# Penerapan Algoritma Pencocokan String Boyer-Moore dan Knuth-Morris-Pratt (KMP) dalam Pencocokkan DNA

Khaidzir Muhammad Shahih 13512068
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*13512068@std.stei.itb.ac.id*

***Absrak*—DNA merupakan materi genetik setiap makhluk hidup. DNA setiap makhluk hidup unik, tidak ada yang sama kecuali kembar identik. Hal ini berlaku pada manusia. Setiap manusia memiliki DNA yang berbeda satu sama lain. Saat ini, pemrosesan kode DNA menjadi hal yang penting, salah satunya adalah pencocokkan kode DNA dari suatu antara dua sampel untuk menentukan apakah kedua sampel berasal dari individu yang sama. Pengaplikasiannya salah satunya dalam kasus tindak kriminal. Pencocokkan DNA dapat dijadikan untuk membuktikan kesalahan diduga tersangka, atau bahkan sebaliknya, dapat menghapus tuduhan kepada terduga. Dalam makalah ini dibahas DNA secara umum, penerapan pencocokkan DNA. Dalam fokusnya, makalah ini tertuju pada pencocokkan DNA untuk membuktikan apakah kedua sampel DNA berasal dari individu yang sama dengan menerapkan pencocokkan string. Algoritma pencocokkan string yang dibahas adalah algoritma Boyer-Moore dan Knuth-Morris-Pratt, serta perbandingan performansi kedua algoritma untuk diterapkan dalam pencocokkan DNA.**

***Kata kunci* – DNA, pencocokkan DNA, pencocokkan string, Boyer-Moore, Knuth-Morris-Pratt**

## I. PENDAHULUAN

DNA merupakan salah satu substansi genetik setiap makhluk hidup. Setiap makhluk hidup memiliki DNA yang berbeda satu sama lainnya. DNA inilah yang memberi sifat kepada makhluk hidup dan membedakan satu individu dengan individu lainnya. Dari DNA ini pula, keberadaan suatu spesies dapat dirunut asal – usulnya, proses evolusi, serta hubungannya dengan spesies lain.

Pencocokan DNA merupakan hal yang cukup penting, karena dengan berbagai masalah yang menyangkut identitas sesorang dapat diselesaikan dengan mengidentifikasi dan mencocokkan beberapa sampel DNA. Misalnya untuk mengetahui hubungan darah beberapa orang, mencari identitas pelaku kriminal, membebaskan seseorang dari tuduhan kejahatan. Dengan pencocokkan DNA dapat membantu menyelesaikan masalah tersebut.

Karena kode – kode genetik yang terkandung dalam DNA sangat banyak jumlahnya, akan sangat sulit bila mencocokkan DNA secara manual. Oleh karena itu, kode – kode genetik dalam DNA dapat direpresentasikan menjadi sebuah string yang sangat panjang. Untuk mencocokkan dua buah sampel DNA tentu saja tinggal mencocokkan kedua DNA tersebut yang telah direpresentasikan menjadi string. Karena itu, algoritma pencocokkan string yang mangkus akan sangat berguna. Banyak algoritma yang digunakan untuk pencocokkan string, dua diantaranya adalah algoritma Knuth-Morris-Pratt (KMP) dan algoritma Boyer-Moore.

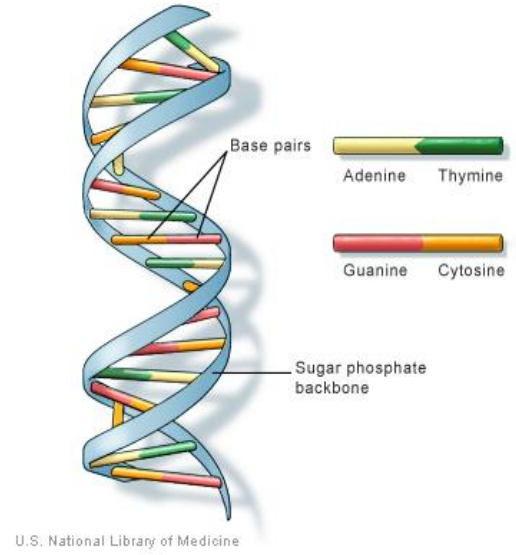
Dalam pencocokkan string, kita gunakan istilah teks, yaitu string yang dijadikan patokan dalam pencocokkan. Istilah kedua yaitu pattern, string yang digunakan untuk pencarian apakah ada yang sama dengan dirinya di dalam teks.

## II. TEORI DASAR

### A. DNA

DNA (deoxyribonucleic acid) adalah material genetik yang terdapat pada hampir semua makhluk hidup. Adanya DNA membuat setiap makhluk hidup di dunia ini unik. Setiap sel pada manusia memiliki DNA yang sama. Sebagian besar DNA terdapat dalam nukleus. Karena banyaknya DNA jumlah DNA, dan kecilnya ukuran nukleus, DNA di dalam nukleus ini terkemas dengan sangat rapi, yang disebut kromosom.

Informasi dalam DNA yang disimpan sebagai kode genetik, merupakan rangkaian 4 buah basa : adenin(A), guanin(G), sitosin(C), dan timin(T). Susunan basa – basa ini menentukan sifat – sifat yang dimiliki makhluk hidup. Bentuk dari DNA sendiri seperti anak tangga yang terpilin, atau yang biasa disebut *double helix.*




Gambar 1 Bentuk DNA[2]

Dalam tubuh manusia terdapat sekitar 3 miliar basa – basa tersebut. Seperti yang disebutkan sebelumnya, DNA setiap makhluk hidup membedakan satu dengan lainnya, termasuk manusia. Tetapi, untuk semua manusia, lebih dari 99% susunan basa mereka sama. Hanya sebagian kecil sisanya yang berbeda antar individu, yang membuat kita unik satu sama lainnya.

B. Pencocokkan DNA

Pencocokkan DNA merupakan cara yang sangat ampuh untuk beberapa penerapan seperti :

- Pengetesan hubungan darah
- Pengetesan forensik
- Terapi gen
- *Genetic genealogy*[3]

Seperti yang sudah dijelaskan, lebih dari 99% DNA manusia sama, hanya sebagian kecil saja yang berbeda. DNA yang berbeda ini disebut penanda genetik (*genetic markers*). Tes hubungan darah, tes forensik, dan tes genetik mencari kesamaan pada penanda genetik ini.

Pada pengetesan hubungan darah, pencocokkan DNA tidak seluruh rangkaian basanya yang sama, karena tidak mungkin semuanya sama jika individunya berbeda. Misalnya untuk menentukan hubungan anak dengan orang tuanya, DNA si anak dicocokkan dengan DNA kedua orangtuanya. Jika 50% cocok dengan sang ayah dan 50% lagi cocok dengan sang ibu, maka benar anak tersebut anak kedua si ayah dan si ibu. Jadi, tidak benar – benar yang sama. Lain halnya seperti pada kasus kriminal, jika diambil sampel genetik di TKP, maka pencocokkan harus semuanya cocok untuk menentukan sampel milik siapa. Dalam makalah ini yang dibahas adalah pencocokkan yang kedua dengan menerapkan algoritma pencocokkan string Boyer-Moore dan Knuth-Morris-Pratt.

C. Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) membandingkan karakter pada pattern dengan teks dengan orientasi dari kiri ke kanan. Jika mendapat ketidakcocokan karakter antara pattern dengan teks, maka pattern akan "digeser" ke kanan. Besarnya pergeseran tidak per satu karakter, melainkan berdasarkan posisi karakter yang tidak cocok. Nilai pergeseran ini dihitung terlebih dahulu sebelum pencocokan pattern dengan teks dilakukan.

Seberapa jauh pattern digeser ditentukan oleh sifat pattern itu sendiri, bukan berdasarkan pada teks. Dalam sebuah string, dikenal istilah suffix (awalan) dan prefix (akhiran). Misalkan A adalah alfabet dan $x = x_1x_2...x_k$, $k \in N$, adalah string yang panjangnya k yang dibentuk dari karakter – karakter di dalam alfabet A. Awalan (prefix) dari x adalah upa-string u dengan

$$u = x_1x_2...x_{k-1},\ k \in \{1, 2, ..., k-1\}$$

dengan kata lain, x diawali u. Akhiran (suffix) dari x adalah upa-string v dengan

$$v = x_{k-b}x_{k-b+1}...x_k,\ k \in \{1, 2, ..., k-1\}$$

dengan kata lain, x diakhiri v.[1]

Pinggiran (*border*) dari x adalah upa-string sedemikian sehingga

$r = x_1x_2x_3...x_{k-1}$ dan $r = x_{k-b}x_{k-b+1}...x_k$, $k \in \{1, 2, ..., k-1\}$

dengan kata lain, pinggiran dari x adalah upa-string yang keduanya awalan dan juga akhiran sebenarnya dari x.[1]

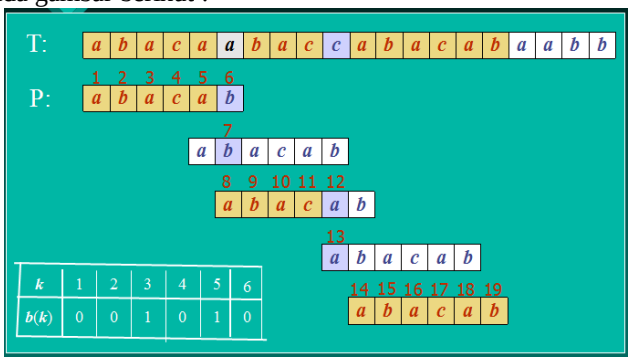
Algoritma KMP pertama – tama menghitung fungsi pinggiran untuk pattern P. Fungsi pinggiran menunjukkan seberapa besar pattern P harus digeser terhadap teks jika terjadi ketidakcocokan. fungsi pinggiran ini murni hanya bergantung pada pattern saja, tidak melibatkan teks, sehingga fungsi pinggiran ini dihitung diawal sebelum pencocokan dilakukan. Fungsi pinggiran b(j) didefinisikan sebagai ukuran awalan dari P yang merupakan akhiran dari P[1..j] [1]. Sebagai contoh misalkan pattern P adalah "aabaabba". Maka fungsi pinggiran dari P dapat ditulis sebagai berikut :

| j | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| P(j) | a | a | b | a | a | b | b | a |
| b(j) | 0 | 1 | 0 | 1 | 2 | 3 | 0 | 1 |

Tabel 1 Fungsi pinggiran untuk pattern "aabaabba"

Setelah dihitung fungsi pinggirannya untuk pattern P, selanjutnya tinggal mencocokkannya dengan teks. Jika terjadi ketidakcocokan, pergeseran yang dilakukan merujuk pada fungsi pinggiran yang sudah kita hitung. Misalnya teks yang akan dicocokkan dengan P diatas adalah "aabaababaabaabba", mula – mula pattern P ditempatkan diawal teks. Setelah itu, maka akan terjadi ketidakcocokkan pada P karakter ke 7. Dengan begitu, karakter P pada indeks 1-6 sama dengan teks, tetapi di karakter 7 berbeda. Karena itu, kita lihat nilai b(6) untuk P yang bernilai 3. Maka, P digeser sejauh panjang karakter yang sama, 6 dikurangi b(6), yaitu 6 – 3 = 3. P digeser sejauh 3 karakter ke kanan. Selanjutnya dilakukan pencocokkan lagi, dimulai dari P indeks ke b(6) + 1 = 4 begitu seterusnya hingga pattern ditemukan kecocokkan dalam teks atau tidak ditemukan kecocokkan hingga akhir

teks. Untuk lebih jelasnya, untuk contoh lain dapat dilihat pada gambar berikut :




Gambar 2 Proses pencocokkan string algoritma Knuth-Morris-Pratt[4]

D. Algoritna Boyer - Moore

Berbeda dengan KMP yang membandingkan pattern dimulai dari kiri, algoritma Boyer-Moore membandingkan pattern dengan teks dari kanan ke kiri. Sama seperti KMP, algoritma Boyer-Moore melakukan pergeseran pattern bergantung karakteristik pattern itu sendiri.

Algoritma Boyer-Moore didasarkan pada dua teknik, yaitu :

1. The looking-glass technique
   Mencari pattern P di teks dengan bergerak mundur, dimulai dari akhir pattern
2. The character-jump technique
   Jika ketidakcocokkan terjadi, maka pattern akan "loncat" sesuai kondisi yang terjadi.

Dalam algoritma Boyer-Moore ada 3 kondisi, misalkan karakter yang tidak cocok dengan teks adalah x, maka kondisi – kondisi tersebut antara lain sebagai berikut

Kondisi 1

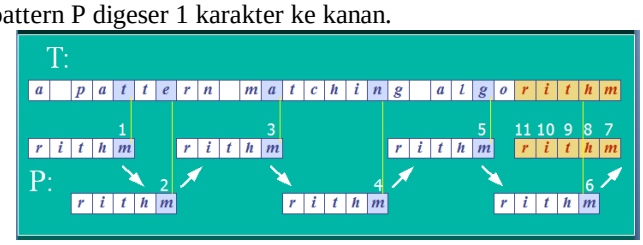
Jika pattern P mengandung x, maka pattern digeser ke kanan sehingga x terakhir di P sejajar dengan teks di posisi yang tidak cocok tadi.

Kondisi 2

Jika pattern P mengandung x, tetapi tidak memungkinkan untuk digeser ke kanan seperti kondisi 1, maka pattern P digeser ke kanan sejauh 2 karakter.

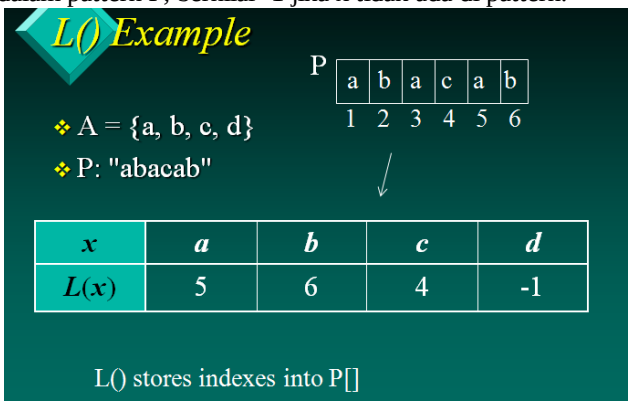
Kondisi 3

Jika kondisi 1 dan kondisi 2 tidak terpenuhi, maka pattern P digeser 1 karakter ke kanan.



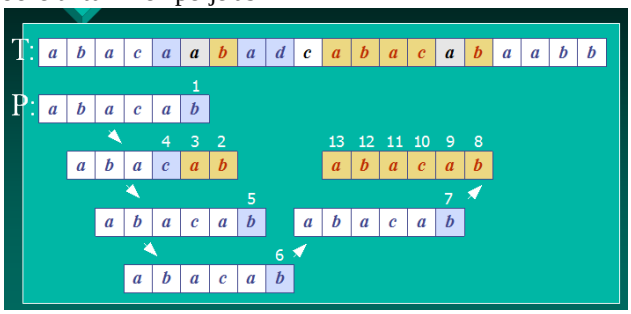

Gambar 3 Proses pencocokkan string algoritma Boyer-Moore[4]

Karena proses pergeseran berdasarkan karakter yang tidak cocok, maka diperlukan sebuah tabel yang menyimpan posisi kemunculan terakhir setiap alfabet dalam pattern. Fungsi yang menghitung ini disebut *Last Occurence Function* L(x), yang memetakan setiap alfabet A ke sebuah integer, yang menyatakan posisi kemunculan x dalam pattern P, bernilai -1 jika x tidak ada di pattern.




Gambar 4 Fungsi *Last Occurence* untuk pattern "abacab" [4]

Berikut contoh lain penggunaan algoritma Boyer-Moore untuk memperjelas




Gambar 5 Proses pencocokkan string algoritma Boyer-Moore 2[4]

## III. Analisis Algoritma Pencocokkan String Knuth-Morris-Pratt dan Boyer-Moore dalam Pencocokkan DNA

Dalam pencocokkan DNA, DNA dipresentasikan sebagai string dengan alfabet {A, C, G, T}, yang merepresentasikan basa dalam DNA.

Dalam makalah ini akan diuji beberapa pencocokkan string menggunakan algoritma KMP dan Boyer-Moore dengan teks dan pattern yang serupa dengan string yang merepresentasikan basa DNA.

Pengujian pertama dilakukan dengan teks sepanjang 100 karakter {A, C, G, T}. Teks pertama yang diujikan sebagai berikut:

```
ATTCGGAGAAACTTGTAGGCGACGG
GGTTAACCTGGACGCATTATAGATCC
TAACACGCTGTCCCAGCCGTGGCTCT
ACTTGGTCAAGACCCCAATTTGA
```

Dengan pattern sepanjang 25 karakter {A, C, G, T}:

```
GGGGTTAACCTGGACGCATTATAGA
```

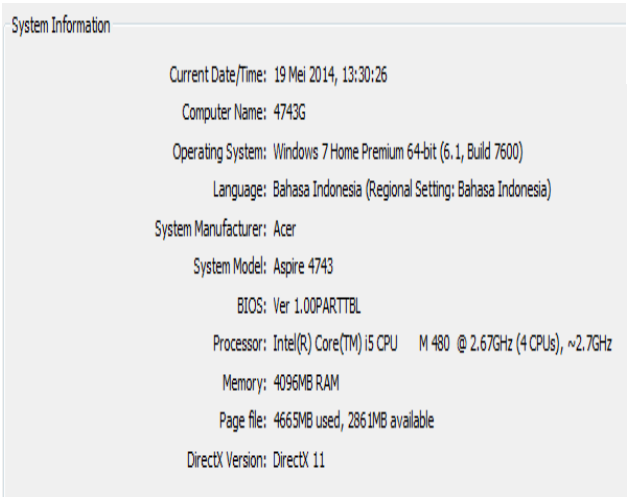
Screenshot hasil pengujian:

```
TEKS:
ATTCGGAGAAACTTGTAGGCGACGGGGTTAACCTGGACGCATTATAGATCCTAACACGCTGTCCCAGCCGTGGCTCTACTTGGTCAAGACCCCAA

PANJANG TEKS = 102
PATTERN = GGGGTTAACCTGGACGCATTATAGA

PANJANG PATTERN = 25
HASIL:
[KMP]Pattern ditemukan pada indeks 23
[KMP]Jumlah operasi perbandingan = 55
[KMP]Waktu eksekusi: 1748.872 mikrosekon

[BM]Pattern ditemukan pada indeks 23
[BM]Jumlah operasi perbandingan = 33
[BM]Waktu eksekusi: 1341.969 mikrosekon
```

Gambar 5 Hasil uji untuk teks dengan panjang 100 karakter

Terlihat untuk pengujian pertama algoritma Boyer-Moore lebih cepat dan jumlah perbandingan

Semua pengujian dalam makalah ini dilakukan dengan laptop Acer Aspire 4743G prosesor core i5 2.67GHzdijalankan di sistem operasi Windows 7 Home Premium.




Gambar 6 Informasi sistem yang digunakan untuk pengujian

Untuk kasus uji berikutnya dilakukan dengan panjang uji teks meningkat 10 kali dengan panjang 1000, 10000, 100000, dan 1000000 dengan panjang pattern seperempat panjang teks. Pattern dan teks digenerate oleh program.

```
TEKS:
ATGACTTGTCTAGGTTGCCGGTCTCCACATAGTCACAAACCTGCTATCACTCACTCCAAAGGGGTCTGACCGTATCTGTGG

PANJANG TEKS = 1002
PATTERN = ACGCAGTTCCGACGCAGGACAGCTTGCTTTGGCTGGGTATCTAATATCCATCGCTATTAAGACTGTGGTCA

PANJANG PATTERN = 250
HASIL:
[KMP]Pattern ditemukan pada indeks 126
[KMP]Jumlah operasi perbandingan = 405
[KMP]Waktu eksekusi: 2130.751 mikrosekon

[BM]Pattern ditemukan pada indeks 126
[BM]Jumlah operasi perbandingan = 288
[BM]Waktu eksekusi: 1955.595 mikrosekon
```

Gambar 7 Hasil uji untuk teks dengan panjang 1000 karakter

```
TEKS:
AAACTTGGACTGACACAACTACCCTGGTATTAGCGCGATTATAGTATTGTGTCACAGGA

PANJANG TEKS = 10002
PATTERN = TGCCCAACCTCTGCCCAAGTCACAGTACTTATCAACTGGAACTATCACT

PANJANG PATTERN = 2500
HASIL:
[KMP]Pattern ditemukan pada indeks 684
[KMP]Jumlah operasi perbandingan = 3338
[KMP]Waktu eksekusi: 3789.158 mikrosekon

[BM]Pattern ditemukan pada indeks 684
[BM]Jumlah operasi perbandingan = 2743
[BM]Waktu eksekusi: 2809.82 mikrosekon
```

Gambar 8 Hasil uji untuk teks dengan panjang 10000 karakter

```
PANJANG TEKS = 100002
PATTERN = TGAGGTGTCTTACTTGCGTAAAAAAGAACCAAGTCCCGTAATGCCGCT

PANJANG PATTERN = 25000
HASIL:
[KMP]Pattern ditemukan pada indeks 2742
[KMP]Jumlah operasi perbandingan = 28446
[KMP]Waktu eksekusi: 14949.137 mikrosekon

[BM]Pattern ditemukan pada indeks 2742
[BM]Jumlah operasi perbandingan = 25736
[BM]Waktu eksekusi: 11076.443 mikrosekon
```

Gambar 9 Hasil uji untuk teks dengan panjang 100 ribu karakter

```
TEKS:
CCCTTAAGTATCGCTTAAACGGTTGAATTGCACATCGCTTTCGTCATCTGGGACATACAAAGAGTCCACGCGGT

PANJANG TEKS = 1000002
PATTERN = ACCACTCAGCTTACCCATGGCCTGTCAAAGCTGACTTTAGCAGGGTCTCTACTCAGGTGACAAC

PANJANG PATTERN = 250000
HASIL:
[KMP]Pattern ditemukan pada indeks 623623
[KMP]Jumlah operasi perbandingan = 1029600
[KMP]Waktu eksekusi: 27233.969 mikrosekon

[BM]Pattern ditemukan pada indeks 623623
[BM]Jumlah operasi perbandingan = 512154
[BM]Waktu eksekusi: 13915.136 mikrosekon
```

Gambar 10 Hasil uji untuk teks dengan panjang 1 juta karakter

Berikut tabel hasil pengujian

| Algoritma | Panjang teks | | | | |
|---|---|---|---|---|---|
| | 100 | 1000 | 10000 | 100000 | 1000000 |
| KMP (waktu eksekusi(mikrosekon) | 1748.872 | 2130.751 | 3789.158 | 14949.137 | 27233.969 |
| BM (waktu eksekusi (mikrosekon) | 1341.969 | 1955.595 | 2809.82 | 11076.443 | 13915.136 |

Tabel 2 Perbandingan waktu eksekusi

| Algoritma | Panjang teks | | | | |
|---|---|---|---|---|---|
| | 100 | 1000 | 10000 | 100000 | 1000000 |
| KMP (jumlah perbandingan) | 55 | 405 | 3338 | 28446 | 623623 |
| BM (jumlah perbandingan) | 33 | 288 | 2743 | 25736 | 512154 |

Tabel 3 Perbandingan operasi perbandingan karaktaer

Tambahan panjang teks sebaanyak 2 disebabkan adanya karakter new line yang dibuat oleh teks editor. Jadi sebenarnya panjang karakter {A,C,G,T} dari teks adalah kelipatan 10.

Dalam kelima kasus uji ditunjukkan algoritma Boyer-Moore lebih mangkus untuk pencocokkan string DNA. Terlihat semakin panjang pattern dan teks semakin terlihat kesenjangan performansi antara kedua algoritma.

Sebagai perbandingan kita tinjau *preprocessing* kedua algoritma. Algoritma KMP menghitung semua fungsi pinggiran, yaitu prefiks dan sufiks yang sama dan terpanjang dalam pattern. Maka akan dihitung fungsi borderan b(1) hingga b(N-1), dengan N adalah panjang pattern.

Dalam algoritma Boyer-Moore, *preprocessing* yang dilakukan adalah menghitung *last occurence* setiap karakter dalam alfabet, yaitu indeks kemunculan terakhir karakter dalam teks.

Kode untuk menghasilkan fungsi pinggiran diberikan sebagai berikut:

```
private static int[] computeFail(String pattern) {
    int fail[] = new int[pattern.length()];
    fail[0] = 0;

    int m = pattern.length();
    int j = 0;
    int i = 1;

    while(i < m) {
        if (pattern.charAt(j) == pattern.charAt(i)) {
            fail[i] = j+1;
            i++;
            j++;
        } else if (j > 0) {
            j = fail[j-1];
        } else {
            fail[i] = 0;
            i++;
        }
    }
    return fail;
}
```

Kode untuk menghasilkan fungsi pinggiran

Sedangkan untuk menghitung fungsi *last occurence* sebagai berikut:

```
private static int[] buildLast(String pattern) {
    int last[] = new int[128];
    for (int i=0; i<128; i++)
        last[i] = -1;

    for (int i=0; i<pattern.length(); i++)
        last[pattern.charAt(i)] = i;

    return last;
}
```

Kode untuk menghasilkan fungsi *last occurence*

Secara umum, kedua algoritma memiliki kompleksitas yang sama, yaitu O(n). Tetapi, dalam algoritma fungsi *last occurence* hanya melakukan proses *assignment* ke dalam array, sedangkan pada algoritma menghitung fungsi pinggiran diatas dilakukan operasi perbandingan dan *assignment* sehingga membuat algoritma menghitung fungsi pinggiran lebih banyak melakukan proses.

Selain dalam hal *preprocessing*, performansi juga sangat ditentukan oleh proses pencocokkan. Dalam kasus ini, algoritma Boyer-Moore mampu melakukan operasi perbandingan yang lebih sedikit daripada KMP. Ini disebabkan menggunakan algoritma Boyer-Moore mampu melakukan pergeseran pattern yang lebih jauh daripada KMP, namun tetap menghasilkan hasil yang sama.

## IV. KESIMPULAN

Dalam setiap kasus pengujian yang dilakukan, algoritma Boyer-Moore terbukti lebih efisien untuk melakukan pencocokkan string DNA. Algoritma Boyer-Moore pada kelima kasus memperlihatkan waktu eksekusi yang lebih cepat dan jumlah operasi perbandingan yang lebih sedikit daripada algoritma Knuth-Morris-Pratt.

Dalam data uji yang dihasilkan, terlihat semakin panjang teks dan pattern, semakin terlihat kesenjangan performansi antara kedua algoritma.

Kesenjangan performansi ini dikarenakan *preprocessing* algoritm Boyer-Moore yang lebih sederhana daripada algoritma Knuth-Morris-Pratt. Selain itu, dalam proses pencocokkan pun algoritma Boyer-Moore terbukti lebih efisien, terlihat dari jumlah operasi perbandingan algoritma Boyer-Moore yang lebih sedikit daripada algoritma Knuth-Morris-Pratt. Hal – hal diatas membuat waktu eksekusi algoritma pencocokkan string Boyer-Moore lebih cepat daripada algoritma pencocokkan string Knuth-Morris-Pratt dalam kasus string yang merepresentasikan basa DNA.

## V. UCAPAN TERIMA KASIH

Pertama – tama saya ucapkan terima kasih kepada Tuhan Yang Maha Esa karena atas rahmat-Nya makalah ini bisa terselesaikan. Saya berterima kasih juga kepada kedua orang tua saya yang telah membesarkan saya sampai saat ini. Tidak lupa terima kasih saya ucapkan kepada Pak Rinaldi Munir dan Bu Masayu selaku dosen Mata Kuliah IF2211 Strategi Algoritma yang telah member pengetahuan kepada saya tentang semua teori yang ada dalam makalah ini.

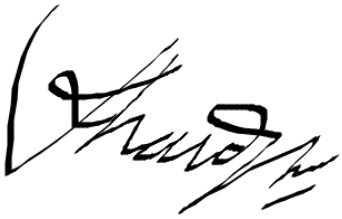
## REFERENSI

[1] Rinaldi Munir, *Diktat Kuliah IF2211*. 2009
[2] http://ghr.nlm.nih.gov/handbook/basics/dna diakses 17 Mei 2014
[3] http://www.bbc.co.uk/science/0/20205874 diakses 17 Mei 2014
[4] Slide kuliah Rinaldi Munir 2014 http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2013-2014-genap/String%20Matching%20dengan%20Regex.pdf
[5] http://www.biotechnologyonline.gov.au/popups/int_dnaprofiling.html diakses 18 Mei 2014
[6] http://science.howstuffworks.com/life/genetic/dna-evidence4.htm diakses 18 Mei 2014
[7] http://www.bbc.co.uk/science/0/20205874 diakses 18 Mei 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



Khaidzir Muhammad Shahih - 13512068