Penggunaan Algoritma Knuth-Morris-Pratt untuk Pengecekan Ejaan

Andreas Dwi Nugroho - 13511051
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511051@std.stei.itb.ac.id

***Abstrak*—Ketika ingin melakukan pencarian dengan *keyword* tertentu pada mesin pencari, terkadang kita salah mengetikkan ejaan *keyword* yang ingin kita cari sehingga hasil pencarian yang ditampilkannya pun kadang tidak ada atau tidak sesuai dengan apa yang kita cari. Namun biasanya mesin pencari akan melakukan pengecekan ejaan *keyword* dan memberikan pilihan *keyword* yang menyerupai dengan *keyword* yang kita masukkan tapi dengan ejaan yang benar. Pencarian *keyword* yang mirip tersebut dapat dilakukan dengan menerapkan algoritma *pattern* matching, salah satunya algoritma Knuth-Morris-Pratt (KMP).**

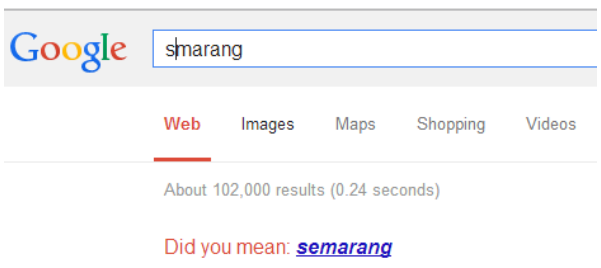
Makalah ini akan membahas tentang pencarian *string* yang menyerupai *string* tertentu untuk membantu pengecekan ejaan dengan uji coba menggunakan algoritma KMP.

***Kata Kunci*—KMP, ejaan, *pattern matching*, pencarian.**

I. PENDAHULUAN

Mesin pencari sangat dibutuhkan saat ini. Hampir semua orang terbantu untuk mencari informasi melalui mesin pencari. Hanya dengan memasukkan *keyword*, maka akan muncul ratusan hasil yang terkait dengan *keyword* tersebut. Namun terkadang beberapa orang tidak sengaja memasukkan *keyword* dengan ejaan yang salah, mungkin *keyword* tersebut kurang sejumlah huruf atau salah mengetikkan beberapa huruf dalam *keyword* tersebut. Hal ini berdampak pada hasil pencarian yang muncul. Mungkin hasil pencarian yang keluar tidak sesuai dengan apa yang kita maksud atau kadang-kadang pencarian dengan *keyword* tersebut tidak memberikan hasil sama sekali.

Namun jika hal itu terjadi, biasanya mesin pencari akan memberi tahu kita bahwa ejaan *keyword* yang kita masukkan ada yang salah. Contohnya pada Google Search seperti berikut ini:



Dalam contoh tersebut penulis mengetikkan *keyword* dengan ejaan yang salah, tapi aplikasi Google Search memberikan rekomendasi mengenai *keyword* yang mirip dengan *keyword* yang penulis masukkan. Dan memang benar *keyword* yang diberikan Google tersebut sebenarnya *keyword* yang penulis cari.

Untuk pengecekan apakah ejaan suatu *keyword* itu benar atau salah bisa dilakukan dengan mencari dalam kamus. Jika suatu *keyword* tidak terdapat dalam kamus, bisa berarti *keyword* tersebut mungkin salah ejaan. Untuk memberikan rekomendasi *keyword* yang sesuai dengan yang kita masukkan dapat dilakukan dengan mencari *keyword* dalam kamus yang mempunyai kemiripan dengan masukan *keyword* dengan ejaan yang salah sebelumnya. Untuk mencari kemiripan tersebut bisa dilakukan dengan melakukan pencarian *substring* dari *keyword* yang ingin dicari terhadap kata-kata dalam kamus. Semakin banyak *substring* yang ditemukan pada suatu kata, kemungkinan besar kata tersebut merupakan kata yang dicari. Pencocokan *substring* tersebut bisa dilakukan dengan menggunakan algoritma KMP.

II. ALGORITMA KNUTH MORRIS PRATT

Algoritma Knuth-Morris-Pratt dikembangkan oleh D. E. Knuth, bersama-sama dengan J. H. Morris dan V. R. Pratt. Dalam algoritma pencarian *string* dengan menggunakan brute force, ketika ditemukan ketidakcocokan *pattern* dengan teks, maka *pattern* tersebut digeser satu karakter ke kanan. Sedangkan jika menggunakan algoritma KMP (Knuth-Morris-Pratt), informasi yang digunakan untuk melakukan jumlah pergeseran setiap terjadi ketidakcocokan akan dikelola dalam algoritma ini. Kemudian algoritma ini menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak seperti algoritma brute force yang hanya digeser satu karakter. Dengan algoritma KMP, waktu pencarian dapat dikurangi secara signifikan.

Dalam algoritma KMP, terdapat beberapa definisi yang digunakan :

Misalkan *A* adalah alfabet dan $x = x1x2\ldots xk$, $k \in \mathbf{N}$, adalah *string* yang panjangnya k yang dibentuk dari karakter-karakter di dalam alfabet *A*.

- Awalan (*prefix*) dari *x* adalah upa-*string* (*substring*) *u* dengan $u = x1x2\ldots xk - 1$, $k \in \{1,2, \ldots, k - 1\}$ dengan kata lain, *x* diawali dengan *u*.
- Akhiran (*suffix*) dari *x* adalah upa*string* (*substring*) *u* dengan $u = xk - b\ xk - b + 1 \ldots xk$, $k \in \{1, 2, \ldots, k - 1\}$ dengan kata lain, *x* diakhiri dengan *v*.
- Pinggiran (*border*) dari *x* adalah upa-*string* (*substring*) *r* sedemikian sehingga $r = x1\ x2\ldots xk - 1$ dan $u = xk - b\ xk - b + 1 \ldots xk, k \in \{1, 2, \ldots, k = 1\}$, dengan kata lain, pinggiran dari *x* adalah upa-*string* yang keduanaya awalan dan juga akhiran sebenarnya dari *x*.

Algoritma KMP melakukan proses awal terhadap *pattern* P dengan menghitung fungsi pinggiran yang mengindikasikan pergeseran s terbesar yang mungkin dengan menggunakan perbandingan yang dibentuk sebelum pencarian *string*. Dengan menggunakan fungsi pinggiran ini, dapat dicegah pergeseran yang tidak berguna seperti yang terjadi pada algoritma brute force. Fungsi pinggiran tersebut hanya bergantung pada karakter-karakter di dalam *pattern*, bukan karakter-karakter yang ada di dalam teks.

Fungsi pinggiran b(j) didefinisikan sebagai ukuran awalan terpanjang dari P yang merupakan akhiran dari P[1..j]. Sebagai contoh, misalnya *pattern* P = abcabd. Nilai fungsi pinggiran untuk setiap karakter dalam *pattern* P sebagai berikut:

j	1	2	3	4	5	6
P[j]	a	b	c	a	b	d
b(j)	0	0	0	1	2	0

Algoritma untuk menghitung pinggiran sebagai berikut:

```
        procedure HitungPinggiran
(input m : integer, P : array[1..m] of
char, output b : array[1..m] of integer)
{ Menghitung nilai b[1..m] untuk pattern P[1..m] }

Deklarasi
k,q : integer

Algoritma:
b[1]←0
q←2
k←0
for q←2 to m do
  while ((k > 0) and (P[q] ≠ P[k+1])) do
     k←b[k]
  endwhile
  if P[q]=P[k+1] then
     k←k+1
  endif
  b[q]=k
endfor
```

Selanjutnya dilakukan percobaan pencocokan *pattern* P dengan sebuah teks T = abcabcabd.
Awal pencocokan digambarkan sebagai berikut:

Teks	a	b	c	a	b	c	a	b	d
Pattern	a	b	c	a	b	d			

Kemudian ditemukan ketidakcocokan pada posisi ke 6. Jumlah pergeseran setelah itu ditentukan oleh pinggiran dari awalan P yang bersesuain. Awalan yang diperoleh yaitu abcab, dengan panjang $l = 5$ dan pinggiran terpanjang untuk *string P*[1..5] adalah ab dengan nilai fungsi pinggirannya 2 sehingga jarak pergeseran selanjutnya adalah l – b = 5 – 2 = 3. Setelah digeser sebanyak 3 karakter akhirnya *pattern P* ditemukan dalam teks *T*.

Teks	a	b	c	a	b	c	a	b	d
Pattern				a	b	c	a	b	d

III. PENGGUNAAN ALGORITMA KMP PADA PENGECEKAN EJAAN

Untuk penggunaan algoritma KMP dalam pengecekan ejaan suatu *keyword* dilakukan prinsip kerja sebagai berikut :

1. Aplikasi akan menerima input berupa kata (yang digunakan sebagai *keyword* pencarian aplikasi pencarian) dengan ejaan yang sengaja disalahkan.
2. Kemudian untuk setiap *keyword* di dalam basis data yang sudah disiapkan dilakukan pencarian *substring* dari input dengan menggunakan algoritma KMP.
3. *Substring* yang dicari merupakan semua kemungkinan *substring* dari input yang diambil mulai karakter index ke 0 dalam input sampai index terakhir secara berurutan.
   Misal input : “meja”
   Substring yang mungkin didapat :

me
mej
meja
ej
eja
ja

4. Dari pencarian *substring* terhadap semua daftar *keyword*, diambil *keyword* dengan jumlah banyaknya *substring* yang ditemu paling banyak.

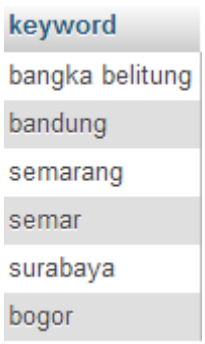
m	e	j	a
m	e		
m	e	j	
m	e	j	a
	e	j	
	e	j	a
		j	a

Semakin banyak *substring* yang ditemukan dalam sebuah *keyword* maka kemungkinan besar *keyword* tersebut merupakan *keyword* yang dicari.

5. Selanjutnya *keyword* tersebut ditampilkan sebagai kemungkinan hasil perbaikan ejaan pada input yang salah.

IV. UJI COBA APLIKASI

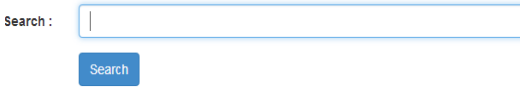
Dalam beberapa pengujian yang akan dilakukan, digunakan basis data yang memuat daftar *keyword* yang untuk uji coba. Berikut daftar sebagian *keyword* yang disimpan dalam basis data:



Gambar 2 Daftar *keyword* dalam basis data

Berikut ini adalah hasil uji coba aplikasi yang dikhususkan untuk menunjukkan pengecekan ejaan pada semacam aplikasi pencarian :

- Tampilan awal untuk menerima input berupa *keyword* yang ingin dicari

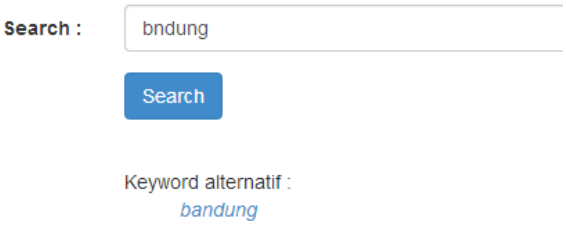


Gambar 3 Tampilan aplikasi

- Aplikasi menerima input berupa *keyword* yang sengaja ejaannya disalahkan

Gambar 4 Masukan aplikasi

- Aplikasi mencari kata yang sesuai dan mendekati dengan input serta menampilkan hasilnya.



Gambar 5 Hasil pengecekan

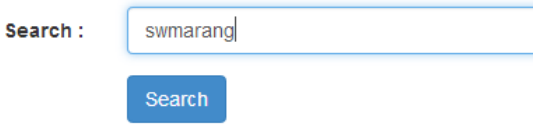
Dari hasil pencarian, diperoleh *keyword* yang mendekati dengan input dan ditampilkan pada daftar *keyword* alternatif seperti gambar diatas. Berdasarkan pengujian diatas, diperoleh hasil berupa kata “bandung” dari masukan kata “bndung” yang ejaan sengaja disalahkan dengan mengurangi salah satu huruf. Hal itu dikarenakan jumlah pencarian *substring* dari kata “bndung” paling banyak ditemukan dalam kata “bandung” dibandingkan dengan kata lainnya. Berikut ini merupakan proses pencarian *substring* yang ditemukan dalam kasus ini:

b	a	n	d	u	n	g
		n	d			
		n	d	u		
		n	d	u	n	
		n	d	u	n	g
			d	u		
			d	u	n	
			d	u	n	g
				u	n	
				u	n	g
					n	g

Kemudian dilakukan pengujian juga untuk *keyword* pencarian yang berbeda dan dengan ejaan yang sengaja

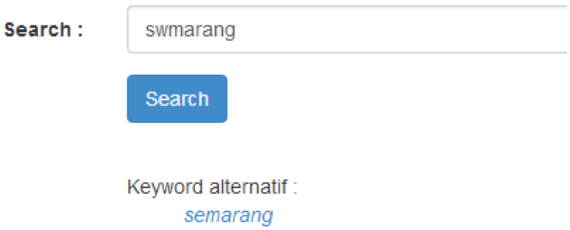
disalahkan untuk menunjukkan proses pengecekan ejaan. Misalnya dengan *keyword* "swmarang".

- Pengujian dengan menggunakan *keyword* "swmarang".



Gambar 6 Masukan untuk pengujian

- Hasil yang diperoleh dari pengecekan ejaan.

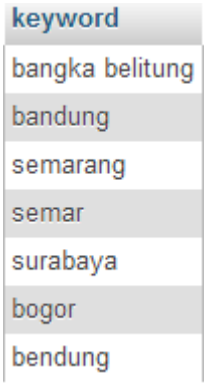


Gambar 7 Hasil pengujian

- Pada pengujian sebelumnya, digunakan masukan berupa kata yang dihilangkan salah satu hurufnya. Pengujian kali ini menggunakan masukan kata yang salah satu hurufnya dirubah. Dan diperoleh hasil seperti gambar diatas. Dari hasil tersebut didapat kata yang mendekati dengan masukan.

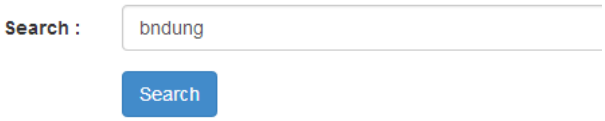
Selain itu, dilakukan juga pengujian dalam kasus dimana terdapat lebih dari satu *keyword* yang memiliki jumlah *substring* ditemukan sama. Berikut ini hasil dari pengujian tersebut.

- Sebelumnya dalam basis data ditambahkan sebuah *keyword* yang memiliki kemiripan dengan *keyword* lain untuk dipakai dalam pengujian ini, misalnya "bendung".



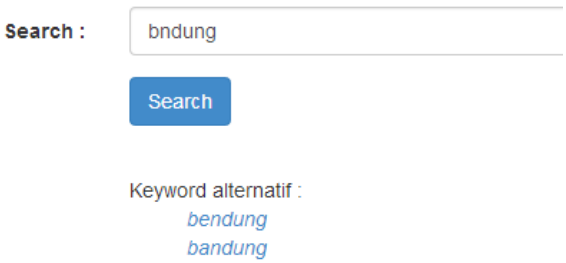
Gambar 8 Penambahan *keyword*

- Kemudian dilakukan pengujian dengan menggunakan *keyword* "bndung" pada aplikasi.



Gambar 9 Masukan pengujian

- Dari proses pengujian, diperoleh hasil seperti ini:



Gambar 10 Dua buah hasil pengujian

Berdasarkan hasil pengujian, ditampilkan sebanyak 2 buah hasil berupa *keyword* yang mirip dengan masukan. Hal itu dikarenakan jumlah *substring* yang ditemukan diantara 2 *keyword* tersebur sama banyaknya.

Dari beberapa pengujian yang dilakukan,, diperoleh hasil yang diinginkan. Untuk pencarian *string* yang memiliki kemiripan dengan *string* lain dapat dilakukan dengan pencocokan *substring* dengan setiap *string* dalam basis data kemudian *string* yang mempunyai jumlah *substring* yang ditemukan paling banyak maka *string* tersebut merupakan *string* yang dicari.

V. KESIMPULAN

Setelah dilakukan beberapa kali uji coba terhadap penggunaan algoritma KMP pada aplikasi pengecekan ejaan dapat disimpulkan bahwa algoritma ini dapat digunakan untuk mencari *string* yang mempunyai kemiripan dengan *string* tertentu dengan mencocokan semua kemungkinan *substring*.

Namun aplikasi ini dirasa masih kurang efisien dalam proses pencocokan *string*. Maka dari itu untuk kedepannya perlu pengembangan lebih lanjut supaya menjadi lebih baik lagi.

REFERENSI

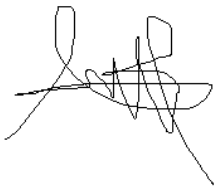
[1] Munir, Rinaldi, Diktat Kuliah IF2211 Strategi Algoritma. Program Studi Teknik Informatika ITB, 2013.

[2] http://fivedots.coe.psu.ac.th/Software.coe/LAB/PatMatch/Pattern Matching.ppt. 19 Desember 2013

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, 20 Desember 2013



Andreas Dwi Nugroho
13511051