Perbandingan Kinerja Algoritma Pencocokan *String* Perkiraan untuk Prediksi Lokasi *Off-Target* pada Sistem CRISPR-Cas9

Aria Judhistira - 13523112
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: judhistiraaria@gmail.com , 13523112@std.stei.itb.ac.id

***Abstrak*—Sistem CRISPR-Cas9 merupakan teknologi yang dimanfaatkan untuk melakukan penyuntingan DNA dengan presisi tinggi. Tantangan yang muncul dari teknologi ini adalah efek *off-target* yang dapat menyebabkan komplikasi fatal. Efek *off-target* dapat dimodelkan secara komputasi sebagai persoalan pencocokan *string* perkiraan. Algoritma-algoritma pencocokan *string* didesain untuk menyelesaikan permasalahan tersebut, termasuk algoritma Brute Force, algoritma KMP dengan partisi pola, dan algoritma Bitap. Hasil pengujian yang mengukur kinerja ketiga algoritma tersebut dari segi waktu eksekusi dan penggunaan memori menunjukkan bahwa algoritma Bitap memberikan hasil kinerja yang paling optimal, dengan waktu eksekusi dan penggunaan memori yang relatif rendah dibandingkan algoritma lainnya.**

***Keywords*—CRISPR-Cas9, Brute Force, KMP, Partisi Pola, Bitap**

I. PENDAHULUAN

CRISPR-Cas9 (*Clustered Regularly Interspaced Short Palindromic Repeats – Cas9*) merupakan teknologi rekayasa genetika revolusioner yang memungkinkan modifikasi DNA dengan tingkat presisi tinggi. Adanya teknologi ini memungkinkan penyuntingan genetika bahkan pada organisme hidup. Sistem CRISPR-Cas9 bekerja dengan dua komponen utama, yaitu protein Cas9 dan molekul pemandu RNA (gRNA). Protein Cas9 berfungsi untuk memotong DNA, sedangkan gRNA, yang memiliki sekuens DNA yang komplementer dengan sekuens DNA target, berfungsi untuk memandu Cas9 dalam memilih bagian DNA yang akan dipotong. Dalam proses ini, gRNA akan memandu protein Cas9 untuk menelusuri genom hingga menemukan lokasi DNA yang cocok untuk sekuensnya. Ketika kecocokan ditemukan, protein Cas9 akan memotong kedua untai ujung pada sekuens DNA tersebut sehingga memberi ruang untuk menyisipkan materi genetik baru.

Permasalahan pencocokan *string* merupakan salah satu persoalan fundamental dalam dunia komputasi. Sederhananya, permasalahan ini melibatkan pencarian semua kemunculan suatu *string* pendek (pola) pada suatu *string* yang lebih besar (teks). Terdapat berbagai algoritma yang dikemukakan oleh berbagai ahli untuk menyelesaikan permasalahan ini, seperti algoritma Knuth-Morris-Pratt, algoritma Boyer-Moore, dll. Selain itu, variasi dari permasalahan pencocokan *string* adalah pencocokan *string* perkiraan (*approximate string matching*) yang memiliki batas toleransi ketidakcocokan karakter. Setiap algoritma pencocokan *string* memiliki keunikan cara kerja masing-masing. Dengan begitu pun, masing-masing algoritma memiliki kelebihan dan kekurangan, tergantung pada kasus penggunaannya.

Pengaplikasian teknologi CRISPR-Cas9 pada skala genomik membuahkan suatu tantangan komputasi. Salah satu isu efektivitas dan keamanan dari teknologi CRISPR-Cas9 adalah efek *off-target* yang menyebabkan gRNA dapat berikatan dengan lokasi lain yang memiliki sekuens mirip. Efek ini dapat berakibat fatal jika terjadi dan dapat menyebabkan berbagai mutasi yang tidak diinginkan dan sulit untuk diprediksi. Untuk memitigasi risiko efek *off-target* ini, diperlukan analisis komputasi menggunakan algoritma pencocokan *string* perkiraan yang mampu mendeteksi semua potensi lokasi *off-target*. Makalah ini bertujuan untuk mengeksplorasi dan membandingkan kinerja beberapa algoritma pencocokan *string* perkiraan, yaitu algoritma Brute Force, algoritma KMP dengan partisi pola, dan algoritma Bitap, dalam kasus ini serta menganalisis *trade-off* performa dari setiap algoritma.

II. LANDASAN TEORI

A. Rekayasa Genetika dan SistemCRISPR-Cas9

Rekayasa genetika merupakan serangkaian teknologi dalam bidang biologi molekuler yang memungkinkan ilmuwan untuk melakukan perubahan presisi pada materi genetik suatu organisme, seperti tumbuhan, bakteri, dan hewan. Perubahan pada materi genetik menghasilkan modifikasi pada sifat-sifat fisik yang teramati, serta memengaruhi risiko terhadap suatu penyakit. Prinsip kerja teknologi rekayasa genetika biasa diumpamakan sebagai "gunting molekuler" yang memotong untai DNA pada lokasi yang spesifik. Setelah pemotongan dilakukan, para peneliti dapat melakukan beberapa jenis modifikasi, antara lain penghapusan (delesi), penyisipan (insersi), atau penggantian (substitusi) sekuens DNA pada titik potong tersebut [1].

Teknologi rekayasa genetika mulai dikembangkan pada akhir abad ke-20. Seiring dengan kemajuan riset, sebuah

metode bernama CRISPR (Clustered Regularly Interspaced Short Palindromic Repeats) diperkenalkan pada tahun 2009. Teknologi CRISPR secara signifikan mempermudah proses DNA *editing*. Dibandingkan dengan metode-metode sebelumnya, proses CRISPR lebih sederhana, cepat, biaya lebih terjangkau, serta tingkat akurasi yang lebih tinggi. Oleh karena efisiensi dan efektivitasnya, CRISPR telah menjadi perangkat utama yang diadopsi secara luas di berbagai penelitian terkait rekayasa genetika [1].

CRISPR-Cas9 adalah teknologi rekayasa genetika yang memungkinkan para ilmuwan untuk memodifikasi DNA organisme secara presisi, termasuk bakteri, tumbuhan, dan hewan [2]. Di alam, sistem CRISPR-Cas9 berfungsi sebagai sistem kekebalan adaptif bagi prokariota (bakteri dan archaea), melindungi mereka dari elemen genetik penyerang seperti virus dan plasmid. Ketika bakteri diserang oleh virus, ia menangkap sepotong kecil DNA virus dan mengintegrasikannya ke dalam lokus CRISPR miliknya sebagai sekuens "spacer". Proses ini menciptakan memori genetik dari infeksi masa lalu, memungkinkan bakteri untuk mengenali dan menghancurkan virus yang sama pada serangan berikutnya. Para ilmuwan telah memanfaatkan mekanisme pertahanan alami ini untuk menciptakan sebuah alat yang dapat memotong DNA di lokasi spesifik untuk memperbaiki mutasi secara permanen atau mengganggu gen penyebab penyakit [2][3].

A.1 Mekanisme Kerja CRISPR-Cas9

Mekanisme sistem CRISPR-Cas9 bergantung pada tiga komponen utama, yaitu protein Cas9, *single guide* RNA (sgRNA), dan Protospacer Adjacent Motif (PAM). Protein Cas9 adalah endonuklease yang memotong DNA. Protein ini memiliki dua domain nuklease yang berbeda, yang dikenal sebagai RuvC dan HNH, masing-masing bertanggung jawab untuk membelah satu untai dari heliks ganda DNA [3]. SgRNA adalah molekul rekayasa yang mengarahkan protein Cas9 ke target yang presisi di dalam genom [2]. Molekul ini berisi sekuens 20-nukleotida yang dapat diprogram dan komplementer dengan sekuens DNA target yang ingin diedit oleh ilmuwan. Dalam aplikasi laboratorium, sgRNA ini adalah fusi yang disederhanakan dari dua RNA yang ada secara alami, yaitu CRISPR RNA (crRNA) dan *trans-activating* CRISPR RNA (tracrRNA) [3].

Proses *editing* dimulai ketika protein Cas9, yang berikatan dengan sgRNA, membaca genom. Aksi sistem ini bergantung pada keberadaan sekuens PAM, yaitu sekuens pendek dan spesifik, yang terletak persis di hilir DNA target. Cas9 hanya mengikat dan memotong DNA jika terlebih dahulu mengenali sekuens PAM ini. Setelah PAM teridentifikasi, heliks ganda DNA akan terbuka, memungkinkan sgRNA untuk berikatan dengan untai DNA targetnya yang komplementer dan terbentuk struktur R-loop. Ikatan dan pembentukan R-loop ini mengaktifkan domain nuklease RuvC dan HNH pada protein Cas9. Hal ini membelah kedua untai DNA untuk menciptakan pemutusan untai ganda (*double-strand break* atau DSB). Setelah DNA terpotong, mekanisme perbaikan DNA alami sel diaktifkan untuk memperbaiki kerusakan tersebut, seperti Non-Homologous End Joining (NHEJ) dan Homology-Directed Repair (HDR) [2][3].

A.2 Efek *Off-Target*

Efek *off-target* adalah perubahan yang tidak terduga, tidak diinginkan, atau bahkan merugikan yang terjadi pada genom akibat salah pemotongan situs DNA oleh kompleks CRISPR/Cas9. Pemotongan yang tidak disengaja ini dapat terjadi karena protein Cas9 diketahui dapat mentolerir ketidakcocokan (*mismatch*) antara sgRNA dengan DNA genomik. Efek *off-target* merupakan masalah utama dalam penerapan CRISPR/Cas9 karena berpotensi membahayakan. Oleh karena itu, kemampuan untuk memprediksi dan mendeteksi situs *off-target* di seluruh genom sangat penting untuk memastikan keamanan terapi gen. Prediksi yang akurat memungkinkan para ilmuwan untuk merancang sgRNA yang lebih spesifik dan mengembangkan turunan CRISPR/Cas9 dengan presisi yang ditingkatkan sehingga dapat meminimalkan risiko mutasi yang tidak diinginkan sebelum diaplikasikan pada manusia [4].

B. Hamming Distance

Hamming Distance merupakan metode pengukuran perbedaan dua *string* yang dikemukakan oleh Richard Hamming. Sederhananya, Hamming Distance adalah jumlah posisi pada dua *string* dengan panjang sama yang memiliki karakter berbeda. Sebagai contoh, jika terdapat kata "seru" dan kata "sari", maka Hamming Distance kedua kata tersebut adalah 3. Jika Hamming Distance antara dua buah *string* bernilai 0, maka pastilah kedua *string* tersebut sama [5].

C. Algoritma Pencocokan String

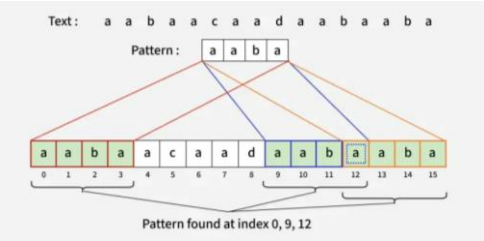
Algoritma pencocokan *string* adalah algoritma yang didesain untuk menyelesaikan persoalan pencocokan *string*. Terdapat tiga algoritma pencocokan *string* yang dibahas pada makalah ini, yakni algoritma Brute Force, algoritma KMP dengan partisi pola, dan algoritma Bitap.

B.1 Algoritma Brute Force

Algoritma Brute Force untuk pencocokan *string* mengikuti konsep algoritma Brute Force pada umumnya, yakni dengan mengecek setiap posisi pola pada teks satu per satu. Algoritma ini mampu mendapatkan hasil, tetapi dengan waktu eksekusi yang lama karena perlu mengiterasi setiap karakter pada teks.

B.2 Algoritma KMP dengan partisi pola

Algoritma KMP dengan partisi pola merupakan adaptasi dari algoritma KMP original. Algoritma KMP (Knuth-Morris-Pratt) melakukan penggeseran pola yang dibandingkan dengan teks dengan lebih intuitif dibandingkan algoritma Brute Force. Prinsip utama yang membantu penggeseran ini adalah fungsi pinggirannya yang mencari LPS (*Longest Proper Prefix which is also a Suffix*) pada pola.



GAMBAR 2.1 Ilustrasi algoritma KMP
Sumber: [6]

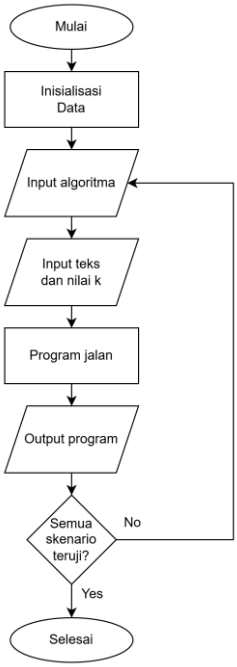
Di sisi lainnya, partisi pola merupakan algoritma yang membagi sebuah pola menjadi pola yang lebih kecil lagi. Hal ini pun memungkinkan algoritma KMP untuk digunakan pada pencocokan *string* perkiraan. Prinsip kerja algoritma KMP dengan partisi pola adalah untuk membagi pola dengan panjang *m* menjadi *seed* dengan panjang masing-masing *seed* sebesar *k*+1, dengan *k* adalah batas toleransi ketidakcocokan. Lalu, algoritma KMP akan mencari semua lokasi kemunculan *seed* pada teks. Semua posisi tersebut dianggap sebagai kandidat yang kemudian diverifikasi dengan cara menghitung Hamming Distance-nya.

B.3 Algoritma Bitap

Algoritma Bitap merupakan suatu algoritma pencocokan *string* perkiraan yang bekerja dengan memanfaatkan operasi *bitwise* [8]. Prinsip kerja algoritma ini melibatkan sebuah tabel *bitmask* yang dibuat untuk setiap karakter unik pada pola. Kemudian, sebuah vektor status ditetapkan yang merepresentasikan status kecocokan setiap posisi pada pola. Vektor status ini diperbarui dengan operasi *bitwise*. Setiap karakter pada teks kemudian diiterasikan, dengan vektor status diperbarui pada setiap iterasi. Pada akhirnya, kecocokan ditemukan jika bit pada posisi terakhir (*most significant bit*) pada salah satu elemen vektor status aktif, yang menandakan bahwa terdapat kecocokan pola terhadap teks dengan batas toleransi yang diizinkan.

III. METODE

A. Kerangka Penelitian



GAMBAR 3.1 Diagram alir kerangka penelitian
Sumber: Dokumen penulis

Penelitian ini dilaksanakan dengan mengikuti alur kerja pada Gambar 3.1. Penelitian dimulai dengan persiapan data yang akan diuji, diikuti oleh pemilihan algoritma dan skenario uji yang berisi teks dan jumlah ketidakcocokan huruf. Setelah proses *input* selesai, program akan dijalankan dan mengeluarkan *output* hasil untuk uji coba untuk skenario dan algoritma yang dipilih. Bila setiap kombinasi algoritma dan skenario uji belum dipilih, maka penelitian akan mengulangi proses *input* dari pemilihan algoritma hingga setiap kombinasi sudah diproses. Data yang didapatkan dari hasil uji coba akan dicatat dan kemudian dianalisis.

B. Objek Penelitian

Objek penelitian yang digunakan terdiri dari algoritma pencocokan *string* dan data uji yang ditentukan.

B.1 Algoritma yang Diuji

Penelitian ini membandingkan kinerja tiga algoritma pencocokan *string*:

- Algoritma Brute Force dengan Hamming Distance
- Algoritma KMP dengan Partisi Pola
- Algoritma Bitap

B.2 Data Uji

Data uji yang digunakan pada penelitian adalah suatu sekuens DNA sintetis yang dihasilkan secara acak. Sekuens DNA terdiri dari karakter ‘A’, ‘G’, ‘T’, dan ‘C’. Terdapat tiga ukuran teks yang digunakan, yaitu 100 KB, 1 MB, dan 10 MB. Selain itu, pola yang digunakan berupa pola dengan panjang 20 karakter yang diambil secara acak. Pola tersebut akan

digunakan untuk masing-masing ukuran teks untuk memastikan objektivitas kinerja masing-masing algoritma.

C. Desain Eksperimen

Penelitian ini dirancang untuk mengukur pengaruh variabel-variabel tertentu terhadap kinerja suatu algoritma pencocokan *string*.

C.1 Variabel Independen

- Ukuran teks, yaitu besar teks yang digunakan dalam pengujian dalam *byte*. Pengujian dilakukan pada tiga ukuran teks berbeda, yaitu 100 KB, 1 MB, dan 10 MB.
- Jumlah ketidakcocokan (k), yaitu jumlah ketidakcocokan huruf yang ditoleransi. Pengujian dilakukan pada tiga nilai ketidakcocokan berbeda, yaitu k = 1, 2, dan 3.

C.2 Variabel Dependen

- Waktu eksekusi, yaitu waktu yang dibutuhkan oleh program menggunakan suatu algoritma untuk menyelesaikan pencarian dalam milidetik (ms).
- Penggunaan memori, yaitu jumlah memori maksimum yang digunakan oleh program selama eksekusi dalam *kilobytes* (KB)

D. Lingkungan Pengujian

Implementasi algoritma dan pengujian dilakukan menggunakan bahasa pemrograman Python versi 3.12.8 pada sistem operasi Windows 10. Perangkat keras yang digunakan untuk pengujian melibatkan spesifikasi CPU berupa 11th Gen Intel® Core(TM) i5-1135G7 @ 2.40 GHz, RAM sebesar 8 GB, dan GPU berupa Intel® Iris® Xe Graphics.

IV. IMPLEMENTASI

Implementasi pencarian dengan algoritma *brute force* dengan Hamming Distance relatif sederhana dibandingkan algoritma-algoritma lain. Prinsip utama dari algoritma *brute force* dalam pencocokan *string* adalah untuk membandingkan pola dengan teks yang mencoba setiap kemungkinan (*exhaustive search*), yaitu melalui penggeseran pola dengan teks sebanyak satu huruf untuk setiap iterasi. Pada setiap iterasi perbandingan, program akan menghitung Hamming Distance pada pola dengan potongan teks yang dibandingkan. Jika besar Hamming Distance di bawah ambang batas nilai toleransi ketidakcocokan (k), maka potongan teks tersebut dianggap sesuai dan dimasukkan pada daftar hasil.

```
ALGORITMA hamming_distance(s1, s2)
BEGIN
  IF panjang(s1) ≠ panjang(s2) THEN
    RETURN infinity
  END IF

  distance ← 0
  FOR i ← 0 TO panjang(s1) - 1 DO
    IF s1[i] ≠ s2[i] THEN
      distance ← distance + 1
    END IF
  END FOR

  RETURN distance
END

ALGORITMA search_bruteforce(teks, pola, k)
BEGIN
  positions ← array kosong
  n ← panjang(teks)
  m ← panjang(pola)

  // Sliding window untuk setiap posisi yang mungkin
  FOR i ← 0 TO (n - m) DO
    substring ← teks[i...i+m-1]
    distance ← hamming_distance(pola, substring)
    IF distance ≤ k THEN
      tambahkan i ke positions
    END IF
  END FOR

  RETURN positions
END
```

Selanjutnya, implementasi algoritma Knuth-Morris-Pratt dengan partisi pola dibantu dengan implementasi algoritma KMP untuk *exact matching* atau pencocokan persis. Secara keseluruhan, implementasi algoritma KMP dengan partisi pola dibagi menjadi dua bagian utama, yaitu tahap partisi dan tahap verifikasi.

Tahap partisi dimulai dengan membagi pola menjadi k+1 bagian. Setiap bagian yang dibagi tersebut dianggap sebagai sebuah *seed*. Lalu, untuk setiap bagian pola, program akan memanfaatkan algoritma KMP untuk pencocokan persis untuk mencari semua lokasi kemunculan bagian pola tersebut pada teks. Untuk menghindari duplikasi kandidat, setiap potongan teks yang cocok dengan bagian pola dimasukkan ke dalam suatu himpunan (*set*).

Tahap selanjutnya pada algoritma KMP dengan partisi pola adalah tahap verifikasi kandidat-kandidat yang ditemukan. Setelah setiap kandidat dikumpulkan, program mengiterasi setiap kandidat dan menggunakan Hamming Distance untuk melakukan verifikasi terakhir terhadap nilai toleransi ketidakcocokan. Kandidat-kandidat potongan teks

yang lolos tahap verifikasi akhirnya dimasukkan ke dalam daftar hasil akhir.

```
ALGORITMA search_adaptive_kmp(teks, pola, k)
BEGIN
  positions ← array kosong
  m ← panjang(pola)

  // Kasus khusus: jika k >= panjang pola
  IF k ≥ m THEN
    FOR i ← 0 TO (panjang(teks) - m) DO
      tambahkan i ke positions
    END FOR
    RETURN positions
  END IF

  // Langkah 1: Partisi Pola
  num_parts ← k + 1
  part_length ← m dibagi num_parts (pembagian integer)
  candidate_positions ← set kosong

  // Cari setiap bagian menggunakan KMP
  FOR part_idx ← 0 TO (num_parts - 1) DO
    start_idx ← part_idx × part_length

    IF part_idx = (num_parts - 1) THEN
      // Bagian terakhir mengambil sisa karakter
      seed ← pola[start_idx...akhir]
    ELSE
      seed ← pola[start_idx...start_idx + part_length - 1]
    END IF

    // Skip jika seed terlalu pendek
    IF panjang(seed) = 0 THEN
      CONTINUE
    END IF

    // Cari semua kemunculan seed menggunakan KMP
    seed_positions ← kmp_search(teks, seed)

    // Hitung posisi awal pola dari posisi seed
    FOR EACH seed_pos IN seed_positions DO
      pattern_start ← seed_pos - start_idx
      IF pattern_start ≥ 0 AND (pattern_start + m) ≤
panjang(teks) THEN
        tambahkan pattern_start ke candidate_positions
      END IF
    END FOR
  END FOR

  // Langkah 2: Verifikasi
  FOR EACH pos IN candidate_positions DO
    substring ← teks[pos...pos + m - 1]
    IF hamming_distance(pola, substring) ≤ k THEN
      tambahkan pos ke positions
    END IF
  END FOR

  RETURN positions (terurut)
END
```

Algoritma terakhir yang diimplementasikan adalah algoritma Bitap atau algoritma Shift-Or. Prinsip kerja algoritma ini cukup unik dibandingkan algoritma pencocokan *string* lainnya karena mengandalkan representasi bit dan operasi *bitwise* untuk melakukan pencocokan. Secara garis besar, algoritma ini dibagi menjadi dua tahap, yaitu tahap pra-pemrosesan dan tahap iterasi dan deteksi kecocokan.

Tahap pra-pemrosesan pada algoritma Bitap melibatkan pengecekan panjang pola. Jika panjang pola melebihi batas praktis operasi *bitwise*, yaitu 64 bit, maka algoritma akan beralih ke algoritma Brute Force. Jika tidak, algoritma akan membentuk sebuah *lookup table* yang memetakan setiap karakter ASCII (0-255) ke sebuah *bitmask integer*. Setiap bit dalam mask merepresentasikan posisi kemunculan karakter tersebut dalam pola. Selanjutnya, dua *array* bernama state dan new_state yang berukuran k+1 diinisialisasi dengan nilai 0, dimana k adalah nilai toleransi *edit distance* maksimum. Elemen *state[j]* merepresentasikan semua posisi dalam pola yang dapat dicapai dengan maksimal j operasi edit.

Pada tahap iterasi, program mengiterasi setiap karakter dalam teks. Algoritma memperbarui new_state[0] untuk *exact match*, kemudian memperbarui new_state[j] untuk j=1 hingga k dengan mengombinasikan empat operasi: *match* (karakter cocok), substitusi, insersi, dan delesi. Setelah pembaruan, state dan new_state ditukar untuk iterasi berikutnya. Pada akhir setiap iterasi, program kemudian memeriksa apakah bit ke-(m-*1*) pada *state* aktif, dengan m adalah panjang pola. Bit ini menandakan bahwa pola berhasil dicocokkan dengan memerhatikan batas toleransi ketidakcocokan. Jika ditemukan, posisi awal pola dihitung dan dicatat ke dalam daftar hasil akhir.

```
ALGORITMA search_bitap(teks, pola, k)
BEGIN
  positions ← array kosong
  n ← panjang(teks)
  m ← panjang(pola)

  // Kasus khusus: pola kosong
  IF m = 0 THEN
    RETURN positions
  END IF

  // Batasan: pola tidak boleh lebih dari 64 karakter (ukuran
bit register)
  IF m > 64 THEN
    RETURN search_bruteforce(teks, pola, k)
  END IF

  // Inisialisasi
  MAX_CHAR ← 256
  char_mask ← array berukuran MAX_CHAR berisi 0
```

```
    // Pra-pemrosesan: membuat bitmask
    FOR i ← 0 TO (m - 1) DO
        char ← pola[i]
        char_mask[kode_ASCII(char)] ← char_mask[kode_ASCII(char)] ATAU (1 << i)
    END FOR

    // Inisialisasi state arrays
    state ← array berukuran (k + 1) berisi 0
    new_state ← array berukuran (k + 1) berisi 0

    // Mask untuk mendeteksi match lengkap
    pattern_match_mask ← 1 << (m - 1)

    // Proses setiap karakter dalam teks
    FOR i ← 0 TO (n - 1) DO
        char ← teks[i]
        mask ← char_mask[kode_ASCII(char)]

        // Exact match (0 mismatches)
        new_state[0] ← ((state[0] << 1) ATAU 1) DAN mask

        // Approximate matches (1 sampai k mismatches/edits)
        FOR j ← 1 TO k DO
            new_state[j] ← (
                ((state[j] << 1) ATAU 1) DAN mask ATAU
                (state[j - 1] << 1) ATAU
                new_state[j - 1] ATAU
                state[j - 1]
            )
        END FOR

        // Tukar state arrays untuk iterasi berikutnya
        temp ← state
        state ← new_state
        new_state ← temp

        // Cek apakah ada match dengan toleransi k
        IF (state[k] DAN pattern_match_mask) ≠ 0 THEN
            pos ← i - m + 1
            IF pos ≥ 0 AND pos tidak ada dalam positions THEN
                tambahkan pos ke positions
            END IF
        END IF
    END FOR

    RETURN positions
END
```

V. Hasil Pengujian

Pengujian dilakukan sebanyak tiga kali untuk masing-masing kombinasi yang kemudian akan diambil nilai rata-rata dari waktu eksekusi dan penggunaan memori. Berikut adalah hasil pengujian yang dilakukan untuk masing-masing algoritma dengan setiap kombinasi nilai *k* (toleransi ketidakcocokan) dan ukuran teks.

Tabel 5.1 Hasil Algoritma Brute Force

k	Ukuran teks	Waktu eksekusi (ms)	Penggunaan memori (KB)
1	100 KB	679.1	0.68
1	1 MB	10161.2	0.68
1	10 MB	79056.0	0.68
2	100 KB	710.8	0.73
2	1 MB	10233.5	0.73
2	10 MB	86899.9	0.73
3	100 KB	682.2	0.68
3	1 MB	8880.4	0.68
3	10 MB	83175.1	0.68

Tabel 5.2 Hasil Algoritma KMP dengan Partisi Pola

k	Ukuran teks	Waktu eksekusi (ms)	Penggunaan memori (KB)
1	100 KB	186.1	1.07
1	1 MB	2883.0	1.85
1	10 MB	28454.2	5.72
2	100 KB	289.9	5.04
2	1 MB	4047.7	82.33
2	10 MB	42473.3	1181.31
3	100 KB	405.4	53.90
3	1 MB	5326.2	893.65
3	10 MB	53414.1	4827.02

Tabel 5.3 Hasil Algoritma Bitap

k	Ukuran teks	Waktu eksekusi (ms)	Penggunaan memori (KB)
1	100 KB	345	2.53
1	1 MB	5231.5	2.61
1	10 MB	33199.8	2.62
2	100 KB	348.8	2.61
2	1 MB	4811.8	2.70
2	10 MB	46085.7	2.73
3	100 KB	449.2	2.71
3	1 MB	4758.0	2.81
3	10 MB	52220.6	2.93

VI. Pembahasan

Hasil uji coba untuk masing-masing algoritma membuahkan dua variabel dependen yang ingin diteliti, yaitu lamanya waktu eksekusi dan besarnya penggunaan memori. Analisis untuk hasil uji coba setiap algoritma didasarkan oleh pengaruh dari dua variabel independen pada eksperimen ini, yaitu ukuran teks dan jumlah toleransi ketidakcocokan karakter.

Hasil eksekusi waktu untuk algoritma Brute Force dengan Hamming Distance cukup sesuai dengan kompleksitas waktu teoritisnya. Berdasarkan hasil pada Tabel 5.1, terjadi peningkatan waktu signifikan seiring bertambahnya ukuran teks. Sebagai contoh, untuk nilai $k = 1$, waktu eksekusi melonjak dari 679.1 ms untuk teks 100 KB hingga 79056.0 ms untuk teks 10 MB. Tren peningkatan yang curam ini juga ditemukan pada nilai $k = 2, 3$. Adanya peningkatan drastis

pada algoritma Brute Force karena kesederhanaan algoritmanya yang membandingkan setiap karakter satu per satu. Di sisi lainnya, penggunaan memori untuk algoritma ini cukup minimum, yakni pada rentang 0.68 hingga 0.73 KB, karena tidak perlu menyimpan informasi apapun selain hasil akhir.

Hasil waktu eksekusi algoritma KMP dengan partisi pola yang dapat dilihat pada Tabel 5.2 membuahkan peningkatan waktu eksekusi terhadap ukuran teks yang sangat rendah dibandingkan algoritma Brute Force. Hal ini karena strategi algoritma yang memungkinkan pemrosesan karakter dengan lebih intuitif. Namun, algoritma ini menunjukkan sensitivitas yang cukup tinggi terhadap peningkatan nilai *k.* Sebagai contoh, pada teks berukuran 1 MB, terdapat tren peningkatan waktu eksekusi berdasarkan nilai *k*, yaitu dari 2883.0 ms untuk $k = 1$, 4047.7 untuk $k = 2$, hingga 5326.2 untuk $k = 3$. Tren ini pun dapat diamati pada ukuran teks 100 KB dan 10 MB juga. Peningkatan yang mengikuti meningkatnya nilai *k* disebabkan oleh strategi partisi pola yang, dengan meningkatnya nilai *k,* membuat *seed* atau potongan pola menjadi lebih pendek. *Seed* yang menjadi lebih pendek membuat pola menjadi lebih umum sehingga terjadi ledakan jumlah kandidat yang perlu diverifikasi.

Pembuatan *seed* yang lebih pendek dengan meningkatnya nilai *k* pun menjadi penyebab dari penggunaan memori yang terbesar dibandingkan algoritma-algoritma lainnya. Pada nilai $k = 1$, peningkatan penggunaan memori tidak terjadi secara signifikan. Namun, peningkatan penggunaan memori menjadi lebih terlihat pada hasil nilai $k = 2$ dan $k = 3$. Terjadinya peningkatan penggunaan memori drastis seiring bertambahnya ukuran teks dan nilai *k* disebabkan oleh pembuatan *seed* yang lebih pendek dan lebih umum yang menyebabkan peningkatan jumlah kandidat secara drastis. Karena kandidat-kandidat tersebut perlu disimpan sebelum diverifikasi, penggunaan memori untuk algoritma ini melonjak secara signifikan dibandingkan algoritma-algoritma lainnya.

Hasil algoritma terakhir, yaitu algoritma Bitap, pada Tabel 5.3 menunjukkan kinerja yang paling seimbang dalam segi waktu eksekusi dan penggunaan memori. Secara keseluruhan, algoritma ini membuahkan hasil dengan waktu eksekusi yang sedikit lebih tinggi dibandingkan algoritma KMP dengan partisi pola. Penggunaan memori untuk algoritma ini jauh lebih kecil dibandingkan algoritma KMP dengan partisi pola untuk nilai $k > 1$. Hal ini karena algoritma Bitap tidak perlu menyimpan data yang banyak sehingga penggunaan memorinya relatif konsisten untuk setiap kasus. Selain itu, waktu eksekusi yang relatif cepat didukung oleh penggunaan operasi *bitwise* yang sangat cepat pada tingkat CPU.

VII. KESIMPULAN

Sistem CRISPR-Cas9 merupakan teknologi rekayasa genetika yang memungkinkan penyuntingan genetika. Terdapat tantangan yang muncul dari proses sistem ini, yaitu risiko efek *off-target* yang muncul akibat gRNA yang berikatan dengan sekuens DNA yang mirip dengan targetnya. Untuk memitgasi risiko ini, dapat dimanfaatkan algoritma pencocokan *string* untuk menganalisis potensi lokasi *off-target* pada sekuens DNA. Tiga algoritma pencocokan *string* dengan pendekatan fundamental berbeda, yakni Brute Force, KMP dengan partisi pola, dan Bitap, telah diimplementasikan dan dievaluasikan kinerjanya dari segi eksekusi waktu dan penggunaan memori. Hasil pengujian menunjukkan bahwa algoritma KMP dengan partisi pola membuahkan hasil waktu eksekusi yang paling rendah dibandingkan algoritma-algoritma lainnya, dengan *trade-off* berupa peningkatan penggunaan memori yang drastis. Di sisi lainnya, algoritma Brute Force menghasilkan penggunaan memori yang paling rendah, tetapi dengan waktu eksekusi yang jauh lebih besar dibandingkan algoritma lainnya. Algoritma terakhir, yaitu algoritma Bitap, menunjukkan kinerja yang paling optimal, dengan waktu eksekusi dan penggunaan memori yang relatif rendah dibandingkan kedua algoritma lainnya.

VIII. UCAPAN TERIMA KASIH

Puji syukur saya ucapkan kepada Tuhan Yang Maha Esa karena sebab kuasa dan bimbingan-Nya, makalah ini dapat diselesaikan sesuai dengan tujuannya. Saya juga mengucapkan terima kasih kepada teman-teman dan keluarga saya yang telah memberikan dukungan selama proses penyusunan makalah ini. Ucapan terima kasih khusus saya sampaikan kepada dosen pengampu mata kuliah IF2211 Strategi Algoritma Dr. Ir. Rinaldi Munir, M.T yang telah mengajar dan memberi saya pengetahuan berharga dalam penyusunan makalah ini. Sekiranya makalah ini bisa bermanfaat dan dapat menjadi kontribusi positif bagi ilmu pengetahuan di masa mendatang.

REFERENSI

[1] National Human Genome Research Institute, "What Is Genome editing?," *Genome.gov*, Aug. 15, 2019. https://www.genome.gov/about-genomics/policy-issues/what-is-Genome-Editing

[2] T. Li *et al.*, "CRISPR/Cas9 therapeutics: Progress and Prospects," *Signal Transduction and Targeted Therapy*, vol. 8, no. 1, pp. 1–23, Jan. 2023, doi: https://doi.org/10.1038/s41392-023-01309-7.

[3] I. Gostimskaya, "CRISPR–Cas9: A History of Its Discovery and Ethical Considerations of Its Use in Genome Editing," *Biochemistry (Moscow)*, vol. 87, no. 8, pp. 777–788, Aug. 2022, doi: https://doi.org/10.1134/s0006297922080090.

[4] C. Guo, X. Ma, F. Gao, and Y. Guo, "Off-target Effects in CRISPR/Cas9 Gene Editing," *Frontiers in Bioengineering and Biotechnology*, vol. 11, no. 1143157, Mar. 2023, doi: https://doi.org/10.3389/fbioe.2023.1143157.

[5] GeeksforGeeks, "Hamming Distance between two strings," *GeeksforGeeks*, Sep. 13, 2023. https://www.geeksforgeeks.org/dsa/hamming-distance-two-strings/

[6] GeeksforGeeks, "KMP algorithm for pattern searching," *GeeksforGeeks*, Feb. 25, 2025. https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/

[7] GeeksforGeeks, "Java program to implement BiTaP algorithm for string matching," *GeeksforGeeks*, Mar. 22, 2023. https://www.geeksforgeeks.org/java/java-program-to-implement-bitap-algorithm-for-string-matching/

[8] L. Metcalf and W. Casey, Cybersecurity and applied mathematics. Amsterdam: Elsevier, 2016.

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, 24 Juni 2025

Aria Judhistira - 13523112