Penerapan Algoritma String Matching untuk Mendeteksi Penyakit dalam Darah

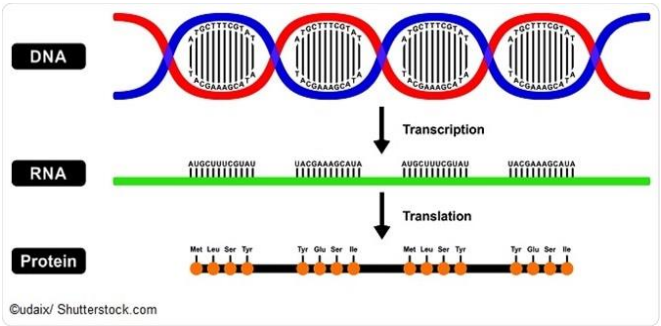
Vanson Kurnialim - 13522049[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522049@mahasiswa.itb.ac.id

***Abstract*—Darah memiliki banyak informasi penting yang mengindikasikan kondisi dan identitas dari seseorang. Salah satu informasi penting yang dapat diambil dari darah adalah protein. Di dalam tubuh seseorang terdapat banyak sekali protein dengan kebutuhan dan fungsinya masing-masing. Setiap protein dapat diubah menjadi sebuah rantai protein atau sekuens berdasarkan urutan asam amino pembentuknya. Penyakit dan kondisi yang ada pada tubuh manusia seringkali memiliki indikasi atau biasa disebut biomarker. Seluruh informasi inilah yang digunakan dalam mendeteksi penyakit dalam darah. Metode yang dapat digunakan untuk mencapai hal tersebut adalah metode *string matching* dengan berbagai algoritma di dalamnya.**

***Keywords*—Darah, Penyakit, Sekuens Protein, Biomarker, *String Matching*, algoritma.**

I. Latar Belakang

Di era kemajuan teknologi medis dan bioinformatika, analisis darah telah menjadi metode utama untuk diagnosis dan pemantauan berbagai penyakit. Hal ini dilakukan karena darah merupakan medium yang kaya akan informasi biologis. Ada banyak hal yang dapat diekstraksi dari sampel darah seseorang, seperti protein, DNA, RNA, hormon, metabolit, dan materi biomolekul lainnya. Seluruh data tersebut dapat digunakan sebagai indikator kesehatan ataupun tanda potensi penyakit seseorang.



Gambar 1. *Ilustrasi rangkaian DNA, RNA, dan rantai protein*
(sumber : https://www.news-medical.net/life-sciences/Amino-Acids-and-Protein-Sequences.aspx)

Sebuah penyakit, bakteri, mutasi, atau kondisi lainnya memiliki sebuah penanda di dalam darah, atau lebih tepatnya, di dalam salah satu informasi yang terkandung dalam darah. Cara kerja *blood test* adalah dengan menganalisa seluruh informasi tersebut dan mencari kemunculan-kemunculan dari penanda tersebut dalam darah pasien yang dites.

Dikarenakan banyaknya data yang perlu diuji dalam suatu sampel darah, maka diperlukan metode pengujian yang cepat, efisien, dan tepat agar kondisi pasien dapat terdiagnosis dengan baik.

Dalam makalah ini, kita akan membahas penggunaan metode *String Matching* dengan algoritma *Brute Force*, algoritma Knuth-Morris-Pratt atau KMP, dan algoritma Boyer-Moore untuk digunakan sebagai metode pengujian darah. Meninjau banyaknya jenis data yang dapat diekstrak dari darah, kita hanya akan membahas lebih dalam terkait deteksi penyakit berdasarkan rantai protein dalam darah.

II. Landasan Teori

A. *Darah*

Darah merupakan kombinasi dari plasma dan sel-sel yang beredar di seluruh tubuh. Cairan ini berperan sebagai pemasok komponen penting bagi tubuh seperti gula, oksigen, dan hormon menuju sel dan organ dala tubuh. Darah juga berfungsi sebagai pertahan tubuh terhadap virus atau bakteri yang berbahaya bagi tubuh serta membawa limbah bahan kimi hasil metabolisme tubuh.

Pemeriksaan darah atau tes darah digunakan untuk menganalisa sel, bahan kimia, protein, atau zat lain dalam darah. Melalui tes darah, dokter dapat mendiagnosis penyakit dan memantau penyakit serta mencari jenis pengobatan yang tepat. Berikut ini adalah beberapa contoh jenis tes darah :

1. *Complete Blood Count (CBC)*
   Tes ini mengukur berbagai bagian darah, termasuk sel darah merah dan putih, trombosit, dan hemogoblin. Fokus dari pengujian ini adalah komposisi dan proporsi dari bagian-bagian dalam darah tersebut.
2. Panel Metabolisme Dasar
   Dilakukan sekelompok tes untuk mengukur bahan kimia tertentu dalam darah, termasuk glukosa, kalsium, dan elektrolit.
3. Tes Spesifik Lain
   Selain jenis-jenis pengujian darah umum seperti yang dijelaskan di atas dan banyak lagi, terdapat tes-tes darah yang dikhususkan untuk mendeteksi suatu kelainan, penyakit, ataupun mutasi yang dimiliki seseorang. Misal tes pengujian penyakit *Cystic Fibrosis*, salah satu tes yang dilakukan adalah pengambilan sampel darah pada bayi baru lahir untuk

mengukur suatu zat kimia.

Segala pengujian ini dapat dilakukan karena darah mempunyai berbagai informasi biologis yang dapat dianalisis secara mendalam. Selain itu, suatu penyakit ataupun kondisi lainnya mempunyai indikasi biologis atau *biomarker*.

B. *Biomarker*

Biomarker adalah molekul biologis yang ditemukan dalam darah, cairan tubuh, atau jaringan yang merupakan tanda dari suatu proses yang normal ataupun abnormal, termasuk suatu kondisi atau penyakit. Terdapat beberapa jenis biomarker seperti berikut :

1. Biomarker Kerentanan

   Menunjukkan potensi berkembangnya suatu penyakit atau kondisi medis pada seseorang yang saat ini tidak memiliki penyakit atau kondisi medis yang tampak secara klinis. Sering juga disebut sebagai biomarker risiko. Contohnya adalah *Germline* BRCA1 PV yang menginformasikan tentang risiko kanker di masa depan.
2. Biomarker Diagnostik

   Digunakan untuk mendeteksi atau memastikan keberadaan suatu penyakit atau kondisi tertentu atau mengidentifikasi individu dengan subtype penyakit tersebut. Sebagai contoh ekspresi HER2 pada kanker payudara.

Terdapat berbagai biomarker lain seperti biomarker jenis pemantauan, prognostik, prediktif, farmakodinamik, dan keamanan namun tidak dijelaskan lebih lanjut.

C. *Protein Sequence*

Protein adalah molekul yang dapat ditemukan di setiap sel-sel tubuh termasuk di dalam darah. Protein terbuat dari ratusan hingga ribuan senyawa kecil yang disebut asam amino. Asam amino ini sering disebut sebagai struktur primer protein dan disusun dalam urutan linier tertentu.

Pengurutan protein adalah teknik dasar dalam biologi molekuler dan biokimia yang melibatkan penentuan urutan asam amino yang tepat dalam molekul protein. Dengan demikian, protein atau kumpulan protein dapat direpresentasikan sebagai sebuah *string* yang panjang.

Berikut adalah informasi terkait konversi asam amino menjadi rantai protein :

IUPAC amino acid code	Three letter code	Amino acid
A	Ala	Alanine
C	Cys	Cysteine
D	Asp	Aspartic Acid
E	Glu	Glutamic Acid
F	Phe	Phenylalanine
G	Gly	Glycine
H	His	Histidine
I	Ile	Isoleucine
K	Lys	Lysine
L	Leu	Leucine
M	Met	Methionine
N	Asn	Asparagine
P	Pro	Proline
Q	Gln	Glutamine
R	Arg	Arginine
S	Ser	Serine
T	Thr	Threonine
V	Val	Valine
W	Trp	Tryptophan
Y	Tyr	Tyrosine

Gambar 2. *Tabel konversi asam amino*
(https://www.bioinformatics.org/sms/iupac.html)

D. *Brute Force*

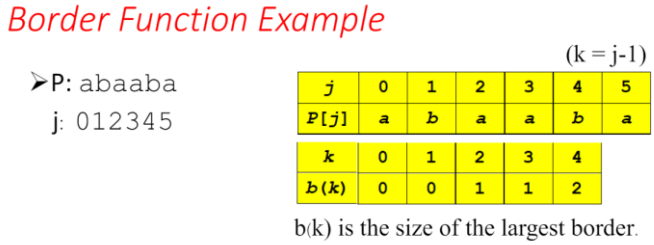
Brute Force adalah suatu algoritma *exhaustive search* atau pencarian sampai habis dan merupakan algoritma dengan pendekatan pencarian yang paling simpel. Sesuai dengan namanya, algoritma ini akan mencari dengan ‘paksa’ dengan mencari satu persatu semua kemungkinan yang mungkin. Dalam kasus *string matching*, algoritma ini membandingkan sebuah *pattern* kepada sebuah *string* mulai dari karakter pertama sampai karakter terakhir secara berurutan sampai semua karakter telah dilakukan perbandingan.

E. *Knuth-Morris-Pratt*

Algoritma Knuth-Morris-Pratt (KMP), merupakan metode pencarian kata atau string matching yang efisien untuk menemukan suatu substring (pola) di dalam string dengan variasi karakter sedikit. KMP menghindari pencocokan karakter yang sudah diketahui akan gagal, sehingga mengurangi jumlah perbandingan yang diperlukan. Memanfaatkan properti dari prefiks dan sufiks dari pola yang sedang dicari, algoritma ini dapat menentukan banyak karakter yang dilewati menggunakan border function.

Algoritma string matching Knuth-Morris-Pratt (KMP) memeriksa *string* dari kiri ke kanan. Jika karakter yang diperiksa match, algoritma akan memeriksa karakter selanjutnya sampai akhir string. Jika tidak, akan dicari panjang prefix yang merupakan suffix dari substring kumpulan karakter *pattern* yang telah diperiksa. Lalu *pattern* akan digeser satu kali ke kanan beserta tambahan pergeseran sejumlah panjang prefix jika ada. Pemeriksaan selanjutnya akan dimulai dari karakter setelah prefix.

Border function adalah fungsi untuk mencari besar prefix terbesar yang sama dengan suffixnya. Dengan menggunakan fungsi ini, maka dapat disimpan informasi *border* untuk tiap prefix yang mungkin dalam sebuah *pattern*. Berikut adalah ilustrasi terkait border function :



Gambar 3. *Ilustrasi penggunaan border function dalam bentuk tabel*
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

F. *Boyer-Moore*

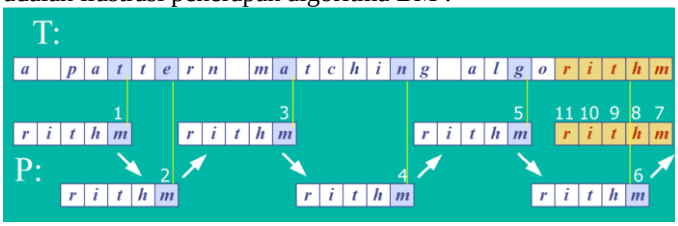
Algoritma Boyer-Moore (BM), merupakan metode pencarian kata atau *string matching* yang efisien dengan jumlah variasi karakter cukup banyak, misalnya pada bahasa Inggris. Algoritma ini menggunakan fungsi heuristic karakter baik dan fungsi heuristic karakter buruk. Ketika terjadi

ketidakcocokan karakter, algoritma Boyer-Moore menggunakan informasi dari karakter yang tidak cocok untuk menentukan seberapa jauh *pattern* dapat digeser ke kanan. Jika terjadi ketidakcocokan setelah beberapa karakter cocok, pola dapat digeser ke kanan berdasarkan bagian dari pola yang sudah cocok. Algoritma ini melakukan pergeseran yang memastikan bahwa sufiks yang cocok terus digunakan untuk pencocokan berikutnya.

Algoritma ini memeriksa karakter mulai dari karakter paling kanan, bergerak menuju karakter sebelah kirinya. Jika karakter yang diperiksa *match* maka karakter yang diperiksa akan terus bergeser ke kiri. Teknik ini disebut dengan *looking-glass technique.*

Terdapat teknik lain dalam algoritma ini yaitu teknik *character-jump.* Ada 3 jenis penanganan yang berbeda untuk teknik tersebut. Penanganan pertama adalah ketika sebuah karakter *mismatch* atau tidak sama, terdapat *last occurrence* dari karakter pada *string* yang sedang dibandingkan di sebelah kiri karakter *pattern* yang sedang dibandingkan. *Pattern* akan digeser ke kanan hingga karakter *last occurrence* tersebut berada pada posisi karakter *string* yang dibandingkan. Karakter *last occurrence* adalah karakter tersebut yang muncul terakhir kali pada *pattern*. Misal sebuah *pattern* "informatika", maka *last occurrence* dari karakter 'i' berada pada indeks ke-8.

Penanganan kedua adalah ketika kejadian yang sama terjadi, namun *pattern* tidak mungkin digeser hingga karakter *last occurrence* karena karakter tersebut berada di bagian kanan karakter *pattern* yang sedang dibandingkan. Maka *pattern* hanya akan digeser ke kanan sekali. Penanganan terakhir adalah ketika karakter *string* yang dibandingkan tidak ada dalam *pattern*. Maka jika karakter *string* yang dibandingkan berada pada indeks i, *pattern* akan digeser hingga karakter pertamanya berada pada indeks i + 1. Berikut adalah ilustrasi penerapan algoritma BM :



Gambar 4. *Ilustrasi penerapan algoritma BM* (sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

G. CFTR dan CBAVD

Cystic Fibrosis Transmembrane Conductance Regulator (CFTR) merupakan sebuah protein yang terdiri dari 1480 asam amino. Protein CFTR adalah jenis protein tertentu yang disebut saluran ion. Saluran ion menggerakkan atom atau molekul yang bermuatan listrik dari dalam sel ke luar, atau dari luar sel ke dalam. Di paru-paru, saluran ion CFTR menggerakkan ion klorida dari dalam sel ke luar sel. Untuk keluar dari sel, ion klorida bergerak melalui bagian tengah tabung yang dibentuk oleh protein CFTR.

Semua orang memiliki protein ini dalam tubuhnya, namun pada beberapa orang, gen pembawa instruksi membuat protein CFTR tidak normal atau memiliki mutasi gen. Tentunya hal ini memberikan efek samping bagi tubuh manusia. Salah satu contoh mutasi gen CFTR adalah hilangnya rantai protein pada urutan sekuens 404-464, lebih tepatnya rantai berikut ini :

```
GFGELFEKAKQNNNNRKTSNGDDSLFFSNFSLLGTPV
LKDINFKIERGQLLAVAGSTGAGK
```

Gambar 5. *Sekuens yang hilang pada protein CFTR* (sumber : dokumen pribadi)

Manifestasi klinikal yang utama akibat hilangnya sekuens protein tersebut adalah infertilitas, yang mencakup 1-2% kasus infertilitas pria. Mutasi ini disebut sebagai *Congenital bilateral absence of the vas deferens* (CBAVD) yang biasanya mengikuti mutasi protein CFTR lain.

III. Analisis dan Implementasi

A. Sekuens Protein dan Biomarker Mutasi CFTR

Pada kesempatan ini, akan dilakukan pengujian deteksi biomarker mutasi gen CFTR yang menyebabkan CBAVD, pada sebuah sekuens protein CFTR. Berikut adalah cuplikan rantai protein CFTR normal dengan panjang 1480 dan biomarker yang digunakan dengan panjang 10 :

```
MQRSPLEKASVVSKLFFSWTRPILRKGYRQRLELSDI
YQIPSVDSADNLSEKLEREWDRELASKKNPKLINALR
RCFFWRFMFYGIFLYLGEVTKAVQPLLLGRIIASYDP
DNKEERSIAIYLGIGLCLLFIVRTLLLHPAIFGLHHI
GMQMRIAMFSLIYKKTLKLSSRVLDKISIGQLVSLLS
NNLNKFDEGLALAHFVWIAPLQVALLMGLIWELLQAS
AFCGLGFLI...
```

Gambar 6. *Sekuens protein CFTR normal* (sumber : dokumen pribadi)

```
AFWEETSLLM
```

Gambar 7. *Biomarker mutasi gen CBAVD pada protein CFTR* (sumber : dokumen pribadi)

Segala informasi terkait sekuens protein CFTR beserta mutasinya dapat ditinjau pada laman *UniProt* dengan kode entri P13569.

B. Penerapan Kode

Pada implementasi ini, akan diuji penggunaan algoritma KMP, BM, dan Brute Force untuk mendeteksi biomarker pada sekuens protein CFTR dan algoritma manakah yang paling efisien. Bahasa pemrograman yang digunakan untuk membuat program ini adalah C#.

```
public static bool BFSolve(string sequence, string pattern)
{   StringBuilder seq = new StringBuilder(sequence) ;
    StringBuilder pat = new StringBuilder(pattern) ;
    int len_seq = seq.Length ;
    int len_pat = pat.Length ;

    for (int i = 0 ; i < len_seq ; i++) {
        for (int j = 0 ; j < len_pat ; j++) {
            count += 1 ;
            if (seq[i+j] == pat[j]) {
                if (j == len_pat - 1) {
                    return true ;
                }
                continue ;
            }
            else {
                break ;
            }
        }
    }
    return false ;
}
```

Gambar 8. *Implemetasi Brute Force*
(sumber : dokumen pribadi)

Program ini berjalan dengan sangat sederhana. Dilakukan iterasi bersarang untuk tiap karakter pada sekuens dan juga pada *pattern* atau biomarker. Selama karakter yang dibandingkan tidak sama, maka karakter akan terus digeser sesuai urutan.

```
public static bool KMPSolve(string sequence, string pattern) {
    int[] border_list = makeBorder(pattern) ;
    StringBuilder src = new StringBuilder(sequence) ;
    StringBuilder sub = new StringBuilder(pattern) ;
    int len_src = src.Length ;
    int len_sub = sub.Length ;

    if (len_src <= len_sub) {
        return false ;
    }

    int t_index = 0 ;
    int size = 0 ;
    while (t_index <= len_src-len_sub) {
        int p_index ;
        for (p_index = size ; p_index < len_sub ; p_index++) {
            if (p_index == len_sub - 1 && sub[p_index] == src[t_index+p_index]) {
                count += 1 ;
                return true ;
            }
            else {
                count += 1 ;
                if (sub[p_index] == src[t_index+p_index]) {
                    continue ;
                }
                else {
                    int border = border_list[p_index] ;
                    if (p_index == 0) {
                        t_index += 1 ;
                    }
                    else {
                        t_index += p_index - border ;
                        size = border ;
                    }
                    break ;
                }
            }
        }
    }
    return false ;
}
```

Gambar 9. *Implemetasi KMP* (sumber : dokumen pribadi)

Program ini diimplementasikan menurut alur algoritma KMP pada umumnya. Untuk mendukung performa waktu program, tabel *border* atau *border function* disiapkan di awal program agar fungsi *border* tidak perlu dipanggil berulang-ulang sepanjang keberjalanan program. Berikut adalah cuplikan kode terkait yang digunakan :

```
static int[] makeBorder(string word) {
    StringBuilder sb = new StringBuilder(word) ;
    int len_sb = sb.Length ;
    int[] border = new int[len_sb];
    for (int i = 0 ; i < len_sb ; i++) {
        border[i] = BorderFunction(sb.ToString(0,i)) ;
    }
    return border ;
}
```

Gambar 10. *Make border function result into an array*
(sumber : dokumen pribadi)

```
static int BorderFunction(string word) {
    if (word.Length <= 1) {
        return 0 ;
    }
    else {
        StringBuilder prefix = new StringBuilder(word) ;
        prefix.Remove(prefix.Length - 1 , 1) ;
        StringBuilder sufix = new StringBuilder(word) ;
        sufix.Remove(0,1) ;
        while (prefix.Length > 0) {
            if (prefix.Equals(sufix)) {
                return prefix.Length ;
            }
            else {
                prefix.Remove(prefix.Length - 1 , 1) ;
                sufix.Remove(0,1) ;
            }
        }
        return 0 ;
    }
}
```

Gambar 11. *Border Function* (sumber : dokumen pribadi)

Program *string matching* dengan algoritma BM yang diimplementasikan memiliki sedikit perbedaan dibanding alur algoritma BM pada umumnya. Jika biasanya pergeseran hanya dilakukan dengan fokus pada *last occurrence* atau kemunculan karakter yang terakhir, program ini mengoptimasi jumlah perbandingan agar lebih sedikit. Perbedaannya berada pada pencarian *last occurrence*, dimana selama terdapat karakter target pada bagian kiri karakter perbandingan, walaupun karakter tersebut bukan merupakan *last occurrence*, *pattern* akan tetap digeser sehingga karakter tersebut berada pada posisi perbandingan.

```
public static bool BMSolve(string sequence, string pattern) {
    StringBuilder seq = new StringBuilder(sequence) ;
    StringBuilder pat = new StringBuilder(pattern) ;
    int len_seq = seq.Length ;
    int len_pat = pat.Length ;

    int index = len_pat - 1 ;

    while (index <= len_seq - 1) {
        int pat_index = len_pat - 1 ;
        for (int i = 0 ; i < len_pat ; i++) {
            count += 1 ;
            if (seq[index - i] == pat[pat_index - i] && i == len_pat - 1) {
                return true ;
            }
            if (seq[index - i] != pat[pat_index - i]) {
                int char_in = isIn(pat.ToString(0, len_pat - i), seq[index - i])
;
                if (char_in != -1) {
                    index += len_pat - i - char_in - 1 ;
                    break ;
                }
                char_in = isIn(pat.ToString(len_pat - i, i), seq[index - i]) ;
                if (char_in != -1) {
                    index += 1 ;
                    break ;
                }
                else {
                    index += len_pat - i ;
                    break ;
                }
            }
        }
    }
    return false ;
}
```

Gambar 12. *Implementasi BM* (sumber : dokumen pribadi)

C. Hasil Implementasi dan Analisis

Berikut adalah tampilan hasil pengujian menggunakan sekuens protein CFTR yang bermutasi.

```
PS C:\ITB\Akademik\Tingkat 2\sem 2\Stima\Makalah> ./BF
|| Using Brute Force Algorithm ||
Biomarker ditemukan pada sequence!
Total perbandingan : 437
Execution time : 2.1098 ms

PS C:\ITB\Akademik\Tingkat 2\sem 2\Stima\Makalah> ./KMP
|| Using KMP Algorithm ||
Biomarker ditemukan pada sequence!
Total perbandingan : 435
Execution time : 3.1058 ms

PS C:\ITB\Akademik\Tingkat 2\sem 2\Stima\Makalah> ./BM
|| Using BM Algorithm ||
Biomarker ditemukan pada sequence!
Total perbandingan : 62
Execution time : 2.7537 ms
```

Gambar 13. *Hasil implementasi dengan sekuens bermutasi* (sumber : dokumen pribadi)

Berikut adalah tampilan hasil pengujian menggunakan sekuens protein CFTR yang normal.

```
PS C:\ITB\Akademik\Tingkat 2\sem 2\Stima\Makalah> ./BF
|| Using Brute Force Algorithm ||
Biomarker tidak ditemukan pada sequence!
Total perbandingan : 1574
Execution time : 1.9745 ms

PS C:\ITB\Akademik\Tingkat 2\sem 2\Stima\Makalah> ./KMP
|| Using KMP Algorithm ||
Biomarker tidak ditemukan pada sequence!
Total perbandingan : 1554
Execution time : 2.718 ms

PS C:\ITB\Akademik\Tingkat 2\sem 2\Stima\Makalah> ./BM
|| Using BM Algorithm ||
Biomarker tidak ditemukan pada sequence!
Total perbandingan : 195
Execution time : 2.5248 ms
```

Gambar 14. *Hasil implementasi dengan sekuens normal* (sumber : dokumen pribadi)

Berdasarkan hasil pengujian di atas, terlihat biomarker berhasil ditemukan pada sekuens protein mutasi dan tidak ditemukan pada sekuens protein normal. Hal ini menunjukkan program atau algoritma *string matching* ini berhasil sesuai dengan hasil yang diperkirakan, setidaknya pada kasus deteksi penyakit CBAVD.

Tujuan dari pembuatan makalah ini adalah untuk mengimplementasikan algoritma *string matching* dalam kasus pendeteksian penyakit atau kondisi dalam darah dan juga menentukan algoritma mana yang paling tepat. Oleh karena itu, terdapat 2 aspek yang dapat kita bandingkan untuk ketiga algoritma tersebut, yaitu waktu eksekusi dan total perbandingan. Perbandingan yang dimaksud di sini adalah perbandingan atau komparasi karakter dari sekuens dan karakter dari biomarker.

Meninjau waktu eksekusi tiap algoritma yang digunakan, Brute Force merupakan algoritma yang paling cepat dan memiliki perbedaan waktu yang cukup jauh dari algoritma lainnya, dengan algoritma BM *modified* kedua dan KMP paling lambat. Namun aspek waktu eksekusi ini diragukan sebagai aspek penilaian yang valid karena kurangnya perbedaan waktu dari pencarian sekuens normal dan sekuens mutasi. Panjang sekuens normal adalah 1480 karakter dan sekuens mutasi sepanjang 1410 karakter karena sekuens yang hilang akibat mutasi sebanyak 70 karakter. Pada kasus protein mutasi, biomarker ditemukan kurang lebih pada karakter ke-400, sehingga karakter yang dilewati berjumlah sekitar 400 karakter. Berbeda dengan pengujian pada sekuens protein normal dimana tidak ditemukan biomarker, pencarian dilakukan sampai sekuens habis atau karakter yang dilewati merupakan keseluruhan sekuens. Menimbang terdapat hingga 3x lipat perbedaan jumlah karakter yang dilewati, walaupun tidak berbanding lurus dengan waktu eksekusi, setidaknya diperkirakan adanya selisih waktu yang cukup antara kedua kasus.

Diduga adanya waktu *overhead* dari sistem yang menyebabkan waktu eksekusi tidaklah murni waktu pencarian. Berdasarkan kecurigaan ini, telah dilakukan tes pengujian dengan sekuens panjang 1 dan biomarker sepanjang 1 agar hanya perlu dilakukan satu kali perbandingan. Sesuai dengan prediksi, hasil eksekusi program kurang lebih sama dengan percobaan-percobaan sebelumnya. Maka dari itu, diperlukan

sekuens dengan jumlah karakter yang lebih banyak secara signifikan agar mendapatkan hasil eksekusi yang layak untuk menentukan performa dari algoritma.

Berdasarkan konsiderasi tersebut, waktu eksekusi tidak dianggap sebagai aspek penilaian algoritma. Mungkin hal ini terjadi akibat kualitas perangkat yang digunakan atau bahasa pemrograman C# yang digunakan.

Melihat aspek penilaian yang tersisa, total perbandingan yang paling sedikit dimiliki oleh algoritma Boyer-Moore, baik pada protein normal ataupun mutasi. Total perbandingan akan terlihat kontras semakin panjang karakter yang dilewati, seperti pada gambar 10. Hal ini sesuai dengan perkiraan karena algoritma BM-*modified* ini dibuat secara khusus untuk meminimalisir jumlah perbandingan. Maka, algoritma dengan efektivitas pencarian tertinggi berdasarkan total perbandingan sudah jelas adalah BM, dengan KMP dan BF berada pada tingkat efektivitas yang tidak jauh berbeda.

IV. KESIMPULAN

Algoritma *string matching* merupakan metode yang sangat bermanfaat terutama pada bidang kesehatan dan bioinformatika. Mendeteksi penyakit dan kondisi melalui darah merupakan cara yang efektif dalam diagnosis kesehatan pasien. Dari tiga algoritma Brute Force, Knuth-Morris-Pratt, dan Boyer-Moore, yang paling sesuai untuk digunakan sebagai metode pencarian biomarker pada sebuah sekuens protein adalah Boyer-Moore yang telah dimodifikasi seperti yang telah diuraikan di makalah ini. Algoritma ini sudah dilengkapi dengan proses penggeseran karakter yang efektif untuk mengurangi perbandingan karakter yang tidak perlu. Algoritma ini juga dapat mengatasi panjang sekuens protein yang panjang dengan jumlah perbandingan yang tetap minimal.

Metode-metode tersebut dapat digunakan untuk membantu proses deteksi penyakit dan kondisi pada manusia. Namun masih terdapat beberapa hal yang dapat dikembangkan lagi untuk meningkatkan akurasi dan efektivitas waktu.

V. PENUTUP

Terima kasih dan puji syukur penulis haturkan sebesar-besarnya kepada Tuhan yang mahakuasa, atas berkat dan rahmatnya, penulis diperbolehkan untuk menyelesaikan makalah ini tepat waktu dan dengan baik. Penulis juga mengucapkan terimakasih kepada dosen mata kuliah Matematika Diskrit, Dr. Ir. Rinaldi Munir, M. T., atas bimbingan dan pengajarannya selama di kelas. Tidak lupa penulis juga berterimakasih lagi kepada dosen Dr. Ir. Rinaldi Munir, M. T., karena telah memfasilitasi penulis dengan *website* berisi materi-materi sehingga dapat dijadikan bahan pembelajaran. Terima kasih juga kepada seluruh pihak yang karyanya penulis jadikan referensi dalam pembuatan makalah ini.

Penulis mengucapkan permintaan maaf apabila ada kesalahan dalam penulisan makalah ini.

REFERENCES

[1] Munir, R. (2021). Pencocokan String. Diakses Juni 12, 2024, pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

[2] Siloam Hospitals. (n.d.). Apa Itu Cystic Fibrosis. Diakses June 12, 2024, pada https://www.siloamhospitals.com/informasi-siloam/artikel/apa-itu-cystic-fibrosis

[3] News Medical. (n.d.). Amino Acids and Protein Sequences. Diakses June 12, 2024, pada https://www.news-medical.net/life-sciences/Amino-Acids-and-Protein-Sequences.aspx

[4] Halodoc. (n.d.). Darah. Diakses June 12, 2024, pada https://www.halodoc.com/kesehatan/darah

[5] Oncology Nursing Society. (n.d.). Biomarkers. Diakses June 12, 2024, pada https://www.ons.org/genomics-taxonomy/biomarkers

[6] Creative Proteomics. (n.d.). Protein Sequencing: Significance, Methods, Applications. Diakses June 12, 2024, pada https://www.creative-proteomics.com/resource/protein-sequencing-significance-methods-applications.htm

[7] PubMed Central. (2022). Article PMC8873976. Diakses June 12, 2024, pada https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8873976/

[8] UniProt. (n.d.). UniProt: The Universal Protein Resource. Diakses June 12, 2024, pada https://www.uniprot.org

[9] Bioinformatics.org. (n.d.). IUPAC Codes for Nucleic Acid. Diakses June 12, 2024, pada https://www.bioinformatics.org/sms/iupac.html

Pranala YouTube : https://youtu.be/I6EGdHme3y8

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, 10 Juni 2024

Vanson Kurnialim
13522049