Perluasan Algoritma Boyer-Moore sebagai Algoritma Pencocokan *String*/Pola pada *Array* Multidimensional

Agil Fadillah Sabri - 13522006[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522006@std.stei.itb.ac.id

***Abstract*—Algoritma Boyer-Moore merupakan salah satu algoritma pencocokan pola yang efisien untuk *string* satu dimensi, dikenal karena kemampuannya mengurangi jumlah perbandingan yang diperlukan dalam proses pencocokan. Namun, dengan berkembangnya aplikasi yang melibatkan data multidimensi seperti citra, video, dan data ilmiah, muncul kebutuhan untuk memperluas algoritma ini agar dapat menangani pencocokan pola pada *array* multidimensi. Makalah ini mengkaji perluasan algoritma Boyer-Moore ke dalam konteks pencocokan pola pada *array* multidimensi. Dengan mengadopsi analogi bahwa setiap baris atau elemen dari dimensi terluar *array* multidimensi sebagai karakter tunggal dalam *string* satu dimensi, proses pencocokan pola dapat dilakukan dengan metode yang mirip dengan algoritma Boyer-Moore tradisional. Implementasi ini diuji dengan beberapa contoh kasus, menunjukkan bahwa perluasan algoritma Boyer-Moore ini mampu dalam menangani pencocokan pola pada *array* multidimensi. Kesimpulan dari penelitian ini menekankan bahwa dengan penyesuaian yang tepat, algoritma Boyer-Moore dapat diaplikasikan secara efisien pada data multidimensi, membuka peluang baru untuk analisis data dalam berbagai bidang.**

***Kata Kunci*—Algoritma Boyer-Moore; pencocokan pola; *array* multidimensi.**

I. PENDAHULUAN

Pencocokan *string*/pola (*string/pattern matching*) adalah salah satu masalah fundamental dalam ilmu komputer yang memiliki aplikasi luas di berbagai bidang seperti pengolahan teks, bioinformatika, pengenalan pola, dan analisis data. Algoritma Boyer-Moore, yang diperkenalkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977, adalah salah satu algoritma pencocokan pola yang paling efisien dan banyak digunakan. Algoritma ini terkenal karena kemampuannya untuk melakukan pencarian dengan kompleksitas waktu yang mendekati linear dalam praktiknya, terutama ketika digunakan pada teks yang panjang dan pola yang relatif pendek.

Namun, sebagian besar penelitian dan aplikasi Algoritma Boyer-Moore difokuskan pada pencocokan pola pada *string* satu dimensi. Dengan perkembangan teknologi dan meningkatnya kompleksitas data yang harus diolah, kebutuhan akan algoritma pencocokan pola yang dapat bekerja pada data multidimensional menjadi semakin dibutuhkan. *Array* multidimensional, yang sering digunakan untuk merepresentasikan gambar, video, data ilmiah, dan informasi kompleks lainnya, memerlukan pendekatan yang lebih canggih dan spesifik dibandingkan dengan *string* satu dimensi.

Makalah ini bertujuan untuk mengeksplorasi perluasan Algoritma Boyer-Moore untuk digunakan sebagai algoritma pencocokan pola pada *array* multidimensional. Perluasan ini diharapkan dapat menggabungkan efisiensi dan keandalan Algoritma Boyer-Moore dengan kemampuan untuk menangani data dalam format yang lebih kompleks. Dalam makalah ini, akan dibahas berbagai aspek teknis dari adaptasi ini, termasuk perubahan yang diperlukan pada struktur algoritma, strategi penanganan data multidimensional, serta evaluasi kinerja algoritma yang diusulkan.

Melalui penelitian ini, diharapkan dapat memberikan kontribusi signifikan dalam bidang pencocokan pola dan membuka jalan bagi aplikasi praktis yang lebih luas dari algoritma pencocokan pola pada *array* multidimensional. Dengan demikian, makalah ini tidak hanya memperluas pemahaman tentang Algoritma Boyer-Moore tetapi juga menawarkan solusi yang relevan untuk tantangan pencocokan pola dalam konteks data yang semakin kompleks.

II. LANDASAN TEORI

A. Pencocokan *String*/Pola (*String/Pattern Matching*)

a. Definisi Pencocokan *String/* Pola

Pencocokan *string*/pola adalah proses menemukan satu atau beberapa kejadian dari suatu pola (*pattern*) dalam teks atau *array* yang lebih besar (*text*). Tujuan utamanya adalah menemukan satu atau beberapa posisi di mana pola tersebut muncul dalam teks. Konsep ini sangat fundamental dalam ilmu komputer dan memiliki aplikasi luas dalam berbagai bidang seperti pengolahan teks, pencarian informasi, bioinformatika, dan pengenalan pola.

Dalam konteks pencocokan pola, *string* teks (T) adalah urutan simbol atau karakter dari suatu alfabet tertentu, yang umumnya berukuran sangat besar. Adapun pola (P) adalah urutan karakter yang lebih pendek yang dicari dalam *string* T (biasanya P jauh lebih kecil dari T).

Masalah pencocokan pola, dimana diberikan sebuah teks T dengan panjang n dan sebuah pola P dengan panjang m, tujuan dari pencocokan pola adalah menemukan semua posisi (atau posisi pertama) dalam T di mana P muncul sebagai *substring*.

b. Algoritma Pencocokan *String*/Pola

Dalam ilmu komputer, algoritma pencocokan *string*, kadang-kadang disebut algoritma pencarian *string*, adalah kelas penting dari algoritma *string* yang mencoba untuk menemukan tempat di mana satu atau beberapa *string* (juga disebut pola) ditemukan dalam *string* teks yang lebih besar. Terdapat beberapa algoritma pencocokan *string*, yaitu:

1. Algoritma Brute Force

   Algoritma pencocokan pola paling sederhana adalah algoritma brute force, yang memeriksa setiap kemungkinan posisi di mana pola dapat terjadi dalam teks. Kompleksitas waktu dari algoritma ini adalah $O(n * m)$, yang bisa sangat tidak efisien untuk teks yang panjang dan pola yang besar.
2. Algoritma Knuth-Morris-Pratt (KMP)

   Algoritma KMP meningkatkan efisiensi pencocokan pola dengan menggunakan informasi dari perbandingan sebelumnya untuk menghindari perbandingan yang tidak perlu. KMP menggunakan tabel lompatan (*partial match table*) untuk mempercepat pencarian dan memiliki kompleksitas waktu $O(n + m)$.
3. Algoritma Boyer-Moore

   Algoritma Boyer-Moore adalah salah satu algoritma pencocokan pola yang paling efisien untuk pencarian dalam teks panjang. Algoritma ini menggunakan dua aturan utama: "*bad character rule*" dan "*good suffix rule*" untuk melewati beberapa karakter sekaligus dalam teks, sehingga mengurangi jumlah perbandingan yang diperlukan. Kompleksitas waktu terburuk dari algoritma ini adalah $O(n * m)$, tetapi dalam praktek, ia seringkali jauh lebih cepat.
4. Algoritma Rabin-Karp

   Algoritma ini menggunakan hashing untuk menemukan pola dalam teks. Rabin-Karp mengubah pola dan setiap *substring* teks yang diperiksa menjadi hash dan membandingkan hash tersebut. Hal ini memungkinkan pencarian yang cepat dengan kompleksitas waktu rata-rata $O(n + m)$, namun dalam kasus terburuk bisa menjadi $O(n * m)$.

c. Aplikasi Pencocokan *String*/ Pola

Algoritma pencarian pola memiliki banyak aplikasi, termasuk:

1. Pemrosesan Teks: Mencari kata kunci dalam dokumen, menemukan dan mengganti teks, pemeriksaan ejaan, dan deteksi plagiarisme.
2. Pengambilan Informasi: Menemukan dokumen yang relevan dalam database, pencarian web, dan penambangan data.
3. Bioinformatika: Mencari urutan DNA dalam genom, analisis protein, dan analisis ekspresi gen.
4. Keamanan Jaringan: Mendeteksi pola berbahaya dalam lalu lintas jaringan, deteksi intrusi, dan analisis malware.
5. Data Mining: Mengidentifikasi pola dalam kumpulan data besar, segmentasi pelanggan, dan deteksi penipuan.

B. Algoritma Boyer-Moore

a. Definisi Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah salah satu algoritma pencocokan pola yang paling efisien dan banyak digunakan, terutama untuk mencari *substrings* dalam *string* yang panjang. Algoritma ini dikembangkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977 dan sejak saat itu menjadi salah satu algoritma pencocokan pola yang paling efektif dalam praktik.

b. Konsep Dasar Algoritma Boyer-Moore

Algoritma Boyer-Moore bekerja dengan cara yang berbeda dari banyak algoritma pencocokan pola lainnya. Alih-alih membandingkan pola dengan setiap posisi dalam teks secara berurutan, algoritma ini menggunakan dua aturan utama untuk mengabaikan beberapa karakter sekaligus, sehingga mengurangi jumlah perbandingan yang diperlukan.

1. Aturan *Bad Character*

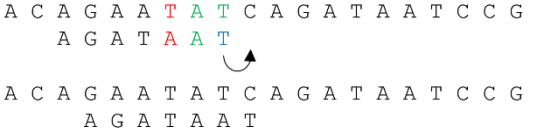
   Jika terjadi ketidaksesuaian karakter antara pola dan teks, algoritma menggunakan informasi tentang posisi terakhir dari karakter teks yang tidak cocok dalam pola untuk menggeser pola ke depan. Karakter teks yang tidak cocok ini disebut dengan *bad character*.

   Ketika terjadi ketidakcocokan, maka pola akan digeser hingga ketidakcocokan menjadi kecocokan atau pola P bergeser melewati karakter yang tidak cocok tersebut. Dalam hal ini, terdapat 3 kasus mengenai bagaimaan pergeseran terjadi.

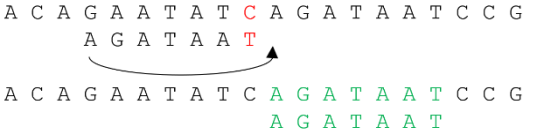
   - Jika *bad character* muncul di dalam pola dan kemunculan terakhirnya terjadi sebelum tempat terjadinya ketidakcocokan (*mismatch*), maka pola akan digeser sedemikian rupa agar *bad character* tersebut cocok dengan karakter pada pola.

     Gambar 1. Pergeseran *Bad Character* Tipe 1
     Sumber: Koleksi Pribadi
   - Jika *bad character* muncul di dalam pola dan kemunculan terakhirnya terjadi setelah tempat terjadinya ketidakcocokan (*mismatch*), maka pola akan digeser sebanyak satu karakter ke kanan.

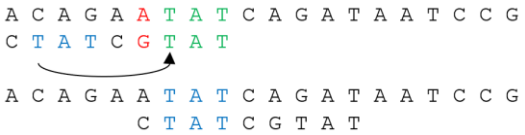
     Gambar 2. Pergeseran *Bad Character* Tipe 2
     Sumber: Koleksi Pribadi
   - Jika *bad character* tidak muncul di dalam pola, maka pola akan digeser melewati *bad character*.

     Gambar 3. Pergeseran *Bad Character* Tipe 3
     Sumber: Koleksi Pribadi

2. Aturan *Good Suffix*

Jika terjadi ketidaksesuaian pada sebuah karakter dalam pola, algoritma mencari kesesuaian dengan *suffix* dari bagian yang cocok dalam pola. Ini memungkinkan pola untuk digeser ke posisi di mana *suffix* yang telah cocok sebelumnya dapat ditemukan lagi di pola.



Gambar 4. Pergeseran *Good Suffix*
Sumber: Koleksi Pribadi

Selain itu, hal lain yang membedakan antara algoritma Boyer-Moore dengan algoritma lainnya yaitu proses pencocokan tiap-tiap karakternya. Algoritma Boyer-Moore melakukan pencocokan dari karakter terakhir pola (dari belakang) dan bergerak menuju karakter pertama pola (menuju depan). Teknik ini disebut dengan teknik *looking-glass*.

c. Implementasi Algoritma Boyer-Moore

Algoritma Boyer-Moore terdiri dari dua fase utama:

1. Fase Pra-pemrosesan:
   - Membangun tabel heuristik *bad character* atau *good suffix* (tergantung keinginan) berdasarkan pola yang diberikan.
   - Tabel *bad character* akan mencatat posisi kemunculan terakhir dari setiap karakter dari alfabet yang digunakan dalam pola.
   - Tabel *good suffix* akan mencatat berapa jauh pola dapat digeser ketika terjadi ketidaksesuaian pada posisi tertentu.
2. Fase Pencocokan:
   - Memulai pencocokan dari awal teks.
   - Pencocokan dilakukan dengan bergerak dari indeks terakhir pola menuju indeks pertama pola (*backwards*/mundur)
   - Menggunakan tabel heuristik untuk menggeser pola secara efisien setiap kali terjadi ketidaksesuaian.

d. Kompleksitas Waktu Algoritma Boyer-Moore

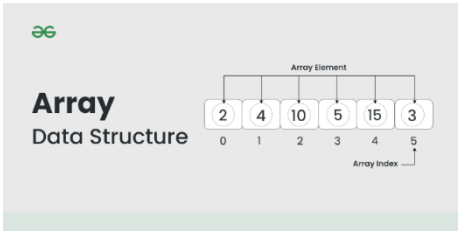
1. Kompleksitas Pra-pemrosesan: $O(m + |\Sigma|)$, di mana m adalah panjang pola dan $|\Sigma|$ adalah ukuran alfabet.
2. Kompleksitas Pencocokan: $O(n/m)$ dalam kasus terbaik dan $O(n * m)$ dalam kasus terburuk, di mana n adalah panjang teks. Namun, dalam praktiknya, algoritma ini sangat efisien dan sering kali mendekati $O(n)$.

C. *Array*

a. Definisi *Array*

Array/larik adalah salah satu struktur data fundamental dalam ilmu komputer yang digunakan untuk menyimpan sejumlah elemen yang memiliki tipe data yang sama. *Array* memungkinkan akses yang cepat dan efisien ke elemen-elemen individual menggunakan indeks. Dalam banyak bahasa pemrograman, biasanya indeks sebuah *array* dimulai dengan 0.

Array adalah salah satu struktur data tertua dan paling penting, yang digunakan oleh hampir setiap program. Selain itu *array* juga digunakan untuk mengimplementasikan banyak struktur data lainnya, seperti *list*, *string*, dll.



Gambar 5. Struktur Data *Array*
Sumber: https://www.geeksforgeeks.org/array-data-structure-guide/

b. Jenis-Jenis *Array*

1. *Array* Satu Dimensi

   Array satu dimensi adalah kumpulan elemen yang diatur dalam urutan linier. Setiap elemen dalam *array* dapat diakses melalui indeks numerik yang biasanya dimulai dari nol.
2. *Array* Multidimensi

   Array multidimensi pada dasarnya merupakan sebuah *array* yang elemen-elemennya juga merupakan sebuah *array*, yang memungkinkan penyimpanan data dalam bentuk lebih dari satu dimensi. *Array* dua dimensi adalah bentuk paling umum dari *array* multidimensi, sering digunakan untuk merepresentasikan matriks atau tabel.

III. NOTASI DAN TERMINOLOGI

Dalam bagian ini, kita mendefinisikan simbol dan notasi yang akan digunakan di seluruh makalah, terutama pada bagian BAB IMPLEMENTASI.

A. *Array*

1. *Array* akan didefinisikan dalam notasi kurung kurawal yang di dalamnya berisi elemen-elemen *array* yang masing-masing dipisahkan dengan tanda koma. Nama *array* akan menggunakan huruf besar. Contoh:
   A = {a, b, c}
   B = {{a, b}, {c, d}, {e, f}}
   C = {{{a, b}, {c, d}},{{e, f}, {g, h}}}
2. Pengaksesan elemen-elemen *array* dilakukan dengan menuliskan nama *array* yang diikuti oleh tanda kurung siku untuk setiap dimensi *array*.
   $$A[i_1][i_2][i_3]\ldots[i_k]$$
   Pada notasi di atas, A merupakan nama *array*, i_1 merupakan indeks lokasi untuk dimensi pada lapisan pertama/terluar *array*, i_2 merupakan indeks untuk dimensi pada lapisan kedua, i_3 merupakan indeks untuk lapisan ketiga, dan i_k merupakan indeks untuk lapisan ke-*k* dari *array*.
3. Indeks *array* selalu dimulai dari 0.

B. Teks dan Pola

Simbol-simbol berikut akan digunakan seterusnya pada BAB IMPLEMENTASI.

1. T : Teks.
2. P : Pola.
3. n : panjang teks T (untuk dimensi lapisan terluar).
4. m : panjang pola P (untuk dimensi lapisan terluar).
5. Σ : himpunan karakter. Karakter yang akan digunakan adalah seluruh karakter ASCII (128 karakter).

IV. IMPLEMENTASI

A. Algoritma Boyer-Moore pada Proses Pencocokan Pola pada *String* Satu Dimensi

Dalam algoritma Boyer-Moore pada makalah ini, akan digunakan aturan *Bad Character* dalam tahap pra-pemrosesan, yang akan mendaftarkan setiap kemunculan terakhir dari setiap karakter ASCII pada pola.

a. Tahap Pra-Pemrosesan

Dalam tahap pra-pemrosesan, akan dibuat tabel utama berdasarkan aturan *bad character*, yang disebut dengan tabel *last occurrence*, yang akan mencatat posisi terakhir dari setiap karakter ASCII di dalam pola. Jika suatu karakter tidak muncul di dalam pola, maka posisi karakter tersebut akan diberi nilai -1. Berikut adalah kode dalam bahasa pemrograman Python untuk membuat tabel *last occurrence*.

```
def lastOccurence(pattern: list[str]) -> list[int]:
  last = [-1] * 128
  m = len(pattern)
  for i in range(m):
    last[ord(pattern[i])] = i
  return last
```

Dalam algoritma tersebut, kita melakukan traversal sebanyak $|\Sigma|$ (dalam hal ini $|\Sigma| = 128$), yang kemudian diikuti oleh traversal sebanyak panjang dari *pattern* (m). Maka dalam tahap pra-pemrosesan, kompleksitas algoritmanya adalah $O(m+|\Sigma|)$.

b. Tahap Pencocokan/Pencarian

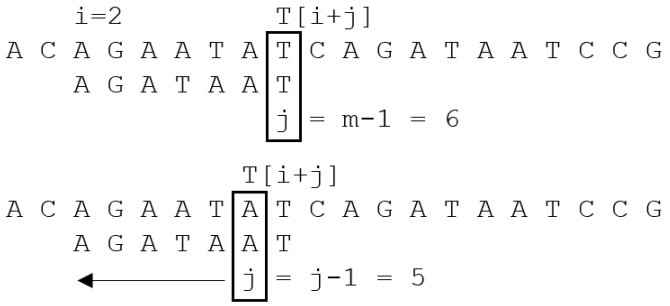
Dalam tahap pencocokan ini, kita akan mencari semua indeks di dalam teks dimana *pattern* muncul. Berikut adalah implementasinya dalam bahasa Python.

```
def boyerMooreSearch(text: list[str],
                     pattern: list[str])
                     -> list[int]:
  n = len(text)
  m = len(pattern)
  last = lastOccurence(pattern)
  occurrences = []

  i = 0 # text index
  while i <= n - m:
    j = m - 1 # pattern index
    while j >= 0 and pattern[j] == text[i + j]:
      j -= 1

    if j < 0: # match
      occurrences.append(i)
      i += m
    else: # mismatch
      i += max(1, j - last[ord(text[i + j])])

  return occurrences
```

Dalam kode di atas, *i* menjadi indeks dimana posisi karakter pertama dalam *pattern* sejajar terhadap teks.

Gambar 6. Ilustrasi i (i = 2)
Sumber: Koleksi Pribadi

Adapun *j* menjadi indeks tempat kita mengakses *pattern* pada saat proses pencocokan pola.



Gambar 7. Ilustrasi Pencocokan Pola
Sumber: Koleksi Pribadi

Pattern P muncul di dalam *Text T* ditandai saat indeks *j* bernilai negatif. Pada saat itu terjadi, maka kita akan mencatat indeks *i* tempat munculnya *pattern* tersebut di dalam teks.

Adapun saat terjadi *mismatch* (terjadi ketidakcocokan karakter antara teks dan pola), maka akan dilakukan proses *jump character*. Teknik *jump character* ini dilakukan dengan menambah nilai indeks *i* agar posisi *pattern* bergeser ke kanan. Adapun besarnya pergeseran ditentukan oleh nilai tabel *last occurrence* untuk karakter yang *mismatch* tersebut.

Tabel *last occurrence* untuk contoh di atas:

A	C	G	T	Lainnya
5	-1	1	6	-1

Adapun penambahan nilai *i* (besarnya pergeseran *pattern*) dapat dilihat pada 3 kasus berikut:

1. Karakter teks yang *mismatch* muncul di dalam pola dengan kemunculan terakhir terjadi sebelum tempat terjadinya *mismacth*.

Gambar 8. *Mismacth* Tipe 1
Sumber: Koleksi Pribadi

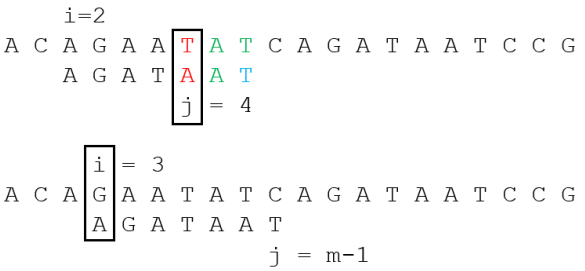
Dari contoh di atas, dapat kita lihat bahwa *i* harus digeser sejauh 2 pergeseran ke kanan agar karakter G pada teks cocok dengna karakter G pada pola. Nilai 2 ini sebenarnya diperoleh dari:

$$shift = j - last\_occurence(mismatch)$$

Ket : *shift* : pergeseran *pattern*

i = i + shift
i = i + (j – last_occurence(G))
i = 0 + (3 - 1)
i = 2

2. Karakter teks yang *mismatch* muncul di dalam pola dengan kemunculan terakhir terjadi setelah tempat terjadinya *mismacth*.

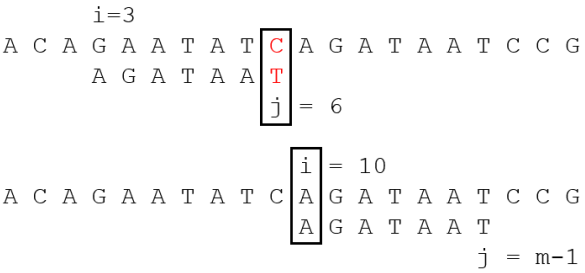


Gambar 9. *Mismacth* Tipe 2
Sumber: Koleksi Pribadi

Pada tipe *mismatch* ini, besarnya pergeseran selalu 1, karena nilai *shift* berdasarkan persamaan sebelumnya akan bernilai negatif, sedangkan pergeseran ke kiri tidak dapat dilakukan.

$$shift = 1$$

3. Karakter teks yang *mismatch* tidak muncul di dalam pola.



Gambar 10. *Mismacth* Tipe 3
Sumber: Koleksi Pribadi

Pada tipe *mismatch* ini, *i* harus digeser sedemikian rupa sehingga posisi *i* saat ini tepat di sebelah kanan karakter yang *mismatch*. Besar pergeseran ini sama dengan (*j*+1).

shift = *j* + 1
shift = *j* - (-1)

Nilai -1 sama dengan nilai *last occurrence* dari karakter yang tidak muncul di dalam pola, sehingga nilai *shift* dapat dinyatakan sebagai:

$$shift = j - last\_occurence(mismatch)$$

i = *i* + *shift*
i = *i* + (*j* – *last_occurence(C)*)
i = *3* + (*6* – *(-1)*)
i = *10*

dari 3 kondisi di atas, besarnya nilai pergeseran *pattern* dapat disederhanakan menjadi:

$$shift = max(1, j - last\_occurence(mismatch))$$

B. Perluasan Algoritma Boyer-Moore pada Proses Pencocokan Pola pada *String* Dua Dimensi

a. Penganalogian *String* Dua Dimensi Dengan *String* Satu Dimensi

Berikut adalah contoh pola dan teks dua dimensi.

```
Teks:
T G C T T G C A C T G G A G G A G C G C
G A G G A A A C T G G C T C T G C T C G
C A A C T C G G C A A C T G G C A C T G
G A C G G A C T T C A C G G T G A C G G
C T C C C A G G C C A G A T A T G A G T
C C C C G T T A T C A T T A C G A T A C
A A T A T A C G A A C A G T A C C C A T
G T G C C A C A C G T C G G T C C A C T
G G T A T A A G T A C C A G T T G C C T
G T G C C G C A C G T G A G G G C G C A
T G C T T G C A C T G G A G G A G C G C

Pola:
C T C G
G T A C
A C T G
A C G G
```

Gambar 11. Teks dan Pola 2 Dimensi
Sumber: Koleksi Pribadi

Proses pencocokan pola pada *string* dua dimensi lebih rumit daripada proses pencocokan pola pada *string* satu dimensi. Untuk mempermudahnya, dapat dilakukan analogi seperti berikut, dengan tujuan untuk mengubah pandangan mengenai *string* dua dimensi tersebut seolah-olah seperti hanya satu dimensi.

1. Setiap baris pada teks dan *pattern* dianalogikan seperti satu buah karakter tunggal pada *string* satu dimensi. Dengan analogi tersebut, contoh pada **Gambar 11.** akan berubah menjadi:

```
T[0]    P[0]
T[1]    P[1]
T[2]    P[2]
T[3]    P[3]
T[4]
T[5]
T[6]
T[7]
T[8]
T[9]
T[10]
```

Gambar 12. Analogi *String* 2 Dimensi menjadi 1 Dimensi
Sumber: Koleksi Pribadi

2. Untuk tabel *last occurrence*, dapat dibuat dengan menggunakan fungsi *hash*, dengan *key*-nya adalah elemen dari tiap-tiap baris *pattern* (berupa *string* satu dimensi) dan *value*-nya adalah indeks kemunculan terakhir di *pattern*.
3. Proses pencocokan pola selanjutnya dapat dilakukan seolah-olah seperti proses pencocokan pola untuk *string* 1 dimensi.

b. Langkah-Langkah Penyelesaian Masalah

Secara umum, langkah-langkah untuk melakukan proses pencocokan pola untuk *string* dua dimensi adalah sebagai berikut:

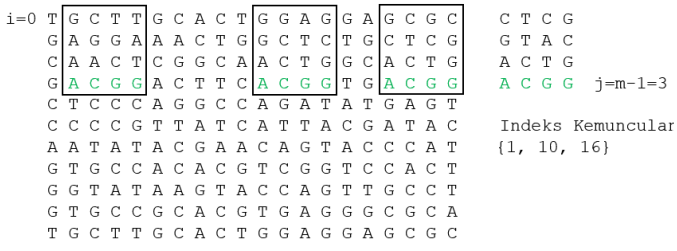
1. Lakukan pra-pemrosesan untuk menghasilkan tabel *last occurrence* dari *pattern* dengan menggunakan fungsi *hash*.
2. Proses pencocokan pola dimulai dari indeks $i = 0$ hingga $i > (n-m)$ (dengan n banyaknya baris pada teks dan m banyaknya baris pada pola).
3. Proses pencocokan pola dengan teks dimulai dari indeks $j = m\text{-}1$, dengan mencari semua indeks kemunculan P[$j = m$-1] di dalam T[$i+j = i + m$-1] dengan memanggil fungsi/prosedur proses pencocokan pola dengan algoritma Boyer-Moore untuk *string* satu dimensi. Simpan setiap indeks tersebut ke dalam suatu tabel kemunculan.

   Contoh berdasarkan **Gambar 11.**

Gambar 13. Pencocokan Pola pada Indeks j=m-1
Sumber: Koleksi Pribadi

Berdasarkan gambar di atas, indeks kemunculan pola P[m-1] pada T[i+(m-1)] adalah {1, 10, 16}.

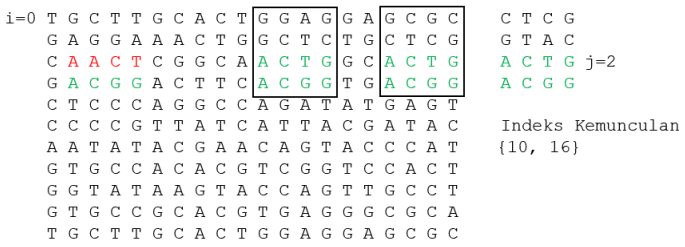
4. Untuk setiap indeks kemunculan, lakukan proses pencocokan pola untuk baris ke-(j-1) dengan secara langsung mengakses dari kolom pada indeks kemunculan tersebut saja, tidak perlu dari kolom ke-0. Lakukan proses pencocokan hingga $j < 0$ atau hingga ditemukan ketidakcocokan, yaitu saat P[j_k] tidak muncul di dalam T[$i+j_k$].



Gambar 14. Kolom Tempat Pencocokan Pola Untuk Baris j-1 dan Seterusnya
Sumber: Koleksi Pribadi

Pada gambar diatas, proses pencocokan pola cukup dilakukan pada kolom-kolom yang berada di dalam kotak.

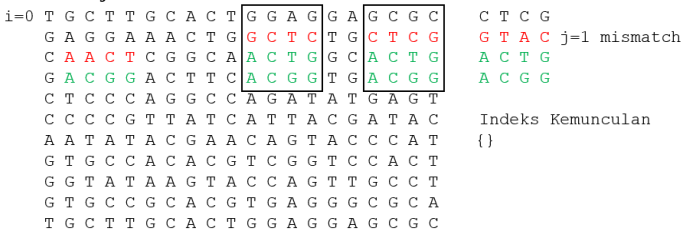
5. Jika pada suatu indeks kemunculan tertentu *string* pada baris tertentu pada *pattern* tidak sesuai dengan yang terdapat di dalam teks, hapus indeks kemunculan tersebut agar tidak perlu diperiksa kembali pada proses pencocokan baris berikutnya.



Gambar 15. Ketidakcocokan Pola pada Indeks Kemunculan = 1 untuk $j = 2$
Sumber: Koleksi Pribadi

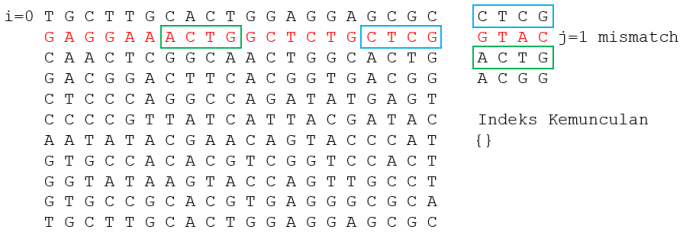
Karena pada kolom 1 terjadi ketidaksesuaian antara pola P[j] dengan teks T[i+j], maka hapus 1 dari tabel indeks kemunculan.

6. Jika indeks kemunculan terus dihapus hingga tabel menjadi kosong, maka pada baris tersebut dikatakan terjadi *mismatch*.



Gambar 16. Mismatch pada Baris j = 1
Sumber: Koleksi Pribadi

7. Saat terjadi *mismatch*, lakukan pencarian baris *pattern* yang mungkin muncul pada baris teks tempat terjadinya *mismatch*. Jika terdapat baris *pattern* yang muncul pada baris teks tersebut, ambil *value last occurrence*-nya dari tabel *hash*. Jika tidak, maka *value last occurrence*-nya adalah -1. Jika terdapat lebih dari satu baris *pattern* yang muncul, maka ambil baris *pattern* dengan *last occurrence* terbesar.



Gambar 17. Baris *Pattern* yang Muncul Pada Baris Teks Tempat Terjadi *Mismatch*
Sumber: Koleksi Pribadi

Pada contoh di atas, pada baris teks T[$i+j$] = T[0+1] = T[1], muncul pola pada baris *pattern* P[2] = [A, C, T, G] pada kolom ke-6, dan *pattern* P[0] = [C, T, C, G] pada kolom ke-16. Maka diambil baris dengan *value last occurrence* terbesar, yaitu baris P[2].

8. Aturan pergeseran *pattern*, sama seperti pada aturan pergeseran *pattern* pada algoritma Boyer-Moore untuk *string* 1 dimensi.

$$shift = max(1, j - last\_occurence(mismatch))$$

Berdasarkan contoh sebelumnya, maka:

shift = *max*(1, *j-last_occurence*(*mismatch*))

shift = *max*(1, 1 – 2)

shift = *max*(1, -1)

shift = 1

sehingga:

i = *i* + *shift*

i = 0 + 1

i = 1

```
    T G C T T G C A C T G G A G G A G C G C
i=0 G A G G A A A C T G G C T C T G C T C G    C T C G
    C A A C T C G G C A A C T G G C A C T G    G T A C
    G A C G G A C T T C A C G G T G A C G G    A C T G
    C T C C C A G G C C A G A T A T G A G T    A C G G  j=m-1=3
    C C C C G T T A T C A T T A C G A T A C
    A A T A T A C G A A C A G T A C C C A T
    G T G C C A C A C G T C G G T C C A C T
    G G T A T A A G T A C C A G T T G C C T
    G T G C C G C A C G T G A G G G C G C A
    T G C T T G C A C T G G A G G A G C G C
```

Gambar 18. Hasil Pergeseran
Sumber: Koleksi Pribadi

Ulangi langkah 3-8 hingga seluruh bagian teks telah diproses.

c. Implementasi Program

1. Fungsi/Prosedur Pembantu

```
def hashMap(pattern: list[str]) -> int:
# hash map sederhana yang menjumlahkan:
# nilai ASCII setiap karakter dalam
# pattern yang dikalikan dengan indeksnya
  sum = 0
  for i in range(len(pattern)):
    sum += ord(pattern[i]) * i
  return sum

def isSame(text: list[str],
           pattern: list[str],
           kolom: int) -> bool:
# Mencocokkan text pada kolom tertentu
# dengan pattern
  for i in range(len(pattern)):
    if text[i+kolom] != pattern[i]:
      return False
  return True
```

2. Tahap Pra-Pemrosesan

Tabel *last occurrence* akan dibuat menggunakan fungsi *hash map* yang akan mengubah suatu *string* menjadi suatu bilangan bulat tertentu.

```
def lastOccurence2D(pattern: list[list[str]])
                    -> dict[int, int]:
  dict = {}
  for i in range(len(pattern)):
    dict[hashMap(pattern[i])] = i
  return dict
```

Dalam algoritma tersebut, memanggil fungsi untuk melakukan *hash map* dari *string* menjadi *integer*. Adapun kompleksitas waktu dari fungsi hashMap adalah $O(2*m-1) = O(m)$. Maka untuk tahap pra-pemrosesan, kompleksitas waktunya adalah $O(m*m) = O(m^2)$.

3. Tahap Pencocokan/Pencarian

Dalam tahap pencocokan ini, kita akan mencari semua indeks di dalam teks dimana *pattern* muncul. Berikut adalah implementasinya dalam bahasa Python.

```
def boyerMooreSearch2D(text: list[list[str]],
                       pattern: list[list[str]])
                       -> list[list[int]]:
  n = len(text)
  m = len(pattern)
  last : dict[str, int] = lastOccurence2D(pattern)
  occurrences = []

  i = 0 # text index
  while i <= n - m:
    j = m - 1 # pattern index
    indeks_kemunculan = boyerMooreSearch(
                        list(text[i+j]),
                        list(pattern[j]))
    j -= 1

    isTrue = True # is indeks_kemunculan masih ada
    while j >= 0 and isTrue:
      if indeks_kemunculan == []:
        isTrue = False
      else:
        new_indeks_kemunculan = []
        for kolom in indeks_kemunculan:
          if isSame(text[i+j], pattern[j], kolom):
            new_indeks_kemunculan.append(kolom)
        indeks_kemunculan = new_indeks_kemunculan
        j -= 1

    if j < 0: # match
      idx = [i, indeks_kemunculan[0]]
      occurrences.append(idx)
      i += m
    else: # mismatch
      maxValue = -1
      kolomText = len(text[0])
      kolomPattern = len(pattern[0])
      for ptrn in range(kolomText-kolomPattern+1):
        hashPtrn = hashMap(
                 text[i+j][ptrn:ptrn+kolomPattern])
        if last.get(hashPtrn) != None:
          maxValue = max(maxValue, last[hashPtrn])
      i += max(1, j - maxValue)
```

Berikut adalah beberapa hasil pencocokan pola yang dilakukan:

```
   0                   1
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0  T G C T T G C A C T G G A G G A G C G C    G C G C
1  G A G G A A A C T G G C T C T G C T C G    C T C G
2  C A A C T C G G C A A C T G G C A C T G    A C T G
3  G A C G G A C T T C A C G G T G A C G G    A C G G
4  C T C C C A G G C C A G A T A T G A G T    Indeks Kemunculan: [[0, 16], [5, 11]]
5  C C C C G T T A T C A G C G C G A T A C
6  A A T A T A C G A A C C T C G C C C A T
7  G T G C C A C A C G T A C T G C C A C T
8  G G T A T A A G T A C A C G G T G C C T
9  G T G C C G C A C G T G A G G G C G C A
```

Gambar 19. Hasil Uji Coba 1
Sumber: Koleksi Pribadi

```
   0                   1
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0  T G C T T G C A C T G G A G G A G C G C    G A A C
1  G A G G A A A C T G G C T C T G C T C G    A C G T
2  C A A C T C G G C A A C T G G C A C T G    G T A C
3  G A C G G A C T T C A C G G T G A C G G    Indeks Kemunculan: [[6, 7]]
4  C T C C C A G G C C A G A T A T G A G T
5  C C C C G T T A T C A T T A C G A T A C
6  A A T A T A C G A A C A G T A C C C A T
7  G T G C C A C A C G T C G G T C C A C T
8  G G T A T A A G T A C C A G T T G C C T
9  G T G C C G C A C G T G A G G G C G C A
```

Gambar 20. Hasil Uji Coba 2
Sumber: Koleksi Pribadi

C. Perluasan Algoritma Boyer-Moore pada Proses Pencocokan Pola pada *String* N Dimensi

a. Penganalogian *String* N-Dimensi Dengan *String* Satu Dimensi

Seperti pada bagian sebelumnya, untuk mempermudah pemahaman, dapat dilakukan analogi seperti berikut, dengan tujuan untuk mengubah pandangan mengenai *string* N dimensi seolah-olah seperti hanya satu dimensi.

1. Setiap elemen dari dimensi terluar *array* N Dimensi, yaitu A = [], baik pada teks maupun pada *pattern* dianalogikan seperti satu karakter tunggal pada *string* satu dimensi. Dengan analogi tersebut, maka proses pencocokan *string* akan bekerja seperti berikut:
   T = T[0]T[1]T[2]…T[*n*]
   P = P[0]P[1]P[2]…P[*m*]
   Dengan masing-masing T[*i*] dan P[*i*] merupakan *array* (N-1) dimensi.
2. Untuk tabel *last occurrence*, dapat dibuat dengan menggunakan fungsi *hash*, dengan *key*-nya adalah P[*i*] (berupa *string* (N-1) dimensi) dan *value*-nya adalah indeks kemunculan terakhir di *pattern*.
3. Proses pencocokan pola selanjutnya dapat dilakukan seolah-olah seperti proses pencocokan pola untuk *string* 1 dimensi.

b. Langkah-Langkah Penyelesaian Masalah

Algoritma Boyer-Moore pada proses pencocokan pola pada *string* N dimensi memiliki sifat yang mirip seperti fungsi rekursif. Namun alih-alih memanggil dirinya sendiri, dalam algoritma ini, algoritma Boyer-Moore N dimensi akan memanggil fungsi algoritma Boyer-Moore (N-1) dimensi. Proses pemanggilan fungsi ini akan terus dilakukan hingga fungsi algoritma Boyer-Moore satu dimensi dipanggil. Jadi dapat dikatakan algoritma Boyer-Moore satu dimensi akan menjadi basis pemanggilan fungsi algoritma Boyer-Moore N dimensi.

Adapun langkah-langkah lebih lengkapnya yaitu:

1. Lakukan pra-pemrosesan untuk menghasilkan tabel *last occurrence* dari *pattern* dengan menggunakan fungsi *hash*.
2. Proses pencocokan pola dimulai dari indeks $i = 0$ hingga $i > (n\text{-}m)$ (dengan i merupakan pengakses indeks dimensi terluar teks).
3. Proses pencocokan pola dengan teks dimulai dari indeks $j = m\text{-}1$ (dengan j merupakan pengakses indeks dimensi terluar *pattern*), dengan mencari semua indeks kemunculan P[$j = m$-1] di dalam T[$i+j = i + m$-1] dengan memanggil fungsi/prosedur proses pencocokan pola algoritma Boyer-Moore untuk *string* (N-1) dimensi (proses yang mirip dengan rekursif). Simpan setiap indeks tersebut ke dalam suatu tabel kemunculan.
4. Untuk setiap indeks kemunculan, lakukan proses pencocokan pola untuk baris ke-(j-1) dengan kembali memanggil fungsi/prosedur proses pencocokan pola algoritma Boyer-Moore untuk *string* (N-1) dimensi. Indeks kemunculan pada tahap ini mungkin berbeda dengan indeks kemunculan pada tahap 3. Lakukan operasi *intersection* untuk mengambil indeks kemunculan yang sama pada tahap 3 dan 4. Hasil *intersection* ini akan menjadi tabel kemunculan yang baru.
5. Ulangi langkah 4 hingga nilai $j < 0$ atau hingga hasil *intersection*-nya merupakan himpunan kosong.
6. Jika nilai $j < 0$, maka simpan nilai i pada saat itu, karena pola berhasil ditemukan di dalam teks. Lanjutkan proses untuk pencarian lokasi pola di tempat lain dengan i yang baru adalah i yang lama di tambah m.
   $$i = i + m$$
7. Jika nilai $j > 0$ dan hasil *intersection*-nya adalah himpunan kosong, maka pada j tersebut dikatakan terjadi *mismatch*.
8. Saat terjadi *mismatch*, lakukan pencarian elemen *pattern* yang mungkin muncul pada elemen teks tempat terjadinya *mismatch*. Jika terdapat *pattern* pada *indeks* tertentu yang muncul pada tempat *mismatch* tersebut, ambil *value last occurrence*-nya dari tabel *hash*. Jika tidak, maka *value last occurrence*-nya adalah -1. Jika terdapat lebih dari satu elemen *pattern* yang muncul, maka ambil elemen *pattern* dengan *last occurrence* terbesar.
9. Lakukan pergeseran *pattern*, dengan aturan pergeseran *pattern* sama seperti pada aturan pergeseran *pattern* pada algoritma Boyer-Moore untuk *string* 1 dimensi.

   $$shift = max(1, j - last\_occurence(mismatch))$$

10. Ulangi langkah-langkah di atas hingga seluruh bagian teks telah diproses.

V. KESIMPULAN

Dalam makalah ini telah berhasil memperluas algoritma Boyer-Moore dari proses pencocokan pola pada *string* satu dimensi menjadi pencocokan pola pada *array* multidimensi. Perluasan ini dilakukan dengan mengadaptasi konsep dasar algoritma Boyer-Moore ke dalam konteks *array* dua dimensi dan N dimensi, dengan memperhatikan kompleksitas tambahan yang muncul pada struktur data yang lebih kompleks.

Dengan adaptasi ini, algoritma Boyer-Moore yang semula hanya efektif untuk *string* satu dimensi kini dapat digunakan untuk *array* multidimensi dengan mempertahankan efisiensi dan kecepatan yang menjadi keunggulan utama dari algoritma ini. Implementasi dan pengujian lebih lanjut diperlukan untuk mengevaluasi performa algoritma ini dalam berbagai kondisi dan ukuran data.

VI. UCAPAN TERIMA KASIH

Pertama-tama, penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya penulis dapat menyelesaikan makalah ini dengan baik dan tepat waktu. Penulis juga mengucapkan terima kasih kepada orang tua karena selalu memberikan semangat dan dukungan, termasuk dalam proses pembuatan makalah ini. Tidak lupa, penulis juga mengucapkan terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc. sebagai dosen pengampu mata kuliah IF2211 Strategi Algoritma kelas K02, Semester II Tahun 2023/2024, karena telah memberikan bimbingan dan ilmu pengetahuan yang berharga dalam penyelesaian makalah ini. Penulis juga mengucapkan terima kasih kepada teman-teman yang telah membantu dalam penyelesaian makalah ini.

REFERENSI

[1] Boyer, R. S., & Moore, J. S. 1977. *A Fast String Searching Algorithm*. Communications of the ACM, 20(10). https://dl.acm.org/doi/pdf/10.1145/359842.359859. (Diakses 10 Juni 2024).

[2] Hare. 2024. *Array Data Structure Guide*. https://www.geeksforgeeks.org/array-data-structure-guide/. (Diakses 10 Juni 2024).

[3] Munir, Rinaldi. 2024. *Pencocokan String* (*String/Pattern Matching*). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf. (Diakses 10 Juni 2024).

[4] Rathi, Aarti. 2024. *Boyer Moore Algorithm for Pattern Searching*. https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/. (Diakses 10 Juni 2024).

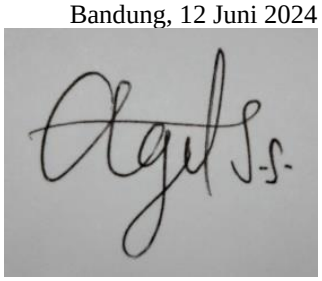
[5] Unknown. (n.d). *Java Multi-Dimensional Arrays: Multidimensional Arrays*. https://www.w3schools.com/java/java_arrays_multi.asp. (Diakses 10 Juni 2024).

[6] Unknown. 2024. *Pattern Searching*. https://www.geeksforgeeks.org/algorithms-gq/pattern-searching/. (Diakses 10 Juni 2024).

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, 12 Juni 2024



Agil Fadillah Sabri
(13522006)