Perbandingan 3 Algoritma Pattern Matching dalam DNA *sequencing*

Muhammad Gema Akbar 13510099[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]mgemaakbar@gmail.com

***Abstract*—DNA *sequencing* adalah proses menentukan urutan dari nukleotida dalam molekul DNA. Hal ini mencakup semua metoda atau teknologi yang digunakan dalam menentukan urutan dari empat basa yaitu adenine, guanine, cytosine, dan thymine. Makalah ini membahas bagiamana peran algoritma *pattern matching* yaitu *brute-force, Knuth-Morris-Pratt,* dan Boyer-Moore, mulai dari metoda sampai analisis ketiga algoritma tersebut dari beberapa segi salah satunya adalah waktu eksekusi dan efisiensi.**

Kata Kunci: Pattern Matching, DNA sequening, brute force, Knuth-Morris Pratt, Boyer-Moore, bioinformatika

I. INTRODUCTION

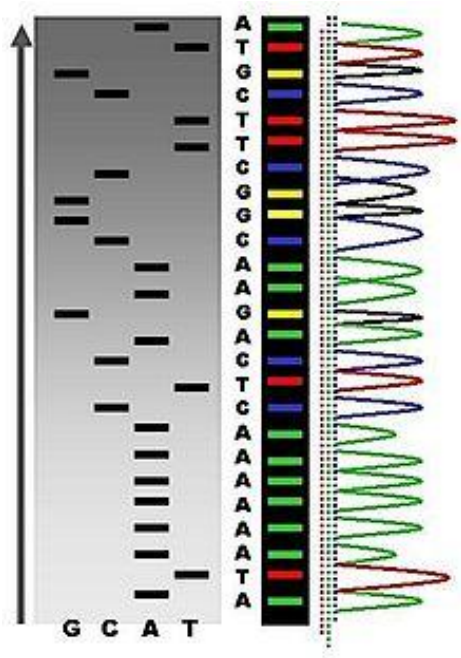
Bioinformatika adalah area penelitian interdisiplin yang berada diantara ilmu biologi dan ilmu komputasi. Tujuan utama dari bioinformatika adalah untuk mengetahui informasi biologis yang tersembunyi dalam data dan mendapatkan pengetahuan tentang ilmu biologi dari organisme, yang dapat berpengaruh pada bidang lain seperti kesehatan, agrikultur, lingkungan, energi dan bioteknologi. Ada banyak aplikasi lain dari bioinformatika, termasuk memprediksi rantai protein, mempelajari bagaimana ekspresi gen dalam banyak spesies, dan membangun model dari sel-sel. Seiring bertambahkuatnya imu komputasi dan meluasnya database dari informasi genetis dan molekuler, bidang bioinformatika secara drastis tumbuh dan berubah, membuat kita dapat membuat model dari banyak kompleksitas.

DNA adalah akronim dari *deoxyribonucleic acid*. DNA disebut double helix karena molekulnya berbentuk 2 *strand*. DNA mengandung 3 komponen: *deoxyribosa* (gula dengan 5 karbon), urutan dari fosfat, 4 basa nitrogen. Empat basa di DNA adalah adenin (A), thymine (T), guanine (G), dan cytosine (C). Deoxyribose menggabungkan fosfat untuk membentuk tulang punggung dari DNA. Thymine dan adenine selalu berpasangan, dan guanine dan cytosine basa juga selalu berpasangan.

Setiap manusia memliki gen yang unik. Gen terdiri dari DNA, maka urutan DNA dari manusia itu unik. Tapi, sebenarnya 99,9% dari urutan DNA adalah mirip, yang berarti perbedaannya hanyalah 0.1%. DNA terkandung dalam sel hidup dari suatu organisme, dan DNA adalah pembawa dari kode genetik suatu organisme. Karena organisme harus bereplikasi, maka harus ada kode genetik yang unik untuk melakukan itu. Kode genetik tersebut adalah informasi, yang akan dibutuhkan dalam pertumbuhan biologis dan keturunan.

Urutan DNA adalah representasi dari kode genetik yang terkandung dalam suatu organisme. Peneliti biologi molekul harus membandingkan bagian dari urutan DNA. Urutan DNA adalah representasi dari *string* nukleotida yang ada dalam strand di DNA. Misalnya: ATGCGATACAAGTTGTGA.

Istilah DNA *sequencing* mencakup metode biokimia yang digunakan untuk menentukan urutan dari basa nukleotida, adenine, guanine, cytosine, thymine, dalam DNA oligonukleotida. Urutan dari DNA membentuk informasi genetis yang dapat di turunkan yang menjadi dasar dalam program perkembangan adri semua organsime. Menentukan urutan DNA berguna dalam penelitian dasar dalam mempelajari proses fundamental biologis, dan jugabidang terapan seperti penelitian diagnostik atau forensik. Karena DNA adalah kunci dari organisme hidup, pengetahuan tentang urutan DNA berguna dalam hampir semua bidang di biologi. Contohnya, dalam ilmu medis pengetahuan ini dapat digunakan untuk mengenali, mendiagnosis, dan mengembangkan perawatan untuk penyakit genetis.



Banyak sel-sel kanker menunjukan banyak variasi dari alterasi genetik, mulai dari mutasi ber skala kecil hingga penyimpangan kromosom yang besar.

II. PENDETEKSIAN PENYAKIT

Ketika kita tahu bahwa urutan tertentu adalah sebab dari suatu penyakit, banyaknya urutan tersebut menentukan intensitas dari penyakit tersebut. Karena DNA memiliki database yang besar, kita membutuhkan algoritma yang efisien untuk mencari tahu urutan tertentu dari DNA. Kita harus mencari banyaknya pengulangan dan index awal dan index akhir dari urutan tersebut, yang dapat digunakan untuk diagnosis

III. PENCARIAN POLA

Dokumen teks banyak muncul di komputasi modern kaerna banyak digunakan untuk berkomunikasi dan dalam publikasi informasi. Dari sudut pandang algoritma, dokumen seperti itu dapat dilihat dianggap sebagai string dari karakter. Untuk melakukan pencarian dan pemrosesan dari data seperti itu membutuhkan metode yang efisien dalam menangani string karakter.

II.1Operasi String

Inti dari algoritma untuk pemrosesan teks adalah metode yang digunakan untuk string karakter. Karakter string dapat datang dari banyak sumber, termasuk aplikasi saintifik. Berikut adalah contoh dari string:

P="CGTAAACTGCTTTAATCAAACGC"

Beberapa operasi pemrosesan string melibatkan pembagian string menjadi string-string yang lebih kecil.

Masalah Pattern-Matching

Dalam masalah pattern-matching, misalnya kita memiliki teks string T dengan panjang *n* dan sebuah pattern string *P* dengan panjang *m*, and want to find whether *P* adalah substring dari *T*. Ide dari pencocokan adalah adanya substring dari *T* dimulai dari suatu index *i* yang cocok dengan *P*, karakter per karakter, agar

T[i] = P[0], T[i+1]=P[1], ...T[i+m-l]=P[m-l].

P = T[i...i+m-l]

Maka, output dari algoirtma patter-matching adalah bahwa pattern *P* tidak ada di T, atau jika ada, maka output adalah indeks di T dari substring yang cocok dengan *P*.

Contoh: Jika kita memiliki

T = “abacaabaccabacabaabb”

dan string pattern

P = "abacab"

Maka P adalah substring dari T. Yaitu P = T[10..15]

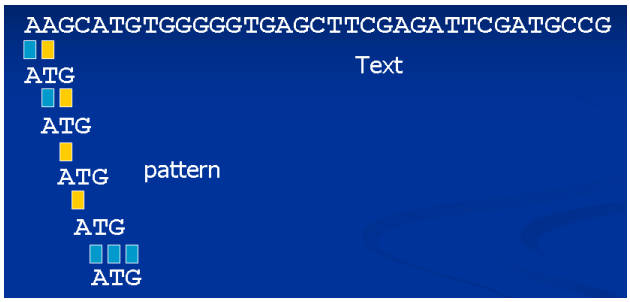
Ada beberapa algortima pattern-matching. Algoritma-algoritma tersebut dapat digunakan untuk menentukan sekuens dalam DNA dari gen, yaitu:

- Brute-Force
- Boyer-Moore
- Knuth-Morris – Pratt

Algoritma Brute Force

Algoritma brute force adalah teknik pencarian yang bersifat *brutal*, karena melibatkan cara dimana pergeseran pattern ketika pencocokan dilakukan secara satu persatu per-karakter yang ada di teks yang akan dibandingkan. Pergeseran pattern seperti ini sebenarnya membuat algoritma brute force menjadi kuat, tapi sering-dalam beberapa kasus pergeseran seperti ini hanya malah menjadi ‘boros’, karena banyak pergeseran yang sebenarnya tidak perlu.

Dalam kasus terburuk, brute force dapat memiliki kompleksitas waktu O(mn), dimana m adalah panjang pattern, dan n adalah panjang teks.

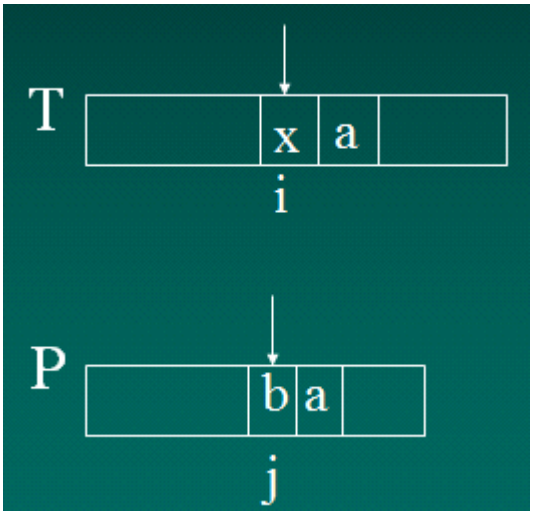


Algoritma Boyer-Moore

Algoritm pattern-matching Boyer-Moore dilakukan dengan 2 cara, yaitu:

1. Pencarian dilakukan dengan cara menelusuri karakter secara mundur, (tidak seperti yang dilakukan di brute force) yaitu dilakukan dari karakter paling belakang pattern.

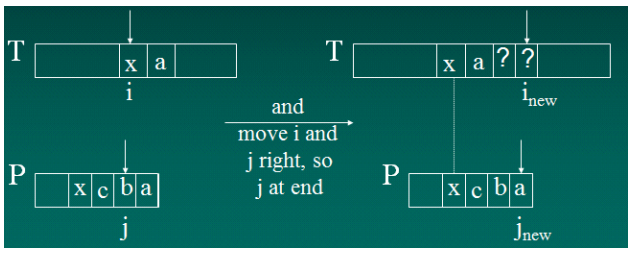
2. Ketika ketidakcocokan terjadi di T[i] == x, karakter P[j] tidak sama dengan T[i]



Setelah menghadapi ketidakcocokan seperti diatas, cara yang digunakan ada 3, bergantung dengan kasusnya seperti apa. Ketiga kasus (dan caranya) yaitu:

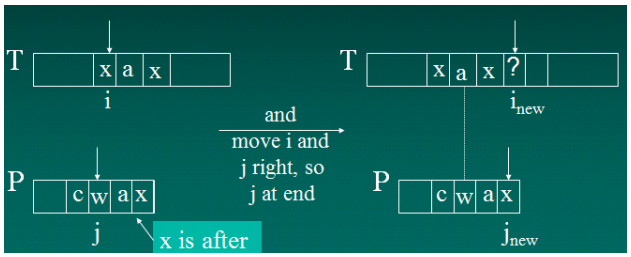
Kasus 1:

Jika P mengandung x disuatu bagian dari P, geser ke kanan untuk membuat kemunculan terakhir dari x dalam P segaris dengan T[i]



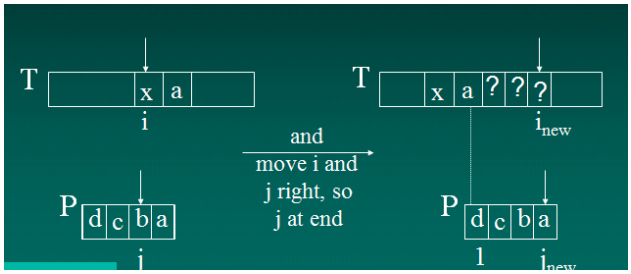
Kasus 2:

Jika kemunculan terakhir dari xdalam P tidak dapat dibuat segaris dengan T[i], maka geser P 1 karakter ke T[i+1]

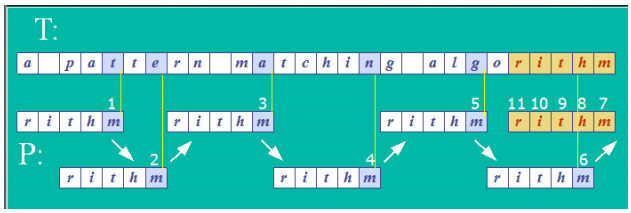


Kasus 3:

Jika kasus 2 dan kasus 1 tidak dapat dilakukan atur agar P[1] segaris dengan T[i+1]



Contoh



Waktu eksekusi kasus terburuk dari Boyer-Moore adalah O(mn+A) dimana m adalah panjang pattern dan n adalah panjang teks, dimana A adalah variasi karakter yang ada dalam teks/pattern. (yang dimaksud variasi disini misalnya adalah di angka biner, karena hanya ada 1 dan 0 maka A = 2)

Algoritma Knuth-Morris-Pratt

Dalam mempelajari performansi kasus terburuk dari brute-force dan boyer-moore pada satu contoh spesifik dari masalah ini , kita menemukan inefisiensi yang besar. Secara spesifik kita dapat melakukan banyak perbandingan ketika menguji pengganti potensial untuk pattern, tetapi jika kita menemukan karakter pattern yang tidak cocok dengan teks, lalu kita membuang semua informasi yang diperoleh dari perbandingan yang telah dilakukan dan mulai kembali dari awal dengan penggantian dari pola. Algoritma KMP menghindari pembuangan informasi seperti demikian sehingga dapat memiliki waktu eksekusi O(m+n) dimana m adalah panjang pattern dan n adalah panjang pattern.

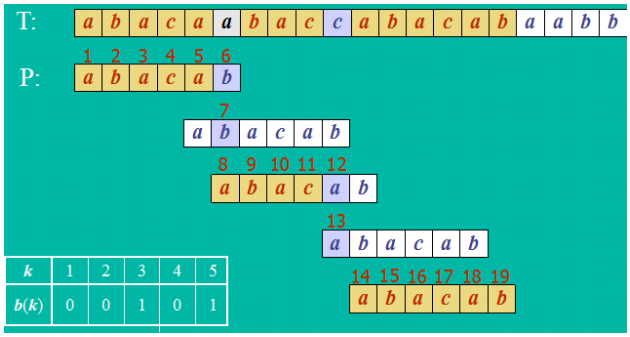
Failure Function

Ide utama dari algoritma KMP adalah untuk mempreproses pattern P untuk menghitung *failure function f* untuk menunjukan pergeseran dari P yang tepat sehingga kita dapat memanfaatkan perbandingan yang telah dilakukan. *Failure function* l(j) dapat didefiniskan sebagai panjang dari prefix terpanjang dari P yang juga adalah suffix dari P. Kita dapat menggunakaan konvensi bahwa l(0)=0. Pentingnya dari fungsi ini adalah untuk mengetahui dimana substring diulang didalam pattern.

```
F[0] <- 0
i <- 1
j <- 0
while i < m
       if P[i] = P[j]
              F[i] <- j + 1
              i <-i + 1
              j <- j + 1
       else if j > 0 then
              j <- F[j - 1]
       else
              F[i] <- 0
              i <- i + 1
```

Algoritma KMP satu persatu memoroses teks T dengan pattern P. Setiap ada kecocokan, indeks di-increment. Jika ada ketidakcocokan dan kita telah menemukan beberapa kecocokan sebelum ketidakcocokan terjadi, kita memanggil *failure function* untuk menentukan index baru di P dimana kita harus melanjutkan pengecekan P terhadap T. Jika sebelumnya tidak ada kecocokan sama sekali, naikan indeks untuk T dan indeks untuk P dibiarkan diawal. Kita mengulang proses ini sampai kita menemukan kecocokan dari P dalam T untuk T mencapai n, yaitu panjang dari T (menunjukan bahwa kita tidak menemukan P di T).

Bagian utama dari algoritma KMP adalah loop-while, yang melakukan perbandingan antara karakter di T dan karakter di P untuk setiap iterasi. Tergantung dengan hasil dari perbandingan ini, algortima dapat bergerak ke karaker selanjutnya dalam T dan P, atau memanggil *failure function* untuk mencari kandidat baru di P, atau mengulang semuanya di indeks selanjutnya dari T.



k	1	2	3	4	5
b(k)	0	0	1	0	1

```
Algorithm KMPMatch(T, P)
F <- failureFunction(P)
i <- 0
j <- 0
while i < n
       if T[i] = P[j]
              if j = m - 1
              return i - j { match }
              else
                     i <- i + 1
                     j <- j + 1
       else
              if j > 0
                     j <- F[j - 1]
              else
                     i <- i + 1
return (-1)
```

Dari ketiga algoritma diatas, berdasarkan waktu eksekusi, algoritma yang dapat melakukan pencarian pattern dengan efisien dalam kasus DNA sequencing adalah Algoritma Knuth-Morris-Pratt. Algoritma Brute force sudah jelas kekurangannya dalam efisiensi karena pengecekan yang terlalu boros. Algoritma Boyer-Moore memiliki waktu eksekusi O(mn+A) dan nilai A (variasi karakter) pada pencarian sekuens DNA bernilai 4. Algoritma Boyer-Moore bekerja semakin cepat jika nilai A adalah besar,tetapi karena pada kasus ini, nilai A hanya 4, tidak seperti kasus dalam pencarian string dalam teks seperti teks berita yang dapat melibatkan semua alfabet. Sedangkan waktu eksekusi dari algoritma Knuth-Morris-Pratt tidak tergantung dengan banyaknya variasi alphabet. Ditambah lagi, karena sekuens DNA memiliki panjang yang sangat besar, algoritma Knuth-Morris-Pratt dapat mengatasi keadaan seperti ini dibandingkan dengan Boyer-Moore.

V. CONCLUSION

Kesimpulannya adalah algoritma Knuth-Morris-Pratt memiliki performansi lebih baik dibandingkan kedua algoritma lainnya. Hal ini disebabkan karena sedikitnya jumlah alphabet pada DNA *sequencing* (hanya 4 huruf).

REFERENCES

[1] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/kuthMP.htm
[2] http://www.inf.fh-flensburg.de/lang/algorithmen/pattern/kmpen.htm
[3] http://www.ics.uci.edu/~eppstein/161/960227.html
[4] http://cs.indstate.edu/~kmandumula/presentation.pdf
[5] www.cs.utexas.edu/~**moore**/best-ideas/string-searching/index.html

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, 21 Desember 2012

ttd

Muhammad Gema Akbar
13510099