Perbandingan Implementasi Algoritma KMP dan Karp-Rabin GST dalam *Multiple String-Matching* untuk Nilai Hash

Puanta Della Maharani - 13507135
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if17135@students.if.itb.ac.id

Fungsi hash, di bidang kriptografi, adalah fungsi yang memiliki beberapa sifat keamanan tambahan sehingga dapat dipakai untuk tujuan keamanan data. Hashing sering juga dipakai sebagai acuan untuk meringkas teks menjadi lebih pendek. Permasalahan muncul ketika nilai hash yang dihasilkan ternyata cukup panjang dan memiliki perbedaan yang sangat kecil saat dibandingkan dengan yang lain.

Pada makalah ini, akan dibandingkan implementasi algoritma string-matching yang dipakai, yaitu Knuth Morris Pratt (KMP) dan Karp-Rabin Greedy String Tiling. Kedua algoritma ini dikenal sebagai algoritma string matching dengan kompleksitas paling kecil.

Kata kunci: Hash, KMP, Karp-Rabin Greedy String Tiling.

I. Pendahuluan

Dalam berkomunikasi, tidak semua informasi dapat diketahui oleh orang selain pengirim dan penerima informasi. Agar informasi tidak langsung diketahui, pesan yang dikirim harus dienkripsi terlebih dahulu. Terdapat banyak cara dan metode dalam melakukan enkripsi. Enkripsi biasanya menggunakan kunci untuk mendekripsi ciphertext menjadi plaintextnya. Seiring berkembangnya zaman, muncul pula teknik menyembunyikan bagian pesan tanpa menggunakan kunci. Teknik ini disebut dengan *Hashing* menggunakan fungsi Hash.

Fungsi hash, di bidang kriptografi, adalah fungsi yang memiliki beberapa sifat keamanan tambahan sehingga dapat dipakai untuk tujuan keamanan data. Umumnya digunakan untuk keperluan autentikasi dan integritas data. Fungsi hash adalah fungsi yang secara efisien mengubah string input dengan panjang berhingga menjadi string output dengan panjang tetap yang disebut nilai hash. Contoh fungsi hash yang sering dipergunakan adalah MD4, MD5, SHA-0, SHA-1, SHA-256, dan lain-lain. Hashing digunakan untuk menyembunyikan isi dari suatu teks yang berukuran besar dengan menerjemahkannya ke dalam nilai hash. Nilai hash ini yang nantinya akan menjadi acuan dari suatu teks yang diragukan keaslian. Caranya, nilai hash teks yang diragukan akan dibandingkan dengan nilai hash teks yang asli. Nilai hash yang relatif lebih pendek dari teks cenderung lebih efisien untuk diuji kesamaannya dibandingkan teks asli yang relatif lebih panjang.

Hashing sering juga dipakai sebagai acuan untuk meringkas teks menjadi lebih pendek. Peringkasan ini bisa menjadi awal mulanya pemrosesan suatu bahasa pada teks tertulis. Teks dengan ukuran panjang dapat diringkas menjadi nilai hash yang berisi sekumpulan kode yang merepresentasikan materi dan isi teks tersebut.

Permasalahan muncul ketika nilai hash yang dihasilkan ternyata cukup panjang dan memiliki perbedaan yang sangat kecil saat dibandingkan dengan yang lain. Bagi pengguna teknik hashing yang merepresentasikan isi dan materi teksnya, perbedaan nilai dapat merujuk ke bagian yang dianggap sudah tidak sama seperti teks asli. Oleh kerena itu, pada makalah ini, saya mencoba untuk mengimplementasi algoritma string-matching untuk nilai hash untuk melihat kesamaan antara nilai-nilai hash.

Pada makalah ini, akan dibandingkan pula implementasi algoritma string-matching yang dipakai. Adapun algoritmanya adalah Knuth Morris Pratt (KMP) dan Karp-Rabin GST. Kedua algoritma ini dikenal sebagai algoritma string matching dengan kompleksitas paling kecil. Mari kita lihat bagaimana proses perbandingan dan hasil implementasinya.

II. Landasan Teori

A. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth Morris Pratt (KMP) dikembangkan oleh D. E. Knuth, bersama dengan J. H. Morris dan V. R. Pratt. Untuk pencarian string dengan algoritma Brute Force, setiap kali ditemukan ketidakcocokan dengan teks, maka pattern akan digeser satu karakter ke kanan. Berbeda dengan algoritma Brute Force, informasi yang digunakan masih dipelihara untuk melakukan jumlah pergeseran. Algoritma menggunakan

informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karakter Perbandingan antara algoritma *brute force* dengan algoritma KMP ditunjukkan dengan perpindahan posisi pattern terhadap posisi teks. Dimana dalam hal pencocokan string, pattern yang dicocokkan berawal dari kiri teks.

Kompleksitas algoritma pencocokan string dihitung dari jumlah operasi perbandingan yang dilakukan. Kompleksitas waktu terbaik dari algoritma brute force adalah $O(n)$. Kasus terbaik terjadi jika pada operasi perbandingan, setiap huruf pattern yang dicocokkan dengan awal dari teks adalah sama. Kompleksitas waktu terburuk dari *brute force* adalah $O(mn)$.

Secara sistematis, langkah-langkah yang dilakukan algoritma Knuth-Morris-Pratt pada saat mencocokkan string:

1. Algoritma Knuth-Morris-Pratt mulai mencocokkan pattern pada awal teks.
2. Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
   a. Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
   b. Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian menggeser pattern berdasarkan tabel next, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

Algoritma ini menemukan semua kemunculan dari pattern dengan panjang n di dalam teks dengan panjang m dengan kompleksitas waktu O(m+n). Algoritma ini hanya membutuhkan O(n) ruang dari memory internal jika teks dibaca dari file eksternal. Semua besaran O tersebut tidak tergantung pada besarnya ruang alfabet.

Berikut ini merupakan pseudocode untuk preproses dan algoritma KMP.

```
procedure preKMP(
    input P : array[0..n-1] of char,
    input n : integer,
    input/output kmpNext : array[0..n] of integer
)
Deklarasi:
i,j: integer

Algoritma
  i := 0;
  j := kmpNext[0] := -1;
  while (i < n) {
     while (j > -1 and not(P[i] = P[j]))
        j := kmpNext[j];
     i:= i+1;
     j:= j+1;
     if (P[i] = P[j])
        kmpNext[i] := kmpNext[j];
     else
        kmpNext[i] := j;
     endif
  endwhile
```

Gambar 1. Pseudocode preproses algoritma KMP

```
procedure KMPSearch(
    input m, n : integer,
    input P : array[0..n-1] of char,
    input T : array[0..m-1] of char,
    output ketemu : array[0..m-1] of boolean
)
Deklarasi:
i, j,next: integer
kmpNext : array[0..n] of interger
Algoritma:
preKMP(n, P, kmpNext)
```

```
i:=0
while (i<= m-n) do
    j:=0
    while (j < n and T[i+j] = P[j]) do
       j:=j+1
    endwhile
    if(j >= n) then
       ketemu[i]:=true;
    endif
    next:= j - kmpNext[j]
    i:= i+next
endwhile
```

Gambar 2. Pseudocode algoritma KMP

B. Algoritma Karp-Rabin GST

Algoritma Running Karp-Rabin Greedy String Tiling pertama kali dipakai pada sistem Neweyes, yaitu sebuah sistem untuk alignment biosequences nukleotida dan asam amino Algoritma ini merupakan pengembangan lebih lanjut dari algoritma Greedy String Tiling dengan penerapan algoritma Karp-Rabin pada fase scanpattern.

Algoritma Karp-Rabin diajukan oleh Richard M. Karp dan Michael O. Rabin dalam literaturnya yang berjudul "Efficient Randomized Pattern-Matching Algorithms". Algoritma tersebut dapat menangani masalah string matching sebagai berikut: Untuk sebuah set pasangan string {(X(i),Y(i))} terdefinisi, jika memungkinkan, tentukan r yang memenuhi X(r)=Y(r). Pada algoritma yang diajukan Karp dan Rabin, suatu nilai fingerprint dihitung untuk setiap string. Nilai fingerprint tersebut lebih pendek dari string yang bersangkutan. Yang kemudian akan dibandingkan bukanlah string itu sendiri, melainkan nilai fingerprint yang telah dihitung. Perhitungan nilai fingerprint untuk pembandingan itulah yang diterapkan pada algoritma Running Karp-Rabin Greedy String Tiling ini.

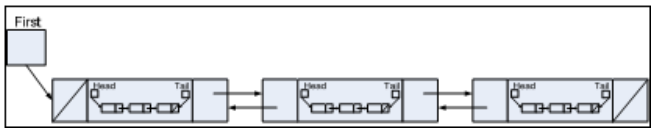
Penerapan algoritma Karp-Rabin pada fase scanpattern algoritma GST dijelaskan berikut ini. Jika |P| merupakan panjang salah satu substring pada pattern string P, maka nilai hash untuk setiap substring yang panjangnya |P| pada text string akan dibandingkan dengan nilai hash untuk substring dari pattern P tersebut. Jika kedua nilai hash tersebut identik, maka substring dari pattern P dan substring dari text string tersebut akan dibandingkan per elemen.

Untuk lebih detilnya, algoritma Running Karp-Rabin diuraikan per tahap sebagai berikut:

1. Untuk setiap substring dari pattern string yang mempunyai panjang s dan belum ditandai, nilai hash dihitung. Misalnya untuk substring P[p..p+s-1], dengan p antara 1 sampai |P|-s. Nilai hash juga dibuat untuk setiap substring yang panjangnya s dan belum ditandai pada text string.
2. Sebuah hashtable digunakan untuk menurunkan cost O(n2) pembandingan. Hashtable tersebut menyimpan nilai hash setiap substring pada text string dan posisi token pertama pada setiap substring tersebut.
3. Setiap nilai hash untuk pattern string dibandingkan dengan nilai hash pada text string. Untuk nilai hash yang sama, terdapat kemungkinan adanya kesamaan antara substring yang bersangkutan. Setelah ditemukan nilai hash yang sama, pembandingan per token dilakukan untuk tokens pada posisi berikutnya seperti pada algoritma GST. Pasangan substring yang sama tersebut juga akan dikonversi menjadi maximal-matches seperti pada algoritma GST.

Panjang pasangan yang dicari (disebut sebagai search-length s) dikurangi pada setiap akhir iterasi sampai minimum-match-length dicapai. Algoritma fase scanpattern pada RKR-GST mempunyai sebuah parameter yaitu search-length s.

Struktur yang digunakan untuk menyimpan maximal-matches adalah sebuah double-linked-list of queues. Maximal-matches yang panjangnya sama disimpan dalam sebuah queue. *List of queues* diurutkan berdasarkan panjang maximal-matches yang disimpan dengan urutan mengecil.



Gambar 3. Data Double Linked-list of Queues
untuk Menyimpan Maximal-matches pada Algoritma Karp-Rabin GST

Salah satu aspek yang penting adalah menentukan nilai yang tepat untuk parameter s yang merupakan search-length. Nilai yang lebih kecil dari setengah panjang P sudah cukup. Alasannya karena maximal-matches yang sangat panjang jarang ditemukan, sehingga secara umum nilai inisialisasi untuk s akan menghasilkan beberapa pass kosong pada scanpattern sampai sebuah match ditemukan. Alasan kedua yaitu jika maximal-match yang panjang, penciptaan tile dari string ini akan membutuhkan banyak token dari pattern string dan text string. Karena itu, scan harus dihentikan dan dimulai kembali dengan nilai s yang lebih besar yaitu sama dengan ukuran maximal-match panjang tersebut. Hal ini menunjukkan bahwa nilai s bisa diinisialisasi dengan nilai konstan yang kecil (misalnya s bernilai 20 pada contoh) daripada bergantung pada panjang string.

Algoritma untuk fase markstrings sama dengan yang digunakan pada GST yang telah di-tuning namun maximal-match dibaca dari list of queues dan mempunyai parameter search-length s. Semua pasangan yang dihasilkan dari proses hashing akan diuji per elemen sebab kesamaan nilai hash tidak menjamin bahwa substring yang berkorespondensi akan sama. Hal ini karena sudah diketahui dari hasil pengujian bahwa KR-hashing jarang sekali gagal, sehingga pengujian per komponen akan lebih efisien jika ditempatkan di fase markstrings.

Kompleksitas kasus terburuk penggunaan algoritma ini yaitu O(n3) dengan n merupakan panjang string input. Namun telah diuji bahwa kondisi yang dapat menyebabkan kasus terburuk hampir tidak mungkin terjadi. Dengan minimum-match-length yang bernilai 3, kompleksitas algoritma yang dihitung pada pengujian tersebut yaitu O(n0.90), hampir linier. Secara umum, estimasi kompleksitas algoritma yang sering muncul pada prakteknya adalah antara O(n) sampai O(n2).

Berikut ini adalah pseudocode untuk preproses markstring dan algoritma Karp-Rabin GST.

```
procedure markstrings (P[0..M],T[0..N] : tokenString, s: integer)
{membandingkan setiap token pada maximal-matches yang telah didapatkan pada
fase scanpattern}
{token string P[1..M] merupakan token string yang lebih pendek dibandingkan
dengan token string T[1..N]}
KAMUS
  p: integer {posisi token pada pattern string}
  t: integer {posisi token pada text string}
  maxmatch_list : double-linked list of queues of match
                  {List berisi maximal-match}
  L: integer {ukuran maximal-match pada queue di iterasi tersebut}
ALGORITMA
   starting with the first queue in maxmatch_list,
   while there is a non-empty queue do
      if the current queue is empty then
          drop to next queue {maximal-match dengan ukuran lebih kecil}
      else
          if all tokens in match(p,t,L) are unmarked then
             if P[p+j]=T[t+j] for all j←0 to s-1 then
                for j←0 to L-1 do
                   mark_token(P[p+j])
                   mark_token(T[t+j])
                length_of_tokens_tiled ← length_of_tokens_tiled + L
          else if (L - length of marked tokens in match(p,t,L)) ≥ s then
             add unmarked portion to maxmatch_list
          delete match(p,t,L) from queue
```

Gambar 4. Fungsi markstrings(s) pada Algoritma RKR-GST

```
procedure RunningKarpRabinGreedyStringTiling(P[0..M],T[0..N] : tokenString,
minimum_match_length : integer)
{top-level algorithm untuk Running Karp-Rabin Greedy String Tiling}
{token string P[1..M] merupakan token string yang lebih pendek dibandingkan
dengan token string T[1..N]}
{mencari substring yang bernilai sama dan panjangnya lebih besar dari
minimum_match_length}
KAMUS
   s : integer {search-length}
   stop : boolean
   Lmax : integer
ALGORITMA
   s ← 20 {inisialisasi search-length}
   stop ← false
   repeat
      Lmax ← scanpattern(s) {the size of largest maximal-matches}
                            {found in this iteration}
      if Lmax>(2*s) then  {Very long string}
         s ← Lmax  {don't mark tiles but try again with larger s}
      else
         markstrings(s) {Create tiles from matches takes from list of queues}
         if s>(2*minimum_match_length) then
            s ← s div 2
         else if s > minimum_match_length then
            s ← minimum_match_length
         else
            stop ← true
   until stop
```

Gambar 5. Top-level Algorithm untuk RKR-GST

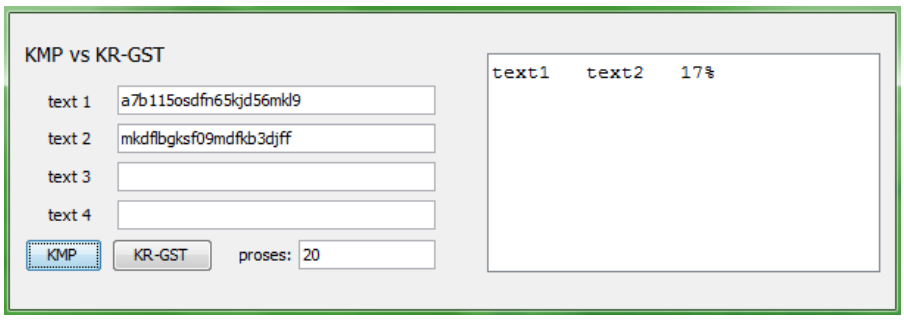
III. IMPLEMENTASI ALGORITMA KMP DAN KARP-RABIN GST

Kedua algoritma diimplementasikan ke dalam suatu aplikasi desktop. Pada aplikasi ini, dapat dilihat di sebelah kiri terdapat 4 buah kotak penulisan teks untuk hash yang akan dibandingkan. Pengguna harus mengisi minimal 2 di antaranya. Hash yang dibandingkan harus kurang dari 150 karakter. Di bawahnya terdapat pilihan algoritma yang akan digunakan. Ada pilihan KMP untuk algoritma Knuth Morris Pratt dan KR-GST untuk Karp Rabin Greedy String Tiling. Di sebelah pilihan algoritma terdapat kotak proses. Nantinya jumlah proses yang dilakukan akan terlihat di kotak ini. Di sebelah kanan, terdapat kotak teks yang akan menampilkan hasil perbandingan oleh algoritma yang telah dipilih. Hasil perbandingan akan ditunjukkan dengan persentase kemiripan.

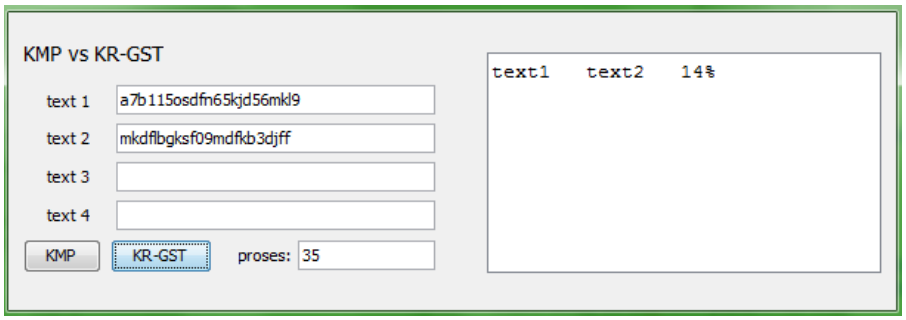
Gambar 6. Aplikasi

Untuk mencoba jumlah teks minimal yang dimasukkan, kita bisa mencoba memasukkan nilai hash seperti pada table ini.

Teks 1	a7b115osdfn65kjd56mkl9
Teks 2	mkdflbgksf09mdfkb3djff
Teks 3	-
Teks 4	-



Gambar 7. Hasil perbandingan dengan KMP

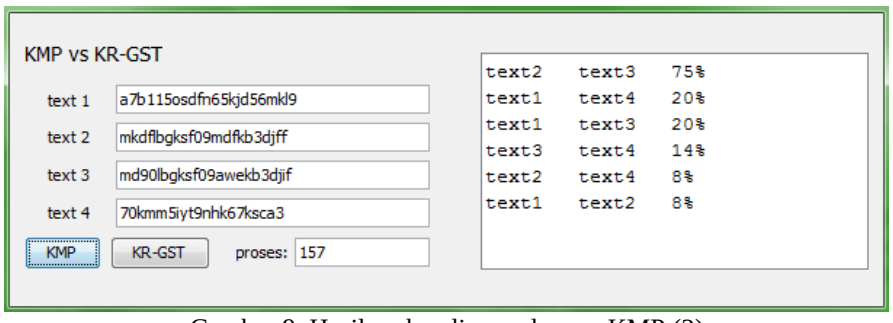


Gambar 8. Hasil perbandingan dengan KR-GST

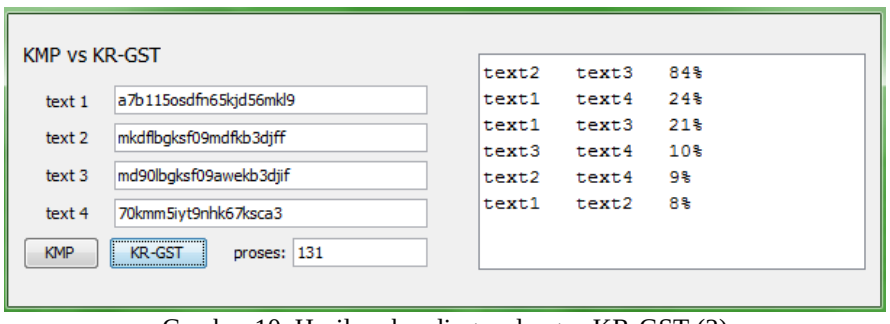
Setelah dibandingkan dengan dua algoritma yang tersebut, kita bisa melihat hasilnya bahwa KMP mengeluarkan nilai 17% dan KR-GST mengeluarkan nilai 14%. Proses yang dilakukan juga berbeda, yaitu 20 proses dengan algoritma KMP dan 35 proses dengan algoritma KR-GST.

Selanjutnya, kita akan mencoba membandingkan 4 buah nilai hash sekaligus. Nilai hash yang akan dibandingkan adalah sebagai berikut.

Teks 1	a7b115osdfn65kjd56mkl9
Teks 2	mkdflbgksf09opfkb3djff
Teks 3	md90lbgksf09awekb3djif
Teks 4	70kmm5iyt9nhk67ksca3



Gambar 9. Hasil perbandingan dengan KMP (2)



Gambar 10. Hasil perbandingan dengan KR-GST (2)

IV. Hasil dan Analisis

Algoritma KMP memiliki kompleksitas pembanding string yang lebih sederhana dan ringkas sehingga proses yang dilalui pun lebih sedikit. Algoritma ini mengalami sedikit inovasi dari algoritma brute-force sehingga tetap mendapat keuntungan, yaitu algoritma yang sederhana. Angka kesamaan yang lebih rendah dari algoritma KR-GST dikarenakan bilangan pembagi yang lebih besar yaitu keseluruhan string yang dimiliki.

Algoritma KR-GST memiliki kompleksitas yang rumit dalam membandingkan string dikarenakan kewajiban untuk mengisi tile dan pattern terlebih dahulu dalam membandingkan string. Akan tetapi, proses yang dilalui saat string yang dibandingkan berjumlah lebih dari 2 berjumlah lebih sedikit. Hal ini dikarenakan variable maximal-match yang menyimpan string dengan kesamaan tertinggi sehingga tidak perlu banyak proses dalam membandingkan 4 buah string.

V. Kesimpulan

Teknik hashing yang merepresentasikan isi dan materi teks yang diringkas memerlukan algoritma pembanding yang dapat menilai kesamaan antara dua buah nilai hash atau lebih.

Setelah mengimplementasi algoritma KR-GST dan KMP, dapat disimpulkan bahwa KMP lebih unggul untuk string-matching antara 2 buah nilai hash karena kompleksitas pembandingnya yang lebih sederhana sedangkan KR-GST lebih unggul untuk string-matching nilai hash yang berjumlah lebih dari 2 buah karena memiliki variable maximal-match yang menyimpan bagian dari nilai hash yang memiliki kesamaan paling besar.

Oleh karena itu, kedua algoritma ini disimpulkan cukup baik dalam perbandingan string-matching nilai hash. Pengguna diharapkan lebih memilih algoritma yang lebih sesuai dengan kondisi nilai hash yang dihadapi.

References

Munir, Rinaldi. 2011. "Bahan Kuliah IF3054 Kriptografi". Departemen Teknik Informatika, Institut Teknologi Bandung

http://id.wikipedia.org/wiki/Kriptografi
waktu akses: 10 Mei 2012 pukul 17.00 WIB

http://id.wikipedia.org/wiki/Algoritma_Knuth-Morris-Pratt
waktu akses: 10 Mei 2012 pukul 17.00 WIB

http://en.wikipedia.org/wiki/Rabin-karp
waktu akses: 10 Mei 2012 pukul 17.00 WIB

http://undip.ac.id/31793/
waktu akses: 10 Mei 2012 pukul 17.00 WIB

http://contohprogramsourcecodemetodealgoritma.com/2011/11/15/tutorial-algoritma-efisien-rabin-karp-contoh-program-source-code/
waktu akses: 10 Mei 2012 pukul 17.00 WIB

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, 12 Mei 2012

Puanta Della Maharani
13507135