Pencarian Komponen PC dalam *Game* PC Building Simulator dengan Algoritma *Brute Force*, Knuth-Morris-Pratt, dan Boyer-Moore

Bastian H. Suryapratama - 13522034
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522034@std.stei.itb.ac.id

***Abstract*—PC Building Simulator adalah sebuah *game* simulasi dalam membuat maupun memperbaiki PC (*personal computer*). Pembelian komponen PC yang tepat merupakan aspek yang sangat penting dalam permainan ini. Untuk mencari komponen PC, dapat digunakan algoritma *string matching brute force*, Knuth-Morris-Pratt, dan Boyer-Moore. Secara umum, algoritma Knuth-Morris-Pratt merupakan algoritma yang paling efisien dalam melakukan pencarian komponen PC dibandingkan algoritma lainnya.**

Keywords—string matching; pattern matching; pencocokan string; brute force; kmp; knuth-morris-pratt; bm; boyer-moore; pc building simulator;

I. PENDAHULUAN

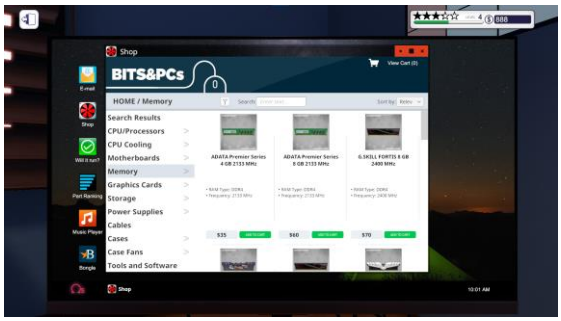
PC Building Simulator adalah salah satu permainan simulasi dalam membuat maupun memperbaiki PC (*Personal Computer*) *Desktop*. Terdapat berbagai persoalan yang harus diselesaikan pemain, mulai dari yang sederhana, seperti membersihkan *malware* dan membersihkan debu pada komponen PC, sampai yang cukup kompleks, seperti melakukan *upgrade* PC agar memenuhi spesifikasi yang dibutuhkan untuk menjalankan suatu perangkat lunak.

PC Building Simulator di-*release* pada *platform* Steam pada tanggal 29 Januari 2019. Permainan ini dikembangkan oleh Claudiu Kiss dan The Irregular Corporation, serta di-*publish* oleh The Irregular Corporation. Permainan ini mendapatkan respon yang sangat baik dari pengguna-penggunanya, dengan lebih dari 90 persen *review* positif dari pengguna hingga pertengahan Juni 2024.

Pembelian komponen PC adalah salah satu mekanisme inti dari permainan ini. Terdapat berbagai komponen PC yang dapat dibeli, seperti CPU, *motherboard*, RAM, *graphics card*, dan lain-lain. Komponen-komponen tersebut memiliki spesifikasi yang beragam. Selain itu, komponen-komponen tersebut juga memiliki *brand* yang berbeda-beda.

Dalam permainan ini, pemilihan komponen PC merupakan keputusan yang sangat krusial. Pemilihan komponen PC yang salah dapat menyebabkan komputer tidak dapat dijalankan. Jika terjadi demikian, uang yang dimiliki akan habis dengan sia-sia karena komponen yang telah dibeli tidak dapat dikembalikan. Selain itu, pelanggan juga akan kecewa sehingga memberikan ulasan (*review*) yang kurang baik. Ulasan yang buruk membuat pemain kesulitan mendapatkan pelanggan baru.

Karena banyaknya komponen PC yang dapat dipilih, terdapat fitur untuk mencari komponen PC yang diinginkan. Pengguna cukup memasukkan suatu kata kunci, kemudian program akan menampilkan komponen-komponen PC yang cocok dengan kata kunci yang diberikan.



Gambar 1.1. Berbagai Komponen PC yang Dapat Dibeli

Sumber: dokumentasi penulis

Pada makalah ini, penulis akan membuat fitur pencarian sederhana untuk mencari komponen PC yang diinginkan. Fitur pencarian tersebut dibuat dengan tiga jenis algoritma, yaitu *brute force*, KMP (Knuth-Morris-Pratt), dan BM (Boyer-Moore). Penulis akan membahas cara kerja ketiga algoritma tersebut. Selain itu, penulis akan membandingkan ketiga algoritma tersebut dari segi efisiensinya.

II. LANDASAN TEORI

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

Misalkan diberikan suatu teks, yaitu sebuah string yang panjangnya n buah karakter, serta suatu *pattern*, yaitu sebuah string yang panjangnya m buah karakter, dengan m <<< n. *String/pattern matching* adalah proses untuk mencari lokasi di

dalam suatu teks yang bersesuaian atau cocok dengan *pattern* yang diberikan.

Contoh:

Teks: `the rain in spain stays mainly on the plain`

Pattern: `main`

Terdapat banyak kasus penggunaan *string matching*, di antaranya pencarian di dalam *text editor*, *web search engine*, analisis citra, serta dalam bidang *bioinformatics*, seperti pencocokan rantai asam amino pada rantai DNA.

B. Algoritma Brute Force pada Pencocokan String

Algoritma *brute force* merupakan pendekatan yang *straightforward* dalam menyelesaikan suatu persoalan. Algoritma *brute force* dapat diterapkan dalam berbagai kasus, seperti pencarian nilai maksimum dan minimum, pencarian suatu nilai di dalam *array*, perkalian dua buah matriks, pengurutan elemen-elemen di dalam *array*, serta pencocokan *string*.

Dalam persoalan pencocokan *string*, berikut ini adalah langkah-langkahnya:

1. Mula-mula, *pattern* disejajarkan (*alignment*) pada awal teks.
2. Lakukan penelusuran dari kiri ke kanan pada *pattern* serta bandingkan setiap karakter pada *pattern* dengan karakter yang bersesuaian pada teks sampai:
   - Semua karakter yang dibandingkan cocok atau sama (pencarian berhasil).
   - Ditemukan karakter yang tidak cocok (pencarian belum berhasil.
3. Jika *pattern* belum ditemukan kecocokannya serta teks belum habis, geser *pattern* satu karakter ke kanan dan ulangi kembali langkah ke-2.

Contoh:

```
Teks: NOBODY NOTICED HIM
Pattern: NOT

  NOBODY NOTICED HIM
1 NOT
2  NOT
3   NOT
4    NOT
5     NOT
6      NOT
7       NOT
8        NOT
```

Gambar 2.1. Contoh Pencocokan *String* dengan *Brute Force*

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

Berikut ini adalah *pseudocode* dalam menyelesaikan persoalan pencocokan string dengan *brute force*:

```
function PencocokanString(input P : string, T : string, m, n : integer, output idx : integer) → integer
{ Luaran: lokasi awal kecocokan (idx) }
Deklarasi
  i : integer
  ketemu : boolean
Algoritma:
  i ← 0
  ketemu ← false
  while (i ≤ n − m) and (not ketemu) do
    j ← 1
    while (j ≤ m) and (P_j = T_{i+j}) do
      j ← j + 1
    endwhile
    { j > m  or  P_j ≠ T_{i+j} }

    if j = m then { kecocokan string ditemukan }
      ketemu ← true
    else
      i ← i + 1 { geser pattern satu karakter ke kanan teks }
    endif
  endwhile
  { i > n − m or ketemu }

  if ketemu then return i + 1 else return −1 endif
```

Gambar 2.2. *Pseudocode* Pencocokan *String* dengan *Brute Force*

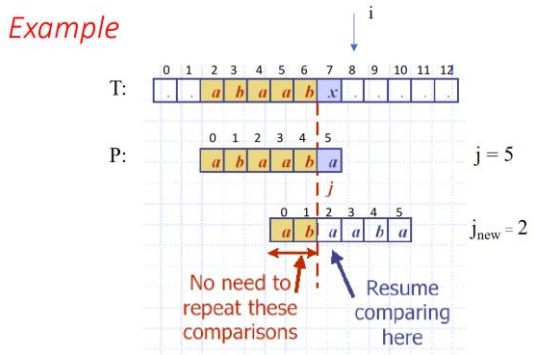
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

Pada kasus terburuknya, algoritma ini memiliki kompleksitas O(mn), sedangkan pada kasus terbaiknya adalah O(n). Namun, secara rata-rata, kompleksitas algoritma ini adalah O(m+n).

C. Algoritma KMP (Knuth-Morris-Pratt)

Algoritma KMP (Knuth-Morris-Pratt) adalah algoritma pencocokan *string* yang dikembangkan oleh Donald Ervin Knuth, Vaughan Pratt, serta James Hiram Morris. Sama seperti algoritma *brute force*, algoritma KMP juga mencari *pattern* di dalam suatu teks dengan urutan dari kiri ke kanan. Akan tetapi, algoritma ini dapat menggeser *pattern* dengan lebih cerdas dibandingkan dengan algoritma *brute force* yang selalu menggeser *pattern* satu karakter ke kanan.

Kunci dari efisiensi algoritma ini adalah kemampuannya untuk menghindari pencocokan ulang yang tidak diperlukan dari suatu *pattern* pada suatu teks. Jika terjadi ketidakcocokan antara teks T dan *pattern* P pada P[j], yang berarti T[i] ≠ P[j], besarnya pergeseran maksimum yang dapat dilakukan untuk menghindari pencocokan ulang yang tidak diperlukan dapat ditentukan dari panjang prefiks P[0..j-1] yang juga merupakan sufiks P[1..j-1].

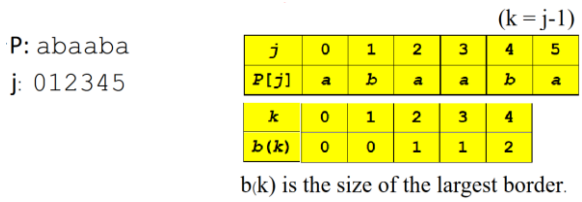


Gambar 2.3. Ilustrasi Pencocokan yang Tidak Diperlukan

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Algoritma ini membutuhkan prapemrosesan terhadap *pattern* untuk mencari *border function* yang akan dipakai dalam menentukan seberapa jauh pergeseran *pattern* yang perlu dilakukan ketika terjadi ketidakcocokan. *Border function* b(k) didefinisikan sebagai ukuran dari prefiks terbesar P[0..k] yang juga merupakan sufiks P[1..k].

Berikut ini merupakan contoh tabel *border function* untuk *pattern* "abaaba":



Gambar 2.4. Contoh Tabel Border Function

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Berikut ini adalah *pseudocode* untuk menghasilkan tabel *border function*:

```
function computeBorder(pattern: String) → array of integer
\
KAMUS
b: array of integer
m, j, i: integer

ALGORITMA
b ← array of integer[0..length(pattern) - 1]
b[0] ← 0
m ← length(pattern)
j ← 0
i ← 1
while (i < m) do
    if pattern[j] = pattern[i] then
        b[i] ← j + 1
        i ← i + 1
        j ← j + 1
    else if j > 0 then
        j ← b[j - 1]
    else
        b[i] ← 0
        i ← i + 1
→ b
```

Gambar 2.5. *Pseudocode Border Function*

Sumber: dokumentasi penulis

Setelah tabel *border function* terbentuk, pencocokan *string* dapat dilakukan. Berikut ini adalah *pseudocode* algoritma KMP:

```
function kmpMatch(text: String, pattern: String) → integer

KAMUS
b: array of integer
n, m, i, j: integer

ALGORITMA
n ← length(text)
m ← length(pattern)
b ← computeBorder(pattern)
i ← 0
j ← 0
while i < n do
    if pattern[j] = text[i] then
        if j = m - 1 then
            → i - m + 1
        i ← i + 1
        j ← j + 1
    else if j > 0 then
        j ← b[j - 1]
    else
        i ← i + 1
→ -1
```

Gambar 2.6. *Pseudocode* Algoritma KMP

Sumber: dokumentasi penulis

Kompleksitas waktu untuk mencari *border function* adalah O(m), sedangkan kompleksitas waktu untuk pencocokan *string* adalah O(n). Secara keseluruhan, algoritma KMP memiliki kompleksitas waktu O(m+n) pada kasus terburuknya. Jika dibandingkan dengan algoritma *brute force*, algoritma ini memiliki kompleksitas waktu yang lebih baik.

D. Algoritma BM (Boyer-Moore)

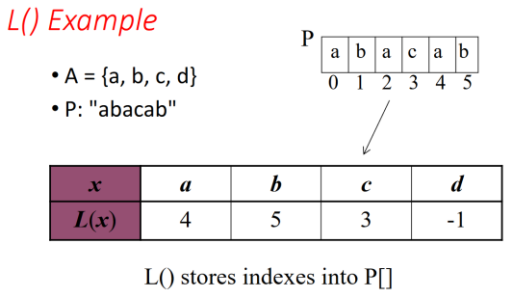
Algoritma BM (Boyer-Moore) adalah algoritma pencocokan *string* yang dikembangkan oleh Robert Stephen Boyer dan J Strother Moore. Berbeda dengan algoritma *brute force* dan KMP, algoritma ini mencari *pattern* di dalam suatu teks dengan urutan dari kanan ke kiri (*looking-glass*).

Efisiensi algoritma ini terletak pada kemampuannya untuk menggeser pattern dengan mempertimbangkan kemunculan terakhir karakter pada *pattern* yang tidak cocok dengan karakter pada teks.

Algoritma ini membutuhkan prapemrosesan terhadap *pattern* untuk mencari *last occurrence function* yang akan dipakai dalam menentukan seberapa jauh pergeseran *pattern* yang perlu dilakukan ketika terjadi ketidakcocokan. *Last occurrence function* l(x) didefinikan sebagai:

- Indeks terbesar i sehingga P[i] = x, atau
- -1 jika tidak ada indeks yang memenuhi kondisi sebelumnya

Berikut ini merupakan contoh tabel *last occurrence function* untuk *pattern* "abacab":



Gambar 2.7. Contoh Tabel *Last Occurence Function*

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Berikut ini adalah *pseudocode* untuk menghasilkan tabel *last occurrence function*:

```
function buildLast(pattern: String) → array of integer

KAMUS
last: array of integer
i: integer

ALGORITMA
last ← array of integer[0..127]
i traversal [0..127]
    last[i] ← -1
i traversal [0..length(pattern) - 1]
    last[pattern[i]] ← i
→ last
```

Gambar 2.8. Pseudocode Last Occurrence Function

Sumber: dokumentasi penulis

Setelah tabel *last occurrence function* terbentuk, pencocokan string dapat dilakukan. Berikut ini adalah *pseudocode* algoritma BM:

```
function bmMatch(text: String, pattern: String) → integer

KAMUS
last: array of integer
n, m, i, j, lo: integer

ALGORITMA
last ← buildLast(pattern)
n ← length(text)
m ← length(pattern)
i ← m - 1
if i > n - 1 then
    → -1
j ← m - 1
do
    if pattern[j] = text[i] then
        if j = 0 then
            → i
        else
            i ← i - 1
            j ← j - 1
    else
        lo ← last[text[i]]
        i ← i + m - min(j, 1 + lo)
        j ← m - 1
while i <= n - 1
→ -1
```

Gambar 2.9. Pseudocode Algoritma BM

Sumber: dokumentasi penulis

Algoritma BM memiliki kompleksitas waktu O(nm + A) pada kasus terburuk, dengan A adalah banyaknya alfabet yang digunakan. Algoritma ini cukup cepat ketika alfabet yang digunakan cukup banyak, misalnya karakter dalam bahasa Inggris, tetapi menjadi lambat ketika alfabet yang digunakan hanya sedikit, misalnya karakter biner. Dibandingkan dengan algoritma *brute force*, algoritma ini jauh lebih cepat ketika digunakan dalam pencarian teks berbahasa Inggris.

III. Implementasi dan Pengujian

Pada makalah ini, implementasi fitur pencarian komponen PC ditulis dalam bahasa pemrograman Python. Secara umum, algoritma *string matching brute force*, KMP, serta BM yang dibuat dalam bahasa pemrograman ini sama dengan *pseudocode* yang telah dituliskan pada bab sebelumnya.

Untuk menguji ketiga algoritma tersebut, penulis menggunakan komponen-komponen PC yang terdapat di dalam *game* PC Building Simulator. Banyaknya komponen PC yang digunakan dalam pengujian ini adalah 2335 komponen. Pengujian dilakukan sebanyak tiga kali untuk masing-masing algoritma. Setiap pengujian dilakukan dengan *pattern* yang berbeda-beda. Keluaran dari pengujian tersebut adalah waktu eksekusi, banyaknya komponen yang ditemukan, serta nama-nama komponen yang ditemukan.

Berikut ini adalah hasil pengujian untuk ketiga algoritma tersebut:

1. *Test Case* 1

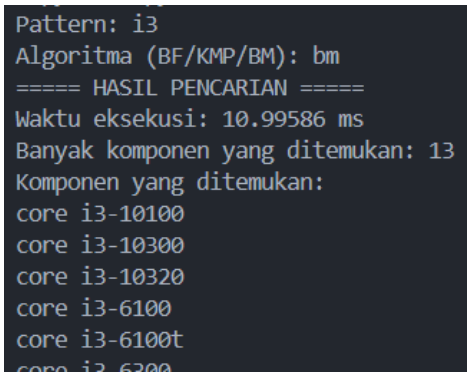
Brute Force

```
Pattern: i3
Algoritma (BF/KMP/BM): bf
===== HASIL PENCARIAN =====
Waktu eksekusi: 8.99243 ms
Banyak komponen yang ditemukan: 13
Komponen yang ditemukan:
core i3-10100
core i3-10300
core i3-10320
core i3-6100
core i3-6100t
```

KMP (Knuth-Morris-Pratt)

```
Pattern: i3
Algoritma (BF/KMP/BM): kmp
===== HASIL PENCARIAN =====
Waktu eksekusi: 10.00333 ms
Banyak komponen yang ditemukan: 13
Komponen yang ditemukan:
core i3-10100
core i3-10300
core i3-10320
core i3-6100
core i3-6100t
```

BM (Boyer-Moore)



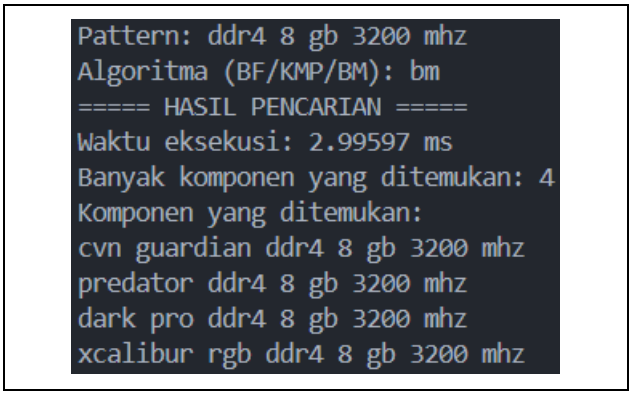
Tabel 3.1. Hasil Pengujian untuk *Test Case* 1

Sumber: dokumentasi penulis

Pada pengujian pertama, semua algoritma tersebut memperoleh hasil pencarian yang sama. Perbedaannya terletak pada waktu eksekusi algoritmanya. Urutan algoritma dari yang paling efisien: *brute force*, KMP, BM.

2. Test Case 2

Brute Force
Pattern: ddr4 8 gb 3200 mhz Algoritma (BF/KMP/BM): bf ===== HASIL PENCARIAN ===== Waktu eksekusi: 3.99232 ms Banyak komponen yang ditemukan: 4 Komponen yang ditemukan: cvn guardian ddr4 8 gb 3200 mhz predator ddr4 8 gb 3200 mhz dark pro ddr4 8 gb 3200 mhz xcalibur rgb ddr4 8 gb 3200 mhz
KMP (Knuth-Morris-Pratt)
Pattern: ddr4 8 gb 3200 mhz Algoritma (BF/KMP/BM): kmp ===== HASIL PENCARIAN ===== Waktu eksekusi: 8.98409 ms Banyak komponen yang ditemukan: 4 Komponen yang ditemukan: cvn guardian ddr4 8 gb 3200 mhz predator ddr4 8 gb 3200 mhz dark pro ddr4 8 gb 3200 mhz xcalibur rgb ddr4 8 gb 3200 mhz
BM (Boyer-Moore)



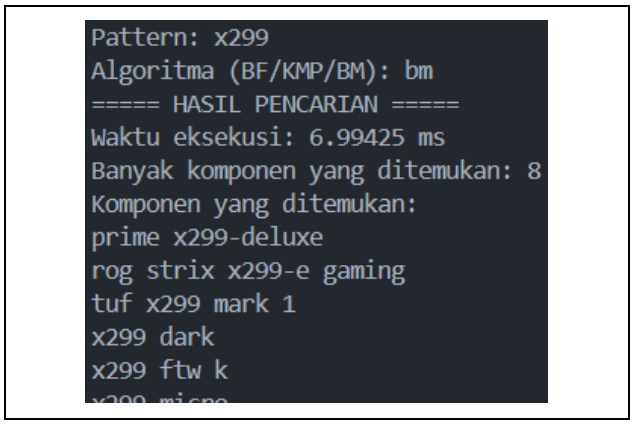
Tabel 3.2. Hasil Pengujian untuk *Test Case* 2

Sumber: dokumentasi penulis

Pada pengujian ke-2, semua algoritma tersebut memperoleh hasil pencarian yang sama. Perbedaannya terletak pada waktu eksekusi algoritmanya. Urutan algoritma dari yang paling efisien: BM, *brute force*, KMP.

3. Test Case 3

Brute Force
Pattern: x299 Algoritma (BF/KMP/BM): bf ===== HASIL PENCARIAN ===== Waktu eksekusi: 8.99029 ms Banyak komponen yang ditemukan: 8 Komponen yang ditemukan: prime x299-deluxe rog strix x299-e gaming tuf x299 mark 1 x299 dark x299 ftw k
KMP (Knuth-Morris-Pratt)
Pattern: x299 Algoritma (BF/KMP/BM): kmp ===== HASIL PENCARIAN ===== Waktu eksekusi: 9.97400 ms Banyak komponen yang ditemukan: 8 Komponen yang ditemukan: prime x299-deluxe rog strix x299-e gaming tuf x299 mark 1 x299 dark x299 ftw k
BM (Boyer-Moore)



Tabel 3.3. Hasil Pengujian untuk *Test Case* 3

Sumber: dokumentasi penulis

Pada pengujian ke-3, semua algoritma tersebut memperoleh hasil pencarian yang sama. Perbedaannya terletak pada waktu eksekusi algoritmanya. Urutan algoritma dari yang paling efisien: BM, *brute force*, KMP.

IV. Analisis dan Pembahasan

Berdasarkan hasil pengujian yang telah dipaparkan pada bab sebelumnya, ketiga algoritma tersebut sama-sama efektif digunakan dalam pencarian komponen PC. Hal ini terbukti dengan hasil pencarian yang selalu sama. Namun, perbedaan yang terlihat dari ketiga algoritma tersebut adalah waktu eksekusinya. Secara umum, algoritma yang memiliki efisiensi yang paling baik adalah algoritma BM.

Jika dilihat berdasarkan kompleksitas algoritmanya, algoritma *brute force* memiliki efisiensi yang kurang baik. Akan tetapi, pada *test case* 1, algoritma ini memiliki waktu eksekusi yang paling cepat. Hal ini disebabkan oleh pendeknya *pattern* yang digunakan pada test case tersebut. Keunggulan yang dimiliki oleh algoritma KMP dan BM tidak berlaku pada *test case* tersebut. *Pattern* tidak dapat bergeser jauh karena panjang *pattern* yang hanya 2 karakter.

Algoritma KMP tidak menunjukkan kinerja yang baik pada semua kasus uji. Menurut kompleksitas waktunya, algoritma ini berpotensi menjadi salah satu algoritma yang efisien. Kurangnya efisiensi algoritma ini dipengaruhi oleh *pattern* yang diberikan. *Pattern* yang diberikan tidak mengandung pasangan prefiks dan sufiks yang cukup panjang karena komponen PC cenderung tidak memiliki kata-kata yang memenuhi kondisi tersebut. Akibatnya, algoritma ini tidak mampu melakukan pergeseran *pattern* secara efisien.

Ketika *pattern* yang diberikan cukup panjang, algoritma BM menunjukkan kinerja yang sangat baik dalam melakukan pencarian komponen PC. Hal ini disebabkan karena algoritma BM mampu menggeser *pattern* dengan cukup jauh ketika *pattern* yang diberikan cukup panjang. Selain itu, alfabet yang digunakan pada pencarian ini cukup beragam, seperti huruf latin, angka, maupun beberapa karakter spesial, sehingga efisiensi algoritma ini meningkat.

V. Penutup

A. Kesimpulan

Algoritma *brute force*, KMP, maupun BM mampu melakukan pencarian komponen PC secara efektif karena dapat menampikan semua komponen PC yang cocok dengan kata kunci yang diberikan. Perbedaannya terletak pada efisiensi algoritma tersebut. Secara umum, algoritma yang paling efisien adalah algoritma BM karena mampu melakukan pergeseran *pattern* secara lebih efisien dibandingkan algoritma lainnya. Namun, untuk kasus *pattern* yang pendek, algoritma *brute force* mampu memberikan kinerja yang relatif baik karena algoritma KMP maupun BM tidak dapat melakukan pergeseran *pattern* secara efisien.

B. Saran

Makalah ini tidak sempurna serta terdapat kekurangan-kekurangan yang dapat diperbaiki. Berikut ini adalah beberapa saran dari penulis:

1. Melakukan pengujian algoritma dengan *pattern* yang lebih beragam serta daftar teks (komponen PC) yang lebih banyak sehingga data uji yang diperoleh lebih *reliable*.
2. Mencari algoritma pencocokan *string* lain yang berpotensi memiliki efisiensi lebih baik dibandingkan algoritma *brute force*, KMP, maupun BM.

Ucapan Terima Kasih

Penulis memanjatkan puji dan syukur kepada Tuhan Yang Maha Esa atas penyertaan-Nya sehingga penulis dapat menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih kepada pihak-pihak yang mendukung penulis dalam penulisan makalah ini, di antaranya:

1. Seluruh tim pengajar IF2211 Strategi Algoritma Institut Teknologi Bandung, terutama Ibu Dr. Nur Ulfa Maulidevi sebagai dosen pengampu kelas K02.
2. Teman-teman yang telah mendukung saya dalam pembuatan makalah ini.
3. Orang tua yang selalu memberikan dukungan kepada saya sampai makalah ini selesai dibuat.

Penulis berharap makalah ini bermanfaat bagi siapapun yang membaca makalah ini.

Referensi

[1] Boyer, Robert S., dan J Strother Moore. 1997. *A Fast String Searching Algorithm*. https://www.cs.utexas.edu/~moore/publications/fstrpos.pdf. Diakses pada 12 Juni 2024.

[2] Munir, Rinaldi. 2022. *Bahan Kuliah IF2211 Strategi Algoritma: Algoritma Brute Force (Bagian 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf. Diakses pada 12 Juni 2024.

[3] Munir, Rinaldi. 2021. *Bahan Kuliah IF2211 Strategi Algoritma: Pencocokan String (String/Pattern Matching)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-

2022/Algoritma-Brute-Force-(2022)-Bag1.pdf. Diakses pada 12 Juni 2024.

[4] Steam. 2019. *PC Building Simulator*. https://store.steampowered.com/app/621060/PC_Building_Simulator/. Diakses pada 12 Juni 2024.

[5] UC Irvine. 1996. *Knuth-Morris-Pratt string matching*. https://ics.uci.edu/~eppstein/161/960227.html. Diakses pada 12 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

Bastian H. Suryapratama (13522034)