Aplikasi *Boyer-Moore String* Search dalam Mesin Pencari Lokal

Kevin Supendi 13514094
Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13514094@std.stei.itb.ac.id

***Abstract*—Mesin Pencari atau *Search Engine*, *online* atau pun local, sangat membantu dalam mencari informasi. Setiap sistem operasi sudah dilengkapi dengan fitur *Desktop Search*, untuk mencari *file-file* yang tersimpan dalam disk. Algoritma *Boyer-Moore* bisa diaplikasikan untuk mencocokkan nama *folder* dan *file* dengan *keyword* yang dimasukkan pengguna. Hasil pencocokkan berupa *path* dari *folder* dan *file* yang mempunyai *keyword* tersebut.**

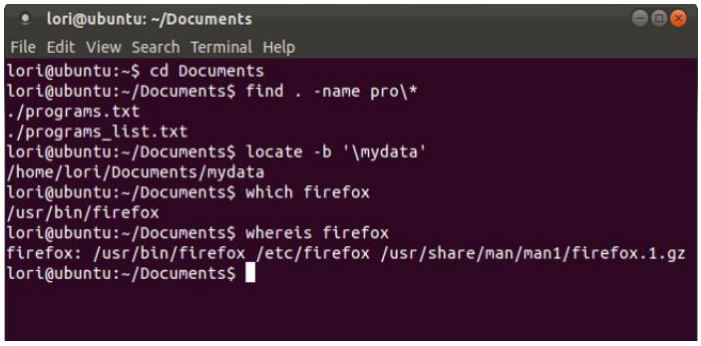
Keywords—Desktop Search, Boyer-Moore String Matching, Keyword

I. PENDAHULUAN

Desktop Search, berbeda dengan istilah *Search* Engine, yang biasa mengacu pada pencarian pada konten *web*, *Desktop Search* berarti pencarian terhadap *file* yang terdapat dalam satu komputer. Hal ini sangat berguna bagi pengguna untuk mencari *file* lama yang terkubur dalam *folder*, atau untuk menemukan di mana letak *folder* suatu aplikasi. Fitur *Desktop Search* sudah digunakan sejak dulu sampai sekarang dalam setiap sistem operasi.

Pada sistem operasi Windows 10 misalnya, Start Menu ditemani oleh *Cortana*, yang bisa mencari untuk local files maupun yang ada di web, dan banyak fitur lainnya.

Hal serupa juga bisa ditemui di sistem operasi lain seperti Android, iOS, MacOS, Linux, ChromeOS. Fitur *Search* juga bisa dijalankan dari *command line*, misalnya command *find* pada sistem Linux.



Gambar 1 : Contoh *command* pencarian pada linux

Sumber : http://cdn.howtogeek.com/wp-content/uploads/2012/04/00_different_commands_for_finding_items.png

Pencarian *file* tentunya dapat lebih optimal jika menggunakan algoritma *Boyer-Moore*. Dengan mencocokkan nama path sebagai String, program pencari bisa menjelajah seluruh direktori secara rekursif dan menemukan file yang cocok dengan *keyword* yang diberikan.

Penulis ingin mencoba membuat suatu program untuk menerima input sebuah *keyword*, yang kemudian dicocokkan dengan semua file yang ada di seluruh disk. Kebanyakkan program hanya mencari *best results* saja sehingga terkadang hasil pencarian kurang memuaskan. Penulis ingin agar program yang dibuat menuliskan hasil pencarian yang *complete* dari seluruh *logical hard drive* (C:\, D:\, dst.).

II. DASAR TEORI

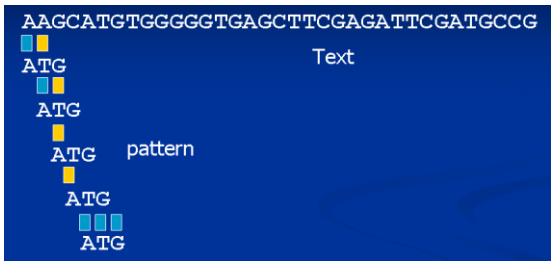
2.1 Algoritma String Matching

Algoritma String Matching adalah bagian dari algoritma pemrosesan *String*. Hal ini menjadi kebutuhan karena banyak data di dunia nyata bukan dalam bentuk numerik, tetapi merupakan kombinasi dari karakter dan symbol juga. Algoritma *String Matching* berarti mencocokkan suatu *pattern*, suatu *String* dengan panjang n, dan suatu *text*, biasanya *String* dengan panjang lebih dari n. Inti dari *String Matching* adalah untuk

mencari apakah terdapat suatu subset berupa *pattern* dari suatu *text* sembarang.

1) Naïve or Brute Force Algorithm

Sesuai dengan namanya, algoritma ini mencocokkan string dengan cara yang naif. Setiap karakter pada *pattern* dicocokkan satu per satu. Jika ada kesalahan, maka pencarian akan diulang dengan indeks awal pencarian ditambah dengan satu.



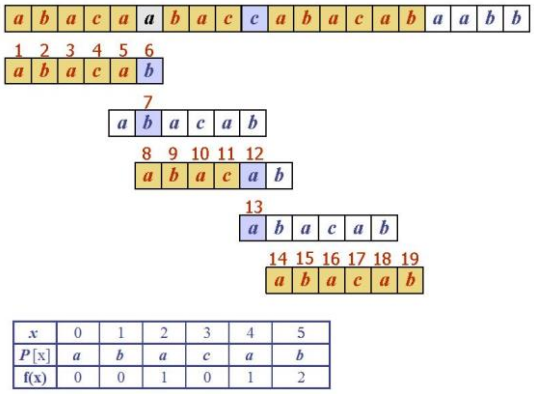
Gambar 2 : Contoh *Brute Force String Matching*, saat ditemukan kesalahan pencarian akan diulang dari karakter pertama pada *pattern*.

Sumber :
http://compbio.chemistry.uq.edu.au/mediawiki/upload/f/f4/Brute_force_schema.png

Algoritma *Brute Force* ini cepat untuk teks dengan jumlah simbol yang banyak. Algoritma ini menjadi lambat untuk file dengan symbol yang sedikit, contohnya biner dan susunan rantai protein.

2) Knuth-Morris-Pratt Algorithm

Serupa dengan algoritma *Brute Force*, algoritma KMP ini mencocokkan setiap karakter secara sekuensial. Perbedaannya adalah saat ditemukan kecocokkan, algoritma KMP tidak kembali ke indeks awal pencarian, tetapi melanjutkan dari indeks ditemukan kesalahan itu, lalu menggeser karakter yang akan dibandingkan sesuai fungsi pinggiran.



x	0	1	2	3	4	5
P[x]	a	b	a	c	a	b
f(x)	0	0	1	0	1	2

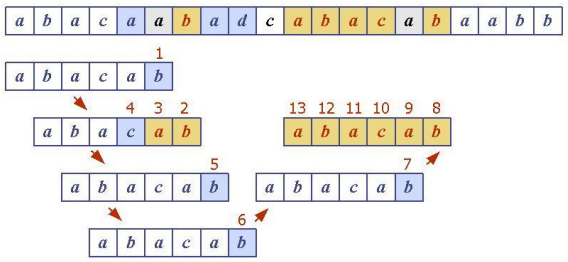
Gambar 3 : Contoh Algoritma KMP, pencarian berlanjut di indeks ditemukannya kesalahan, dan menggeser karakter yang dibandingkan sesuai dengan fungsi pinggiran f(x).
Sumber :
https://koding4fun.files.wordpress.com/2010/05/kmpexample.jpg

Algoritma KMP jauh lebih cepat dibandingkan *Brute Force*, dan lebih baik untuk *String* dengan jumlah simbol yang sedikit.

3) Boyer-Moore Algorithm

Algoritma *Boyer-Moore* memanfaatkan *looking glass technique*. Jika karakter belakang *pattern* sudah tidak sesuai dengan teks, maka karakter bagian depan sudah tidak perlu dicek lagi karena akan tetap salah. Algoritma *Boyer-Moore* mengecek mulai dari karakter bagian belakang, lalu jika ditemukan ketidakcocokkan, maka indeks pencarian bisa berpindah sesuai dengan kondisi. Ada tiga kondisi yang mungkin :

1. Karakter yang tidak cocok tidak terdapat pada *pattern*, maka *pattern* akan dipindahkan melewati karakter itu.
2. Karakter yang tidak cocok terdapat pada *pattern*, dan di sebelah kanan indeks *pattern* yang salah. *Pattern* digeser menyesuaikan karakter itu.
3. Karakter yang tidak cocok terdapat pada *pattern*, tetapi tidak memungkinkan untuk menyesuaikan dengan *pattern* karena indeks karakter ada di sebelah kiri *pattern*. Maka *pattern* digeser ke kanan sekali.



Gambar 4 :Contoh Algoritma *Boyer-Moore*, kondisi 1 terlihat pada langkah 6, kondisi 2 ada pada langkah 1, dan kondisi 3 ada pada langkah 2.
Sumber :
https://koding4fun.files.wordpress.com/2010/05/complete_example.jpg

III. Aplikasi Boyer-Moore Pada Program Pencari

Program pencari dibuat dengan Bahasa C#, dan kakas Visual Studio. Hasil dari aplikasi adalah sebuah file *executable* (*.exe*) yang dijalankan seperti *command* pada *command prompt* dengan sebuah argument berupa *pattern/keyword*. Jika *environment PATH* ditambah dengan *path* program ini, maka program ini dapat dijalankan di folder mana pun dari *command prompt*.

Program terdiri dari dua fungsi, yang pertama untuk mencari seluruh direktori dari path yang ada secara rekursif, kemudian yang kedua adalah fungsi String Matching Boyer-Moore. Kedua fungsi di atas kemudian digabungkan di dalam program utama. Program utama mempunyai dua variable List of String, yang pertama untuk menyimpan seluruh direktori yang ada, dan yang kedua menyimpan hasil pencocokkan String.

A. Fungsi Recursive Search

Fungsi pertama adalah fungsi rekursif dengan parameter sebuah *path*.

```
procedure searchRecur(string path)
        String[ ] files = getFiles( path)
        foreach( string s in files)
                addToList(s)
        String[ ] dirs. = getDirectories(path)
        if ( dirs.Count() >0)
                foreach ( string z in dirs.)
                        addToList(z)
                        searchRecur(z)
```

Gambar 5 : Pseudo-code untuk mencari semua file dan folder secara rekursif

B. Fungsi String Matching Boyer-Moore

Fungsi kedua adalah fungsi *Boyer-Moore*, dengan algoritma sebagai berikut :

```
function BMMatch ( string text, string pattern) : boolean
        index = pattern.LastChar
        while (!found and index < text.Length)
        if ( text.LastChar == index)
                if(index ==0)
                        return true
                else
                        index—

        else
                if (kondisi1)
                        index = shiftPast()
                else if (kondisi2)
                        index = alignChar()
                else
                        index++
```

Gambar 6 : Pseudo-code Algoritma *String Matching Boyer-Moore*

C. Program Utama

Program utama memeriksa apakah argument valid atau tidak. Jika ada *keyword* yang dimasukkan, maka argument valid dan program akan melanjutkan eksekusi. Program akan mencari seluruh *logical drive* pada *hard disk*, dan memanggil fungsi *SearchRecur* untuk masing-masing *drive*. Biasanya *drive* C:\ akan memakan waktu paling lama.

Setelah fungsi *SearchRecur* selesai, akan menghasilkan List of String yang setiap elemennya berupa path, maka program utama akan memanggil fungsi *Boyer-Moore* untuk mencocokkan setiap elemen List.

Langkah terakhir adalah menampilkan semua elemen yang sesuai dengan *pattern*.

```
function Main (string[ ] args)
        List list;
        List result;
        if ( args.Length == 0 )
                write ( “argument invalid”)
        else
                var drives = getAllDrives( )
                foreach (var drive in drives)
                        searchRecur(drive)
        foreach ( string s in list)
                bool res = BMMatch(s,args[0])
                if (res)
                        addToListRes(s)
        writeList(result)
```

Gambar 7 : Pseudo-code program utama

Setelah program utama berhasil dijalankan dan dikompilasi, maka file *executable* boleh dimasukkan ke dalam *Environment Variables* agar dapat digunakan di direktori mana pun.

IV. HASIL DAN ANALISIS

Program siap untuk dijalankan di *command prompt*. Karena hasil *screenshot* terlalu besar, maka *screenshot* dilampirkan pada daftar gambar. Program dijalankan dua kali, pertama penulis memasukkan input “TugasMakalahStima” yang menghasilkan banyak dokumen dikarenakan adanya file backup dan file *temporary* dari yang otomatis dibuat oleh *Microsoft Word*. Input kedua, penulis ingin mencari file presentasi biologi saat SMA, maka penulis memasukkan input “Spermatogenesis”. Input ketiga, penulis ingin melihat file excel yang ia miliki, maka input berupa string “.xls”.

Program membutuhkan waktu 10 menit hanya untuk menelusuri seluruh direktori, sementara *String Matching* hanya membutuhan waktu hitungan detik. Namun saat program berjalan untuk kedua kali, program hanya membutuhkan waktu 24 detik untuk menelusuri direktori. Hasil yang mengejutkan ini bukan karena input yang berbeda, tetapi ternyata pengaruh dari *cache memory* yang digunakan sistem operasi, sehingga akses data dapat berlangsung lebih cepat.

Untuk input ketiga hasilnya sangat banyak sehingga *command prompt* tidak cukup memuat seluruh hasil yang muncul. Mungkin inilah alasan program *Desktop Search* hanya menampilkan *Best Results*, karena terkadang output yang bisa ditampilkan terlalu banyak.

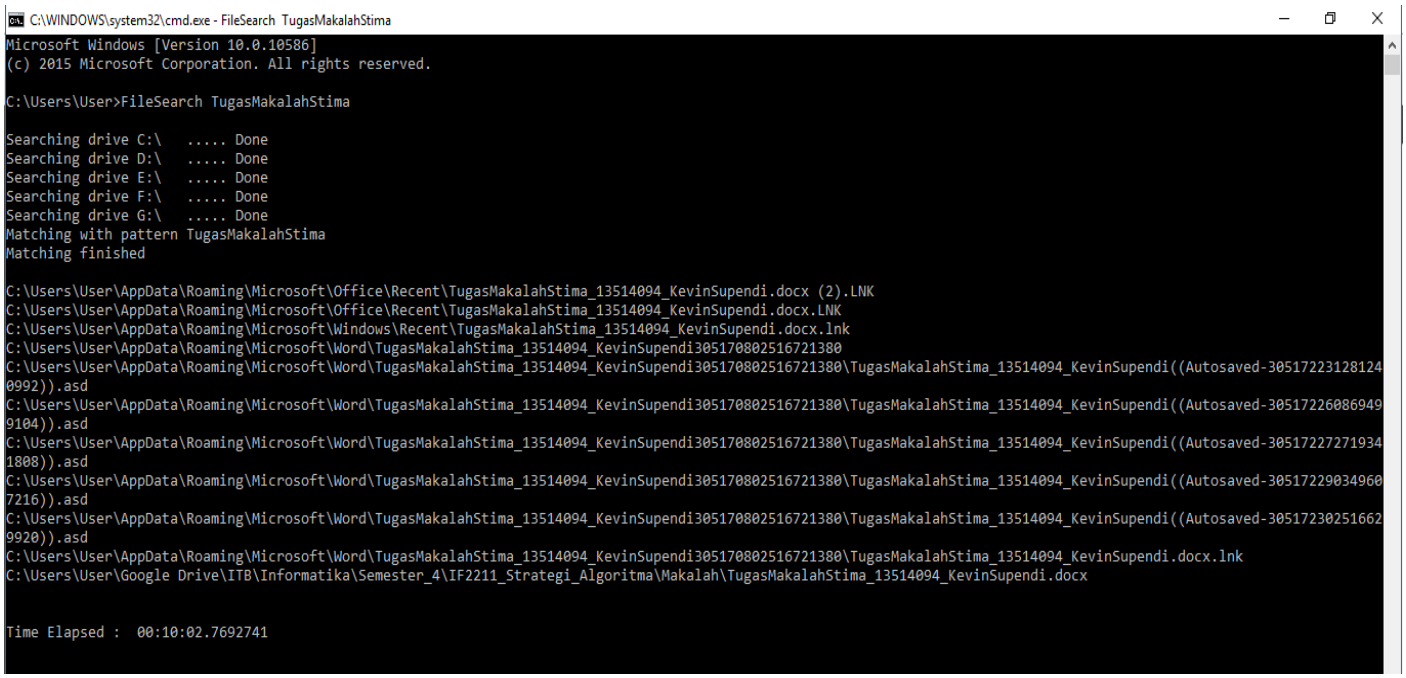
V. KESIMPULAN

Hasil eksekusi program menyatakan bahwa algoritma *String Matching Boyer-Moore* dapat diterapkan untuk program pencari file. Kecepatan membandingkan *String* sangat cepat bahkan untuk membandingkan ratusan ribu *String*.

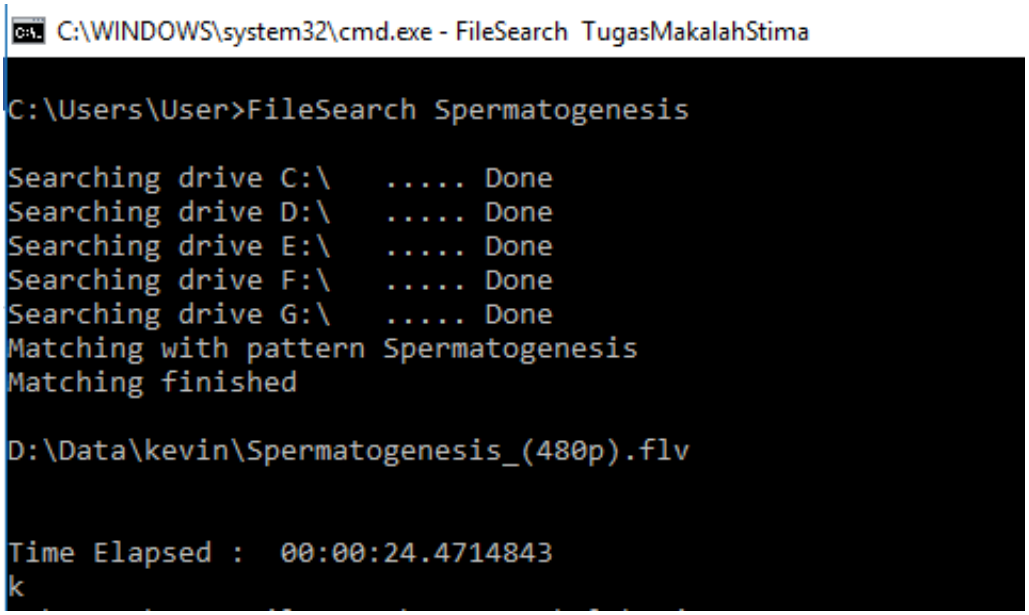
Untuk meningkatkan performansi, program *Desktop Search* memang harus terintegrasi dengan sistem operasi yang berjalan, atau menyimpan memori khusus, sehingga hasil pencarian dapat ditampilkan dalam waktu yang rasional.

Selain itu, mungkin program juga bisa membatasi folder atau disk mana yang dijadikan target pencarian sehingga performansi program lebih cepat.

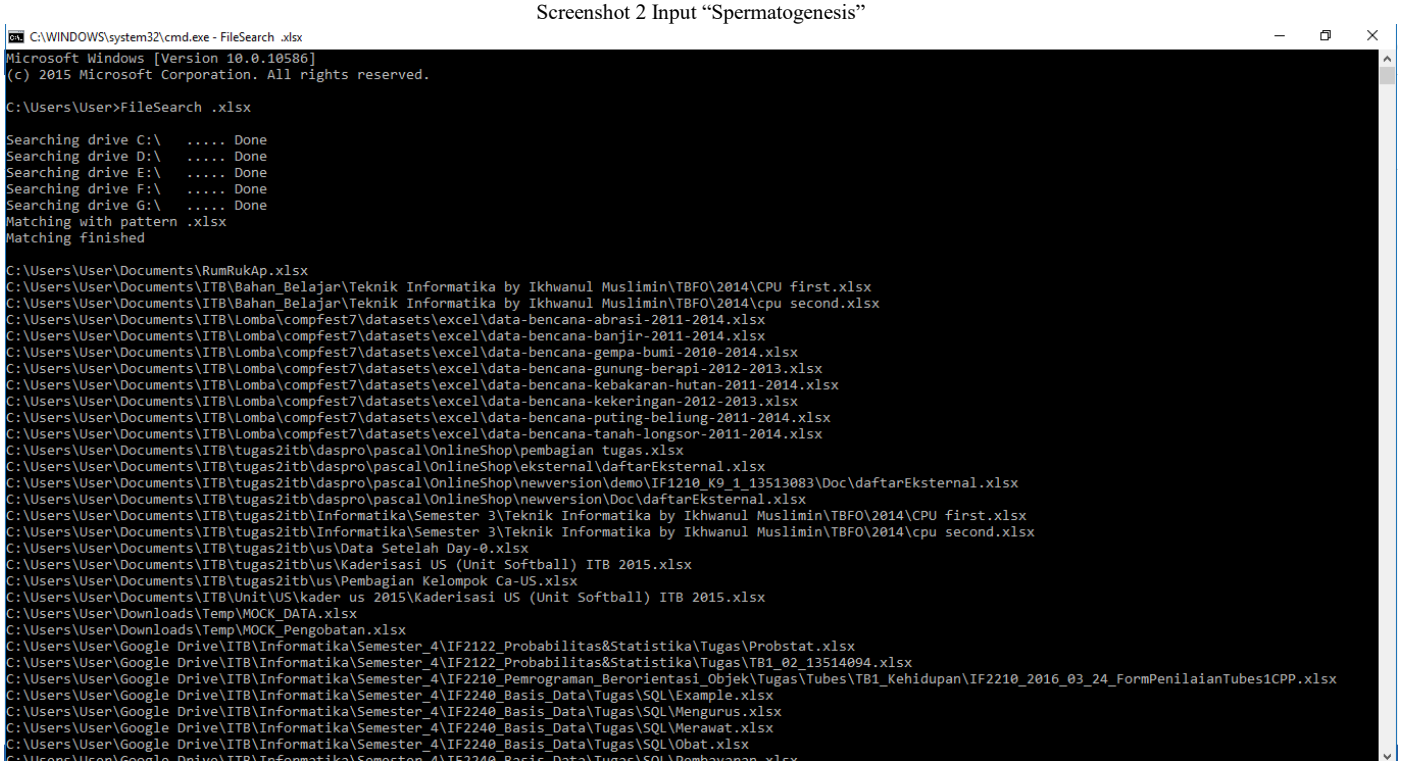
VI. LAMPIRAN



Screenshot 1 – Input “TugasMakalahStima”



Screenshot 2 Input “Spermatogenesis”



Screenshot 3 Input “.xlsx”

[a] Sample of a Table footnote. *(Table footnote)*

VII. Ucapan Terima Kasih

Pertama penulis mau mengucapkan terimakasih kepada Tuhan Yang Maha Esa, sehingga makalah ini dapat selesai tepat waktu. Penulis juga berterima kasih kepada teman-teman dan sumber-sumber yang telah menginspirasi penulis dalam penulisan makalah ini. Penulis juga bersyukur atas orangtua penulis yang telah memberikan kesempatan pada penulis untuk kuliah di ITB. Terimakasih juga kepada Pak Rinaldi dan Bu Ulfa, sebagai dosen IF2211 yang telah mengajar selama setengah tahun ini.

VIII. REFERENSI

[1] Anany Levitin, Introduction to the Design & Analysis of Algorithms, Addison-Wesley, 2003.

[2] https://docs.google.com/viewer?url=http%3A%2F%2Finformatika.stei.itb.ac.id%2F~rinaldi.munir%2FStmik%2F2014-2015%2FPencocokan%2520String%2520(2015).ppt diakses tanggal 7 Mei 2016 10.03 WIB

[3] http://cdn.howtogeek.com/wp-content/uploads/2012/04/00_different_commands_for_finding_items.png diakses tanggal 7 Mei 2016 pk. 10.00 WIB

[4] http://compbio.chemistry.uq.edu.au/mediawiki/upload/f/f4/Brute_force_schema.png diakses tanggal 7 Mei 2016 pk.10.00 WIB

[5] https://koding4fun.files.wordpress.com/2010/05/kmpexample.jpg diakses tanggal 7 Mei 2016 pk. 10.01 WIB

[6] https://koding4fun.files.wordpress.com/2010/05/complete_example.jpg diakses tanggal 7 Mei 2016 pk. 10.02 WIB

IX. 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, 16 Desember 2015

ttd

Kevin Supendi 13514094