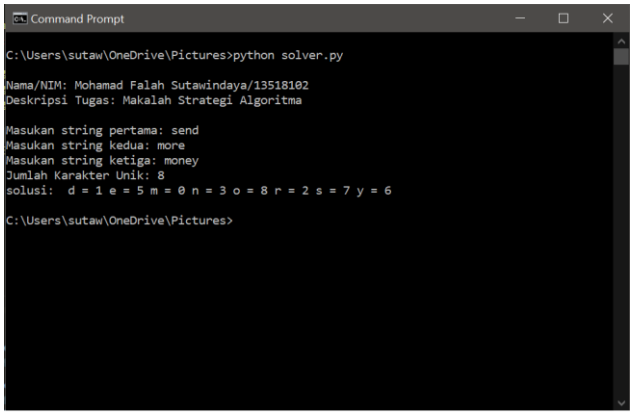
Fungsi mainSolver adalah fungsi utama dari solver puzzle ini, fungsi ini menerima 3 buah string, string ketiga adalah hasil aritmatis dari dua string pertamanya. Fungsi ini akan mengembalikan fungsi permutasi setelah fungsi mainSolver telah mendapatkan karakter-karakter uniknya. Kemudian fungsi permutasi akan mencoba menebak nilai-nilai digit karakter tersebut dengan *trial and error* menggunakan algoritma backtracking.

```
def mainSolver(char1arr, char2arr, char3arr):
# count untuk mendapatkan nilai karakter berbeda
  count=0
# inisiasi variabel panjang karakter input
  c1length = len(char1arr)
```

```
  c2length = len(char2arr)
  c3length = len(char3arr)


# menyimpan frekuensi karakter yang muncul pada
persoalan
  freq = [0]*26


# menyimpan sesuai index karakter (panjang huruf
26)


# untuk karakter baris pertamma
  for i in range(c1length):
     freq[ord(char1[i])-ord('A')]+=1


# karakter baris kedua
  for i in range(c2length):
     freq[ord(char2[i])-ord('A')]+=1


# karakter baris ketiga
  for i in range(c3length):
     freq[ord(char3[i])-ord('A')]+=1


# menyimpan jumlah karakter yang berbeda pada
persoalan
  for i in range(26):
     # ada kemunculan karakter tersebut
     if freq[i]>0:
        count+=1


# bila karakter unik lebih dari 10, tidak dapat
diselesaikan
# digit 0-9 tidak dapat di assign ke lebih dari 10
karakter unik
     if count > 10:
        print("String tidak bisa valid")


# instantiasi solusi simpul awal
# menyimpan default struct node untuk instansiasi
   NodeArr = []
   instanceNode = Node('A', 0)
       for i in range(count):
           NodeArr.append(instanceNode)


# menyimpan setiap karakter unik pada NodeArr
       j=0
       for i in range(26):
           if freq[i]>0:
               NodeArr[j].char=shift('A', i)
               j+=1


# setelah menyimpan karakter unik pada NodeArr,
akan di lakukan permutasi setiap nilai bilangan
pada karakter unik tersebut
    return permutasi(NodeArr, count, 0, char1arr,
char2arr, char3arr)
```

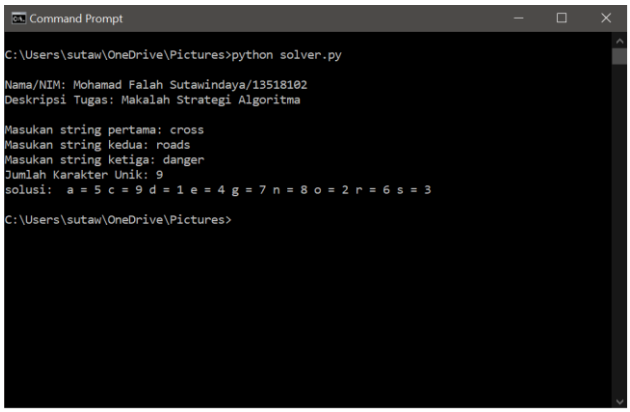
IV. Pengujian Algoritma Backtracking Terhadap Penyelesaian Cryptarithmetic Puzzle

Contoh pengujian berikut untuk beberapa permasalahan umum yang muncul. Program akan terlebih dahulu menerima 3 buah string, string ketiga adalah hasil aritmatis string kedua. Setelah itu program akan memproses untuk mencari digit setiap huruf-hurufnya. Bila keunikan huruf jumlahnya lebih dari 10 yaitu lebih dari jumlah digit maka tidak akan menghasilkan solusi.



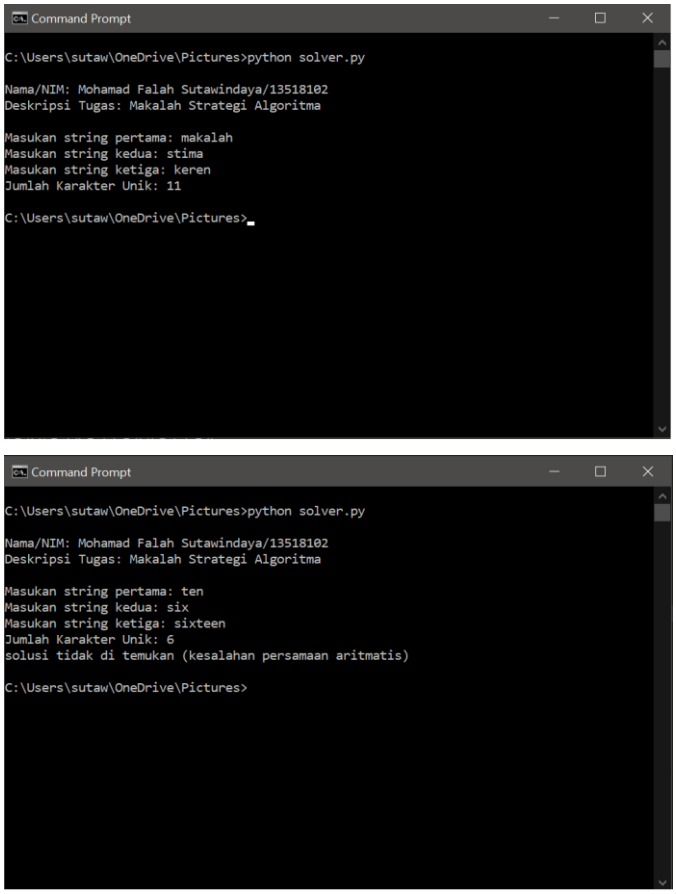
Gambar 3. Pengujian Pertama

Pengujian pertama sukses mencari nilai-nilai tersebut, karena jumlah karakter unik masih kurang dari digit yang ada.



Gambar 4. Pengujian Kedua

Pengujian kedua masih sukses, dan berikut beberapa contoh *edge cases* yang berhasil dikumpulkan.



Gambar 5. Pengujian Ketiga (edge cases)

Gambar diatas adalah contoh *edge cases* yang coba di atasi oleh program. Sejauh ini beberapa masalah-masalah umum dapat di selesaikan dengan baik, beberapa permsalahan khusus masih dapat di teliti lebih jauh lagi mengenai kesalahan dari programnya.

V. KESIMPULAN

Permasalahan *cryparithmatic* dapat diselesaikan menggunakan algoritma backtracking sederhana. Algoritma backtracking memang sangat baik ketika mencoba mencari solusi dimana persoalan berkaitan dengan *constraint*. *Constraint* tersebut dalam permasalahan ini adalah permutasi bilangan tersebut harus sesuai untuk setiap karakter-karakter unik pada input string yang diberikan. Permasalahan ini tidak begitu besar dalam ukuran *input-size* karena keterbatasan digit, untuk itu algoritma backtracking masih cukup baik dibandingkan dengan algoritma brute-force yang meng-enumerasi semua kemungkinan permutasi untuk setiap karakter uniknya.

LINK VIDEO YOUTUBE

https://youtu.be/D_D6TC2P_iw

UCAPAN TERIMA KASIH

Puji syukur dipanjatkan atas kehadirat tuhan YME, atas karunia dan nikmat-Nya penulis dapat menyelesaikan makalah ini. Penulis juga ingin menyampai terima kasih yang sebesar-besarnya kepada orang tua dan teman-teman serta dosen mata kuliah IF2211 Strategi Algoritma karena bimbingan dan ilmu yang disampaikan sangat bermanfaat dalam mewujudkan makalah ini.

REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Algoritma-Runut-balik-(2020).pdf. Diakses: 24 April 2020.

[2] https://www.nagwa.com/en/videos/284126246275/. Diakses 25 April 2020.

[3] https://www.geeksforgeeks.org/solving-cryptarithmetic-puzzles-backtracking-8/. Diakses 25 April 2020.

[4] https://people.revoledu.com/kardi/tutorial/CryptArithmetic/index.html. Diakses 25 April 2020.

[5] https://developers.google.com/optimization/cp/cryptarithmetic. Diakses 25 April 2020.

[6] https://www.geeksforgeeks.org/backtracking-algorithms/. Diakses 25 April 2020.

[7] https://www.javatpoint.com/backtracking-introduction. Diakses 25 April 2020.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2020

Mohamad Falah Sutawindaya
13518102