PERBANDINGAN ALGORITMA KNUTH-MORRIS-PRATT, STRING MATCHING ON ORDERED ALPHABET, dan BOYER-MOORE dalam PENCARIAN UNTAI DNA

Tito Daniswara
13506097

Jurusan Teknik Informatika Institut Teknologi Bandung
Jalan Ganesha 10 Bandung
tito_daniswara@yahoo.com

ABSTRAK

DNA merupakan suatu bagian terkecil dari sel-sel yang dimiliki oleh mahluk hidup. Di dalam DNA tersebut terkandung empat buah protein utama yang menjadi kunci kehidupan manusia, yaitu *adhenine, guanine, cytocine,* dan *thimine*. Setiap manusia mempunyai sel-sel yang masing-masing mempunyai DNA yang sama. Untaian DNA tertentu menggambarkan sifat-sifat dari mahluk hidup tersebut. Melalui algoritma-algoritma pencarian string kita dapat menentukan persamaan-persamaan urutan kemunculan dari suatu DNA, dan dari sana dapat dilihat kemiripan sifat antara empunya *pattern* DNA dengan target DNA yang dicari. Dalam bahasan karya tulis ini, hanya tiga dari sekian banyak jenis algoritma pencarian string yang akan digunakan. Ketiga algoritma itu adalah Knuth Morris Pratt (KMP), String Matching on Ordered Alphabet, dan Boyer-Moore. Di sini akan diberikan satu kasus sebuah target DNA dan pattern DNA yang akan dicari, kemudian akan diselesaikan oleh masing-masing algoritma tersebut. Dari penyelesaian ini dapat dihitung derajat kemangkusan dalam pencarian dengan menggunakan notasi *big-O (*O-Besar*)*. Tentu saja dapat ditentukan pula algoritma mana yang lebih ampuh untuk digunakan dalam pencarian DNA ini karena jumlah untai DNA tersebut amatlah banyak, sehingga kemangkusan pencarian amatlah penting.

Kata kunci: KMP, String, DNA, String Matching on Ordered Alphabet, Boyer-Moore.

1. PENDAHULUAN

Suatu DNA pada sel mahluk hidup mempunyai empat buah protein penting yang menentukan sifat-sifat fisik maupun mental dari seseorang. Empat buah protein yang bernama *adhenine, guanine, cytocine*, dan *thimine* ini amatlah banyak dalam DNA tersebut dan bersifat saling menguntai, seperti string dalam teks. Contoh urutan-urutan protein-protein tersebut misalnya sebagai berikut,

ACGAGCATATCGTCG, GCATACGTAGCAT

dimana A mewakili *adhenine*, G mewakili *guanine*, C mewakili *cytocine*, dan T mewakili *thimine*.

Misalkan ditemukan suatu urutan DNA tertentu, misalnya AGCAGAT yang mewakili suatu sifat tertentu pada mahluk tersebut. Kita dapat mengetahui apakah sifat ini terdapat pada mahluk yang lain. Dengan menggunakan algoritma pencarian string, kita dapat mengetahuinya. Diantara algoritma-algoritma pencarian string tersebut tiga diantaranya adalah Knuth-Morris-Pratt, String Matching on Ordered Alphabet, dan Boyer-Moore.

Sebelum berlanjut ke dalam pembahasan, terlebih dahulu diberikan definisi-definisi yang diperlukan, yaitu:

1. Kata u adalah prefix dari untai w bila terdapat kata v (boleh kosong) sehingga $w = uv$
2. Kata v adalah sufiks dari untai w bila terdapat kata u (boleh kosong) sehingga $w = uv$
3. Kata z adalah subkata dari untai w bila terdapat kata u dan v (boleh kosong) sehingga $w = uzv$
4. Bilangan bulat p adalah periode dari kata w jika i, $0 < i < m-p$, $w[i]=w[i+p]$. Periode terkecil dari w disebut periode, dilambangkan *per(w)*.
5. Kata z adalah pinggiran dari kata w bila terdapat dua kata u dan v sehingga $w = uz = zv$, z adalah prefiks dan sufiks dari w. $|u|=|v|$ adalah periode dari w.
6. Kata w adalah *dasar* jika tidak ada kata z and bilangan bulat k sehingga $w=z^k$

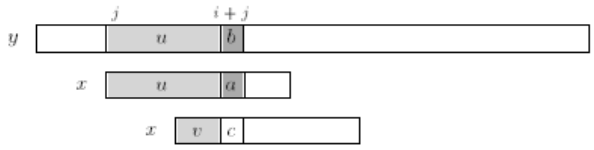
2. METODE

2.1 Pencarian Untai DNA dengan Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) merupakan pengembangan dari algoritma pencarian string secara satu-satu (*Brute Force*). Pada algoritma KMP ini,

informasi mengenai tiap karakter pada pattern yang akan dicari disimpan sebagai fungsi pinggiran.

Pertama-tama kita mencoba pada posisi kiri *j*, yaitu saat 'jendela' diposisikan dalam substring *y[j..j+m-i]*. Asumsikan ketidakcocokan pertama terjadi antara *x[i*] dan *y[i+j]* dengan $0 < i < m$. Kemudian $x[0..i-1] = y[j..i+j-1] = u$ dan $a = x[i] \neq y[i+j] = b$. Saat menggeser, cukup beralasan untuk mengharapkan bahwa prefiks *v* dari pattern cocok dengan sufiks dari sisa *u* pada teks. Prefiks terpanjang *v* yang bisa diambil disebut sebagai penanda pinggiran dari *u*. Maka muncul notasi baru *kmpNext[i]* yang merupakan panjang dari pinggiran terpanjang dari *x[0..i-1]* yang diikuti oleh karakter *c* yng berbeda dari *x[i]* dan -1 jika penanda pinggiran itu tidak ada, untuk $0 < i \leq m$. Kemudian, setelah penggeseran, perbandingan dapat berlanjut di antara karakter *x[kmpNext[i]]* dan *y[i+j]* tanpa menghilangkan okurensi apapun dari *x* dalam *y*, dan menghindarkan runut-balik pada teks (lihat gambar 1).



Gambar 1. Pergeseran pada algoritma KMP, v adalah pinggiran dari u, a ≠ c

Nilai pada *kmpNext[0]* diset menjadi -1. Berikut ini adalah algoritma pencarian pattern DNA GCAGAGAG pada untai DNA GCATCGCAGAGTATACAGTACG.

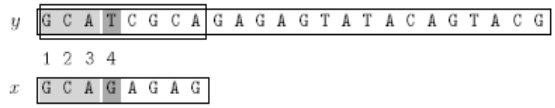
Pertama-tama kita buat fungsi pinggiran untuk pattern

i	0	1	2	3	4	5	6	7	8
x[i]	G	C	A	G	A	G	A	G	
kmpNext[i]	-1	0	0	-1	1	-1	1	-1	1

Tabel 1. Fungsi Pinggiran untai DNA.

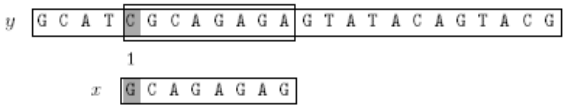
Fase pencariannya adalah sebagai berikut,

- Percobaan pertama



Menggeser sebanyak 4 (*i-kmpNext[i] = 3-(-1)*)

- Percobaan kedua

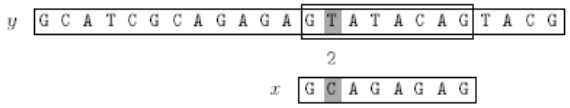


Menggeser sebanyak 1 (*i-kmpNext[i] = 0-(-1)*)

- Percobaan ketiga

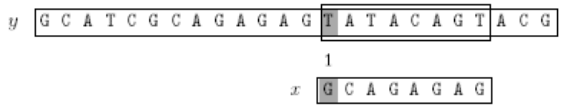
Menggeser sebanyak 7 (*i-kmpNext[i] = 8-1*)

- Percobaan keempat



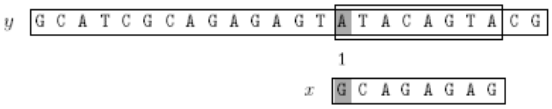
Menggeser sebanyak 1 (*i-kmpNext[i] = 0-(-1)*)

- Percobaan kelima



Menggeser sebanyak 1 (*i-kmpNext[i] = 0-(-1)*)

- Percobaan keenam

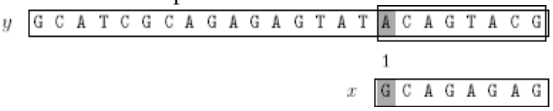


Menggeser sebanyak 1 (*i-kmpNext[i] = 0-(-1)*)

- Percobaan ketujuh

Menggeser sebanyak 1 (*i-kmpNext[i] = 0-(-1)*)

- Percobaan kedelapan



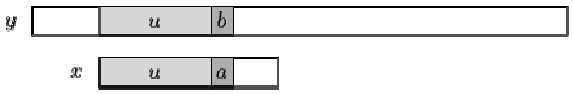
Menggeser sebanyak 1 (*i-kmpNext[i] = 0-(-1)*)

Algoritma Knuth-Morris-Pratt melakukan sebanyak 18 kali perbandingan karakter. Tabel *kmpNext* dapat diselesaikan dengan kompleksitas waktu *O(m)* sebelum melakukan fase pencarian. Fase pencariannya sendiri dapat dilakukan dengan kompleksitas waktu *O(m+n).* Algoritma KMP ini melakukan paling banyak *2n-1* perbandingan karakter selama fase pencarian.

2.2 Pencarian Untai DNA dengan Algoritma String Matching on Ordered Alphabet

Algoritma ini mirip dengan algoritma pencarian satu-satu (brute force), perbedaannya saat dilakukan percobaan untuk menyamakan string dimana 'jendela' diposisikan oleh substring *y[j..j+m-1]*, saat prefiks *u* dari *x* telah ditemukan dan ketidak samaan terjadi antara karakter *a* dalam *x* dengan *b* dalam kata *y* (lihat gambar 2), algoritma ini akan menghitung periode *ub*. Jika tidak

berhasil dalam menemukan periode yang tepat, algoritma ini akan beralih untuk menghitung perkiraannya.



Gambar 2. Percobaan Tipikal pada algoritma String Matching on Ordered Alphabet

Definisi jelas dari algoritma ini adalah sebagai berikut, definsikan $tw^ew`$ sebagai dekomposisi maksimal-sufiks (dekomposisi MS) sebagai *x*, jika:
- $v = w^ew`$adalah sufiks maksimal dari x berdasarkan pengurutan alfabetikal
- *w* adalah dasar
- $e \geq 1$
- *w`* adalah prefiks sebenarnya dari *w*

Lalu didapat $|t| < per(x)$. Jika $tw^ew`$ adalah dekomposisi MS dari kata tidak kosong *x*, maka:
- jika *t* adalah sufiks dari *w*, maka $per(x) = per(v)$
- $per(x) > |t|$
- jika $|t| \geq |w|$ maka $per(x) > |v| = |x| - |t|$
- jika *t* bukan sufiks dari *w* dan $|t| < |w|$, maka $per(x) > min(|v|,|tw^e|)$

Jika *u* adalah sufiks dari *w* maka $per(x)=per(v)=|w|$. Jika *u* bukan sufiks dari *w*, $per(x) > max(|u,min(|v|, |tw^e|)) \geq |x|/2$. Jika tw^ew' adalah dekomposisi MS dari kata tidak kosong *x*, $per(x) = |w|$ dan $e > 1$ maka $tw^{e-1}w'$ adalah dekomposisi MS dari $x' = uw^{e-1}w'$.

Algoritma ini menghitung sufiks maksimal dari prefix pattern yang sama digabung dengan karakter yang sama dari string yang dicari, tiap percobaannya. Ini menghindarkan penghitungan dari awal lagi setelah pergantian panjang *per(w)* dilakukan. Algoritma ini tidak membutuhkan fase untuk melakukan pre-proses.

Sama seperti sebelumnya, algoritma ini akan dicoba untuk mencari pattern DNA GCAGAGAG ke dalam untai DNA GCATCGCAGAGAGTATACAGTACG.
- Percobaan pertama
  GCATCGCAGAGAGTATACAGTACG
  1 2 34
  GCAGAGAG
  Berpindah sejauh 4 karakter

- Percobaan kedua
  GCATCGCAGAGAGTATACAGTACG
  1
  GCAGAGAG
  Berpindah sejauh 1 karakter

- Percobaan ketiga
  GCATCGCAGAGAGTATACAGTACG
  1 2 34 56 7 8
  GCAGAGAG
  Berpindah sejauh 9 karakter

- Percobaan keempat
  GCATCGCAGAGAGTATACAGTACG
  1
  GCAGAGAG
  Berpindah sejauh 1 karakter

- Percobaan kelima
  GCATCGCAGAGAGTATACAGTACG
  1
  GCAGAGAG
  Berpindah sejauh 1 karakter

- Percobaan keenam
  GCATCGCAGAGAGTATACAGTACG
  1
  GCAGAGAG

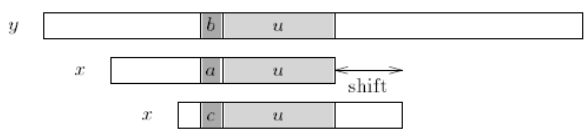
Untai DNA yang dibaca telah habis.

Algoritma String Matching on Ordered Alphabet melakukan 37 kali perbandingan karakter. Pada kasus terburuk, algoritma ini melakukan perbandingan karakter sebanyak *6n+5* perbandingan, menghasilkan kompleksitas waktu pencariannya *O(n).*

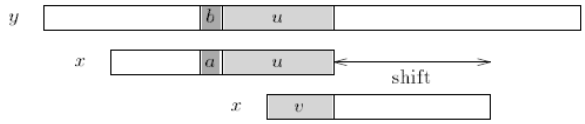
2.3 Pencarian Untai DNA dengan Algoritma Boyer-Moore

Algoritma Boyer-Moore memindai karakter pada untai string dari kanan ke kiri, mulai dari karakter paling kanan. Saat terjadi ketidakcocokan antara karakter (atau tepat cocok), algoritma ini menggunakan dua fungsi yang sebelumnya telah dihitung terlebih dahulu untuk memindahkan 'jendela' ke kanan. Dua fungsi pemindah itu disebut *good-suffix shift* dan *bad-character shift.*

Asumsikan bahwa kesalahan pencocokan terjadi antara karakter $x[i] = a$ dari suatu pattern dan karakter $y[i+j] = b$ dari string target disaat percobaan pencocokan pada posisi *j*. Kemudian, $x[i+1..m-1] = y[i+j+1..j+m-1] = u$ dan $x[i] \neq y[i+j]$. *Good-suffix shift* berguna untuk meratakan segmen $y[i+j+1..j+m-1]$ dengan okurensi terkanan dalam *x* yang didahului oleh karakter yang berbeda dari $x[i]$ (lihat gambar 3). Jika tidak terdapat segmen seperti itu, shift ini berguna untuk meratakan sufiks terpanjang *v* dari $y[i+j+1..j+m-1]$ dengan prefix yang cocok dengan *x* (lihat gambar 4).



Gambar 3. *Good-suffix shift,* u kembali muncul didahului karakter c yang berbeda dengan a



Gambar 4. *Good-suffix shift,* hanya sufiks u yang kembali muncul di x

Bad-character shift meratakan karakter dari string *y[i+j]* dengan okurensi paling kanannya pada *x[0..m-2]* (lihat gambar 5). Jika *y[i+j]* tidak terjadi di pattern *x*, tidak ada okurensi x dalam y yang dapat memasukkan *y[i+j]*, dan bagian paling kiri dari 'jendela' diratakan dengan karakter selanjutnya setelah *y[i+j]*, yaitu *y[i+j+1]* (lihat gambar 6).

Di sini dapat dilihat *bad-character shift* dapat berbentuk negatif, oleh karena itu untuk memindahkan 'jendela', algoritma Boyer-Moore menggunakan maksimum antara *good-suffix shift* dan *bad-character shift.* Secara formal, dua fungsi shift tersebut didefinisikan sebagai berikut.

Fungsi g*ood-suffix shift* disimpan di dalam tabel *bmGs* dengan ukuran *m+1*. Gunakan dua kondisi:

Cs(i,s) : untuk tiap k sehingga i < k < m, s ≥ k atau x[k-s] = x[k], dan

Co(i,s) : jika s < i maka x[i-s] ≠ x[i].

Lalu untuk *0 ≤ i < m*:

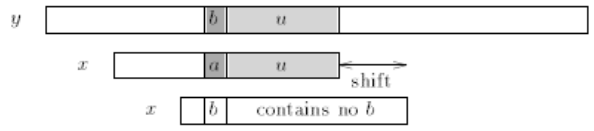
bmGs[i+1] = min{s > 0 : Cs(i,s) dan Co(i,s) tahan}

Dan kita mendefinisikan *bmGs[0]* sebagai panjang dari periode x, Penghitungan dari tabel bmGs menggunakan tabel *suff*, yang didefinisikan sebagai berikut:

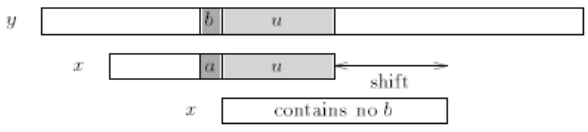
Untuk *1 ≤ i < m, suff[i] = max {k:x[i-k+i..i]} = x[m-k,m-1]*.

Fungsi *bad-character shift* disimpan dalam tabel bmBc dengan ukuran σ. Untuk c Є Σ:

$$bmBc[c] = \begin{cases} \min\{i : 1 \le i < m-1 \text{ dan } x[m-1-i] = c\} & \text{jika } c \text{ terjadi di } x \\ m & \text{untuk lainnya} \end{cases}$$



Gambar 5. *Bad-character shift,* a muncul di x



Gambar 6. *Bad-character shift,* a tidak muncul di x

Pada kasus pencarian DNA dengan untai GCATCGCAGAGAGTATACAGTACG dan pattern yang dicari GCAGAGAG, langkah-langkahnya adalah sebagai berikut,

Pertama dibuat tabel bmBc (untuk *Bad-character shift*)

c	A	C	G	T
bmBc[c]	1	6	2	8

Tabel 2. bmBc

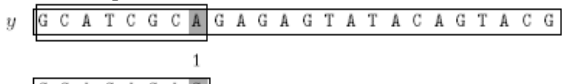
Kemudian tabel bmBs (untuk *good-suffix shift)*

i	0	1	2	3	4	5	6	7
x[i]	G	C	A	G	A	G	A	G
suff[i]	1	0	0	2	0	4	0	8
bmGs[i]	7	7	7	2	7	4	7	1

Tabel 3. bmGs

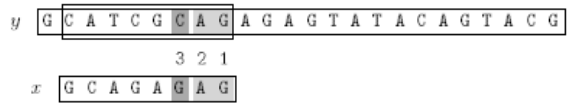
Kemudian, fase pencarian:

- Percobaan pertama



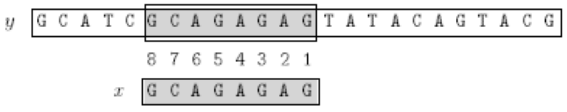
Menggeser sebanyak 1 (*bmGs[7] = bmBc[A]-7+7*)

- Percobaan kedua



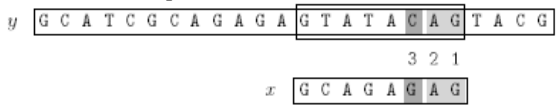
Menggeser sebanyak 4(*bmGs[5] = bmBc[c]-7+5*)

- Percobaan ketiga



Menggeser sebanyak 7 (*bmGs[0]*)

- Percobaan keempat



Menggeser sebanyak 4 (*bmGs[5]=bmBc[c]-7+5*)

- Percobaan kelima

Menggeser sebanyak 7 (*bmGs[6]*)

Algoritma Boyer-Moore melakukan 17 kali perbandingan karakter pada untai DNA tersebut. Tabel *bmBc* dan *bmGs* dapat di hitung sebelumnya dengan waktu *O(m+σ)* sebelum fase pencarian dan membutuhkan

ruang ekstra dalam $O(m+\sigma)$. Kompleksitas waktu pencariannya adalah kuadratik $(O(n^2))$ tapi paling banyak $3n$ perbandingan karakter string yang dilakukan saat pencarian untuk pattern yang tidak periodik.

3.KESIMPULAN

Masing-masing algoritma memiliki keunggulan masing-masing dalam mencari pattern DNA pada sebuah untai DNA. Algoritma *String Matching on Ordered Alphabet* mempunyai kompleksitas waktu terkecil $(O(n))$, karena tidak mempunyai fase pre-proses. Namun pada kasus terburuknya harus melakukan sebanyak $6n+5$ kali perbandingan karakter. Algoritma KMP mempunyai kompleksitas $O(m+n)$ dan paling banyak hanya melakukan $2n-1$ kali perbandingan karakter. Algoritma Boyer-Moore mempunyai kompleksitas waktu terburuk dengan $O(n^2)$ pada fase pencarian dan $O(m+\sigma)$ pada fase pre-proses. Jadi, pencarian untai DNA harus dilihat pada panjang pattern DNA yang akan dicocokkan dan panjang targetnya.

4.REFERENSI

[1] Munir, Rinaldi. "Strategi Algoritmik", Teknik Informatika ITB, 2007.

[2] Iliopoulos, Costas S. dan Thierry Lecroq, "String Algorithmics", King's College London Publications, 2001.