Perbandingan Efisiensi Algoritma Boyer-Moore dan Knuth-Morris-Pratt dalam Deteksi Pola Genetik Huntington's Disease

Fityatul Haq Rosyidi - 13523116[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1] 13523116@std.stei.itb.ac.id FityatulHaqRosyidi25@gmail.com

***Abstract*— Pencarian pola dalam data genetik merupakan langkah penting dalam deteksi dini dan analisis penyakit, termasuk Huntington's Disease. Algoritma Boyer-Moore (BM) dan Knuth-Morris-Pratt (KMP) adalah dua metode populer untuk pencocokan pola pada string panjang . Penelitian ini bertujuan membandingkan efisiensi kedua algoritma dalam mendeteksi pola genetik spesifik terkait Huntington's Disease pada data DNA yang panjang dan kompleks. Eksperimen dilakukan dengan menganalisis waktu eksekusi yang diperlukan algoritma KMP dan BM dalam mendeteksi pola genetik pada Huntington's Disease**

***Keywords*— Knuth-Morris-Prath Algorithm, Boyer-Moore Algorithm, Huntington's Disease**

I. PENDAHULUAN

Algoritma pencocokan string merupakan salah satu instrumen penting dalam analisis data tekstual dan biologis. Dengan kemampuan untuk menemukan pola dalam teks atau data kompleks, algoritma ini memiliki peran esensial dalam pengolahan dokumen, pencarian teks, dan bioinformatika. Dalam bidang bioteknologi, khususnya dalam analisis urutan DNA, algoritma pencocokan string memungkinkan identifikasi variasi genetik dan deteksi mutasi yang menjadi penyebab penyakit.

Penyakit Huntington's Disease (HD) adalah contoh nyata bagaimana analisis urutan DNA digunakan untuk memahami patologi genetik. HD merupakan penyakit neurodegeneratif yang diturunkan secara autosomal dominan akibat mutasi gen HTT. Mutasi ini ditandai oleh ekspansi berlebihan sekuens CAG, yang menyebabkan produksi protein huntingtin yang toksik bagi neuron.

II. LANDASAN TEORI

A. Algortima Pencocokan String

Algoritma Pencocokan String adalah algoritma yang berfokus pada metode untuk menemukan keberadaan dan posisi subtring tertentu di dalam sebuah string yang lebih besar. Konsep ini mendasar dalam berbagai aplikasi pencarian teks, pengolahan dokumen, analisis genomik, dan pengenalan pola. Tujuan pencocokan string adalah untuk menemukan dengan cepat kemunculan string pola dalam teks yang lebih besar, memungkinkan hingga beberapa ketidakcocokan atau perbedaan [1]. Pendekatan dasar dalam pencocokan string menggunakan perbandingan karakter satu per satu, tetapi metode ini menjadi tidak efisien ketika dihadapkan dengan string yang sangat panjang. Oleh karena itu, pengembangan algoritma yang lebih efisien masih menjadi topik pembahasan dan penelitian yang menarik bagi para *computer scientist* di seluruh dunia.

Beberapa algoritma pencocokan string yang terkenal meliputi algoritma Brute-Force, Knuth-Morris-Prath (KMP), Boyer-Moore (BM), dan algoritma Rabin-Karp. Algoritma Brute-Force bekerja dengan membandingkan substring dengan setiap kemungkinan posisi pada string target. Meskipun sederhana, algoritma Brute-Force membutuhkan waktu yang signifikan dalam skenario terburuk. Algoritma KMP menggunakan ide prefix table atau partial mach table untuk memperkecil jumlah operasi perbandingan, sehingga meningkatkan efisiensi pencocokan. Sementara itu, Boyer-Moore menggunakan konsep teknik looking-glass dan character-jump [2] dan melakukan perbandingan dimulai dari karakter paling kanan. Adapun algoritma Rabin-Karp menggunakan teknik hashing untuk membandingkan pola dengan subtring target secara cepat, meskipun perlu penanganan untuk menghindari tabrakan hash.

Keefektifan algoritma-algoritma tersebut sangat bergantung pada ukuran pola, panjang string, dan konteks aplikasinya. Misalnya untuk pencarian teks pendek, algoritma Brute-Force mungkin sudah cukup, tetapi untuk aplikasi seperti pencarian genom atau data dalam skala besar, algoritma yang lebih canggih seperti Boyer-Moore atau Rabin-Karp diperlukan untuk mengurangi waktu eksekusi. Misalnya juga pada konteks pencocokan pola pada string dengan karakter yang heterogen dan beragam seperti bahasa manusia, algoritma Boyer-Moore lebih unggul daripada KMP. Sedangkan untuk pencocokan pola pada string yang memiliki banyak pengulangan, algoritma KMP akan lebih efisien dari Boyer-Moore.

B. DNA Sequence

DNA Sequence atau Urutan DNA adalah urutan nukleotida yang membentuk struktur genetik mahkluk hidup

[3]. Terdiri dari empat basa nitrogen utama: Adenin(A), Timin (T), Guanin (G), dan Siitosin (C). Urutan ini membentuk struktur primer DNA yang menyimpan informasi biologis penting. Nukleotida tersusun dalam rantai polinukleotida, dimana setiap nukleotida terdiri dari gula deoksiribosa, gugus fosfat, dan basa nitrogen.

Pada ilmu Bioteknologi, urutan DNA digunakan untuk mempelajari variasi genetik dalam populasi, mengidentifikasia gen yang menyebabkan munculnya penyakit tertentu, dan mengembangkan terapi gen untuk untuk mengobati kondisi genetik. Dalam bidang forensik, urutan DNA mengidentifikasi individu melalui analisis genetik pada sidik jari.

C. Huntington's Disease

Huntington's Disease (HD) adalah penyakit neurodegeneratif yang ditandai oleh kerusakan progresif pada sel-sel saraf otak, khususnya di area stratium dan cerebral cortex. Penyakit ini memengaruhi gerakan, kognisi, dan emosi serta memiliki sifat herediter yang autosomal dominan, artinya seseorang hanya membutuhkan satu salinan gen yang bermutasi dari salah satu orang tua untuk mewarisi penyakit ini.

Huntington's Disease disebabkan oleh mutasi pada gen HTT yang terletak di kromosom 4. Gen ini mengkode protein huntingtin yang berperan dalam fungsi neuron. Pada individu dengan HD, terjadi ekspansi abnormal dari sekuens trinukleotida CAG dalam gen HTT. Normalnya, sekuens CAG berulang antara 10 hingga 35 kali, tetapi pada individu dengan HD, jumlah pengulangan ini meningkat menjadi lebih dari 36. Semakin banyak jumlah pengulangan CAG, semakin dini gejala penyakit muncul dan semakin berat kondisinya. Protein huntingtin yang bermutasi menjadi toksik bagi sel saraf, menyebabkan kematian neuron secara bertahap. [4]

Penyakit ini biasanya menunjukkan gejala pada usia 30-50 tahun, tetapi bentuk juvenile Huntington's Disease dapat terejadi pada usia lebih muda. Gejalanya meliputi:

1. Gangguan Gerakan : Gerakan tak terkendali atau koreatik (chorea) , kekakuan otot, kesulitan berjalan, dan koordinasi yang buruk.
2. Gangguan Kognitif : Penurunan kemampuan berpikir, sulit dalam berkonsentrasi, kehilangan memori, dan demensia progresif.
3. Gangguan Psikologis : Depresi, perubahan suasana hati, iritabiliitas, dan perilaku impulsif.

Tidak ada obat untuk menyembuhkan Huntington's Disease. Pengobatan yang tersedia bersifat paliatif, membantu mengelola gejala untuk meningkatkan kualitas hidup. Terapi farmakologi seperti tetrabenazine digunakan untuk mengontrol chorea, sementara antidepresan atau antipsikotik membantu mengatasi gejala psikologis. Selain itu, terapi fisik, okupasi, dan psikologis dapat membantu penderita mempertahankan fungsionalitas sehari-hari. HD bersifat progresif, dan prognosisnya bervariasi, dengan harapan hidup rata-rata setelah diagnosis adalah sekitar 15–20 tahun.

D. Algoritma Knuth-Morris-Pratt

Algoritma Knuth–Morris–Pratt (KMP) adalah algoritma pencocokan string yang dirancang untuk menemukan pola dalam teks secara efisien tanpa memeriksa kembali karakter yang telah dibandingkan sebelumnya. Algoritma ini diperkenalkan oleh Donald Knuth, Vaughan Pratt, dan James H. Morris pada tahun 1977, berdasarkan penelitian mereka terhadap struktur string dan kebutuhan untuk meningkatkan efisiensi proses pencocokan pola. Motivasi utama mereka adalah mengatasi keterbatasan algoritma pencocokan tradisional, seperti metode pencarian naif, yang cenderung mengulang pemeriksaan karakter pada teks secara redundan. Dengan memanfaatkan sifat periodisitas dalam pola, algoritma KMP mengurangi jumlah perbandingan yang diperlukan, sehingga meningkatkan kecepatan pencarian.

KMP menawarkan manfaat signifikan dalam efisiensi waktu, khususnya ketika berurusan dengan teks yang panjang atau pola yang sering berulang. Karena memiliki kompleksitas waktu O(n + m), di mana n adalah panjang teks dan m adalah panjang pola, algoritma ini sangat cocok untuk aplikasi yang membutuhkan analisis teks skala besar. Contoh pengaplikasiannya meliputi pencarian teks dalam dokumen digital, analisis urutan DNA di bioinformatika, pemeriksaan plagiarisme dalam karya tulis, dan sistem pencarian berbasis pola seperti dalam aplikasi mesin pencari. Selain itu, algoritma ini digunakan dalam kompresi data, di mana pola berulang dalam data dapat diidentifikasi dengan cepat, serta dalam sistem deteksi virus yang memerlukan pencocokan pola dalam signature virus.

KMP bekerja dengan membagi proses pencocokan menjadi dua tahap utama: pembuatan tabel **prefix function** (atau π-table) dan pencocokan pola terhadap teks. Tabel prefix function digunakan untuk menyimpan informasi tentang panjang prefix yang juga merupakan suffix dari pola hingga posisi tertentu. Dengan tabel ini, jika terjadi ketidaksesuaian saat pencocokan, algoritma tidak perlu kembali ke awal teks, tetapi langsung melompat ke posisi yang sesuai berdasarkan nilai dalam tabel. Proses pencocokan dimulai dari awal teks, dan algoritma membandingkan karakter pola dengan karakter teks satu per satu. Jika terjadi ketidaksesuaian, nilai dalam tabel prefix function menentukan langkah berikutnya, sehingga menghindari pemeriksaan ulang karakter yang sama. Efisiensi ini menjadikan KMP lebih cepat dibandingkan metode pencocokan naif, terutama dalam kasus teks dan pola yang sangat panjang. Berikut ini langkah-langkah lebih detail algoritma KMP :

Pembuatan Tabel Prefix Function (π-table)

Langkah pertama adalah membangun tabel **prefix function**, yang membantu menentukan seberapa jauh pola dapat bergeser jika terjadi ketidaksesuaian.

- Buat array π berukuran sama dengan panjang pola mmm.
- Tetapkan nilai awal π[0]=0
- Mulai dari indeks ke-1 pola, bandingkan karakter dengan karakter sebelumnya.
- Jika ada kecocokan, tingkatkan nilai panjang prefix-suffix yang cocok, dan simpan dalam tabel.

- Jika tidak cocok, gunakan nilai π sebelumnya untuk mencari kecocokan baru.

Contoh:
Untuk pola "ABABAC," π-table adalah:
π=[0,0,1,2,3,0]

Pencocokan Pola dengan Teks

Langkah kedua adalah mencocokkan pola dengan teks menggunakan π-table untuk menghindari pemeriksaan ulang karakter.

- Tetapkan dua indeks: i=0 untuk teks dan j=0 untuk pola.
- Bandingkan karakter teks T[i] dengan karakter pola P[j]:
    - Jika cocok, tingkatkan iii dan j.
    - Jika j mencapai panjang pola m, pola ditemukan di posisi i−j.
- Jika tidak cocok:
    - Gunakan nilai dari π-table untuk menentukan posisi baru j dalam pola.
    - Jika j=0, tingkatkan i untuk memulai pencocokan berikutnya.

Contoh:
Teks: "ABABABCABABABCABABABC"
Pola: "ABABAC"

- Pada iterasi pertama, pola cocok hingga indeks ke-5. Ketidaksesuaian di posisi ke-6 mengacu pada π[5]=3, sehingga jjj melompat ke posisi 3 tanpa memulai ulang.

Hasil Akhir

Proses pencocokan diulang hingga teks selesai dipindai, atau semua kemunculan pola ditemukan.

E. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah salah satu algoritma pencocokan string yang terkenal karena efisiensinya dalam mencari pola dalam teks panjang. Algoritma ini dikembangkan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977. Boyer-Moore dirancang untuk memanfaatkan sifat pola dan teks dengan melakukan perbandingan dari kanan ke kiri, berbeda dengan algoritma pencocokan string lainnya. Motivasi utama pengembangan algoritma ini adalah untuk mengurangi jumlah perbandingan yang diperlukan dengan menggunakan informasi tambahan dari pola itu sendiri. Sebagai hasilnya, Boyer-Moore menjadi sangat efisien, terutama dalam skenario di mana pola memiliki panjang yang signifikan dibandingkan teks.

Boyer-Moore sangat bermanfaat karena kompleksitasnya yang bersifat adaptif terhadap struktur teks dan pola. Algoritma ini sering digunakan dalam aplikasi di mana pencocokan string cepat sangat penting, seperti dalam perangkat lunak pencarian teks, alat deteksi malware, dan aplikasi pengolahan data besar. Dalam bioinformatika, algoritma ini juga digunakan untuk menganalisis sekuens DNA dan RNA, mengidentifikasi motif genetik tertentu. Selain itu, Boyer-Moore diterapkan dalam pengembangan editor teks dan basis data untuk menemukan dan mengganti pola tertentu dalam dokumen atau catatan database yang sangat besar.

Langkah-langkah algoritma Boyer-Moore diawali dengan pembuatan tabel heuristik untuk mempercepat proses pencocokan pola dengan teks. Tabel ini terdiri dari dua aturan utama: **Bad Character Rule** dan **Good Suffix Rule**. **Bad Character Rule** memanfaatkan informasi tentang karakter teks yang tidak cocok dengan pola. Jika terjadi ketidaksesuaian, pola digeser sedemikian rupa sehingga karakter tidak cocok dalam teks dapat cocok dengan posisi lain dalam pola atau keluar dari pengaruh pola. **Good Suffix Rule** digunakan untuk menangani kasus di mana bagian suffix dari pola cocok sebelum terjadi ketidaksesuaian. Dalam aturan ini, pola digeser untuk mempertahankan kesesuaian suffix atau menempatkan ulang pola agar tidak ada kesamaan redundan dengan teks. Setelah tabel heuristik selesai dibuat, algoritma melanjutkan ke tahap pencocokan. Pola ditempatkan pada awal teks, tetapi perbandingan dilakukan dari kanan ke kiri, dimulai dari karakter terakhir pola. Jika semua karakter cocok, algoritma mencatat posisi pola yang ditemukan dalam teks. Namun, jika terjadi ketidaksesuaian, algoritma memanfaatkan tabel heuristik untuk menentukan seberapa jauh pola harus digeser. Aturan yang memberikan pergeseran lebih besar antara **Bad Character Rule** dan **Good Suffix Rule** digunakan untuk memastikan efisiensi maksimum. Proses ini diulangi hingga pola telah dibandingkan dengan seluruh teks atau semua kemunculan pola telah ditemukan. Berikut ini langkah-langkah lebih detail :

Pembuatan Tabel Heuristik:
Algoritma Boyer-Moore menggunakan dua heuristik utama:

- **Bad Character Rule (Aturan Karakter Buruk):**
    - Jika terjadi ketidaksesuaian, gunakan tabel ini untuk menentukan seberapa jauh pola dapat bergeser berdasarkan karakter teks yang tidak cocok.
    - Misalnya, jika karakter tidak cocok ada di teks pada indeks tertentu, pola digeser sehingga karakter itu cocok atau tidak relevan lagi.
- **Good Suffix Rule (Aturan Suffix Baik):**
    - Jika terjadi ketidaksesuaian, gunakan tabel ini untuk menentukan seberapa jauh pola dapat bergeser berdasarkan kesesuaian suffix pola sebelumnya.
    - Pola digeser agar bagian suffix yang cocok sebelumnya tetap sejajar dengan teks, atau pola diposisikan ulang sepenuhnya.

2. Proses Pencocokan:
Setelah tabel heuristik dibuat, pencocokan dilakukan dari kanan ke kiri pada pola:

- **a. Inisialisasi:**
    - Tempatkan pola pada awal teks dan bandingkan karakter terakhir pola dengan teks.
- **b. Bandingkan Karakter:**
    - Jika cocok, pindah ke karakter sebelumnya dalam pola dan ulangi.
    - Jika seluruh pola cocok, posisi pencocokan ditemukan.
- **c. Atur Ulang Posisi:**
    - Jika tidak cocok, gunakan aturan **Bad Character** atau **Good Suffix** untuk menentukan pergeseran pola.
    - Pilih pergeseran yang lebih besar antara dua aturan tersebut untuk efisiensi maksimum.

3. Iterasi dan Penemuan:

- Ulangi proses pencocokan hingga seluruh teks selesai dipindai, atau semua kemunculan pola ditemukan.

III. IMPLEMENTASI

Implementasi dilakukan menggunakan bahasa pemrograman python. Saya memilih bahasa pemrograman ini untuk memudahkan implementasi. Ditambah lagi, program yang akan diimplementasikan tidak begitu kompleks.

A. Implementasi Algoritma KMP

```
# kmp.py

class KMP:
    def __init__(self, text : str):
        self.text = text

    def _compute_prefix_table(self,
pattern : str) -> list[int]:
        m = len(pattern)
        prefix_table = [0] * m
        j = 0

        for i in range(1, m):
                while j > 0 and
pattern[i] != pattern[j]:
                j = prefix_table[j
- 1]

                if pattern[i] ==
pattern[j]:
                j += 1
                prefix_table[i] = j
            else:
                prefix_table[i] = 0

        return prefix_table

    def search(self, pattern : str)
-> int:
        n = len(self.text)
        m = len(pattern)
        j = 0

                  prefix_table   =
self._compute_prefix_table(pattern
)

        for i in range(n):
                while j > 0 and
self.text[i] != pattern[j]:
                j = prefix_table[j
- 1]

               if self.text[i] ==
pattern[j]:
                j += 1

            if j == m:

                return i - m + 1

        return -1
```

Kelas KMP diatas adalah implementasi dari algoritma

Knuth-Morris-Prath. Kelas ini memiliki kontruktor yang menerima teks sequence DNA utama sebagai input (dalam bentuk string).

Terdapat metode _compute_prefix_table yang membangun table prefix. Tabel ini mencatat panjang prefix-sufiks terpanjang yang cocok dalam pola.

Metode search bertugas mencari pola dalam teks dengan memanfaatkan tabel prefiks yang telah dibangun. Algoritma ini berjalan melalui teks, membandignkan pola karakter demi karakter, dan melompat ke posisi yang sesuai ketika pola tidak cocok. Jika pola ditemukan, metode ini mengembalikan indeks awal pencocokan dalam teks. Jika tidak, nilai -1 dikembalikan. Dengan kompleksitas waktu O(n + m), di mana n adalah panjang teks dan m adalah panjang pola.

B. Implementasi Algoritma Boyer-Moore

```
# bm.py

class BM:
    def __init__(self, text : str):
        self.text = text

                                def
_compute_last_occurrence_table(sel
f, pattern: str) -> dict[str, int]:
        last_occurrence = {}
                for   i,   char   in
enumerate(pattern):
            last_occurrence[char] =
i
        return last_occurrence

    def search(self, pattern: str)
-> int:
                last_occurrence   =
self._compute_last_occurrence_tabl
e(pattern)
           n,  m  =  len(self.text),
len(pattern)
        if m == 0:
            return -1

        shift = 0
        while shift <= n - m:
            j = m - 1

                while  j  >=  0  and
pattern[j] == self.text[shift + j]:
                j -= 1

            if j < 0:
                return shift

                      char_shift   =
last_occurrence.get(self.text[shif
t + j], -1)
              shift  +=  max(1,  j  -
char_shift)

        return -1
```

Implementasi algoritma Boyer-Moore menggunakan kelas BM. bekerja dengan membandingkan pola dari kanan ke kiri dan memanfaatkan informasi dari tabel kejadian terakhir (*last occurrence table*) untuk melompat ke posisi yang relevan jika terjadi ketidaksesuaian. Dalam kelas ini, metode _compute_last_occurrence_table membangun tabel yang mencatat indeks terakhir setiap karakter dalam pola. Tabel ini digunakan untuk menentukan seberapa jauh pola dapat digeser saat ketidaksesuaian terjadi, sehingga mengurangi jumlah perbandingan.

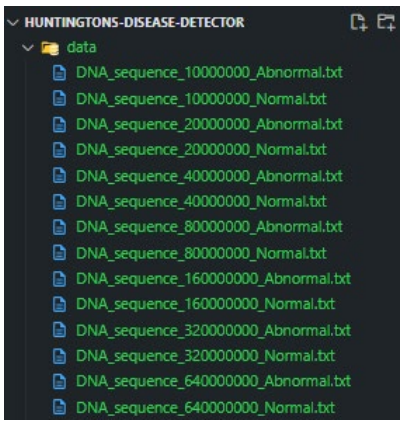
Metode utama search melakukan pencarian pola menggunakan tabel tersebut. Algoritma dimulai dengan menyelaraskan pola di awal teks, lalu membandingkan karakter dari pola dengan teks dari ujung pola ke arah kiri. Jika pola cocok sepenuhnya, indeks awal pencocokan dikembalikan. Jika tidak, algoritma menggunakan informasi dari tabel kejadian terakhir untuk menghitung jarak geser yang optimal, mempercepat proses pencarian.

IV. Pengujian dan Analisis

Pengujian dilakukan dengan membuat 7 sampel DNA dengan panjang bervariasi, mulai dari 10 juta hingga 640 juta, masing-masing untuk kasus normal dan kasus abnormal. Kasus normal adalah kasus dimana sampel tidak memiliki sekuens CAG berulang sebanyak 36 kali, yang dimiliki sampel kasus abnormal.

```
...CAGCAGCAGCAGCAGCAGCAGCAGCAGC
AGCAGCAGCAGCAGCAGCAGCAGCAGCAGCA
GCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG
CAGCAGCAG...
```

Pada repository program, sampel-sampel ini disimpan di folder data/ , dalam file txt dengan format DNA_sequence_{panjang sampel}_{Normal/Abnormal}.txt

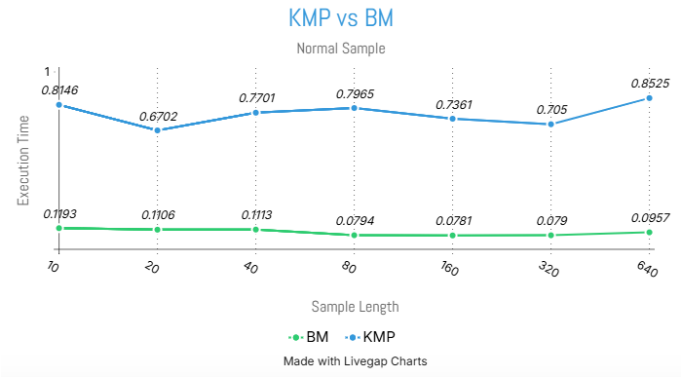


Gambar 1. Sampel DNA sequence yang akan diuji

A. Sampel Normal

No	Panjang Sampel (juta)	Waktu eksekusi	
		KMP (s)	BM (s)
1.	10	0.8146	0.1193
2.	20	0.6702	0.1106
3.	40	0.7701	0.1113
4.	80	0.7965	0.0794
5.	160	0.7361	0.0781
6.	320	0.7050	0.0790
7.	640	0.8525	0.0957

Tabel 1. Hasil pengujian sampel normal

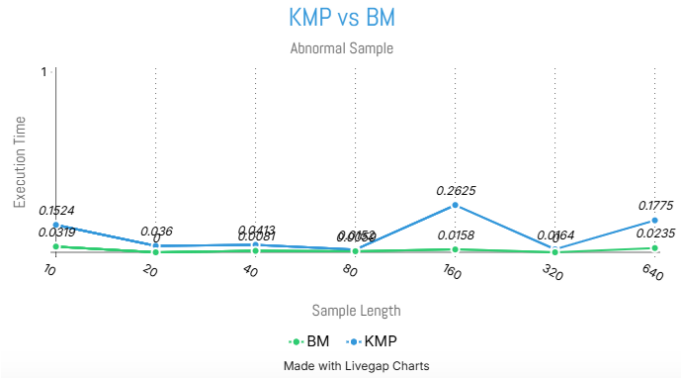


Gambar 2. Grafik Hasil pengujian sampel normal

B. Sampel Abnormal

No	Panjang Sampel (juta)	Waktu eksekusi	
		KMP (s)	BM (s)
1.	10	0.1524	0.0319
2.	20	0.0360	0.0001
3.	40	0.0413	0.0081
4.	80	0.0152	0.0059
5.	160	0.2625	0.0158
6.	320	0.0164	0.0001
7.	640	0.1775	0.0235

Tabel 2. Hasil pengujian sampel abnormal



Gambar 3. Grafik hasil pengujian sampel abnormal

V. Kesimpulan

Dalam perbandingan algoritma KMP dan BM untuk mencari pola penyakit genetik Huntington pada sequence DNA yang panjang, hasil menunjukkan bahwa **algoritma BM lebih efisien dibandingkan KMP**. Efisiensi BM berasal dari kemampuannya untuk melakukan lompatan lebih besar saat ketidaksesuaian ditemukan, memanfaatkan informasi dari *last occurrence table.* Sedangkan KMP menggunakan tabel prefiks untuk memastikan tidak ada perbandingan ulang. Walaupun DNA Huntington memliki banyak pola berulang yang mendukung algoritma KMP, tetap saja harus membandingkan setiap karakter secara berurutan dan lompatan yang dilakukan pun tidak sejauh algoritma BM.

LAMPIRAN

Pranala Github : FityatulhaqRosyidi/huntingtons-disease-detector: this program detect whether a DNA sequence contains huntington's disease, written in python

Pranala Video : https://youtu.be/nepJ1ehsWjA

UCAPAN TERIMA KASIH

Rasa syukur dan terimakasih saya panjatkan ke kehadirat Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga makalah ini dapat selesai dengan baik. Dengan selesainya penyusunan tugas makalah ini, saya ingin mengucapkan rasa terima kasih yang sebesar-besarnya kepada seluruh pihak yang turut serta membantu saya dalam menyelesaikan makalah ini. Terima kasih juga saya ucapkan kepada para dosen pengampu mata kuliah Strategi Algortima. Begitu juga kepada teman-teman saya yang senantiasa mendukung saya selama berkuliah di ITB. Terakhir, Walaupun sederhana, semoga tulisan saya ini dapat bermanfaat bagi perkembangan sains di masa depan.

REFERENSI

[1] S. Asha and S. T. Krishna, "Semantics-based string matching: A review of machine learning models," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 10, no. 3, pp. 15–28, 2021

[2] R. Munir, *Pencocokan String: Teori dan Algoritma*. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf. [Accessed: Jun. 20, 2025].

[3] H. Nishida, "Genome DNA sequence variation, evolution, and function in bacteria and archaea," *Curr. Issues Mol. Biol.*, vol. 15, no. 1, pp. 19–24, 2013.

[4] F. Duyao, C. Ambrose, R. Myers, A. Novelletto, dan F. Persichetti, "Trinucleotide repeat length instability and age of onset in Huntington's disease," *Nature Genetics*, vol. 4, hlm. 387–392, Agu 1993.

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, 31 Desember 2024

Fityatul Haq Rosyidi 13523116