Perbandingan Kecepatan Algoritma *String Matching* dalam *Multiple Word Finder* Berbasis Web

Jason Kanggara - 13520080
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: jasonkanggara19022002@gmail.com

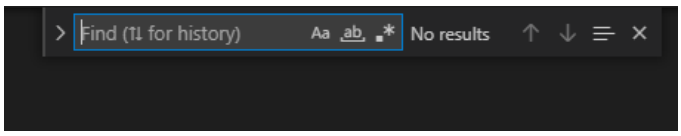
***Abstrak*—Di zaman modern ini, banyak sekali kemudahan yang sudah diberikan dari berbagai macam aplikasi yang terdapat pada gadget apapun. Salah satu fitur yang memberikan pengguna kemudahan dalam mencari suatu kata / informasi adalah fitur *Word Finder*. Fitur *Word Finder* sudah banyak diterapkan pada berbagai macam sektor aplikasi, seperti aplikasi *Teks Editor*, *Browser*, dokumen online, dan masih banyak lagi. Penerapan fitur pencarian kata, atau *Word Finder*, adalah dengan memanfaatkan algoritma *String Matching*. *String Matching*, atau pencocokan *string*, adalah suatu algoritma yang digunakan untuk mencari kata/*pattern* yang terdapat dalam suatu teks. Ada berbagai macam algoritma *string matching* yang ada, diantaranya seperti algoritma *brute force*, algoritma *Knuth-Morris-Pratt*, algoritma *Boyer-Moore*, *Regular Expression*, algoritma *Rabin-Karp*, dan masih banyak lagi.**

Kata Kunci*—*Word Finder, Search, String Matching, Algoritma

I. Pendahuluan

Di era digital ini, teknologi berkembang dengan sangat cepat dan masif. Banyak sekali fitur – fitur yang dapat membantu kita dalam berkegiatan sehari – hari. Salah satu fitur yang sering kita gunakan, khususnya dalam mencari suatu kata yang kita inginkan pada suatu teks/dokumen di internet maupun teks editor adalah fitur *word finder*.

Tanpa fitur *word finder* ini, kita tentu akan kesulitan dalam mencari kata yang kita inginkan dalam suatu teks, apalagi jika teks itu jumlahnya sangat banyak, dapat menghabiskan waktu kita untuk mencari suatu kata. Oleh karena itu, dibuatlah fitur *word finder* yang sudah banyak diaplikasikan pada berbagai macam media seperti *teks editor*, internet, *E-book*, file pdf, dan masih banyak lagi.



Gambar 1.1 Contoh Fitur *Word Finder* Pada Aplikasi *Visual Studio Code*

Pada makalah ini, penulis akan membahas apa itu *word finder* secara singkat, konsep dasar dari algoritma *string matching*, dan bagaimana fitur *word finder* berhubungan dengan algoritma *string matching*. Fitur *word finder* akan diimplementasikan dengan konsep yang berbeda dengan fitur *word finder* secara umum, yaitu *multiple word finder*. Secara umum, fitur *word finder* hanya dapat mencari satu kata, atau sebuah frasa yang saling berhubungan. Namun pada makalah ini, penulis akan mengimplementasikan fitur *word finder* yang dapat mencari beberapa kata berbeda.

II. Landasan Teori

A. *Word Finder*

Word Finder adalah fitur yang sudah sering digunakan pada beberapa aplikasi/media seperti *teks editor*, browser, file pdf, dan lain lain. Fitur *word finder* adalah sebuah fitur yang membantu pengguna dalam mencari kata yang ingin dicari dari serangkaian teks. Secara umum, misalnya di *teks editor*, untuk menjalankan fitur *word finder*, dapat ketikkan *shortcut* pada keyboard CTRL + F. Setelah muncul suatu kotak masukkan, pengguna tinggal memasukkan kata yang dicari, lalu fitur *word finder* dalam aplikasi akan mencari kata tersebut yang terdapat dalam teks. Jika ada akan ditunjukkan posisinya, jika tidak ada maka fitur akan mengirimkan pesan bahwa kata tidak ditemukan.

B. *String Matching*

String Matching atau *Pattern Matching* adalah suatu algoritma yang digunakan untuk mencari adanya kecocokan suatu *string* yang menjadi *pattern* pada *string* acuan. Jika ditemukan *string* yang dicari, algoritma akan mengembalikan posisi *string pattern* pada *string* acuan. Secara definisi, pada *string matching*, diberikan:

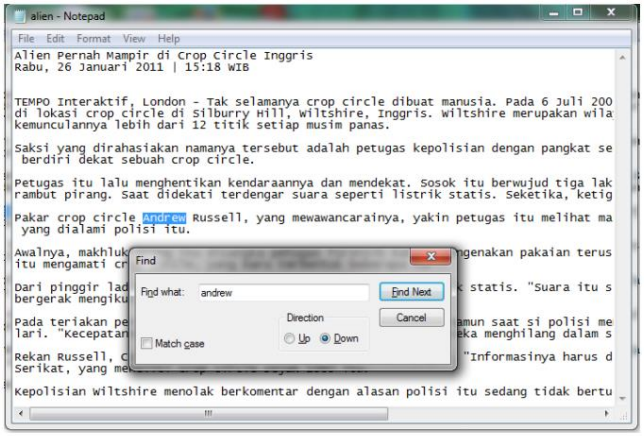
1. T: teks, yaitu *string* yang panjangnya n karakter
2. P: *pattern*, yaitu *string* dengan panjang m karakter, dengan asumsi panjang *pattern* lebih kecil dari panjang teks, yang akan dicari pada teks.

```
T: the rain in spain stays mainly on the plain
P: main
```

Gambar 2.1 Contoh *String/Pattern Matching*
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

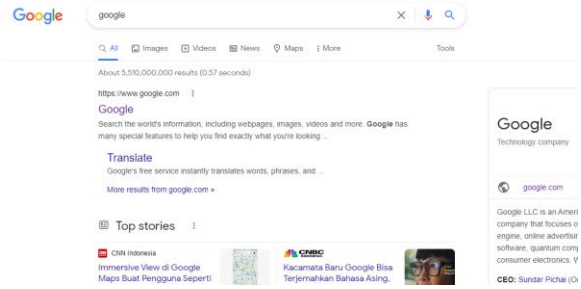
String Matching dapat diaplikasi dalam berbagai macam sektor, seperti:

1. Pencarian kata di dalam *text editor*



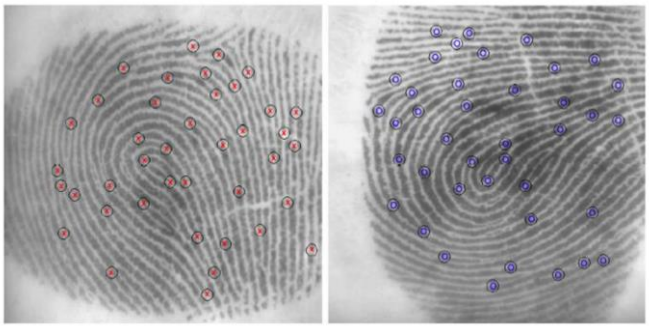
Gambar 2.2 Contoh Pencarian Kata di *Teks Editor*
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

2. *Web Search Engine* (Misal: Google, Bing, dll)



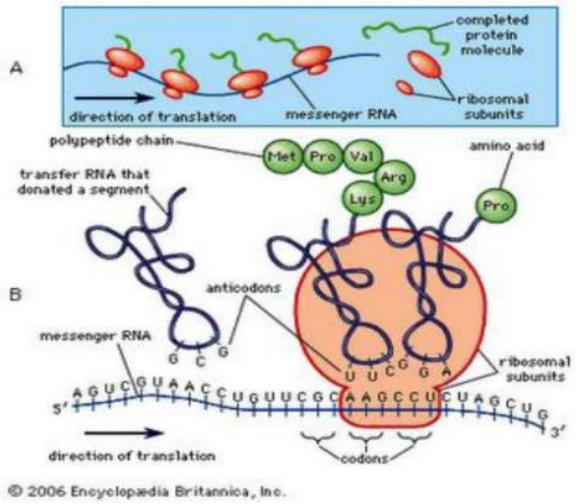
Gambar 2.3 Contoh *Web Search Engine Google*
Sumber: https://www.google.com/

3. Analisis Citra



Gambar 2.4 Contoh Analisis Citra
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

4. *Bioinformatics*
   - Pencocokan rantai asam amino pada rantai DNA

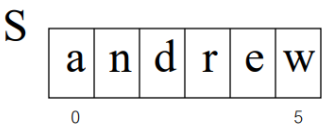


Gambar 2.5 Contoh Pencocokan Rantai Asam Amino dengan Rantai DNA
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Konsep dasar dari Algoritma *String Matching* adalah pengetahuan akan apa itu *String*. Secara sederhana *string* adalah sebuah teks/kata dari kumpulan *character* dengan panjang n. Sebagai contoh, "*Hello World"* adalah sebuah *string* dengan panjang 11, dimana spasi juga merupakan bagian dari *string*.

Pada tipe data *string*, terdapat 2 bagian yang terkait dengan *string*, yaitu *prefix* dan *suffix*. Misalnya terdapat sebuah *string* dengan panjang m. *Prefix* dari *string* tersebut adalah *Substring* dari indeks ke – 0 sampai k. Sedangkan *suffix* dari *string* adalah *Substring* dari indeks ke - k sampai m – 1. "k" dalam contoh sebelumnya adalah indeks sembarang di antara 0 dan m – 1.

Contoh *prefix* dan *suffix* dari sebuah *string* dapat dilihat pada gambar berikut.



Gambar 2.6 Contoh *String* dengan panjang 6
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

- All possible prefixes of S:
  - "a", "an", "and", "andr", "andre", "andrew"
- All possible suffixes of S:
  - "w", "ew", "rew", "drew", "ndrew" , "andrew"

Gambar 2.7 Contoh *Prefix* dan *Suffix* pada *String* di Gambar 7.
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Algoritma *String Matching* sangat variatif, dimana setiap algoritma memiliki penerapan yang berbeda – beda dan juga waktu dan efisiensi yang berbeda – beda. Pada topik ini, algoritma yang akan dibahas adalah Algoritma *Brute Force*, Algoritma *Knuth-Morris-Pratt*, dan Algoritma *Boyer-Moore*. Pembahasan lebih lanjut dari masing – masing algoritma akan dibahas pada subbab selanjutnya.

C. Algoritma Brute Force

Algoritma *Brute Force* merupakan salah satu dari ketiga algoritma *string matching* yang disebutkan pada subbab sebelumnya. Penerapan algoritma *brute force* pada *string matching* dilakukan dengan cara membandingkan *string* pada *pattern* dengan *string* acuan secara kata per kata. Jadi setiap kata pada *pattern* akan diperiksa dengan kata pada *string*, jika berbeda, maka indeks pemeriksaan pada *string* acuan akan dilanjutkan dan dibandingkan lagi dengan *string pattern* sampai ditemukan kata yang sama pada *string* acuan dengan *string pattern*.

Sebagai contoh untuk memperjelas proses *string matching* dengan Algoritma *Brute Force*, misalnya terdapat teks T dengan panjang n (T[0..n-1]), dan sebuah *pattern* P dengan panjang m (P[0..m-1]), dengan syarat m <= n, maka tahapan algoritma *brute force* adalah sebagai berikut.

1. Dilakukan pencocokan pertama pada T[0] dengan P[0].
2. Jika terdapat kecocokan, maka indeks pada T dan P ditambah 1, lalu diulangi proses pencocokan karakter seperti yang dilakukan pada langkah 1. Jika ada ketidakcocokan dalam pemeriksaan, maka indeks kembali seperti indeks yang terdapat pada langkah 1, lalu indeks pada teks T ditambah 1, dan dicocokan kembali dengan *pattern* P pada indeks 0.
3. Pencocokan dilakukan sampai ditemukan *string* pada teks T yang cocok dengan *pattern* P, atau mencapai indeks terakhir pada T, yang berarti tidak adanya kecockan antara P di T.

Contoh gambaran untuk memperjelas proses di atas adalah sebagai berikut

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
```

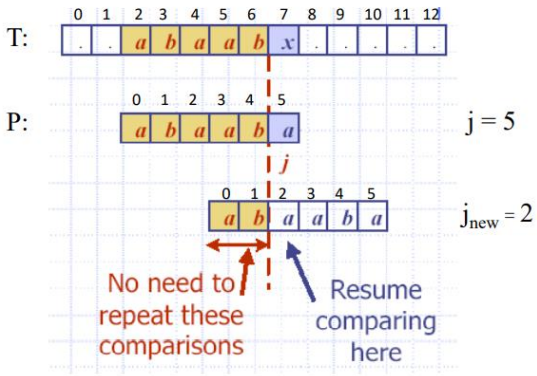
Gambar 2.8 *String Matching* dengan Algoritma *Brute Force*
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Untuk memperjelas penggunaan algoritma *brute force*, *pseudocode* algoritma *brute force* adalah sebagai berikut.

```
function bruteForce(text, pattern: string) -> int
  int n = length(text)
  int m = length(pattern)
  int j

  iterate from i = 0 until (n - m) {
    j = 0
    while (j < m and (char in text at (i+j) == char in pattern at
j)) {
      j++
    }
    if (j == m) {
      return i
    }
  }
  # no match
  return -1
```

D. Algoritma Knuth-Morris-Pratt

Algoritma *Knuth-Morris-Pratt* adalah algoritma *String Matching* yang melakukan pencocokan *pattern* pada teks dari arah kiri ke kanan (seperti algoritma *brute force*). Perbedaan dengan algoritma *brute force* adalah, pada algoritma *brute force*, jika terdapat ketidakcocokan pada pengecekan karakter, indeks pada Teks digeser sebanyak satu kali, atau indeks pada Teks ditambah 1. Pada algoritma KMP, pergeseran dilakukan secara lebih *pintar*, yaitu dengan ke *prefix* terbesar pada *pattern* yang adalah *suffix* dari *pattern*. Contoh gambar pergeseran kata pada algoritma KMP dapat dilihat pada gambar berikut.



Gambar 2.9 Pergeseran kata pada Algoritma *Knuth-Morris-Pratt*
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Algoritma KMP memanfaatkan suatu fungsi pinggiran (*border function*) untuk mendefinisikan ukuran prefix terbesar dari *pattern* yang juga merupakan suffix dari *pattern*. Dengan menggunakan *border function* ini, proses perbandingan karakter pada algoritma KMP menjadi lebih sedikit, sehingga algoritma KMP lebih efisien dibandingkan algoritma *brute force*. *Pseudocode* dari algoritma KMP adalah sebagai berikut.

```
function kmp(text, pattern : string) -> int
  int n = length(text)
  int m = length(pattern)

  int fail[] = computeFail(pattern)

  int i = 0
  int j = 0

  while (i < n) {
    if (char in pattern at j == char in text at i) {
      if (j == m - 1) {
        return i - m + 1
      }
      i++
      j++
    }
    else if (j > 0) {
      j = fail[j - 1]
    }
    else {
      i++
    }
  }
  # no match
  return -1
```

Algoritma KMP ada keuntungan dan kerugian yang juga harus diperhatikan. Keuntungan dari penggunaan algoritma KMP adalah algoritma ini tidak perlu bergerak mundur pada teks. Jadi ada konsep loncatan selama proses pengencekan, tidak seperti algoritma *Brute Force* yang selalu mengulang pemeriksaan jika terjadi *mismatch*. Kerugian dari penggunaan algoritma KMP adalah algoritma ini tidak cocok digunakan ketika ukuran alfabet bertambah karena hal ini dapat meningkatkan terjadinya *mismatch*.

Kompleksitas algoritma KMP terbagi menjadi dua proses, yaitu proses menghitung fungsi pinggiran dan proses pencarian *string*. Untuk proses menghitung fungis pinggiran memiliki kompleksitas waktu O(m) dengan m adalah panjang *string pattern*, dan proses pencarian *string* memiliki kompleksitas waktu O(n) dengan n adalah panjang dari teks. Maka, jika digabung, kompleksitas waktu algoritma KMP adalah O(m + n) yang jauh lebih cepat disbanding kompleksitas waktu algoritma *brute force*.

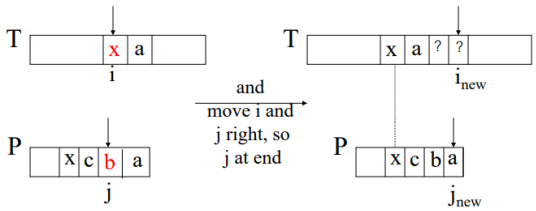
E. Algoritma Boyer-Moore

Algoritma *Boyer-Moore* adalah algoritma *String Matching* yang melakukan pencocokan *pattern* pada teks dengan dua teknik yang berbeda, yaitu teknik *looking-glass* dan teknik *character-jump*.

1. *Looking Glass Technique*
   Teknik *looking glass* adalah teknik pada algoritma *Boyer-Moore* yang mencari *pattern* pada teks dengan bergerak secara mundur pada *pattern*, mulai dari posisi/indeks terakhir.
2. *Character Jump Techinique*
   Teknik *character jump* adalah teknik pada algoritma *Boyer-Moore* yang akan dijalankan saat terjadi ketidakcocokan kata pada teks dengan x, dan saat karakter pada *pattern* tidak sama dengan karakter pada teks pada suatu indeks tertentu.

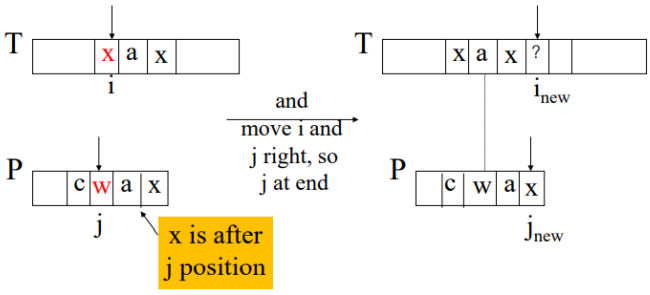
Algoritma *Boyer-Moore* dapat di dibedakan berdasarkan tiga kasus, yaitu

1. Kasus 1
   Jika P memiliki ‘x’ pada suatu posisi, maka dapat dilakukan pergeseran P ke kanan untuk mensejajarkan kemunculan terakhir dari ‘x’ di P dengan T[i], dengan P adalah *pattern* dan T adalah teks .



Gambar 2.10 Algoritma BM Kasus 1
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

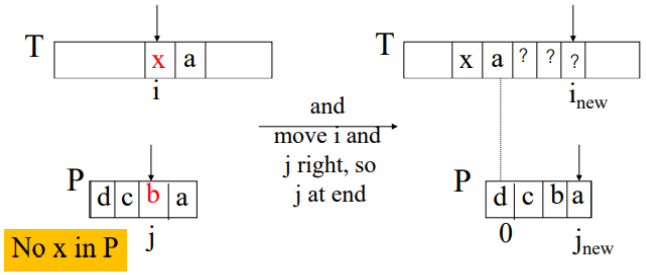
2. Kasus 2
   Jika P memiliki ‘x’ pada suatu posisi tertentu tetapi pergeseran ke kanan sampai kemunculan terakhir tidak memungkinkan, maka geser P ke kanan sebanyak satu karakter ke T[i+1], dengan P adalah *pattern* dan T adalah teks.



Gambar 2.11 Algoritma BM Kasus 2
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

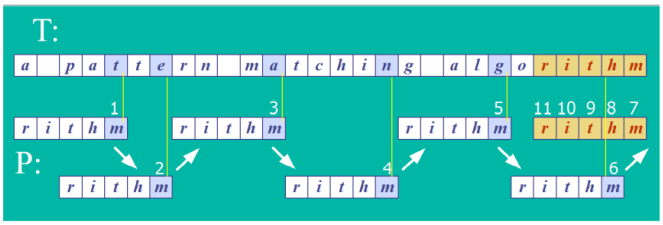
3. Kasus 3

Jika kasus 1 dan kasus 2 tidak memnuhi, maka geser P untuk mensejajarkan P[0] dengan T[i+1], dengan P adalah *pattern* dan T adalah teks.



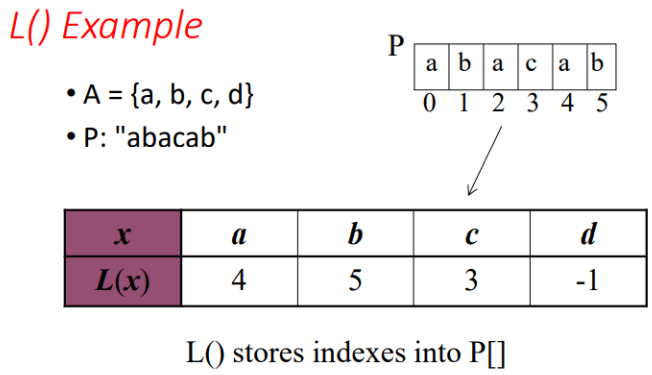
Gambar 2.12 Algoritma BM Kasus 3
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Contoh penerapan algoritma *Boyer-Moore* dapat dilihat pada gambar berikut.



Gambar 2.13 Contoh Pencocokan *String* dengan Algoritma *Boyer-Moore*
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Algoritma *Boyer-Moore*, serupa dengan Algoritma *Knuth-Morris-Pratt*, memanfaatkan suatu fungsi pembantu yang dinamakan *Last Occurrence Function*. *Last Occurrence Function* adalah suatu fungsi yang digunakan dalam algoritma *Boyer-Moore* untuk melakukan *preprocess pattern* P dan alfabet A dan menyimpan indeks kemunculan terakhir alfabet yang terdapat pada *pattern* P. *Last Occurencce Function* dikalkulasikan ketika *pattern* P sedang dalam proses pembacaan. Contoh gambaran dari *Last Occurrence Function* adalah sebagai berikut.



Gambar 2.14 *Last Occurrence Function*

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

Untuk memperjelas penggunaan dari algoritma *Boyer-Moore*, berikut adalah *pseudocode* dari algoritma *Boyer-Moore*.

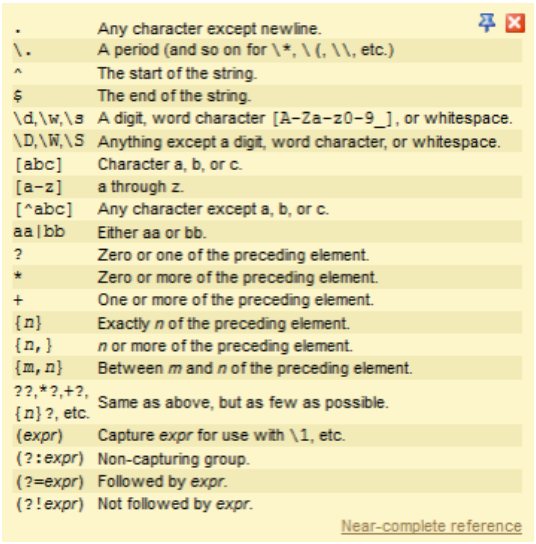
```
function boyerMoore(text, pattern : string) -> int
 int last[] = buildLast(pattern) # Fungsi Last Occurrence
 int n = length(text)
 int m = length(pattern)
 int i = m - 1

 if (i > m - 1) {
  # tidak ada kecocokan jika pattern lebih besar dari text
  return -1
 }

 int j = m - 1
 do {
  if (char in pattern at j == char in pattern at i) {
   if (j == 0) {
    return i # cocok
   } else {
    i--
    j--
   }
  } else {
   int lo = last[char in text at i]
   i = i + m - min(j, 1 + lo)
   j = m - 1
  }
 } while (i <= n - 1)

 return -1
```

Kompleksitas waktu algoritma *Boyer-Moore* memiliki kasus terburuk dengan kompleksitas O(nm + A) dimana O(nm) adalah kompleksitas waktu pencocokan *string* dengan algoritma *Boyer-Moore*, dan O(A) adalah pencarian *last occurrence* masing – masing alfabet pada *pattern* dengan memanfaatkan *Last Occurrence Function*.

Akan tetapi, algoritma *Boyer-Moore* akan cepat ketika alfabet (A) merupakan huruf besar, dan lambat ketika alfabet adalah huruf kecil. Algoritma *Boyer-Moore* jauh lebih cepat dibandingkan algoritma *Brute Force* untuk pencarian pada suatu teks, seperti teks berbahasa Inggris.

F. Regular Expression

Reguler Expression, atau biasa disingkat regex, adalah sekumpulan karakter yang dapat dikonversi menjadi suatu pola berbentuk teks sehari – hari. *Regular Expression* digunakan dalam algoritma *pencocokan string*, dan biasa digunakan untuk proses pencarian suatu *string*/kata. Notasi umum dari regex dapat dilihat pada gambar berikut.



.	Any character except newline.
\.	A period (and so on for *, \(, \\, etc.)
^	The start of the string.
$	The end of the string.
\d,\w,\s	A digit, word character [A-Za-z0-9_], or whitespace.
\D,\W,\S	Anything except a digit, word character, or whitespace.
[abc]	Character a, b, or c.
[a-z]	a through z.
[^abc]	Any character except a, b, or c.
aa|bb	Either aa or bb.
?	Zero or one of the preceding element.
*	Zero or more of the preceding element.
+	One or more of the preceding element.
{n}	Exactly *n* of the preceding element.
{n,}	*n* or more of the preceding element.
{m,n}	Between *m* and *n* of the preceding element.
??,*?,+?, {n}?, etc.	Same as above, but as few as possible.
(*expr*)	Capture *expr* for use with \1, etc.
(?:*expr*)	Non-capturing group.
(?=*expr*)	Followed by *expr*.
(?!*expr*)	Not followed by *expr*.

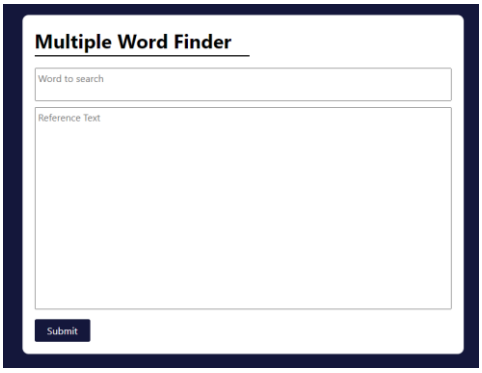
Near-complete reference

Gambar 2.15 Notasi Umum Regex
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

III. PEMBAHASAN

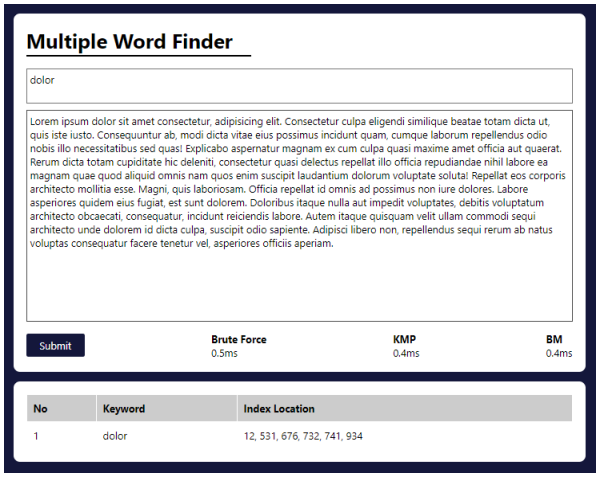
Pada implementasi algoritma *string matching*, akan dibandingkan kecepatan dari ketiga jenis algoritma *string matching* yang sudah disebutkan sebelumnya, yaiut algoritma *Brute Force*, algoritma *Knuth-Morris-Pratt*, dan algoritma *Boyer-Moore* dengan memanfaatkan aplikasi web *Multiple Word Finder* sebagai metode pengetesan dari masing – masing algoritma.

Sebagai media untuk mengetes ketiga algoritma tersebut, penulis telah membuat aplikasi *Multiple Word Finder* berbasis web untuk mempermudah proses pemasukkan teks yang nanti akan diperiksa, dan menampilkan waktu yang dibutuhkan masing – masing algoritma untuk menyelesaikan proses pencocokan *string*, serta posisi dari masing - masing kata yang dicari pada teks. Berikut adalah aplikasi website yang akan digunakan penulis sebagai media perbandingan kecepatan ketiga algoritma *string matching*.



Gambar 3.1 Aplikasi Website *Multiple Word Finder*

Implementasi ketiga algoritma dalam program dibuat dalam bentuk *object-oriented* agar mempermudah proses pencarian sejumlah kata untuk masing – masing algoritma pencocokan *string*. Setiap kelas memiliki fungsi pembantu yang sama, yaitu mengkonversi teks yang akan diperiksa, atau *pattern*, menjadi sebuah *array* kata, yang nantinya setiap array akan dicari apakah berada pada teks referensi atau tidak. Jika kata ditemukan pada teks, maka akan dimasukkan posisi/indeks kata tersebut pada teks ke dalam suatu array. Lalu, setelah semua posisi ditemukan, fungsi utama algoritma *string matching* akan mengembalikan *array* tersebut dan disimpan ke dalam sebuah objek *Map*, untuk memetakan *array* of posisi dengan kata acuannya. Hasil pemetaan tersebut akan ditampilkan pada website dalam bentuk tabel, yang berisi kata pencarian, serta posisi/indeks kata tersebut ditemukan dalam teks. Jadi hasil yang dikeluarkan oleh aplikasi adalah kecepatan dari masing – masing algoritma, serta objek *Map* yang berisi kata pencarian dan posisi ditemukannya. Berikut adalah contoh hasil eksekusi aplikasi *Multiple Word Finder*.



Gambar 3.2 Contoh Eksekusi Aplikasi *Multiple Word Finder*

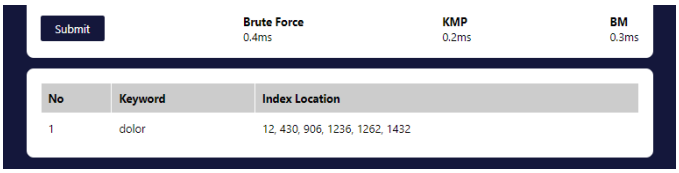
Kode lengkap dari *Multiple Word Finder* dapat diakses pada link berikut:

https://github.com/jasonk19/multiple-word-finder.git

A. *Uji Coba 1*

Percobaan pertama akan dilakukan dengan test case sebagai berikut:

- Kata pencarian sebanyak 1
- Referensi teks sebanyak 200 kata



Gambar 3.3 Uji Coba 1

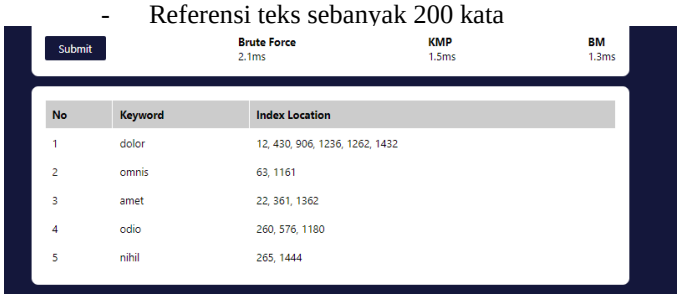
Dari percobaan pada uji coba 1, diperoleh hasil sebagai berikut:

- Algoritma *Brute Force*: 0.4ms
- Algoritma KMP: 0.2ms
- Algoritma BM: 0.3ms

B. Uji Coba 2

Percobaan kedua akan dilakukan dengan test case sebagai berikut:

- Kata pencarian sebanyak 5
- Referensi teks sebanyak 200 kata



Gambar 3.4 Uji Coba 2

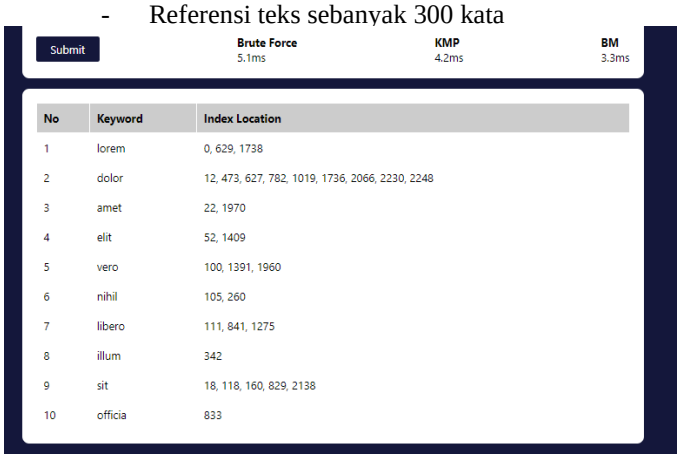
Dari percobaan pada uji coba 2, diperoleh hasil sebagai berikut:

- Algoritma *Brute Force*: 2.1ms
- Algoritma KMP: 1.5ms
- Algoritma BM: 1.3ms

C. Uji Coba 3

Percobaan ketiga akan dilakukan dengan test case sebagai berikut:

- Kata pencarian sebanyak 10
- Referensi teks sebanyak 300 kata



Gambar 3.5 Uji Coba 3

Dari percobaan pada uji coba 3, diperoleh hasil sebagai berikut:

- Algoritma *Brute Force*: 5.1ms
- Algoritma KMP: 4.2ms
- Algoritma BM: 3.3ms

D. Analisis

Setelah menguji aplikasi *Multiple Word Finder* pada 3 test case berbeda, diperoleh kecepatan ketiga algoritma sebagai berikut:

Tabel 3.1 Perbandingan Kecepatan Algoritma

No	*Brute Force*	*Knuth Morris Pratt*	*Boyer Moore*
1	0.4ms	0.2ms	0.3ms
2	2.1ms	1.5ms	1.3ms
3	5.1ms	4.2ms	3.3ms
Rata - rata	2.53ms	1.97ms	1.63ms

Berdasarkan tabel di atas, diperoleh hasil bahwa dalam aplikasi *Multiple Word Finder*, algoritma *Boyer-Moore* memiliki waktu eksekusi tercapat, diikuti oleh algoritma *Knuth-Morris-Pratt*, lalu eksekusi waktu paling lama adalah algoritma *Brute Force*. Dari hasil yang sudah dapat, hal ini sesuai dengan apa yang sudah dijelaskan pada teori, dimana algoritma *Brute Force* memiliki kompleksitas waktu yang lebih lama dibandingkan algoritma KMP dan BM, sedangkan algoritma BM dapat menjadi yang tercepat adalah karena, sesuai dengan teori, cocok dan umumnya cepat untuk melakukan pencarian kata dalam suatu teks.

Akan tetapi, dalam beberapa kasus, walaupun kejadiannya kecil, algoritma *brute force* memiliki waktu yang lebih cepat dibanding algoritma KMP dan BM, hal tersebut dapat diakibatkan karena pengaruh dari device yang mempengaruhi kecepatan suatu algoritma.

IV. Kesimpulan

Algoritma *string matching* merupakan algoritma yang digunakan untuk mencari suatu kata dalam suatu teks yang bersesuaian dengan kata/*pattern* tersebut. Ada berbagai macam algoritma *string matching* yang dapat diaplikasikan, seperti algoritma *Brute Force*, algoritma *Knuth-Morris-Pratt*, algoritma *Boyer-Moore*, algoritma *Rabin-Karp*, dan masih banyak lagi. Setiap algoritma berjalan dengan efisiensi yang berbeda – beda. Pengecekan efisiensi dari setiap algoritma dapat diuji dalam berbagai kasus maupun program yang berbeda – beda, salah satunya adalah pengujian pada makalah ini, yaitu *Multiple Word Finder*. Pada aplikasi *Multiple Word Finder*, dapat dihitung waktu eksekusi masing – masing algoritma untuk nanti dibandingkan waktunya. Walaupun algoritma *string matching* ini efisien dalam melakukan pencarian kata, masih banyak algoritma yang jauh lebih baik untuk nantinya dapat dieksplorasi lebih lanjut, seperti penggunaan *Artificial Intelligence* dalam melakukan pencarian kata.

Video Link at Youtube

https://youtu.be/9xRj2nGSlGw

Ucapan Terima Kasih

Pertama – tama, penulis mengucapkan syukur kepada Tuhan Yang Maha Esa atas segala berkat yang diberikan sehingga penulis dapat menyelesaikan makalah ini dengan baik dan tepat waktu. Penulis juga mengucapkan terima kasih kepada orang tua dan teman – teman yang selalu mendukung selama proses pengerjaan makalah ini. Tidak lupa juga penulis

mengucapkan terima kasih kepada Dr. Nur Ulfa Maulidevi, S.T, M.Sc. selaku dosen K2 mata kuliah Strategi Algoritma yang telah memberi materi serta ilmu sepanjang Semester 2 Tahun Ajar 2021/2022. Penulis menyadari masih adanya kekurangan dan kesalahan kata dalam makalah ini, penulis berharap makalah ini dapat digunakan sebaik – baiknya dan dikembangkan sehingga bermanfaat bagi masyarakat luas, khususnya para pelajar.

REFERENSI

[1] Munir, Rinaldi. "Pencocokan string (String matching/pattern matching)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada tanggal 21 Mei 2022.

[2] Munir, Rinaldi. "Pencocokan string dengan Regular Expression (Regex)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf diakses pada tanggal 21 Mei 2022.

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, 23 Mei 2022

Jason Kanggara
13520080