Pemrograman Dinamis dalam Kedokteran Forensik

Ramandika Pranamulia 13512078
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
ramandika@students.itb.ac.id

***Abstraksi*—Asam Dexyribosa atau yang lebih populer dengan nama DNA adalah sebuah biomolekul yang menyimpan informasi genetik dari sebuah makhluk hidup. DNA terletak didalam inti sel. Dalam ilmu biologi DNA diketahui bersifat unik untuk setiap orang. Sehingga DNA bisa mengidentifikasi seseorang secara spesifik, sifat ini kemudian digunakan dalam bidang kedokteran forensik secara luas. Karena manusia memiliki lebih dari 3 juta pasang basa DNA, para ahli forensik bergantung pada komputasi komputer dalam mencocokkan dua buah DNA. Selain itu jelas bahwa dibutuhkan algoritma yang mangkus dalam membandingkan kedua buah DNA tersebut seghingga pencocokkan DNA menjadi lebih efisien dan tetap akurat.**

***Index Terms*—pattern matching, pemrograman dinamis, Smith-waterman, Needleman-Wunsch**

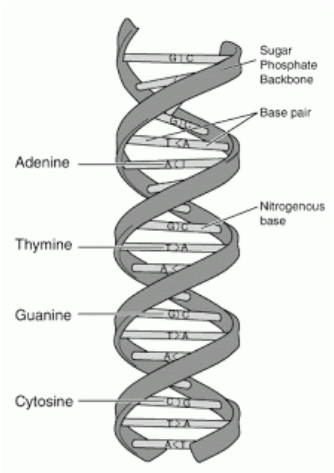
I. PENGENALAN

Tuhan begitu sempurna dapat mencipatakan makhluknya dengan begitu amat sempurna. Salah satunya ditunjukkan dari penciptaan anggota tubuh manusia yang unik sehingga dapat menentukan sifat individu secara khusus contohnya yaitu retina mata, sidik jari dan DNA. Ketiga hal tersebut adalah unik untuk setiap manusia yang artinya tidak ada manusia yang memiliki retina mata,sidik jari ataupun DNA yang sama.

DNA merupakan polimer yang terdiri dari tiga komponen utama yaitu gugus fosfat, gula deoksiribosa, dan basa nitrogen dengan 3 jenis yaitu Adenina(A), Guanina(G), Sitosina(C) dan Timina(T).

Penemuan DNA diawali dengan dugaan bapak genetika yaitu Mandel akan adanya unsur senyawa kimia di dalam sel makhluk hidup yang mewarisi sifat keturunan. Kemudian pada tahun 1869 ilmuwan kimia berkebangsaan Jerman bernama Firedrich Miescher menyelidiki susunan kimia dari nukleus sel. Ia mengentahui bahwa nukleus sel tidak terdiri dari karbohidrat, protein, maupun lemak, melainkan terdiri dari zat yang mempunyai kandungan fosfor yang sangat tinggi. Oleh karena zat itu terdapat di dalam nukleus sel, maka zat itu disebutnya nuklein. Nama itu kemudian dirubah menjadi asa nukleat karena asam ikut menyusunnya.

Penelitian berikutnya dilakukan oleh Fisher pada tahun 1880. Dari hasil risetnya ditemukan adanya zat-zat pirimidin dan purin di dalam asam nukleat. Temuan ini dikembangkan lagi oleh Albreent Kossel yang menhasilkan temuan dua pirimidin yaitu sitosin dan timin dan dua purin yaitu adenin dan guanin di dalam asam nukleat. Selanjutnya pada tahn 1955 Chargaff melalui hidrolisis DNA membuktikan bahwa pada berbagai macam makhluk ternyata banyaknya adenin selalu kira kira sama dengan banyaknya timin, demikian pula dengan sitosin dengan guanin. Hal ini diperkuat oleh penelitian selanjutnya yang dilakukan oleh James Dewey Watson dan Francis H.C. Crick pada tahun 1953. Hasil penelitian tersebut memperlihatkan bahwa DNA tidak berdiri sendiri sebagai suatu rantai tunggal melainkan sebagai dua rantai yang saling berpilin dengan basa pada rantai yang satu melekat pada basa rantai yang lain, yang kemudian lebih dikenal dengan sebutan double helix.



Gambar 1.1 DNA double helix

Pada zaman sekarang DNA digunakan dalam bidang kedokteran forensik untuk mencari pelaku kriminal, dengan mendapatkan DNA dari tempat kejadian dan mencocokannya dengan database di kepolisian.

Hal ini tentu membutuhkan komputer dalam perhitungannya dan juga algoritma yang cepat dan akurat dalam menentukan *pattern matching* antara DNA. Pada umumnya *pattern matching* yang sangat dikenal luas oleh mahasiswa informatika adalah Knuth Morris Prat dan Bayer more. Pada artikel ini saya akan membahas pattern matching antar rantai DNA menggunakan pemrograman dinamis dan beberapa algoritma tambahan untuk mempercepat kerja algoritma program dinamis.

II. Landasan Teori

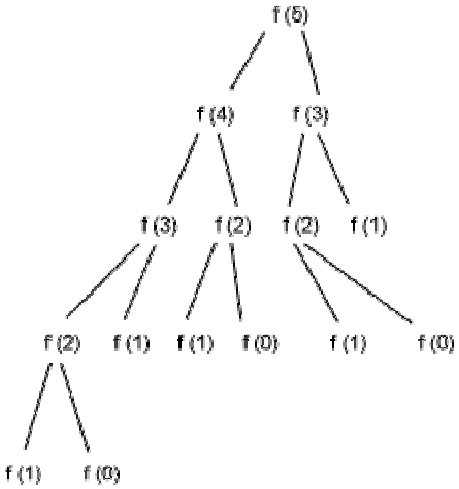
1. Dynamic Programming/Pemrograman dinamis

Pemrograman dinamis adalah salah satu teknik programming yang digemari pada abad ini karena kemampuannya menyelesaikan banyak permasalahan dalam waktu polynomial. Ide utama dari pemrograman dinamis yaitu memecah permasalahan yang kompleks menjadi submasalah yang lebih sederhana. Pemrograman dinamis memakai prinsip nilai optimal pada subpermasalahan sebuah permasalahan merupakan bagian dari hasil optimal sebuah permasalahan. Contoh penggunaan *dynamic programming* yaitu pada pencarian bilangan fibonacci ke n. Bilangan Fibonacci didefinisikan sebagai berikut

F(0)=0
F(1)=1
F(n)=F(n-1)+F(n-2) untuk setiap n>=2

Berikut kode program untuk menyelesaikan permasalahan diatas menggunakan rekursif

```
int Fibonacci(int n){
   if(n==0){return 0;}
  else if(n==1){return 1;}
  else return Fibonacci(n-1)+Fibonacci(n-2)
}
```



Gambar 1.2 Recursive computation Fibonacci numbers

Pada ilustrasi pohon biner diatas terlihat bahwa f(2) dihitung sebanyak tiga kali. Hal tersebut akan menjadi lebih efisien jika kita menyimpan hasil perhitungan f(2) saat pertama kali dihitung. Sehingga selanjutnya kita cukup panggil saja hasil tersebut.

2. Longest Common Subsequence Problem

Ahli biologi yang menemukan suatu gen baru biasanya akan mencari tau gen apa yang paling mirip dengan gen tersebut. Menemukan LCS adalah salah satu cara untuk menentukan seberapa mirip sautu gen terhadap gen yang lain. Semakin besar nilai LCS antara kedua gen maka semakin miriplah kedua gen tersebut.

Perlu diketahui bahwa subsequence berbeda dengan substring. Sebagai contoh ACE merupakan susequence dari ABCDE tetapi bukan substring.

3. Sequence Alignment

Fokus utama dari perbandingan dua DNA adalah sequence allignement. Jika terdapat dua susunan urutan DNA yang memeiliki jumlah sebsequence yang sama lebih dari yang diharapkan, maka ada kemungkinan bahwa kedua DNA tersebut homolog. Homolog merupakan sifat penting dari pasangan DNA, karena jika kedua DNA tersebut homolog maka mereka memiliki nenek moyang yang sama.

4. Global and Local Sequence Alignment

Global sequence alignment adalah pencarian *alignment*/susunan terbaik terhadap seluruh sequence S1 dan S2. Perhatikan dua buah string berikut ini

S1= GCCCTAGCG
S2= GCGCAATG

Jika setiap karakter yang sama bernilai satu poin, spasi bernilai -2 poin, dan karakter yang tidak cocok */mismatch* bernilai -1 maka string S1 dan S2 memiliki poin maksimum dengan kondisisi susunan seperti dibawah ini

S1= GCCCTAGCG
S2= GCGC- AATG (- menandakan spasi)

Maka pada susunan diatas nilai poin totalnya adalah (5*1)+(1*-2)+(3*-1)= 0, yang mana merupakan poin maksimum yang dapat dicapai.

Sedangkan local sequence alignment adalah menyusun S1 agar mirip dengan S2 tetapi tidak harus pas antara S1 dengan S2, perhatikan ilustrasi menggunakan dua buah string diatas (S1 dan S2)

S1=GCCCTAGCG
S2= GCGCAATG

Sehingga total skor untuk local alignment tersebut adalah (3*1) + (0*-2) + (0*-1) = 3. Perhatikan bahwa nilai dari *local alignment* lebih besar dari pada *global alignment* dan secara umum ternyata nilai/skor dari *local alignment* akan selalu lebih besar sama dengan *global alignment*, karena *global alignment* merupakan *local alignment*.

4. Needleman-Wunsch algorithm

Algoritma ini digunakan untuk menghitung global alignment. Idenya mirip dengan algoritma LCS, menggunakan sebuah tabel dua dimensi dengan sebuah

sequence berada diatas tabel (secara horizontal) dan satu sequence lainnya berada di sebelah kiri table tersusun vertikal.

5. Smith-Waterman Algorithm

Algoritma ini merupakan perbaikan dari alogritma Needleman-Wunsch dalam menghitung LCS. Ide dari algoritma ini adalah kita tidak harus mensejajarkan semua sequence.

III. Algoritma dan Pemecahan Masalah

1. LCS Algorithm

Pertama perhatikan bahwa LCS dapat dihitung secara rekursif. Misalkan

- C1 merupakan karakter paling kanan dari S1
- C2 merupakan karkter paling kanan dari S2
- S1' adalah S1 dengan C1 dihilangkan
- S2' adalah S2 dengan C2 dihilangkan

Karenanya terdapat tiga rekursif subproblems:

1. L1=LCS(S1',S2)
2. L2=LCS(S1.S2)
3. L3=(LCS(S1',S2')

Dengan solusinya yaitu nilai terbesar dari L1, L2, L3 digabung C1 jika C1=C2 dan L3 jika C1!=C2. Solusi big-O dari penyelesaian dengan cara program dinamis diatas adalah O(mn) dengan m adalah panjang string S1 dan n adalah panjang string S2.

Untuk menghitung LCS dengan lebih efisien kita menggunakan sebuat tabel yang akan menyusun hasil yang kita cari. Salah satu string disusun secara horizontal di atas table dan yang lainnya secara vertical disebelah kiri table.

		G	C	C	C	T	A	G	C	G
G										
C										
G										
C										
A										
A										
T										
G										

Gambar 3.1.1 Inisialisasi Tabel

Cara pengisian tabel diatas yaitu dari atas ke bawah dan dari kiri ke kanan dan setiap sel yang berisi nilai dari LCS dari S1 dan S2 merupakan *cost* awal yang harus dijumlahkan pada kolom(dengan baris yang lebih besar dari sekarang) yang sama dan baris yan sama (dengan nomor kolom yang lebih besar dari sekarang). Sehingga pada akhirnya setiap sel akan berisi nilai dari subpersoalan dari persoalan aslinya.

		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0									
C	0									
G	0									
C	0									
A	0									
A	0									
T	0									
G	0									

Gambar 3.1.2 Pengisian tabel awal

		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1
C	0	1	2	2	2	2	2	2	2	2
G	0	1	2	2	2	2	2	3	3	3
C	0	1	2	3	3	3	3	3	4	4
A	0	1	2	3	3	3	4	4	4	4
A	0	1	2	3	3	3	4	4	4	4
T	0	1	2	3	3	4	4	4	4	4
G	0	1	2	3	3	4	4	5	5	5

Gambar 3.1.3 Tabel yang sudah terisi penuh

Ketika mengisi sebuah sel beberapa hal yang harus dipertimbangkan adalah

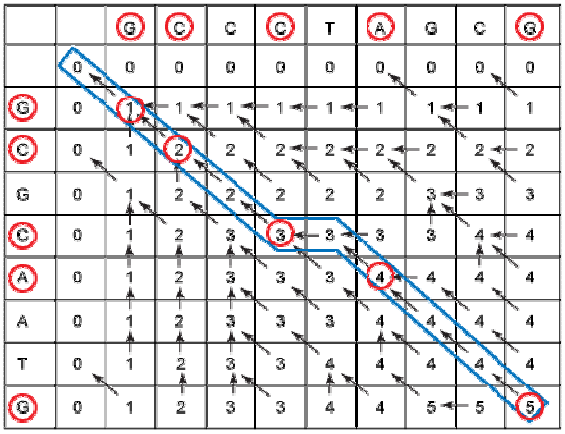
- Sel yang berada disebelah kiri posisi sel yang akan disi
- Sel yang berada diatas sel yang akan diisi
- Sel yang berada serong kiri atas sel yang akan diisi

Tiga nilai dibawah ini akan berpengaruh terhadap nilai tabel yang akan diisi

- V1=nilai sel disebelah kiri sel yang akan diisi
- V2=Nilai sel diatas sel yang akan diisi
- V3=Nilai sel yang berada serong kiri atas sel yang akan diisi

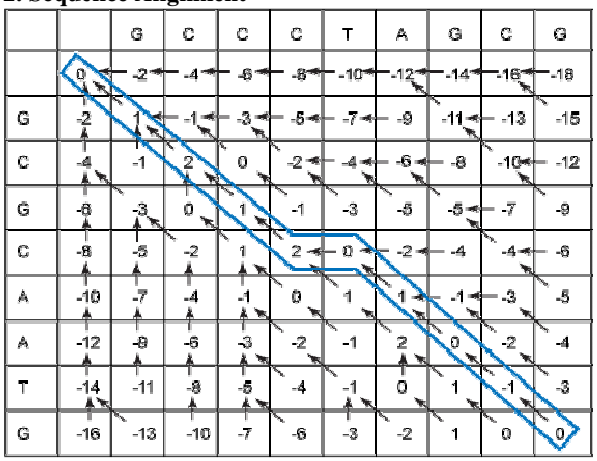
Isi sel dengan maksimum dari ketiga nilai V1,V2, dan V3, dimana untuk V3 ada pengecualian nilai yaitu jika karakter yang berada diatas sel yang akan diisi sama dengan karakter disebelah kiri sel yang akan diisi maka V3+1 selain itu teteap V3 nilainya.

Untuk mengetahui *subsequence* yang sama kita lakukan pencarian dimulai dari sel pojok kanan bawah dengan mengikuti anak panah secara diagonal kearah pojok kanan atas.



Gambar 3.1.4 Pencarian subsequencse(warna merah merupakan sequence yang sama)

2. Sequence Alignment



Gambar 3.2.1 Pengsian subsequence Alignment

Kareana permasalahan sequence alignment merupakan bagian dari LCS maka kita dapat menyelesaikan persoalan ini dengan cara yang hampir sama. Pertama kita harus inisalisasi tabel awal terlebih dahulu yaitu kolom dua dan baris 2. Pada baris dua terlihat bahwa kita membandingkan GCCCTAGCG dengan string kosong atau spasi yang dalam hal ini kita anggap nilainya adalah -2 dan setiap baris atau kolom berikutnya merupakan nilai dari kolom/ bari sebelumnya -2 sehingga kolom 2 bernilai {0,-2,-4,…,-18} begitu juga dengan baris dua. Inisialisasi tersebut dikenal dengan Needleman-Wunsch inizialization code. Berikut merupakan code bahasa C dari inisalisasi diatas

```
double getInitialPointer(int row,int col){
  if(row==0      &&      col!=0)      return
Table[row][col-1];
  else   if   (col==0   &&   row!=0)   return
Table[row-1][col];
       else return 0;
}
```

```
double getInitialScore(int row,int col){
  if(row==0 && col!=0) return col*space;
  else   if   (col==0   &&   row!=0)   return
row*space;
  else return 0;
}
```

Setelah inisialisasi selesai kita dapat melanjutkan pengisian ke sel sel yang masih kosong. Sama seperti algoritma LCS untuk setiap sel kosong kita memiliki tiga pilihan, ambil yang paling maksimum nilainya.

Misalkan S1 dan S2 merupakan dua buah string yang ingin disejajarkan/align dan S1' dan S2' merupakan string hasil pensejajaran. Perhatikan bahwa melanjutkan pencaraian nilai sel dari baris sebelumnya pada kolom yang sama halnya dengan menambahkan satu karakter disebelah kiri S2 sehingga menjadi S2', sambil melongkapi karakter karakter di S1 sehingga tetap di S1'. Karena spasi memiliki nilai -2 maka nilai dari sel sekarang adalah nilai dari sel diatasnya dikurangi 2. Hal tersebut juga berlaku untuk nilai sel yang didapat dari sebelah kiri sel yang sedang diisi.

Sedangkan jika menambahkan suatu karakter pada cell sekarang sehingga menjadi S1' dan menambahkan lagi sehingga menjadi S2' maka hal ini sama dengan berpindah ke sel kosong secara diagonal ke pojok kanan bawah. Skor sel kosong tersebut bergantung pada 2 karakter yang ditambahkan, jika karakter yang ditambahkan cocok maka nilai dari sel kosong tersebut akan menjadi sel diagonal kanan atasnya ditambah satu, namun jika tidak cocok maka akan sama dengan sel diagonal atasnya -1. Berikut adalah kode dalam bahasa C algoritma pengisiannya

```
  void fillInCell(Cell currentCell, Cell
cellAbove, Cell cellToLeft,
  Cell cellAboveLeft) {
  int rowSpaceScore = cellAbove.getScore()
+ space;
  int colSpaceScore = cellToLeft.getScore()
+ space;
  int matchOrMismatchScore =
cellAboveLeft.getScore();
  if (sequence2.charAt(currentCell.getRow()
- 1) == sequence1
  .charAt(currentCell.getCol() - 1)) {
  matchOrMismatchScore += match;
  } else {
  matchOrMismatchScore += mismatch;
  }
  if (rowSpaceScore >= colSpaceScore) {
  if (matchOrMismatchScore >=
rowSpaceScore) {
  currentCell.setScore(matchOrMismatchScore
);
  currentCell.setPrevCell(cellAboveLeft);
  } else {
  currentCell.setScore(rowSpaceScore);
  currentCell.setPrevCell(cellAbove);
  }
  } else {
  if (matchOrMismatchScore >=
colSpaceScore) {
  currentCell.setScore(matchOrMismatchScore
);
```

```
  currentCell.setPrevCell(cellAboveLeft);
  } else {
  currentCell.setScore(colSpaceScore);
  currentCell.setPrevCell(cellToLeft);
  }
}
```

Selanjutnya kita perlu mencari alignment aslinya dan skornya. Skor yang terdapat pada sel pojok kanan bawah merupakan skor maksimum untuk alignment S1 dan S2, sama halnya pada kasus LCS sel pojok kanan bawah berisi banyaknya karakter yang sesuai/ panjang dari LCS.

Kemudian untuk yntuk mendapatkan S1' dan S2' kita harus melakukan penelusuran ulang dari sel paling pojok kanan bawah mengitu tanda panah, dan akan didapatkan kebalikan dari S1' dan S2', maksudnya jika S1'=ATGC maka hasil yang akan didapat adalah CGTA. Berikut merupakan kode dalam bahasa Java untuk mendapatkan reverse dari S1' dan S2'

```
protected boolean traceBackIsNotDone(Cell
currentCell) {
return currentCell.getPrevCell() != null;
}
protected Cell getTracebackStartingCell() {
return scoreTable[scoreTable.length -
1][scoreTable[0].length - 1];
}
```

3. Smith-Waterman

Pada algoritma Smith-Waterman kita tidak fokus terhadap pensejajaran/*aligning* seluruh sequence. Perhatikan bahwa dua buah string dengan panjang nol adalah local alignment dengan skor 0. Sehingga dalam menyusun suatu local alignment kita tidak perlu memperhitungkan sel dengan nilai sama dengan atau kurang dari 0. Hal tersebut dapat menyebabkan alignment berikutnya memiliki skor yang lebih rendah daripada mereset panjang kedua string menjadi 0. Selain itu local alignment tidak perlu berakhir pada salah satu dari kedua sequence, sehingga kita tidak perlu memulai penelusuran kita untuk mencari reverse dari S' dari pojok kanan bawah, kita bisa mulai penelusuran dari sel dengan skor tertinggi. Berikut adalah kod program untuk Smith-Waterman

Smith-Waterman Inizialization

```
int getInitialScore(int row, int col) {
return 0;
}
Cell getInitialPointer(int row, int col) {
return null;
}
```

Smith-Waterman Pengisian sebuah sel

```
void fillInCell(Cell currentCell, Cell
cellAbove, Cell cellToLeft,
Cell cellAboveLeft) {
int rowSpaceScore = cellAbove.getScore() +
space;
int colSpaceScore = cellToLeft.getScore() +
space;
int matchOrMismatchScore =
cellAboveLeft.getScore();
if (sequence2.charAt(currentCell.getRow() -
1) == sequence1
.charAt(currentCell.getCol() - 1)) {
matchOrMismatchScore += match;
} else {
matchOrMismatchScore += mismatch;
}
if (rowSpaceScore >= colSpaceScore) {
if (matchOrMismatchScore >= rowSpaceScore)
{
if (matchOrMismatchScore > 0) {
currentCell.setScore(matchOrMismatchScore);
currentCell.setPrevCell(cellAboveLeft);
}
} else {
if (rowSpaceScore > 0) {
currentCell.setScore(rowSpaceScore);
currentCell.setPrevCell(cellAbove);
}
}
} else {
if (matchOrMismatchScore >= colSpaceScore)
{
if (matchOrMismatchScore > 0) {
currentCell.setScore(matchOrMismatchScore);
currentCell.setPrevCell(cellAboveLeft);
}
} else {
if (colSpaceScore > 0) {
currentCell.setScore(colSpaceScore);
currentCell.setPrevCell(cellToLeft);
}
}
}
if (currentCell.getScore() >
highScoreCell.getScore()) {
highScoreCell = currentCell;
}
```

```
protected boolean traceBackIsNotDone(Cell
currentCell) {
return currentCell.getScore() != 0;
}
protected Cell getTracebackStartingCell() {
return highScoreCell;
}
```

V. KESIMPULAN

Kesimpulan yang bisa kita ambil yaitu informatika bisa digunakan pada cabang biologi yang sekarang lebih dikenal dengan sebutan bioinformatics. Selain itu kita dapat lihat bahwa dynamic programming memegang peranan penting dalam menyelesaikan masalah yang kompleks dan bisa membuat waktu pengerjaan lebih cepat. Terdapat tiga ciri khas persoalan yang dapat diselesaikan dengan dynamic programming yaitu

- Solusi dari persoalan dapat diekspresikan dalam bentuk rekrusif
- Solusi optimal dari sebuah permasalahan dapat dibangun dari solusi optimu subpermasalahannya.

VI. UCAPAN TERIMAKASIH

Ramandika Pranamulia sebagai penulis ingin mengucapkan terima kasih kepada Dr.Ir. Rinaldi Munir,

M.T. dan Ibu Masayu Leylia Khodra,S.T.,M.T. sebagai dosen pembimbing untuk mata kuliah Strategi Algoritma IF 2211. Terimakasih juga penulis ucapankan kepada keluarga serta kerabat yang telah memberikan dukungan dan inspirasi kepada penulis sehingga dapat menyelesaikan karya ilmiah ini.

DAFTAR PUSTAKA

[1] David W. Mount, Bioinformatics : Sequence and Genome Analysis (2nd), Newyork : Cold Spring Harbor Laboratory Press. 2004.

[2] Russell, Peter (2001). *iGenetics*. New York: Benjamin Cummings.

[3] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm tanggal akses 18/5/2014

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, 19 Mei 2014

ttd

Nama dan NIM