Penerapan *String Matching* dan Algoritma *Greedy* dalam Rekomendasi Kata di Mesin Pencarian

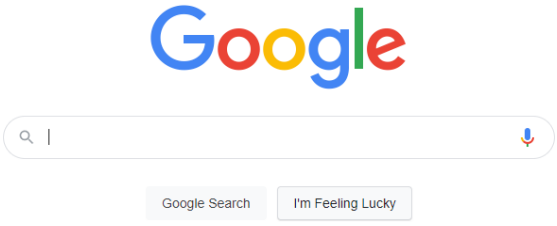
Gayuh Tri Rahutami 13519192
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13519192@std.stei.itb.ac.id

***Abstract*—Mesin pencarian, suatu alat yang melakukan pencarian terhadap suatu basis data berdasarkan masukan pengguna, adalah salah satu aplikasi yang paling sering digunakan pada zaman ini. Tidak jarang suatu mesin pencarian menerapkan algoritma perekomendasian kata berdasarkan kata-kata yang pernah dicari sebelumnya oleh pengguna. Salah satu penerapan yang bisa digunakan untuk membuat algoritma perekomendasian kata adalah dengan mengaplikasikan algoritma *string matching* dan algoritma *greedy*.**

***Keywords*—*String Matching*, Algoritma *Greedy*, Mesin Pencarian**

I. Pendahuluan

Mesin pencarian (*search engine*) merupakan suatu alat yang melakukan pencarian terhadap suatu basis data berdasarkan masukan pengguna. Saat ini, terdapat banyak sekali aplikasi yang menggunakan mesin pencarian untuk melakukan pencarian terhadap basis data internet (Google, Bing, Yandex) ataupun terhadap basis data suatu website (mesin pencarian produk di aplikasi e-commerce, mesin pencarian artikel di aplikasi berita).

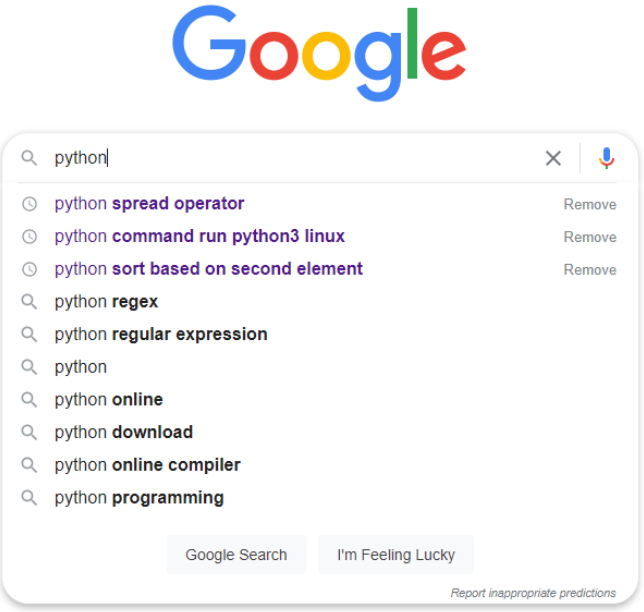


Gambar 1.1 Mesin pencarian Google

sumber: google.com

Tidak jarang mesin pencarian yang diterapkan pada suatu aplikasi memiliki fitur rekomendasi kata untuk memudahkan pengguna dalam melakukan pencarian. Kata-kata yang direkomendasikan oleh mesin pencarian dipilih berdasarkan riwayat pencarian yang telah dilakukan oleh pengguna.

Terdapat banyak cara untuk menerapkan fitur rekomendasi kata tersebut. Salah satunya adalah dengan menerapkan *string matching* untuk mencocokkan *query* dengan kata-kata di riwayat pencarian dan algoritma *greedy* untuk menentukan kata-kata yang akan direkomendasikan.



Gambar 1.2 Fitur rekomendasi kata di Google

sumber: google.com

II. Landasan Teori

1. *String Matching*

String matching atau yang sering disebut juga sebagai *pattern matching* atau *string searching* merupakan algoritma yang digunakan untuk mencari keberadaan suatu *string* di dalam *string* lain yang lebih panjang. Sampai saat ini, sudah ditemukan banyak metode untuk melakukan *string matching*, seperti algoritma Knuth Morris Pratt, algoritma Boyer Moore, dan *Regular Expression*.

1.1. Algoritma Brute Force untuk *String Matching*

Algoritma *Brute Force* merupakan pendekatan yang *straightforward* untuk memecahkan suatu persoalan yang biasanya didasarkan pada pernyataan pada persoalan atau konsep yang dilibatkan. *Brute force* merupakan algoritma

pemecahan persoalan yang paling sederhana tapi bisa memakan waktu yang lama.

Pada *string matching*, algoritma *brute force* dapat diterapkan dengan cara seperti berikut:

1. Diberikan suatu teks T dan kata yang ingin dicari P. Mula-mula P disejajarkan dengan awal dari T.
2. Telusuri P dari kiri ke kanan dan bandingkan tiap karakter pada P dengan kata pada teks hingga semua karakter pada P cocok (pencarian berhasil) atau dijumpai sebuah ketidakcocokan.
3. Jika dijumpai ketidakcocokan, geser P satu ke kanan dan ulangi langkah 2 hingga pencarian berhasil atau hingga akhir dari P sejajar dengan akhir dari T.

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 2.1.1.1 Ilustrasi pergeseran *pattern* pada algoritma *brute force*

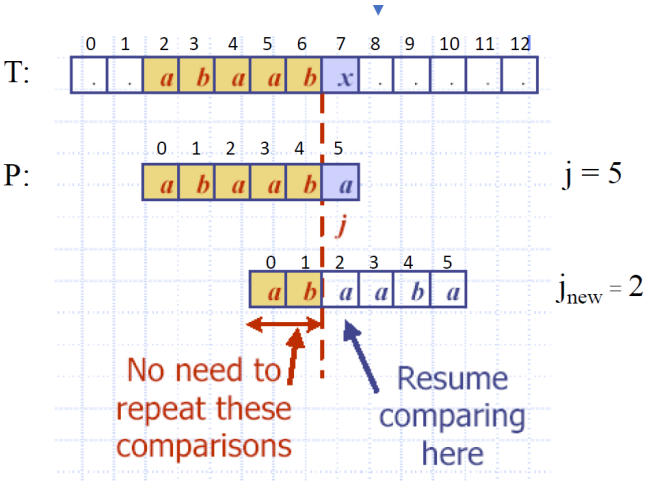
sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm

Pada algoritma *brute force* untuk *string matching* kemungkinan terburuknya adalah ketidakcocokan selalu ditemukan di karakter terakhir dari P sehingga kompleksitas waktu untuk kemungkinan terburuk adalah O(mn) dengan m adalah panjang P dan n adalah panjang T. Sedangkan kemungkinan terbaiknya adalah ketidakcocokan ditemukan di karakter pertama dari P sehingga kompleksitas waktu untuk kemungkinan terburuk adalah O(n). Sedangkan kompleksitas waktu rata-rata adalah O(m+n).

1.2. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt atau yang disebut juga algoritma KMP adalah algoritma *string matching* yang mencari *pattern* di dalam sebuah teks dari kiri ke kanan sama seperti pada algoritma *brute force*. Akan tetapi, pada algoritma KMP ketika terjadi ketidakcocokan antara teks dan kata yang ingin dicari, kata akan digeser sebanyak mungkin agar tidak terjadi perbandingan yang tidak berguna, yaitu sepanjang prefiks terpanjang dari P[0..j-1] yang juga merupakan sufiks dari P[1..j-1].

Untuk menentukan banyak pergeseran yang perlu dilakukan, perlu dihitung *border function* dari kata yang ingin dicari sebelum pencocokan *string* dimulai di mana suatu *border function* b(k) adalah panjang prefiks terpanjang dari P[0..k] yang juga merupakan sufiks dari P[1..k] dan k adalah j-1 di mana j adalah posisi ketidakcocokan terjadi.



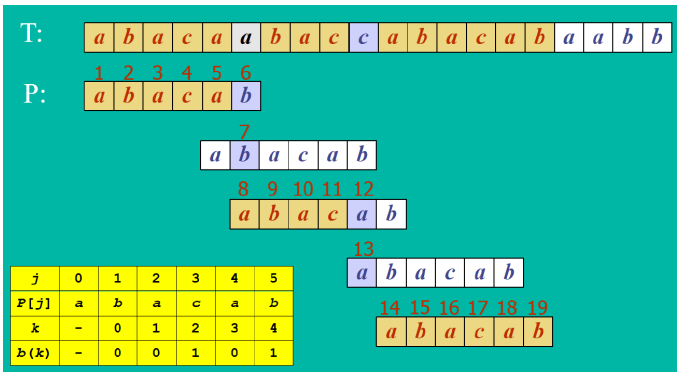
Gambar 2.1.2.1 Ilustrasi pergeseran *pattern* pada algoritma KMP

sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm

Tabel 2.1.2.1 Contoh *border function*

j	0	1	2	3	4	5	6	7	8	9
P[j]	a	b	a	b	a	b	a	b	c	a
k	-	0	1	2	3	4	5	6	7	8
b(k)	-	0	0	1	2	3	4	5	6	0

Setelah *border function* dibuat, pencocokan *string* kemudian dilakukan dengan mencocokkan teks dengan kata yang ingin dicari dari kiri ke kanan. Kemudian, ketika ketidakcocokkan terjadi kata yang ingin dicari akan digeser sehingga b(k) sejajar dengan i.



Gambar 2.1.2.2 Contoh *string matching*

dengan algoritma KMP

sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm

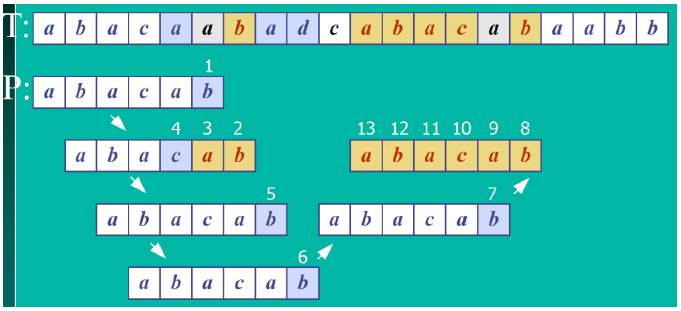
Pada algoritma KMP, untuk menghitung *border function* dibutuhkan kompleksitas waktu O(m) dengan m adalah panjang kata yang ingin dicari dan untuk pencocokan string dibutuhkan kompleksitas waktu O(n) dengan n adalah panjang teks. Sehingga kompleksitas waktu dari algoritma KMP adalah O(m+n)

1.3. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah algoritma *string matching* yang menerapkan dua teknik, yaitu dengan teknik *looking-glass,* teknik pencocokan *string* dengan memulai pencocokan dari huruf terakhir kata yang ingin dicari, dan teknik *character-jump* yang dilakukan setiap terjadi ketidakcocokan karakter.

Pada algoritma Boyer-Moore terdapat tiga kemungkinan yang bisa terjadi ketika ketidakcocokan karakter terjadi, yaitu:

1. Ketika terjadi ketidakcocokan pada P[j] dengan T[i] dengan T[i] = x, P mengandung x, dan kemunculan terakhir x di P berada di sebelah kiri j, maka geser P sehingga T[i] sejajar dengan kemunculan terakhir x di P.
2. Jika P mengandung x tetapi kemunculan terakhir x di P berada di sebelah kanan j, maka geser P sebanyak 1 ke kiri.
3. Jika P tidak mengandung x, maka geser P sehingga P[0] sejajar dengan T[i+1]



Gambar 2.1.2.3 Contoh *string matching*

dengan algoritma Boyer-Moore

sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm

Algoritma Boyer-Moore melakukan pemrosesan awal pada kata yang ingin dicari dengan mencari kemunculan terakhir tiap-tiap karakter yang ada di dalam alfabet di dalam kata tersebut. Fungsi ini disebut sebagai *last occurrence function* L di mana L(x) adalah indeks terbesar sehingga P[i] = x atau -1 jika x tidak ada di dalam P.

Pada algoritma Boyer-Moore, untuk menghitung fungsi *last occurrence function* dibutuhkan waktu O(A) yaitu banyak alfabet dan untuk pencocokan string dibutuhkan kompleksitas waktu O(nm) dengan n adalah panjang teks dan m adalah panjang kata yang ingin dicari. Sehingga kompleksitas waktu algoritma Boyer-Moore adalah O(nm+A). Akan tetapi, perlu dicatat bahwa algoritma Boyer-Moore cepat ketika A besar sehingga baik digunakan untuk pencocokan teks sehari-hari, tetapi lambat untuk pencocokan teks biner.

1.4. Regular Expressions

Regular Expressions (Regex) adalah suatu *string* spesial yang digunakan untuk mendeskripsikan suatu pola dalam pencarian. Regex memiliki karakter-karakter spesial dengan notasi yang tertera di tabel 2.1.4.1.

Tabel 2.1.4.1 Notasi *Regular Expressions*

Karakter	Kegunaan
.	Karakter apapun kecuali *newline.*
^	Awal dari suatu *string.*
$	Akhir dari suatu string.
\d,\w,\s	Suatu digit, suatu *word character* (A-Za-z0-9_), atau *whitespace.*
\D, \W,\S	Karakter apapun kecuali suatu digit, suatu *word character* (A-Za-z0-9_), atau *whitespace.*
[abc]	Karakter a atau b atau c.
[a-z]	Karakter a sampai z.
[^abc]	Karakter selain a, b, dan c.
aa|bb	aa atau bb
?	Nol atau satu kemunculan dari elemen sebelum.
*	Nol atau lebih kemunculan dari elemen sebelum.
+	Satu atau lebih kemunculan dari elemen sebelum.
{n}	Tepat n kemunculan dari elemen sebelum.
{n,}	n atau lebih kemunculan dari elemen sebelum.
{m,n}	m sampai n kemunculan dari elemen sebelum

2. Algoritma *Greedy*

Algoritma *greedy* adalah algoritma yang memecahkan persoalan secara langkah per langkah sedemikian sehingga pada setiap langkah diambil pilihan terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan dengan harapan pemilihan optimum lokal akan berakhir pada optimum global.

2.1. Elemen-Elemen pada Algoritma *Greedy*

1. Himpunan Kandidat (C)

   Berisi kandidat yang bisa dipilih pada setiap langkah.

2. Himpunan Solusi (S)

   Berisi kandidat yang sudah dipilih pada langkah-langkah sebelumnya.

3. Fungsi Solusi

   Menentukan apakah himpunan solusi pada langkah saat ini sudah memberikan solusi.

4. Fungsi Seleksi

   Menentukan kandidat yang akan dipilih pada suatu langkah berdasarkan strategi *greedy* yang telah ditentukan.

5. Fungsi Kelayakan

   Menentukan apakah suatu kandidat layak untuk dimasukkan ke dalam himpunan solusi atau tidak.

6. Fungsi Obyektif

   Menentukan tujuan dari algoritma.

III. Rekomendasi Kata di Mesin Pencarian dengan *String Matching dan Algoritma Greedy*

1. Elemen-Elemen Greedy pada Program Rekomendasi Kata

1. Himpunan Kandidat (C)

   Himpunan kandidat berisi kata-kata yang berada di riwayat pencarian pengguna.

2. Himpunan Solusi (S)

   Himpunan solusi berisi tiga kata yang akan direkomendasikan.

3. Fungsi Solusi

   Fungsi solusi menentukan apakah sudah ditemukan tiga kata yang valid untuk direkomendasikan atau semua kata di riwayat pencarian telah diperiksa.

4. Fungsi Seleksi

   Terdapat 2 fungsi seleksi yang bisa dipakai:

   a. Mengambil kata yang paling terakhir dicari.
   b. Mengambil kata yang paling banyak dicari.

5. Fungsi Kelayakan

   Suatu kata layak untuk dimasukkan ke himpunan solusi apabila kata tersebut diawali dengan masukan pengguna.

6. Fungsi Obyektif

   Mendapatkan maksimal tiga kata untuk direkomendasikan.

2. Langkah-Langkah Umum Program

Program pencarian rekomendasi kata dijalankan tiap kali pengguna mengetikkan 1 karakter ke input. Langkah-langkah program pencarian rekomendasi kata adalah sebagai berikut:

1. Kata-kata yang ada di riwayat pencarian diurutkan berdasarkan fungsi seleksi yang dipilih (tanggal terakhir pencarian atau jumlah pencarian)
2. Telusuri list kata-kata tersebut dari indeks 0. Apabila terdapat suatu kata yang diawali dengan *query* yang telah dimasukkan pengguna, masukkan kata tersebut ke dalam himpunan solusi. (menggunakan *string matching*)
3. Apabila telah ditemukan 3 kata rekomendasi atau seluruh kata-kata di dalam riwayat pencarian telah ditelusuri, tampilkan daftar rekomendasi kata yang telah didapatkan.

IV. Hasil Prorgam

Program rekomendasi kata akan dibuat di atas aplikasi mesin pencarian program televisi yang telah dibuat oleh penulis sebelumnya. Kemudian di dalam aplikasi tersebut telah tersimpan riwayat pencarian seperti yang ditampilkan pada tabel 4.1.

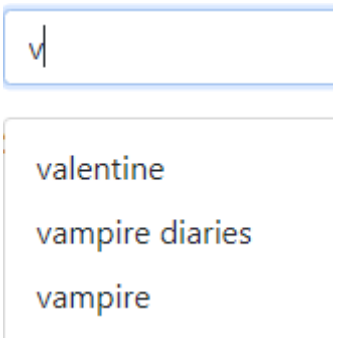
Tabel 4.1 Riwayat Pencarian

Query	Waktu terakhir pencarian	Jumlah pencarian
vincenzo	5/10/2021, 10:36:55 PM	2
love alarm	5/10/2021, 10:34:46 PM	3
first love	5/10/2021, 10:34:37 PM	1
true beauty	5/10/2021, 10:35:00 PM	1
this is my first life	5/10/2021, 10:35:06 PM	1
born this way	5/10/2021, 10:35:13 PM	1
because this is my first life	5/10/2021, 10:34:26 PM	4
boys	5/10/2021, 10:36:15 PM	1
girls	5/10/2021, 10:36:21 PM	2
vampire	5/10/2021, 10:37:06 PM	1
vampire diaries	5/10/2021, 10:37:09 PM	1
valentine	5/10/2021, 10:37:11 PM	1
barbie	5/10/2021, 10:37:54 PM	2

1. Hasil Program Menggunakan Fungsi Seleksi Kata Paling Terakhir Dicari

Dengan menggunakan fungsi seleksi berupa kata yang paling terakhir dicari, didapatkan hasil sebagai berikut:

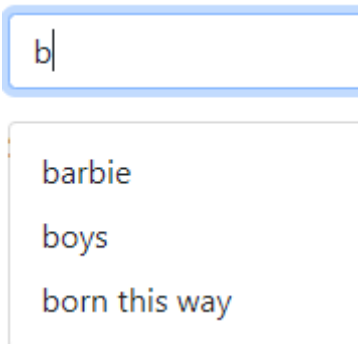
1. Query ‘v’



Gambar 4.1.1.1 Rekomendasi kata query 'v' dengan fungsi seleksi kata paling terakhir dicari.

sumber: Dokumen pribadi

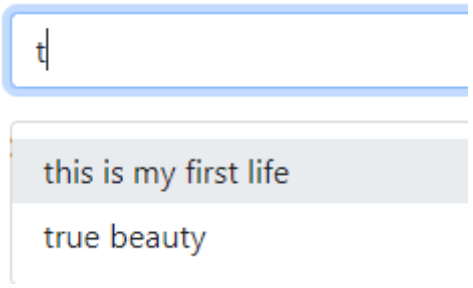
2. Query 'b'



Gambar 4.1.1.2 Rekomendasi kata query 'b' dengan fungsi seleksi kata paling terakhir dicari.

sumber: Dokumen pribadi

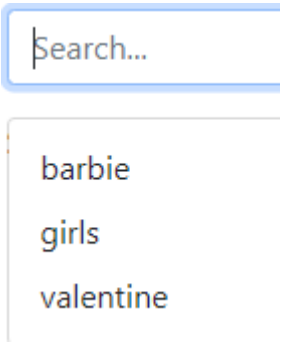
3. Query 't'



Gambar 4.1.1.3 Rekomendasi kata query 'b' dengan fungsi seleksi kata paling terakhir dicari.

sumber: Dokumen pribadi

4. Query kosong



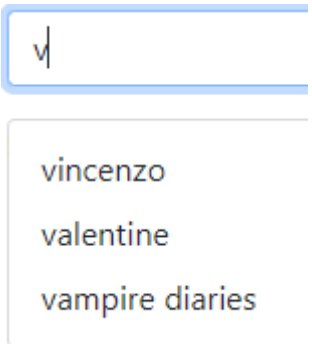
Gambar 4.1.1.4 Rekomendasi kata query kosong dengan fungsi seleksi kata paling terakhir dicari.

sumber: Dokumen pribadi

2. Hasil Program Menggunakan Fungsi Seleksi Kata Paling Banyak Dicari

Dengan menggunakan fungsi seleksi berupa kata yang paling banyak dicari, didapatkan hasil sebagai berikut:

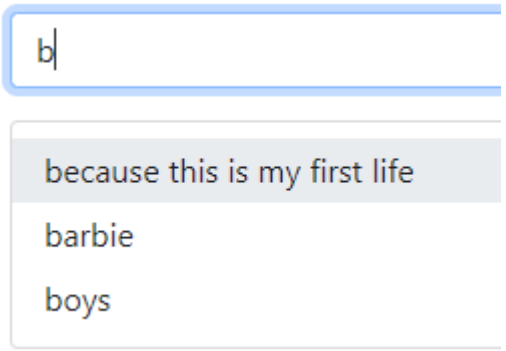
1. Query 'v'



Gambar 4.1.2.1 Rekomendasi kata query 'v' dengan fungsi seleksi kata paling banyak dicari.

sumber: Dokumen pribadi

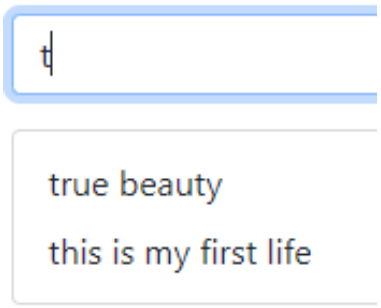
2. Query 'b'



Gambar 4.1.2.2 Rekomendasi kata query 'b' dengan fungsi seleksi kata paling banyak dicari.

sumber: Dokumen pribadi

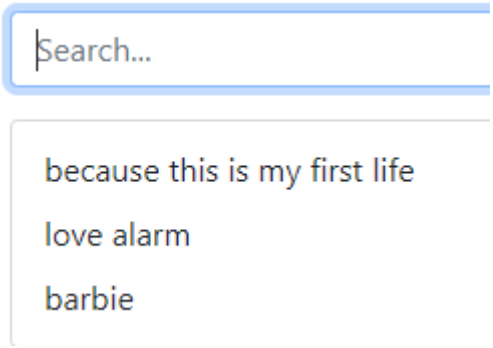
3. Query ‘t’



Gambar 4.1.2.3 Rekomendasi kata query ‘b’ dengan fungsi seleksi kata paling banyak dicari.

sumber: Dokumen pribadi

4. Query kosong



Gambar 4.1.2.4 Rekomendasi kata query kosong dengan fungsi seleksi kata paling banyak dicari.

sumber: Dokumen pribadi

V. Kesimpulan

Suatu mesin pencarian seringkali memberikan rekomendasi kata ketika pengguna sedang memasukkan kata yang ingin dicarinya. Salah satu cara yang bisa digunakan untuk mendapatkan daftar kata rekomendasi adalah dengan menggunakan *string matching* dan algoritma *greedy*. *String matching* digunakan untuk mencari suatu kata di riwayat pencarian pengguna yang diawali dengan kata yang telah diketik oleh pengguna. Sedangkan algoritma *greedy* digunakan untuk menentukan kata yang akan direkomendasikan apabila terdapat lebih dari satu kata yang cocok.

VI. Saran

Tiap-tiap pengguna yang menggunakan suatu aplikasi mesin pencarian biasanya memiliki *pattern* pencarian yang berbeda. Misalnya, terdapat pengguna yang melakukan pencarian yang sama berulang kali dalam jangka waktu pendek, tetapi ada juga pengguna yang melakukan pencarian suatu kata yang sama tiap suatu jangka waktu tertentu. Algoritma *greedy* yang diterapkan pada pengaplikasian ini belum mempertimbangkan *pattern* pencarian yang dilakukan oleh pengguna. Selain itu, algoritma *greedy* pada pengaplikasian ini juga belum mempertimbangkan riwayat pencarian seluruh pengguna aplikasi. Kedepannya, untuk membuat rekomendasi kata yang didapatkan menjadi lebih akurat dapat menerapkan *machine learning* yang mempelajari *pattern* pencarian yang dilakukan oleh pengguna dan melakukan analisis terhadap riwayat pencarian seluruh pengguna aplikasi.

VII. Link

1. Link Aplikasi

   https://wundersmith.github.io/tvShowSearchEngine/index.html

2. Link Source Code

   https://github.com/wundersmith/wundersmith.github.io/tree/master/tvShowSearchEngine

3. Link Video Youtube

   https://youtu.be/V0Bu9aJwbLs

VIII. Ucapan Terima Kasih

Saya mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa karena berkat rahmat dan hidayah-Nya saya bisa menyelesaikan makalah ini dengan tepat waktu tanpa ada kendala yang berarti. Selanjutnya saya ingin berterima kasih kepada orang tua saya yang selalu mendukung saya dalam menghadapi dunia perkuliahan dan juga para dosen pengampu mata kuliah IF2211 Strategi Algoritma, khususnya bapak Rinaldi Munir, dosen pengampu K-04, yang telah memberikan ilmunya kepada saya sehingga saya memiliki pengetahuan yang cukup untuk menyelesaikan makalah ini. Semoga makalah ini dapat bermanfaat bagi para pembaca dan juga saya sendiri.

Referensi

[1] https://www.geeksforgeeks.org/algorithms-gq/pattern-searching/

[2] https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/

[3] https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/

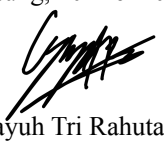
[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm

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, 10 Mei 2021



Gayuh Tri Rahutami

13519192