Penerapan Algoritma *Brute-Force* serta *Backtracking* dalam Penyelesaian *Cryptarithmetic*

Jason Jeremy Iman 13514058
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514058@std.stei.itb.ac.id

***Abstrak*—Cryptarithmetic adalah permainan matematika yang mengharuskan pemain untuk mencari representasi angka dari setiap huruf untuk menghasilkan nilai operasi yang tepat. Dalam penyelesaiannya perncarian solusi permainan ini dibutuhkan waktu yang cukup lama. Untuk itu, dapat dibuat suatu algoritma penyelesaikan dengan bruteforce maupun backtracking untuk mencoba kemungkinan-kemungkinan representasi dalam menghasilkan solusi yang sesuai.**

Keywords—backtracking; bruteforce; cryptarithmetic

I. PENDAHULUAN

Cryptarithmetic yang sering juga dikenal dengan nama verbal arithmetic atau alphametics adalah suatu jenis permainan matematika yang merepresentasikan angka sebagai huruf-huruf. Dalam permainan jenis ini, akan terdapat operasi-operasi aritmatika antar kata-kata yang menghasilkan suatu kata yang baru. Operasi yang terdapat dalam permainan ini dapat berupa penjumlahan, pengurang, pembagian, dan juga perkalian. Dalam permainan ini juga, suatu huruf hanya dapat merepresentasikan suatu angka, dengan kata lain tidak ada huruf yang memiliki nilai yang sama.

Permainan ini diperkirakan berasal dari tahun 1864 dengan nama yang pertama kali dicetuskan oleh seorang puzzlist bernama Simon Vatriquant. Contoh klasik dari permainan Cryptarithmetic ini ditulis tahun 1924 dengan persoalan sebagai berikut, SEND + MORE = MONEY. Selain, persoalan tersebut masih banyak alternatif persoalan dari jenis permainan ini.

Dalam penyelesaiannya suatu permainan Cryptarithmetic ini umumnya diselesaikan dalam waktu yang cukup lama. Hal ini disebabkan banyaknya kemungkinan nilai untuk setiap huruf. Berkaitan dengan hal tersebut, suatu algoritma bruteforce akan memakan waktu yang cukup lama. Untuk itu, dalam penyelesaian permasalahan Cryptarithmetic kali ini, akan digunakan algoritma runutbalik, backtracking.

II. TEORI DASAR

A. Algoritma Brute-Force

Brute-force merupakan suatu jenis pendekatan penyelesaian masalah yang bersifat sederhana, jelas, dan juga langsung. Dapat penyelesaian kombinatorik, algoritma brute-force terdiri dari beberapa langkah, diantaranya,

1. Enumerasi (*list*) setiap seolusi yang mungkin dengan cara yang sistematis.
2. Evaluasi setiap kemungkinan solusi
3. Apabila persoalan telah selesai solusi terbaik yang telah didapatkan akan menjadi hasil algoritma tersebut.

Algoritma brute-force dalam kombinatorik ini sering disebut sebagai algoritma exhaustive search. Di mana akan dilakukan pencocokan untuk semua kemungkinan kunci yang ada. Untuk itu, dalam pengerjaannya algoritma ini mampu menyelesaikan hampir seluruh persoalan. Namun, buruknya algoritma ini merupakan algoritma yang jarang menghasilkan algoritma yang mangkus dan sering tidak diterima karena lamanya pengerjaan.

B. Algoritma Backtracking

Algoritma runut-balik atau yang biasa disebut sebagai backtracking adalah algoritma sistematis dan mangkus. Algoritma ini berlangsung dengan prinsip pergerakan mundur apabila terdapat posisi atau *state* yang tidak lagi memungkinkan.

Berbeda dengan *exhaustive search*, suatu persoalan *backtracking* tidak mengecek semua kemungkinan. Hal ini disebabkan pemangkasan (*pruning*) pada solusi yang telah dinyatakan tidak mungkin.

Terdapat beberapa komponen algoritma runut-balik yang utama diantaranya,

1. Solusi

Solusi dinyatakan sebagai vektor dengan n-tuple,
$X = (x_1, x_2, \ldots, x_n)$, $x_i \in S_i$

2. Fungsi pembangkit nilai x_k

Dinyatakan sebagai T(k), yang akan membangkitkan nilai x_k

3. Fungsi pembatas

Dinyatakan sebagai predikat $B(x_1, x_2, \ldots, x_k)$
jika predikat tersebut tidak mengarah ke solusi
maka akan dibuang

Algoritma backtracking hanya dapat digunakan dalam persoalan-persoalan yang memiliki batasan-batasan yang dapat terlihat sebelum solusi akhir didapatkan. Diantaranya, sudoku, teka-teki silang, juga knapsack.

Aplikasi algoritma backtracking berkaitan erat dengan struktur pohon, biasa digunakan dengan *depth-first search* atau DFS. Dalam algoritma backtracking suatu kondisi digambarkan sebagai suatu node. Dan untuk setiap node, fungsi pembatas dapat menentukan apakah node tersebut masih dapat lanjut atau sudah tidak mungkin mendapatkan solusi.

C. *Crpytarithmetic*

Cryptarithmetic merupakan sebuah jenis permainan matematika dengan angka-angka yang dipresentasikan sebagai huruf atau simbol. Suatu persoalan cryptarithmetic juga digolongkan ke dalam persoalan NP-Complete untuk suatu basis tertentu. Hal ini disebabkan persoalan yang memiliki kompleksitas n! dengan n adalah jumlah karakter yang tersedia.

Suatu persoalan cryptarithmetic mengharuskan setiap huruf memiliki nilai yang unik. Oleh sebab itu, tidak semua kata dapat digunakan dapat permainan ini.

```
    E Y E
  x M A T
  -------
  S Y I A
G M T A
A I R Y
-------
A A S M A A
```

Gambar 2.1: Contoh persoalan Cryptarithmetic

Sebuah cryptarithmetic, seperti pada gambar 2.1, mengharuskan pemain untuk mencari nilai dari setiap huruf berkaitan dengan operasi yang diberikan.

Pada contoh di atas, perkalian dari EYE dan MAT harus menghasilkan SYIA sedangkan penjumlahan dari SYIA, GMTA, dan AIRY harus menghasilkan AASMAA.

Dalam pengerjaannya seorang harus mencari nilai-nilai yang pasti, dengan melihat karakteristik operasi, lalu melakukan deduksi terhadap huruf-huruf yang belum diketahui nilainya.

III. Penyelesaian Cryptarithmetic

A. *Penyelesaian Manual Cryptarithmetic*

Suatu cryptarithmetic umumnya merupakan persoalan yang dikerjakan dalam waktu 10 sampai 12 menit. Pengerjaan persoalan ini bergantung pada kesulitan dan kemampuan seseorang dalam mengerjakannya. Adapun langkah pengerjaan persoalan cryptarithmetic adalah sebagai berikut,

1. Deduksi kemungkinan angka dengan karakteristik khusus. Sebagai contoh, dalam penjumlahan dua buah kata, apabila hasil memiliki jumlah digit yang lebih banyak, maka digit pertama pasti merupakan angka 1.
2. Melakukan percobaan untuk angka-angka yang lain.
3. Ulangi langkah 2 dan 1 sampai mendaptkan jawaban.

Sebagai contoh untuk kasus,

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

Gambar 3.1: Contoh persoalan Cryparithmetic

Langkah pertama adalah menyatakan bahwa M = 1. Karena hanya merupakan penjumlahan dua bilangan, maka nilai M yang hanya mungkin merupakan *carrier* dari S + M. Karena M juga huruf pertama, dipastikan M bukanlah 0.

Dari persoalan tersebut juga dapat diturunkan beberapa fakta diantaranya, pada kolom kedua nilai O harus lebih kecil atau sama dengan M (penjumlahan M dan S lebih dari 10). Karena M = 1 dan tidak boleh ada angka yang sama, maka O = 0.

Dengan fakta bahwa O = 0, maka didapatkan bahwa S adalah 8 atau 9 (penjumlahan S dan M harus lebih dari 10) bergantung pada penjumlahan E dan O. Namun, karena O adalah 0, maka dipastikan jumlah E dan O tidak lebih besar dari 10. Maka, S adalah 9.

Pada kondisi sekarang didapatkan bentuk persamaan sebagai berikut.

```
 9END
 10RE
----+
10NEY
```

Setelah itu dapat juga dipastikan kalau N adalah E + 1 (E dan N tidak mungkin sama, *carrier* dari N + R). Karena N + R = E + 10 dan N = E + 1, didapatkan R = 8.

```
 9END
 108E
```

```
----+
10NEY
```

Dari perhitungan tersebut didapatkan juga D + E lebih dari 10 karena nilai N + R > 10 dan N tidak sama dengan 1 maupun 0 serta E tidak sama dengan 0.

Nilai terkecil yang masih mungkin adalah 2, maka Y minimal adalah 2 dan D + E minimal adalah 12.

Karena nilai yang tersedia untuk menghasikan D + E >= 12 hanyalah 7 5 dan serta 7 6, maka D atau E pasti memiliki nilai 7.

Jika E = 7, maka persamaan N = E + 1, akan menjadi tidak valid (8 sudah digunakan). Maka dari itu, disimpulkan bahwa D = 7.

```
 9EN7
 10RE
----+
10NEY
```

Untuk persamaan N = E + 1, nilai E yang mungkin adalah 4 dan 5. Namun, jika E = 4 nilai Y menjadi tidak valid (1 sudah dipakai), maka E = 5 dan N = 6.

```
 9567
 1085
----+
1065Y
```

Dari data tersebut hanya perlu dilengkapi bahwa Y = 2.

```
 9567
 1085
----+
10652
```

Perhitungan tersebut menghasilkan nilai operasi yang valid dengan O = 0, M = 1, Y = 2, E = 5, N = 6, D = 7, R = 8, serta S = 9.

B. Cryptarithmetic dengan Algoritma Brute-Force

Sebuah persoalan cryptarithmetic dapat diselesaikan dengan menggunakan algoritma brute-force. Suatu solusi dari cryptarithmetic akan didapatkaan dengan mengiterasi setiap kemungkinan dari kombinasi huruf.

Pengerjaan dengan algoritma *brute-force* adalah sebagai berikut.

1. Nyatakan setiap huruf sebagai suatu angka. Masing -masing angka harus berbeda
2. Cek kebenaran dari penjumlahan dua buah kata tersebut dengan kata solusi.
3. Apabila salah representasikan ulang angka pada huruf. Tambahkan satu pada representasi huruf untuk mencegah pengulangan.
4. Ulangi langkat 1 sampai 3, sampai menemukan solusi

Berikut langkah-langkah pengerjaan dengan contoh 3.1.

```
 SEND
 MORE
----+
MONEY
```

Pertama, untuk setiap huruf yang unik dibentuk string baru. SEND + MORE = MONEY. Membuat string baru yaitu, SENDMORY.

Lalu untuk setiap karakter inisialisasi nilai, maka:

```
SENDMORY

01234567
```

Cek apakah penjumlahan dengan angka tersebut sudah benar.

```
 0123
 4561
----+
45217
```

Karena penjumlahan belum benar, maka dilakukan pengecekan terhadap representasi nilai karakter berikutnya.

```
SENDMORY

01234568
```

Lakukan pengecekan terhadap nilai representasi baru.

```
 0123
 4561
----+
45218
```

Penjumlahan masih belum sesuai, lakukan terus penambahan nilai. Apabila nilai satuan sudah mencapai 9, tambahkan 1 pada nilai puluhan dan ubah nilai 9 menjadi 0, sama seperti penjumlahan biasa.

Iterasi penambahan representasi dilakukan terus-menerus sampai terdapat nilai penjumlahan yang sesuai dan nilai mengikuti aturan yang sesuai juga (tidak ada angka duplikat, tidak diawali 0).

Adapun *pseudocode* dari algoritma *brute-force* yang digunakan adalah sebagai berikut.

```
procedure CBruteForce(input string s1, s2,
shasil)

{ mencari solusi dari s1 + s2 = shasil }

Deklarasi:
   string ssolusi
   // string mencatat representasi angka
   string sh
   // string setiap huruf yang ada
   boolean benar
   // mengecek apakah penjumlahan benar
   int i

Algoritma:
   for (i=0; i<panjang(s1); i++)
      if (s1[i] belum ada di sh)
         tambahkan
   for (i=0; i<panjang(s2); i++)
      if (s2[i] belum ada di sh)
         tambahkan
```

```
for (i=0; i<panjang(s3); i++)
   if (s3[i] belum ada di sh)
      tambahkan

for (i=0; i<panjang(sh); i++)
   ssolusi[i] = i // inisialisasi

replace(ssolusi, s1, s2, shasil)
// Mengganti nilai huruf menjadi
   angka yang telah dibuat

while (!benar) do
   add(ssolusi)
   // cari kemungkinan lain dari
      untuk mendapatkan solusi

   replace(ssolusi, s1, s2, shasil)

   benar = check(s1, s2, shasil)
   // Mengecek penjumlahan

-> ssolusi
// Mendapatkan hasil
```

Untuk persoalan SEND + MORE = MONEY yang digambarkan pada gambar 3.1 pengerjaan dengan brute-force akan melakukan permutasi pemilihan m dari n pilihan. Perhitungan nilai kombinasi tersebut mencapai 1.814.400 kemungkinan.

Berkaitan dengan banyaknya jumlah kemungkinan, suatu algoritma brute-force akan memakan waktu yang sangat lama.

C. Cryptarithmetic dengan Algoritma Backtracking

Suatu persoalan cryptarithmetic dapat juga diselesaikan dengan menggunakan algoritma runut-balik. Berbeda dengan algoritma *exhaustive-search* yang mengiterasi setiap kemungkinan, algoritma *backtracking* dapat memisahkan solusi sementara yang tidak mengarah pada solusi sebenarnya.

Langkah-langkah pengerjaan suatu cryptarithmetic dengan algoritma *backtracking* adalah sebagai berikut.

1. Pisahkan setiap bagian dari kata berdasarkan satuannya.
2. Cari solusi yang memungkinkan untuk satuan yang sedang dievaluasi
3. Apabila sudah didapatkan, lanjutkan ke satuan berikutnya dan ulangi langkah 2.
4. Apabila tidak ada solusi yang memungkinkan, mundur satu langkah ke satuan sebelumnya.
5. Ulangi langkah 1 sampai 5, sampai solusi sudah benar pada setiap satuan

Berikut langkah pengerjaan untuk contoh pada gambar 3.1.

```
 SEND
 MORE
----+
MONEY
```

Pertama, buat string karakter dengan pemisahan satuan (satuan = s, puluhan = p, ribuan = r, puluhribuan = q).

```
MSORNYED
qqrppsss
```

Untuk langkah pertama yaitu D, E, dan Y sebagai satuan menjadi komponen pertama. N, M, dan E sebagai puluhan menjadi komponen kedua. Lalu cari setiap kemungkinan agar solusi benar dalam bentuk satuan (1 digit + 1 digit = 1 digit).

Inisialisasi nilai hanya untuk yang sedang diproses, lalu cari nilai yang sesuai.

```
MSORNYED
qqrpp210
```

Tambahkan representasi satuan sampai penjumlahan untuk satuan benar.

```
 SEN2
 MOR1
----+
MONE3
```

Catatan, untuk nilai D = 7, E = 8, dan Y = 5, juga tergolong benar, nilai yang dihitung sepenuhnya hanya satuan (1 digit)

```
 SEN7
 MOR8
----+
MONE5
```

Karena nilai sudah benar inisialisasi untuk puluhan.

```
MSORNYED
qqr40312
```

Apabila terdapat penjumlahan yang menghasilkan nilai puluhan yang benar, maka lanjutkan ke bentuk ratusan.

Penambahan puluhan hanya dilakukan untuk puluhan, tidak untuk satuan. Hal tersebut berlaku untuk ratusan dan setiap representasi lain. Berikut contoh penambahan puluhan.

```
MSORNYED
qqr98312

MSORNYED
qqr40312
```

Namun apabila dengan satuan D, E, Y berturut-turut 2, 1, 3 tidak terdapat solusi dalam bentuk puluhan, lakukan *backtrack*. Tidak ada solusi ditandai dengan kembalinya ke bentuk awal (seperti contoh diatas ppsss 40312 kembali menjadi 40312).

Yaitu, hapus kembali representasi puluhan, dan cari representasi satuan lain yang cocok.

```
MSORNYED
qqrpp532
```

Lakukan proses tersebut berturut-turut sampai akhirnya mencapai jumlah puluhribuan yang tepat.

Adapun *pseudocode* untuk penyelesaian dengan algoritma runut-balik adalah sebagai berikut.

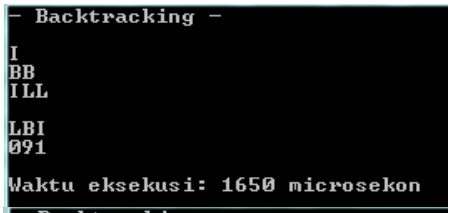
```
procedure CBacktrack(input string s1, s2,
shasil)

{ mencari solusi dari s1 + s2 = shasil }
```

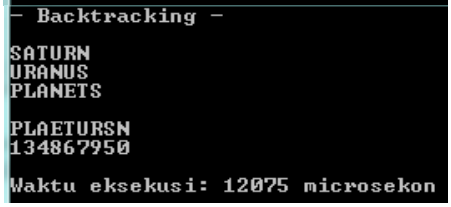
```
Deklarasi:
   string ssolusi
   // string mencatat representasi angka
   string sh
   // string setiap huruf yang ada
   boolean benar
   // mengecek apakah penjumlahan benar
   boolean tbenar
   boolean ulang
   int i
   int level = 1
   // level satuan yang diperiksa

Algoritma:
   for (i=0; i<panjang(s1); i++)
      if (s1[i] belum ada di sh)
         tambahkan
   for (i=0; i<panjang(s2); i++)
      if (s2[i] belum ada di sh)
         tambahkan
   for (i=0; i<panjang(s3); i++)
      if (s3[i] belum ada di sh)
         tambahkan

   for (i=0; i<panjang(sh); i++)
      ssolusi[i] = i // Inisialisasi

   replace(ssolusi, s1, s2, shasil)
   // Mengganti nilai huruf menjadi
      angka yang telah dibuat

   while (!benar) do
      while (!tbenar && !ulang) do
      // cek selama tidak ulang atau
         salah

         add(ssolusi)
         // cari kemungkinan lain dari
            untuk mendapatkan solusi

         replace(ssolusi, s1, s2,
         shasil)

         tbenar = check(s1, s2, shasil)
         // Mengecek penjumlahan

      if (ulang)
          level <- level - 1
      else
          level <- level + 1

   -> ssolusi
   // Mendapatkan hasil
```

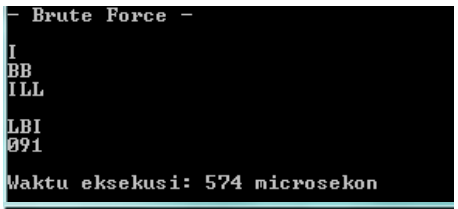
IV. Pengujian dan Analisis

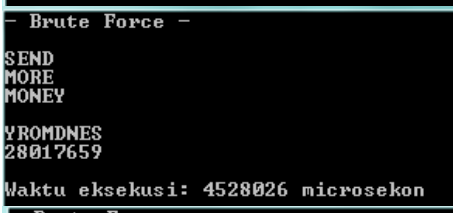
A. Hasil Uji Algoritma Bruteforce dan Backtracking

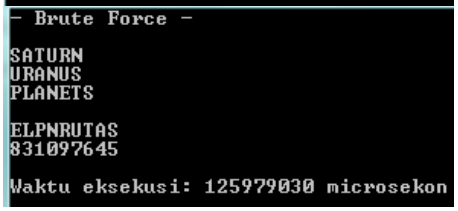
Berikut dilampirkan hasil uji untuk kedua algoritma untuk beberapa kasus.











B. Analisis

Dari data uji yang didapatkan, untuk suatu data yang kecil (3 karakter) algoritma *brute-force* mampu mengimbangi algoritma *backtracking* dan bahkan lebih cepat. Namun, untuk kasus yang lebih besar (8 karakter), kecepatan algoritma *backtracking* 331 kali lebih cepat. Sedangkan untuk kasus 9 karakter, kecepatan *backtracking* mencapai 10.416 kali lebih cepat.

Hal ini menandakan bahwa algoritma *backtracking* jauh lebih efektif untuk kasus yang besar, namun untuk kasus yang relatif kecil, algoritma *bruteforce* dapat dikatakan cukup.

V. Kesimpulan

Cryptarithmetic merupakan suatu persoalan yang dapat diselesaikan dengan menggunakan algoritma *brute-force* maupun *backtracking*. Hal ini disebabkan persoalan tersebut

merupakan persoalan yang membutuhkan percobaan angka atau dengan kata lain penebakan angka untuk representasi setiap huruf. Dalam hal ini, kedua algoritma merupakan algoritma yang mencoba setiap kombinasi dari angka-angka tersebut.

Perbedaan penggunaan kedua algoritma tersebut terdapat pada pemilihan langkah serta pengerjaan. Pada algoritma *brute-force* setiap kemungkinan dicoba sampai mendapatkan hasil. Sedangkan algoritma *backtracking* lebih teratur dan memilih untuk melanjutkan hanya representasi yang dapat membawa pada solusi.

Oleh karena itu, pengerjaan dengan algoritma *backtracking* lebih cepat dibanding algoritma *brute-force*. Namun, untuk kasus yang kecil algoritma *brute-force* dapat dikatakan lebih baik dikarenakan pembuatan yang juga lebih mudah.

Namun, walaupun kecepatan pengerjaan kedua algoritma berbeda. Kedua algoritma tersebut masih dapat mengalahkan kecepatan pengerjaan rata-rata secara manual.

VI. Ucapan Terima Kasih

Pertama penulis ingin mengucapkan terima kasih kepada Tuhan yang Maha Esa karena hanya atas berkat-Nya penulis dapat menyelesaikan makalah ini. Penulis juga ingin menyampaikan terima kasih kepada orang tua yang tanpanya penulis tidak akan mampu mendapat pengetahuan sejauh ini. Penulis juga tidak lupa ingin mengucapkan terima kasih kepada Dr. Ir. Rinaldi Munir, MT serta Dr. Nur Ulfa Maulidevi, ST, M.Sc. yang telah memberikan pengajarana mengenai materi yang dibahas. Selain itu, penulis juga ingin mengucapkan terima kasih kepada teman-teman serta keluarga yang telah membantu melalui masukan, dukungan, serta doa.

References

[1] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Penerbit Informatika.

[2] A, Newell, Studies in Problem Solving, Cryptarithmetic: http://repository.cmu.edu/cgi/viewcontent.cgi?article=2730&context=compsci diakses 6 Mei 2016. Pukul 21.12

[3] Collins, Alphametics Index: http://www.tkcs-collins.com/truman/alphamet/ diakses 6 Mei 2016. Pukul 20.25

Pernyataan

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

Bandung, 6 Mei 2016

Jason Jeremy Iman - 13514058