Penerapan Algoritma *Brute force* dalam Penentuan Keanagraman Dua Buah *String*

A. Bara Timur (13510019)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
bara.timur@itb.ac.id

***Abstraksi*—Pada makalah ini akan dijelaskan penggunaan algoritma *brute force* untuk menentukan apakah kedua *string* yang dibandingkan adalah anagram atau bukan. Termasuk perbandingan antara algoritma yang berbeda dan aplikasinya.**

Kata Kunci*—Anagram, *Brute force, String

I. PENDAHULUAN

Anagram dipakai di beberapa aspek kehidupan contohnya permainan dan puzzle. Selain itu dipakai juga di bidang keamanan informasi. Dalam kehidupan sekarang ini, sering kali informasi yang ada dikodekan agar tidak dibajak atau diambil oleh orang yang tidak berwewenang. Banyak cara untuk mengkodekan suatu pesan agar tidak dimengerti oleh orang lain kecuali orang yang akan menerima pesan tersebut. Salah satu pengkodean adalah pengacakan letak huruf pada pesan tersebut.

Pengacakan dari pesan tersebut menghasilkan sebuah anagram, yang memiliki jumlah tiap huruf sama dengan pesan sebelumnya. Untuk mengecek kesamaan tersebut terdapat beberapa algoritma yang bisa digunakan. Salah satunya adalah *brute force*, yang akan dibahas pada makalah ini.

II. METODE

Untuk memecahkan persoalan mengenai keanagraman suatu *string*, maka dibutuhkan suatu metode yang dapat membandingkan jumlah huruf yang ada pada kedua *string* tersebut. Metode yang paling mudah adalah metode *brute force*, yaitu dengan menghitung semua jumlah tiap huruf pada masing-masing *string*, kemudian membandingkan jumlah tiap hurufnya.

II.1. Algoritma *Brute force*

Brute force adalah sebuah pendekatan yang lempang (straightforward) untuk memecahkan suatu masalah (problem statement) dan definisi konsep yang dilibatkan Algoritma *brute force* memecahkan masalah dengan sangat sederhana, langsung, dan dengan cara yang jelas (obvious way) meskipun bukan merupakan solusi yang paling mangkus. Karakteristik Algoritma *Brute force*.

1. Algoritma *Brute force* umumnya tidak “cerdas” dan tidak mangkus karena ia membutuhkan jumlah langkah yang besar dalam penyelesaiannya. Kadang pula algoritma *Brute force* disebut juga algoritma naïf (naïve algorithm).
2. Algoritma *Brute force* lebih cocok untuk masalah yang berukuran kecil.
3. Meskipun bukan metode yang mangkus, hampir semua masalah dapat diselesaikan dengan algoritma *Brute force*.

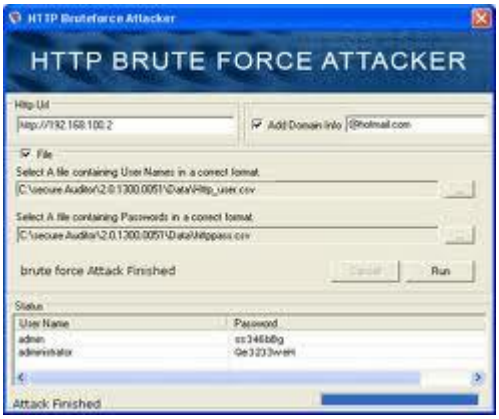
Kekuatan dan Kelemahan Metode *Brute force*
Kekuatan:

1. Metode *brute force* dapat digunakan untuk memecahkan hampir sebagian besar masalah (wide applicability).
2. Metode *brute force* sederhana dan mudah dimengerti.
3. Metode *brute force* menghasilkan algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan *string*, perkalian matriks.
4. Metode *brute force* menghasilkan algoritma baku (standard) untuk tugas-tugas komputasi seperti penjumlahan/perkalian n buah bilangan, menentukan elemen minimum atau maksimum di dalam tabel (list).

Kelemahan:

1. Metode *brute force* jarang menghasilkan algoritma yang mangkus.
2. Beberapa algoritma *brute force* lambat sehingga tidak dapat diterima.
3. Tidak sekontruktif/sekreatif teknik pemecahan masalah lainnya.

Dalam praktiknya, algoritma ini sering digunakan untuk teknik *hacking* atau *cracking*. Karena mencobai semua kemungkinan, maka bukan tidak mungkin sebuah password dapat dibobol. Kenneth Thompson pernah berkata, "When in doubt, use brute-force" (jika ragu, gunakan brute-force).



Gambar 1. Contoh Program *Brute Force*

II.2. Anagram

Menurut wordsmith.com anagram berarti :

[n] Sebuah kata atau frase yang merupakan pengaturan ulang huruf dari kata lain atau frase

[v] Untuk mengatur ulang huruf sedemikian rupa.

Kata angram sendiri diambil dari bahasa latin, yaitu anagrammatismos, ana- (naik, lagi, sebelum, baru) + -gram (kata).

Sedangkan menurut Wikipedia, anagram adalah salah satu jenis permainan kata yang huruf-huruf di kata awal biasanya diacak untuk membentuk kata lain atau sebuah kalimat.

Contoh anagram adalah Informatika = if rat, I'm an OK.

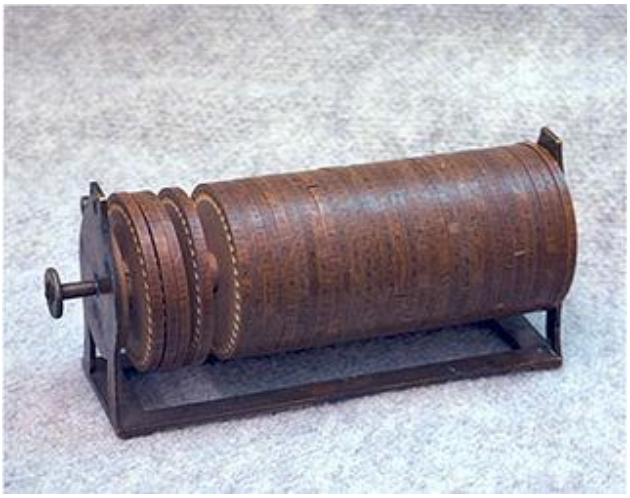
.


Gambar 2. Permainan *Scrabble*

Anagram diaplikasikan di permainan yang berbasiskan puzzle. Beberapa permainan yang menggunakan anagram adalah Anagrams, Scrabble, Countdown, Boggle, Brainteaser, Bananagrams. Inti dari kebanyakan permainan itu adalah membuat sebuah kata atau kalimat dari kumpulan huruf yang ada. Atau mengubah susunan huruf dari kata yang ada menjadi kata baru. Semakin panjang kata, maka skor yang didapat makin besar.

Penggunaan lain adalah sebagai *chipers.* Beberapa teknik penganagraman digunakan untuk menyelesaikan sebuah kriptograms, seperti *permutation cipher, transposition chiper, Jefferson disc.* Guna dari *chipers* ini adalah melindungi isi pesan yang akan dikirim, sehingga hanya penerima saja yang mampu menerjemahkan pesan yang dikirim. Pada *Jefferson disc,* digunakan sebuah alat yang terdiri dari beberapa keping piringan, tiap piringan ini memiliki kombinasi huruf yang berbeda-beda. Biasanya hanya beberapa baris saja yang dapat dibaca, juga kepingan-kepingan pirijgan tersebut sudah diurutkan.

Sedangkan dalam buku karangan Dan Brown, the Da Vinci Code, anagram dipakai untuk menyampaikan pesan kematian. Misalnya, “O, Draconian devil!” menjadi “Leonardo Da Vinci”. Atau dalam buku karangan JK Rowling, Harry Potter, anagram digunakan untuk menamakan karakter tertentu yang memiliki dua identitas. “Tom Marvolo Riddle” dan “I am Lord Voldemort”


Gambar 3. Jefferson disc

Di dunia Barat anagram dianggap permainan saja, sering untuk menyusun sebuah sinonim. Demikian misalnya Salvador Dali, pelukis Spanyol yang surrealis, menyusun kembali huruf-huruf namanya lalu terbaca "Avida Dollars" (menyukai dollar-dollar).Namun, dalam aliran-aliran mistik dari Timur Tengah dan Timur kombinasi huruf-huruf tertentu dianggap mempunyai arti magis. Demikian misalnya dalam ajaran rahasia untuk mencapai kesempurnaan. Baca misalnya Haryati Subadio, fnanasid dhanta (edisi Indonesia 1985).

Menurut de Saussure prinsip anagram merupakan dasar bagi teknik puisi dalam bahasa-bahasa Indo-Eropa. Dengan memilih atau menyusun kembali fonem-fonem atau pasangan fonem dapat disusun kata-kata yang penting, misalnya nama seorang dewa, rumus sebuah mantra rahasia dan sebagainya.

Dalam sementara kalangan teori sastra modern anagram dianggap sebuah kata yang membuka pengertian untuk menafsirkan sebuah teks menurut arti yang terdalam. Di bawah teks yang nampak tersembunyilah sebuah teks lain (hypoteks).

Di dunia maya, juga telah bertebaran *online solver* untuk anagram, antara lain :

- Free Online Anagram Solver (at Letterpress Helper)
- Anagram Solver (at Word Game Helper)
- Xavier's Anagram Solver
- Andy's Anagram solver

- Anagram Genius : Anagram Server
- Internet Anagram Server / I, Rearrangement Servant
- Free Scrabble Dictionary

III. PENERAPAN ALGORITMA

Pada makalah ini akan diterapkan 2 macam algoritma yang berbasis *brute force.* Asumsi yang digunakan adalah panjang dari *string* pasti sama, sehingga tidak dicek panjang *string* pada algoritma yang akan digunakan. Algoritma yang pertama saya namakan ***Counter***. Sedangkan algoritma yang kedua saya namakan ***Selection*.**

Untuk algoritma yang pertama, diterapkan perhitungan jumlah tiap karakter yang ada pada *string* yang pertama dan *string* yang kedua. Hasil dari perhitungan ini dibandingkan, jika sama, maka kedua *string* tersebut adalah anagram.

```
function counterAlg(array of char string1,array
of char string2) -> boolean
{fungsi ini mengembalikan true jika string1
adalah anagram dari string2.
Mengembalikan false jika bukan anagram.
Menggunakan algoritma conter}

KAMUS
Map<char,integer> charCounter1
Map<char,integer> charCounter2

ALGORITMA
{menghitung jumlah tiap karakter, kemudian
dimasukkan ke map (untuk string1)}
i traversal [0..String1.length-1]
 if(charCounter1.containsKey(String1[i])) then
   charCounter1.put(
          String1[i],
          charCounter1.get(String1[i]) + 1)
 else charCounter1.put(String1[i], 1)
{ algoritma yang sama untuk string2
... ... ...
}

{Membandingkan hasil kemudian return}
→ charCounter1.equals(charCounter2)
```

Untuk menyatakan bahwa kedua *map* sama(*equals)*, digunakan dua syarat. Yang pertama, ukuran kedua map sama. Yang kedua, kedua map tersebut memiliki *entry* yang sama. Yang dimaksud dengan *entry* adalah pasangan *key* dan *value*. Pengecekan dilakukan dengan looping untuk melihat kesaamaan *entry*.

Untuk algoritma yang kedua dimulai dengan mengiterasi satu per satu karakter yang terdapat di *string* pertama. Kemudian mencari karakter tersebut di *string* kedua. Jika ada, maka karakter tersebut dihapus dari *string* kedua. Jika tidak ada, maka kedua *string* tersebut bukanlah anagram. Algoritma ini berhenti jika tiap karakter *string* pertama sudah ditelusuri.

```
function selectionAlg(array of char
string1,array of char string2) -> boolean
{fungsi ini mengembalikan true jika string1
adalah anagram dari string2.
Mengembalikan false jika bukan anagram.
Menggunakan algoritma selection}

KAMUS
boolean isAnagram, isContain
integer i1Cek, i2Cek

ALGORITMA
isAnagram ← true
i1Cek ← 0

while (isAnagram AND i1Cek < String1.length)
i2Cek ← 0
  isContain ← false
  while (not(isContain) AND
   i2Cek < String2.length)
    if (String1[i1Cek] = String2[i2Cek]) then
      isContain ← true
      String2 ← DeleteChar(i2Cek, String2)
    else
      i2Cek ← i2Cek + 1
  if (not(isContain))
    isAnagram ← false
    i1Cek ← i1Cek + 1
```

Untuk algoritma DeleteChar, digunakan dasar algoritma *deep copying,* yaitu teknik *copy string* yang meng-*assign* satu per satu karakter ke alamat memori *string* yang baru. Algoritma ini dimodifikasi di beberapa bagian, yaitu mengurangi panjang *string* baru sebanyak 1 dari *string* lama, dan tidak memasukkan karakter tertentu ke *string* yang baru. *String* baru ini adalah kembalian dari fungsi DeleteChar.

```
function DeleteChar(integer charPos,
array of char stringToDel) → array of
char

KAMUS
integer i
array of char
       newString[stringToDel.length -
1]

ALGORITMA
i traversal [0..charPos-1]
  newString[i] = stringToDel[i]
i traversal
      [charPos..stringToDel.length – 1]
  newString[i] = stringToDel[i + 1]

→ newString
```

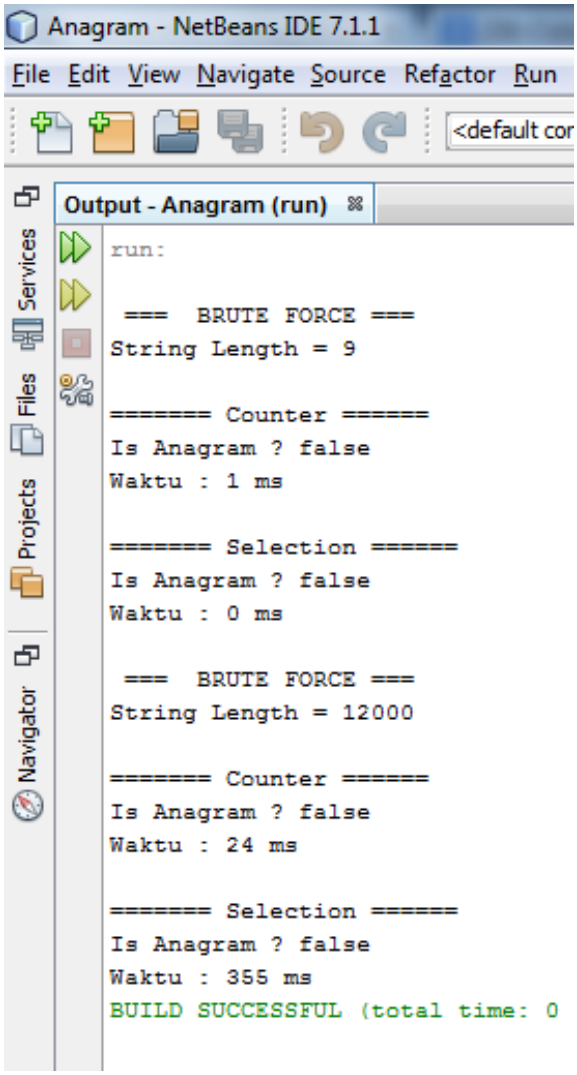
Untuk menguji kedua algoritma ini, maka dibuatlah sebuah program sederhana. Implementasinya menggunakan bahasa Java. Kemudian menggunakan kakas bantu IDE Netbeans 7.1.1.

Dari hasil yang didapat, kedua algoritma ini memiliki kelebihan dan kelemahan masing-masing. Untuk algoritma yang pertama, **counter**, baik digunakan untuk *string* yang sangat panjang. Untuk *string* sepanjang 12000 karakter, yang dibutuhkan

waktu 24-25 ms. Sedangkan untuk algoritma yang kedua, **selection**, dibutuhkan waktu 355-370 ms.

Untuk *string* yang pendek, misalnya satu atau dua kata, hasil yang didapatkan bahwa algoritma **selection** lebih baik, yakni tidak sampai 1 ms. Sedangkan untuk **counter** dibutuhkan lebih dari 1 ms. Walaupun tidak terlalu signifikan, jelas terlihat masing-masing algoritma mempunyai tujuan yang berbeda.

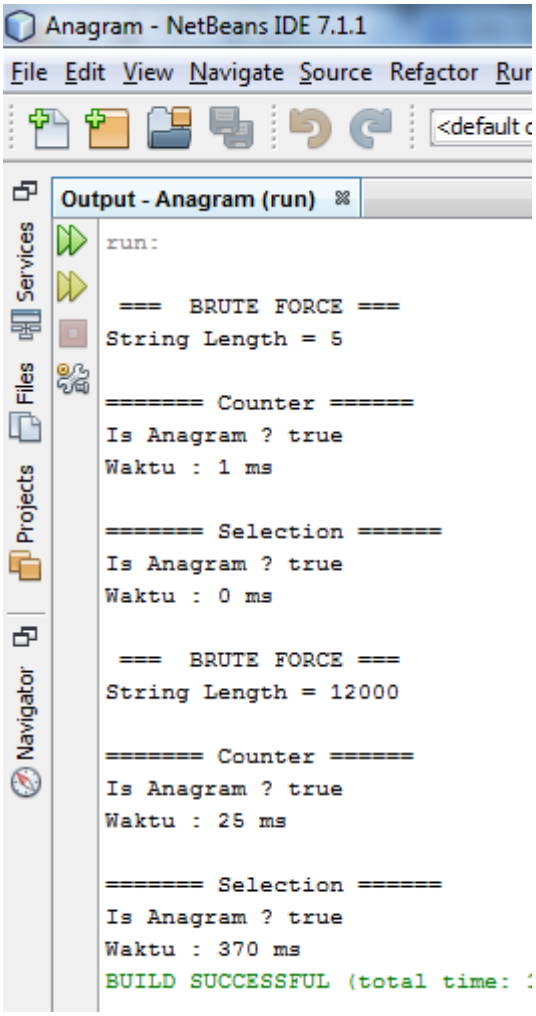
Kedua algoritma ini juga berbeda pada proses berhentinya. Jika algoritma **counter** berhenti pada saat perbandingan mapnya, algoritma **selection** berhenti pada saat ditemukannya perbedaan jumlah huruf pada antara *string* pertama dan *string* kedua.



Gambar 4. Hasil Perbandingan dari kedua algoritma, jika hasilnya salah

Yang membuat proses algoritma **counter** lama, adalah proses pembuatan map untuk menghitung jumlah karakter, serta pemasukan nilainya ke map. Namun ini menjadi kelebihan jika *string* sangat panjang, karena hanya dipanggil satu kali saja.

Yang membuat proses algoritma **selection** lama, adalah proses penghapusan karakter dari *string* kedua. Karena menggunakan *deep copy*., Jika *string* sangat panjang proses ini cukup memakan waktu, karena dilakukan berkali-kali.



Gambar 5. Hasil perbandingan dari kedua algoritma, jika hasilnya benar

Kedua algoritma ini masih bisa diperbaiki lagi sehingga hasil yang didapat menjadi lebih cepat. Untuk algoritma **counter**, bisa dilakukan multithreding, yakni sembari menghitung jumlah huruf, langsung dibandingkan, daripada harus dihitung dahulu, baru dibandingkan.

Untuk algoritma **selection** penghapusan karakter didesain lebih baik lagi dengan cara menghapus cara *deep copy*. Lebih baik digunakan manipulasi string langsung tanpa harus memproses keseluruhan dari string tersebut.

V. KESIMPULAN

Algoritma *Brute Force* dapat digunakan untuk menentukan keanagraman dari kedua buah *string*. Dalam aplikasinya, algoritma *brute force* terdapat beberapa macam.

Algoritma **counter**, memiliki kelebihan membandingkan *string* yang sangat panjang.

Algoritma **selection,** memiliki kelebihan membandingkan *string* yang pendek.

REFERENSI

[1] http://id.wikipedia.org/wiki/Anagram. Diakses 12/12/2012 16:42
[2] http://en.wikipedia.org/wiki/Anagram. Diakses 12/12/2012 17:00
[3] http://www.wordsmith.org. Diakses 17/12/2012 16:24
[4] http://id.wikipedia.org/wiki/Serangan_brutal. Diakses 17/12/2012 16:25
[5] http://en.wikipedia.org/wiki/Brute-force_attack. Diakses 17/12/2012 16:26
[6] Cited in Henry Benjamin Wheatley, Of anagrams: a monograph treating of their history (1862)
[7] http://myindoliterature.blogspot.com/2011/05/anagram.html. Diakses 17/12/2012 16:59
[8] Henry Benjamin Wheatley, On Anagrams (1862) , hal. 58.
[9] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/. Diakses 17/12/2012 17:00

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, 17 Desember 2012

A. Bara Timur
13510019