Laporan Tugas

Penyelesaian *Cryparithmatic* dengan Algoritma *Brute Force*

Fadhil Imam Kurnia - 13515146

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

A. Permasalahan

Cryptarithmatic merupakan pengetahuan dan seni untuk menciptakan dan menyelesaikan teka-teki matematika. Pada cryptarithmatic digit-digit yang ada ditukar dengan huruf-huruf alfabet atau simbol lainnya yang memiliki makna[1]. Contoh permasalahan crypthoarithmatic dengan solusinya diantaranya adalah sebagai berikut:

```
  S E N D          9 5 6 7
+ M O R E        + 1 0 8 5
-----------      -----------
M O N E Y        1 0 6 5 2
```

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

```
  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
```

Jadi, J = 7, U = 9, N = 2, E = 4, L = 0, Y = 6, A = 1, P = 5, R = 8, I = 3

```
    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
```

Jadi, F = 2, O = 9, R = 7, T = 8, Y = 6, E = 5, N = 0, S = 3, I = 1, X = 4

Salah satu cara untuk menyelesaikan cryptarithmatic adalah dengan mencoba semua kemungkinan yang dapat diperoleh[2]. Teknik tersebut biasa disebut dengan istilah *brute force*. Kita dapat mendesain suatu algoritma *brute force* untuk menyelesaikan permasalahan tersebut dan diimplementasikan menggunakan bahasa pemrograman di komputer. Dengan bantuan algoritma *brute force* dan program komputer yang dibuat, kita dapat menyelesaikan *cryptarithmatic* yang ada.

B. Penyelesaian Masalah

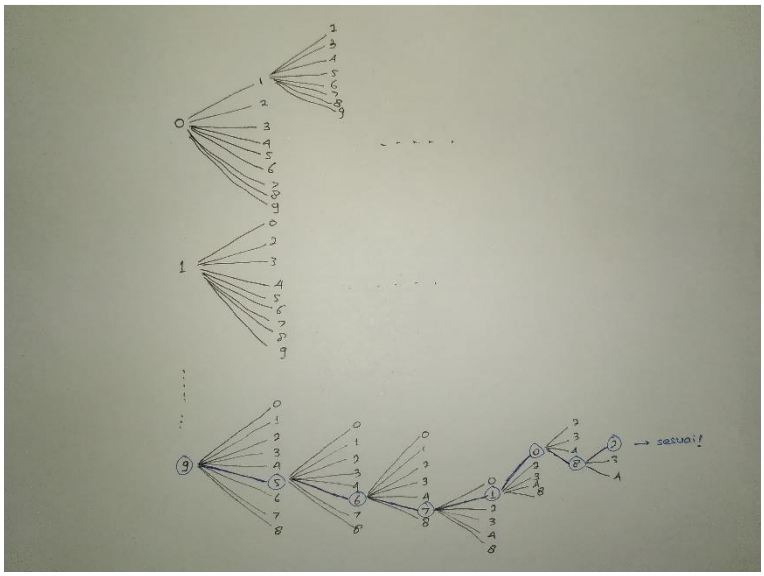
Desain awal algoritma *brute force* yang diperoleh adalah sebagai berikut:

1. Data semua kemunculan huruf dalam crypthoarithmatic yang diinputkan, hindari duplikasi dalam pendataan huruf yang muncul.
2. Pastikan jumlah huruf yang digunakan kurang dari atau sama dengan 10 untuk melenjutkan proses.
3. Untuk huruf pertama coba dengan mensubtitusinya menggunakan angka 0. Kemudian ambil huruf selanjutnya dan coba subtitusi dengan angka 0 sampai 9, namun tidak termasuk angka yang sudah digunakan sebelumnya. Lakukan ini hingga huruf terakhir.
4. Jika telah mencapai huruf terakhir coba evaluasi kedalam persamaan yang diinputkan. Jika sesuai maka permasalahan selesai. Namun jika tidak sesuai, kembali lagi ke langkah 3, namun dengan percobaan angka selanjutnya yang berbeda, yaitu angka 1, 2, 3, 4, 5, 5, 6, 7, 8, dan 9.

Sebagai contoh, kita gunakan crypthoarithmatic berikut:

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

1. Semua kemunculan huruf yang ada adalah: S, E, N, D, M, O, R, Y. Terdapat 8 kemunculan huruf.
2. Jumlah huruf kurang dari atau sama dengan 10, sehingga bisa memiliki solusi. Lanjut ke proses berikutnya.
3. Mencoba semua kemungkinan angka dalam huruf, alur *brute force* yang diperoleh adalah sebagai berikut:

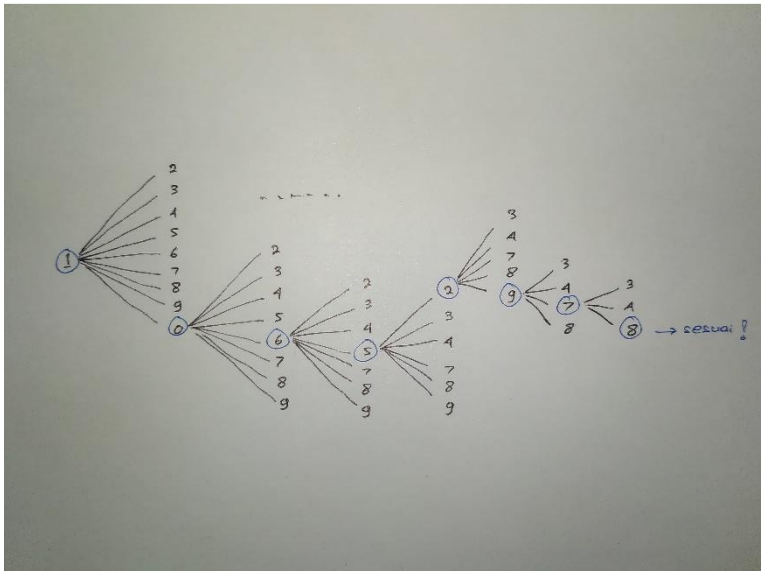


Gambar 1 Proses *brute force* sebelum optimasi yang harus mencoba semua kemungkinan hingga S mencapai angka 9

Jika kita perhatikan dalam penyelesaian kasus tersebut, huruf yang pertama didata adalah S. Padahal huruf S merupakan kode dari angka 9, oleh karena itu dibutuhkan *brute force* hingga angka 9 untuk mencapai solusi dari permasalahan tersebut. Kita dapat mengembangkan algoritma *brute force* tersebut menjadi lebih baik dengan mendata hasil penjumlahan terlebih dahulu. Pada contoh diatas kita sebaiknya mendata huruf M pertama kali karena ada kemungkinan yang lebih besar bahwa huruf pertama pada hasil penjumlahan adalah angka 1. Pada algoritma diatas urutan *brute force* yang dilakukan adalah mulai dari 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Kita sebaiknya melakukannya dengan urutan 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, karena huruf pertama dimungkinkan adalah angka 1. Dengan berbagai pengembangan tersebut, jika digunakan cryptarithmatic pada contoh diatas kita hanya perlu melakukan *brute force* hingga M mencapai angka 1. Sehingga didapat algoritma berikut:

1. Data semua kemunculan huruf pada hasil penjumlahan. Pastikan huruf pertama pada hasil penjumlahan didata pertama kali.
2. Data semua kemunculan huruf pada operan-operan yang ada.
5. Pastikan jumlah huruf yang digunakan kurang dari atau sama dengan 10 untuk melenjutkan proses.
6. Untuk huruf pertama coba dengan mensubtitusinya menggunakan angka 1. Kemudian ambil huruf selanjutnya dan coba subtitusi dengan angka 0 sampai 9, namun tidak termasuk angka yang sudah digunakan sebelumnya. Lakukan ini hingga huruf terakhir.
7. Jika telah mencapai huruf terakhir coba evaluasi kedalam persamaan yang diinputkan. Jika sesuai maka permasalahan selesai. Namun jika tidak sesuai, kembali lagi ke langkah 3, namun dengan percobaan angka yang berbeda, yaitu angka 2, 3, 4, 5, 5, 6, 7, 8, 9, dan 0.

Untuk contoh yang sama dengan sebelumnya, didapatkan urutan huruf: M, O, N, E, Y, S, D, dan R. Proses *brute force* yang dilakukan adalah sebagai berikut.

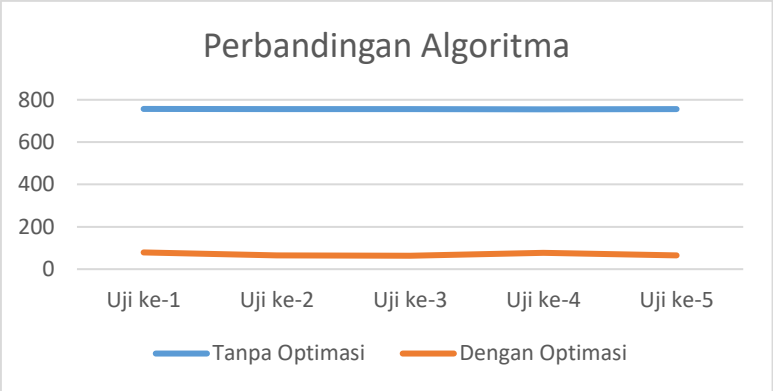


Gambar 2 Proses *brute force* setelah optimasi, huruf M adalah angka 1 sehingga kemungkinan yang dicoba dapat lebih sedikit

Sesuai Gambar 2, kita melakukan proses *brute force* lebih sedikat ketimbang aalgoritma sebelumnya. Perbedaan waktu dengan dilakukannya pengembangan ini dapat dilihat pada Tabel 1.

Uji ke-	Waktu Eksekusi Sebelum Optimasi (ms)	Waktu Eksekusi Setelah Optimasi (ms)
1	756,804	78,986
2	755,556	65,318
3	755,724	63,349
4	754,945	77,578
5	755,424	64,391

Tabel 1 Perbandingan algoritma sebelum dan sesudah optimasi untuk kasus SEND + MORE = MONEY



Gambar 3 Perbandingan algoritma sebelum dan sesudah optimasi untuk kasus SEND + MORE = MONEY

Berdasarkan hasil uji pada Tabel 1 dan Gambar 3, optimasi algoritma tersebut membuat proses *brute force* menjadi sekitar 10 kali lebih cepat. Algoritma yang didesain tersebut akan semakain cepat dieksekusi jika jumlah kemunculan huruf sedikit. Kemungkinan terburuknya adalah jika harus dilakukan semua kemungkinan yaitu sebanyak 10! = 3.628.800 kemungkinan. Jika terdapat N huruf yang harus diproses maka jumlah kemungkinan maksimalnya adalah $\frac{10!}{(10-N)!}$ kemungkinan.

C. Kode Program

Untuk mengimplementasikan algoritma yang telah didesain untuk menyelesaikan cryptarithmatic digunakan bahasa C++. Bahasa tersebut dipilih karena terdapat struktur data yang dapat digunakan untuk membantu mengimplementasikan algoritma *brute force*. Beberapa struktur data yang digunakan diantaranya adalah vector untuk menampung operan dan hasil penjumlahan. Kemudian juga digunakan larik angka untuk menyimpan angka yang direpresentasikan oleh tiap huruf. Berikut merupakan kode program yang telah dibuat:

main.cpp

```
// File  : main.cpp
// Name  : Fadhil Imam Kurnia - 13515146
// Program to solving cryptarithmatic with brute force

#include <iostream>
#include <string>
#include <vector>
#include <list>
#include <time.h>
using namespace std;

list<int> idx_char;
vector<int> idx_char_v;
vector<int> temp;
int angka[26];
string result;
vector<string> operand;
bool selesai = false;

int stringToNumber(string s){
  int hasil = 0,i;
  for(i = 0; i < s.length() ; i++){
    hasil = (hasil*10) + angka[(int) s[i]-'A'];
  }
  return hasil;
}

bool evaluasi(){
  int hasil = stringToNumber(result);
  if(angka[(int)result[0]-'A'] == 0)
    return false;
  int temp = 0;
  for (vector<string>::iterator it = operand.begin() ; it != operand.end(); ++it){
    if(angka[(int) (*it)[0]-'A'] == 0)
      return false;
    int x = stringToNumber(*it);
    temp = temp + x;
  }
  return (temp == hasil);
}

void writeResult(){
  int i = 0;
  for (vector<string>::iterator it = operand.begin() ; it != operand.end(); ++it){
    i++;
    if( i == operand.size())
      cout << *it << "+" << endl;
    else
      cout << *it << endl;
  }
  cout << "--------" << endl;
  cout << result<< endl << endl;

  // menulis hasil
  for (vector<string>::iterator it = operand.begin() ; it != operand.end(); ++it){
    i--;
    if( i == 0)
      cout << stringToNumber(*it) <<  "+" << endl;
    else
      cout << stringToNumber(*it) << endl;
  }
  cout << "--------" << endl;
  cout << stringToNumber(result)<< endl;

}


void checkAll(int offset, int k){
  bool found; int j;

  if(selesai)
    return;
  else if(k == 0){
    if(evaluasi()){
      writeResult();
      selesai = true;
    }
    return;
  }else{
    // brute force dari 1 sampai 9
    for(int i = 1; i<=9; i++){
      found = false; j = 0;
      while(j < temp.size() && !found){
        if(temp[j] == i)
          found = true;
```

```
        else
          j++;
      }
      if(!found){
        angka[idx_char_v[offset]] = i;
        temp.push_back(i);
        checkAll(offset+1,k-1);
        temp.pop_back();
      }
    }

    // brute force untuk angka 0
    found = false; j = 0;
    while(j < temp.size() && !found){
      if(temp[j] == 0)
        found = true;
      else
        j++;
    }
    if(!found){
      angka[idx_char_v[offset]] = 0;
      temp.push_back(0);
      checkAll(offset+1,k-1);
      temp.pop_back();
    }

  }
}


int main(){
  string s; int i;

  // membaca masukan cryptarithmatic bagian operan-operan
  do{
    getline(cin, s);
    i = 0;
    while( i < ( (s[s.length()-1]!='+')? s.length() : s.length()-1)){
      idx_char.push_back( (int) s[i]-'A');
      angka[(int) s[i]-'A'] = i; // inisialisasi
      i++;
    }
    operand.push_back(s);
  }while(s[s.length()-1]!='+');
  operand.back() = operand.back().substr(0, operand.back().length()-1);
  getline(cin, s);
  getline(cin, result);

  // membaca masukan cryptarithmatic bagian hasil penjumlahan
  i = 0;
  while(i<result.length()){
    idx_char.push_back( (int) result[i]-'A');
    angka[(int) s[i]-'A'] = i; // inisialisasi
    i++;
  }

  // mendata kemunculan huruf, namun meletekan awal huruf pada hasil penjumlahan pertama kali
  idx_char.sort();
  idx_char.unique();
  for (list<int>::iterator it = idx_char.begin(); it != idx_char.end(); ++it){
    if(*it == result[0]-'A')
        idx_char_v.insert(idx_char_v.begin(),*it);
    else
      idx_char_v.push_back(*it);
  }

  // memulai bruteforce dengan fungsi rekursif, menampilkan hasilnya, serta menghitung waktunya
  int awal = clock();
  checkAll(0,idx_char.size());
  int akhir = clock();
  if(!selesai)
    cout << "Tidak ada solusi" << endl;
  cout << endl << "waktu: "<< (double)(akhir-awal/(CLOCKS_PER_SEC/1000)) << " ms" << endl;

  return 0;
}
```

Kode program dan beberapa kasus uji dapat dilihat melalui :
https://github.com/fadhilimamk/cryptarithmetic

D. Contoh Hasil Eksekusi Program (Masukan dan Keluaran)

Kasus Uji 1

```
D:\Proyek\Cryptarithmetic>main.exe < in
KYOTO
OSAKA+
--------
TOKYO

41373
32040+
--------
73413


waktu: 46 ms
```

Kasus Uji 2

```
D:\Proyek\Cryptarithmetic>main.exe < in2
APPLE
GRAPE+
--------
CHERRY

63374
90634+
--------
154008


waktu: 171 ms
```

Kasus Uji 3

```
D:\Proyek\Cryptarithmetic>main.exe < in3
BLACK
GREEN+
--------
ORANGE

79208
53446+
--------
132654


waktu: 46 ms
```

Kasus Uji 4

```
D:\Proyek\Cryptarithmetic>main.exe < in4
SEND
MORE+
--------
MONEY

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


waktu: 78 ms
```

Kasus Uji 5

```
D:\Proyek\Cryptarithmetic>main.exe < in5
WORD
WORD+
--------
SONG

1043
1043+
--------
2086


waktu: 46 ms
```

Kasus Uji 6

```
D:\Proyek\Cryptarithmetic>main.exe < in6
TED
HAS
GOOD+
--------
TASTE

134
605
9774+
--------
10513


waktu: 140 ms
```

Kasus Uji 7

```
D:\Proyek\Cryptarithmetic>main.exe < in7
TELL
TALE
TELL
TALE+
--------
LATE

1055
1850
1055
1850+
--------
5810


waktu: 15 ms
```

Kasus Uji 8

```
D:\Proyek\Cryptarithmetic>main.exe < in8
NO
GUN
NO+
--------
HUNT

87
908
87+
--------
1082


waktu: 15 ms
```

E. Ceklist Pengerjaan Tugas

Poin	Ya	Tidak
1. Program berhasil dikompilasi	√	
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.	√	

Daftar Pustaka

[1] Sibarani, Elisa. 2006. Persoalan Cryparithmatic dengan Algoritma Backtracking. Makalah Strategi Algoritmik Teknik Informatika Institut Teknologi Bandung.

[2] Rinaldi Munir, Diktat Kuliah IF2251 Strategi Algoritmik, STEI, 2006.