Aplikasi *String Matching* Pada Fitur *Auto-Correct* dan *Word-Suggestion*

Johan - 13514206
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514026@std.stei.itb.ac.id

Abstraksi — Fitur *Auto-Correct* dan *Word-suggestion* biasa kita jumpai dalam *smartphone,* ataupun aplikasi pemrosesan kata. Fitur ini sangat membantu user untuk mengantisipasi terjadinya kesalahan dalam pengetikan dan dapat mempersingkat waktu pengetikan sebuah kata. Hanya dengan mengetik beberapa huruf, maka akan keluar kemungkinan kata yang diinginkan. Pembuatan fitur ini menggunakan prinsip algoritma *string-matching.*

Makalah ini akan membahas mengenai fitur *auto-correct* dan *word-suggestion* serta bagaimana pencocokan tersebut dilakukan dengan menggunakan algoritma *Knuth-Morrits–Pratt* dan algoritma *Booyer-Moore.*

Keywords—Auto-correct; word-prediction; strng-matching; Knuth-Morris-Pratt; Booyer-Moore

I. PENDAHULUAN

Kemajuan teknologi pada masa sekarang sangatlah beragam. Semakin banyak teknologi yang dibuat untuk mempermudah dan mempercepat pekerjaan manusia. Salah satu nya adalah fitur *auto-correct* dan *word-suggestion* yng sering dijumpai pada *smartphone* dan *text editor*. fitur ini mendukung otomasi penulisan kata-kata. Seperti yang tergambar pada namanya, fitur *auto-correct* berguna untuk memperbaiki kata-kata yang dianggap salah oleh sistem dan memberikan *word-suggestion* kata yang benar dan mirip dengan kata yang dimasukkan pengguna.

Gambar 1 - contoh *auto-correct*

Sedangkan *word-suggestion* itu sendiri berisi prediksi kemungkinan kata-kata apa yang akan diketik oleh pengguna. Dengan mengetikkan beberapa huruf atau seluruh huruf, maka sistem akan memeriksa lalu akan mencari dan mencocokkan huruf-huruf tersebut dengan huruf-huruf suatu kata yang tersedia di dalam database. Jika setelah dicek ternyata di dalam *database* ada, maka akan muncul gambar yang menunjukan *list* dari *suggestion* yang ada.

Gambar 2 - contoh tampilan *word-prediction*

Untuk saat ini, kemampuan dari *auto-correct* dan *word-prediction* sudah ditingkatkan. S*martphone* sekarang pada umumnya sudah mampu menambahkan kosakata baru dengan sendirinya ke dalam *database* berdasarkan kebiasaan penulisan kata-kata pengguna. Dengan adanya ini, maka fungsionalitas *auto-correct* dan *word-prediction* akan bisa bekerja lebih cepat dan lebih baik, sehingga pengguna dapat lebih terbantu.

II. DASAR TEORI

A. *Pattern Matching*

Algoritma pencocokan string adalah algoritma untuk melakukan pencarian semua kemunculan *string* pendek yang biasa di seput ***pattern (P)*** di dalam suatu *string* yang lebih panjang yang disebut ***text (T).***

Contoh :

T : “INSTITUT TEKNOLOGI BANDUNG ITB”

P : “TEKNO”

String matching ini banyak digunakan dalam kehidupan. Misalnya saat pencarian di dalam *Text Editor*, *search engine*, analisis citra, dalam *bioinformatics*, dan dalam fitur *auto-correct* dan *word-prediction*. Algoritma s*tring matching* dapat diklasifikasikan menjadi 3 bagian menurut arah pencariannya.

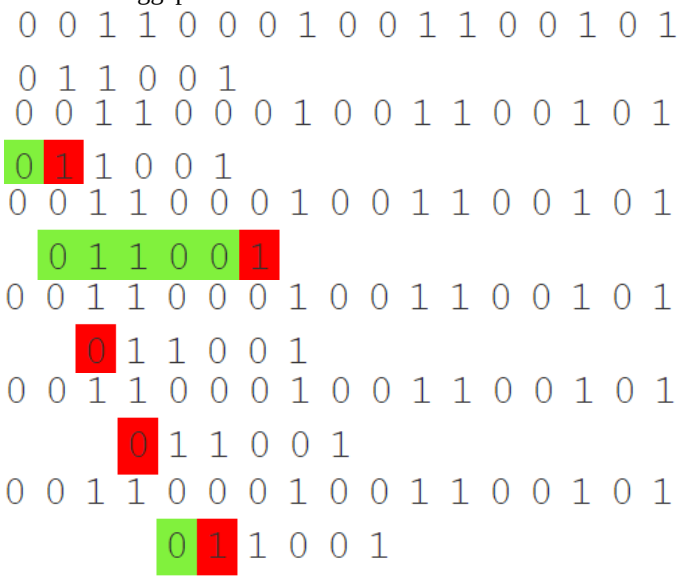
1. Dari kiri ke kanan. Yang termasuk dalam kategori ini adalah algoritma *brute force*, dan algoritma *Knuth-Morris-Pratt.*
2. Dari kanan ke kiri. Yang termasuk dalam kategori ini adalah algoritma dari *Boyer* dan *Moore* yang dikembangkan menjadi algoritma *turbo Boyer-Moore*, algoritma *tuned Boyer-Moore*, dan algoritma *Zhu-Takaoka.*
3. Dari arah yang ditentukan secara spesifik oleh algoritma tersebut. Secara teori, arah ini akan mendapatkan hasil yang terbaik. Yang termasuk dalam kategori ini adalah

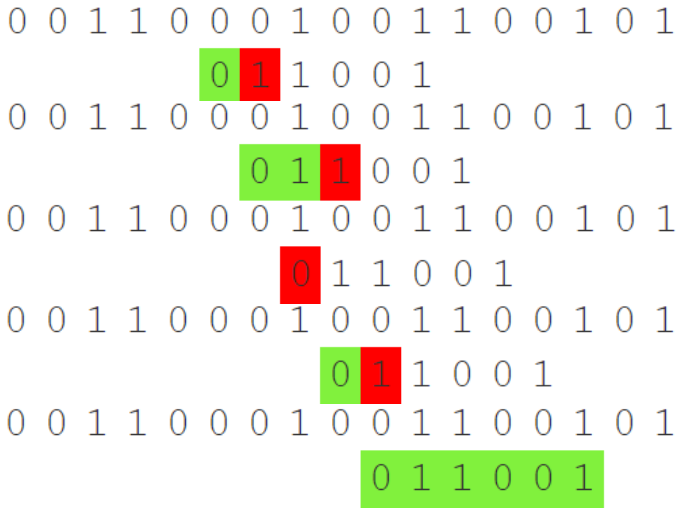
Untuk kali ini, akan dibahas algoritma *brute force, Knuth-Morris-Pratt dan Boyer-Moore.*

B. *Brute Force Algorithm*

Algoritma ini akan selalu menghasilkan solusi, tetapi belum tentu efektif dan optimal. Algoritma ini melakukan pencarian secara bertahap di seluruh rangkaian string. Cara kerjanya sebagai berikut

1. Dimulai dengan mencocokan *pattern* pada awal teks, dari kiri.
2. Dari kiri ke kanan, algoritma ini mecocokan karakter *pattern* satu per satu dengan karakter yang ada di teks.
3. Jika semua karakter pada *pattern* cocok, maka pattern ketemu dan pencocokan dihentikan.
4. Jika karakter di *pattern* dan di teks yang dibandingkan tidak cocok, pattern akan digeser 1 karakter ke kanan dan mengulangi langkah 2 sampai *pattern* berada pada ujung kanan teks.
5. Apabila sampai *string* sudah habis tetapi belum juga ditemukan *pattern* nya, maka masukan dari *user* dianggap tidak ditemukan.





Gambar 3 - mekanisme pencocokan string dengan *brute force*

Jumlah perbandingan yang dilakukan oleh algoritma *brute force* pada kasus terburuknya adalah m(n – m +1). Sehingga kompleksitasnya *O(mn).* Contohnya T: ‘aaaaaaaaaaaaaaah’ dan P: ‘aah’.

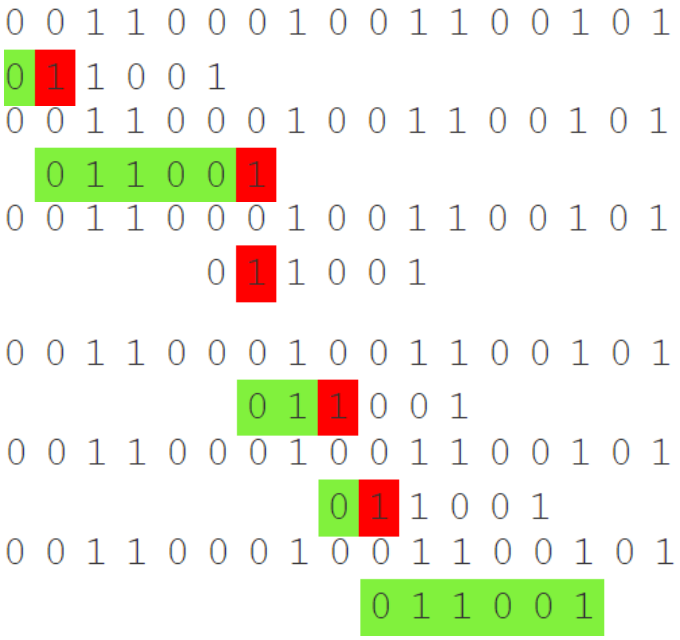
Sedangkan untuk kasus terbaiknya, kompleksitas nya adalah *O(n)* terjadi bila karakter pertama *pattern P* tidak pernah sama dengan kerakter teks T yang dicocokan. Contoh T:’abcde fghijk lmn ooo’ dan P: ‘ooo’.

Dan untuk kasus rata-rata algoritma ini adalah *O(m+n).* Contohnya P: ‘this string example is standard’ P: ‘store’.

C. *Knuth-Morris-Pratt (KMP) Algorithm*

Algoritma KMP ini mambaca *pattern* dari kiri ke kanan, sama seperti *brute force.* Tapi pada algoritma ini pergeseran dilakukan lebih pintar dibandingkan dengan *brute force*. Pergeseran dilakukan dengan mencari *prefix* terpanjang yang juga merupakan *suffix*. Lalu jika terjadi *mismatch* maka bisa langsung menggeser hingga ke *prefix*, tidak perlu satu per satu karakter. Sehingga dalam implementasinya, algoritma ini memerlukan fungsi pembatas *(border function)* yang digunakan untuk menghitung letak suatu ukuran karakter dimana perbandingan harus dilakukan. Secara sistematis, algoritma KMP bekerja seperti ini:

1. Dimulai dengan mencocokan *pattern* pada awal teks, dari kiri.
2. Dari kiri ke kanan, algoritma ini mecocokan karakter *pattern* satu per satu dengan karakter yang ada di teks.
3. Jika semua karakter pada *pattern* cocok, maka pattern ketemu dan pencocokan dihentikan.
4. Jika karakter di *pattern* dan di teks yang dibandingkan tidak cocok, pattern akan digeser berdasarkan *border function* lalu mengulangi langkah 2 sampai *pattern* berada pada ujung kanan teks.
5. Apabila sampai *string* sudah habis tetapi belum juga ditemukan *pattern* nya, maka masukan dari *user* dianggap tidak ditemukan.



Gambar 4 - mekanisme pencocokan dengan algoritma KMP

Sebelum menjalankan algoritma utama, KMP memlakukan preproses terhadap *pattern* untuk mencari kecocokan *prefix* dari *pattern* dengan *pattern* itu sendiri. Preproses ini dinamakan fungsi pinggiran *(border function)* atau biasa disebut *failure function. Border function* didefinisikan sebagai ukuran dari *prefix* terpanjang dari P[1..k] yang juga merupakan *suffix* dari P[1..k].

J = posisi *mismatch* dalam P[ ]

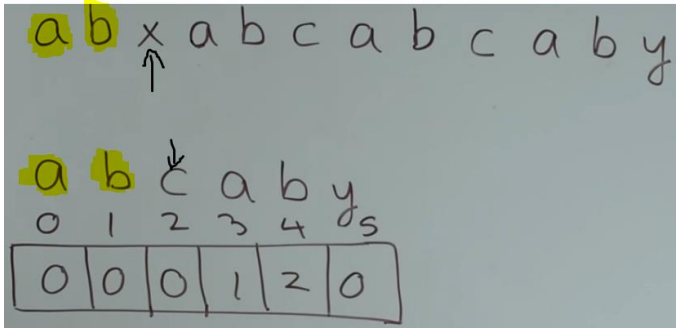
K = posisi sebelum terjadi *mismatch* (k = j-1)

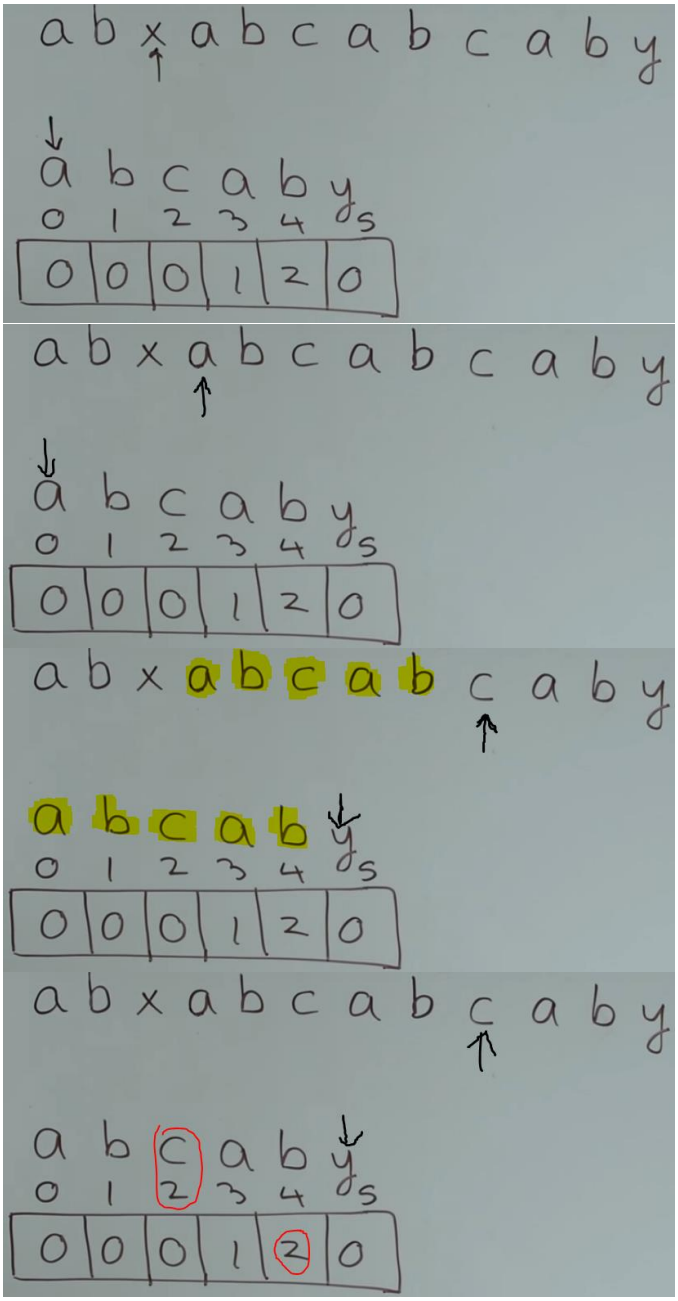
Contoh

P : "a b c a b y"
J : 0 1 2 3 4 5

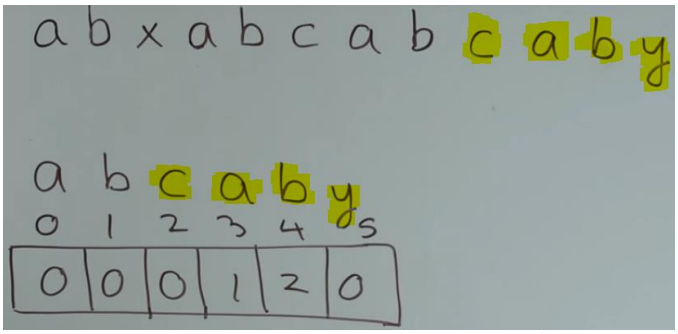
j	0	1	2	3	4	5
P[j]	a	b	c	a	b	y
b[j]	0	0	0	1	2	0

Sehingga saat terjadi *mismatch* di j tertentu, maka akan dilihat dari fungsi pinggirannya. Angka di fungsi pinggiran menandakan posisi mulainya pencocokan baru pada *pattern* di karakter selanjutnya pada text





lihat b(j-1), pencocokan pindah ke posisi tersebut



Gambar 5 - algoritma KMP memanfaatkan *border function*

Untuk komplesitas waktu pada algoritma KMP adalah *O(m+n).* Dengan *O(m)* adalah kompleksitas untuk menghitung *border function* dan *O(n)* untuk pencarian string. Algoritma ini akan efektif jika jenis alphabet yang dicari lebih sedikit seperti contoohnya binary text. Tetapi akan tidak efektif jika jenis alphabet nya sangat beragam.

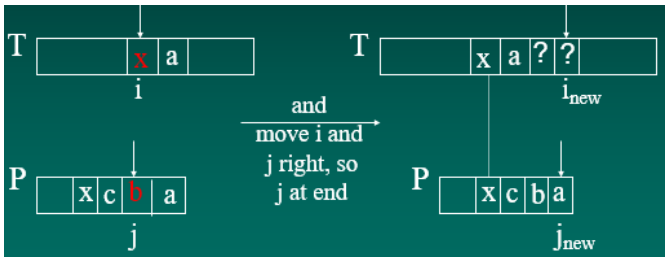
D. Boyer-Moore (BM) Algorithm

Algoritma ini melakukan pencocokan karakter dimulai dari paling kanan. Langkah-langkah yang dilakukan aloritma BM pada saat mecocokan string adalah sebagai berikut:

1. Dimulai dengan mencocokan *pattern* pada awal teks, dari kanan.
2. Dari kanan ke kiri, algoritma ini mecocokan karakter *pattern* satu per satu dengan karakter yang ada di teks.
3. Jika semua karakter pada *pattern* cocok, maka pattern ketemu dan pencocokan dihentikan.
4. Jika karakter di *pattern* dan di teks yang dibandingkan tidak cocok *(mismatch)*, pattern akan digeser dengan memaksimalkan nilai penggeseran *good-suffix* dan penggeseran *bad-character* lalu mengulangi langkah 2 sampai *pattern* berada di ujung text.
5. Apabila sampai *string* sudah habis tetapi belum juga ditemukan *pattern* nya, maka masukan dari *user* dianggap tidak ditemukan.

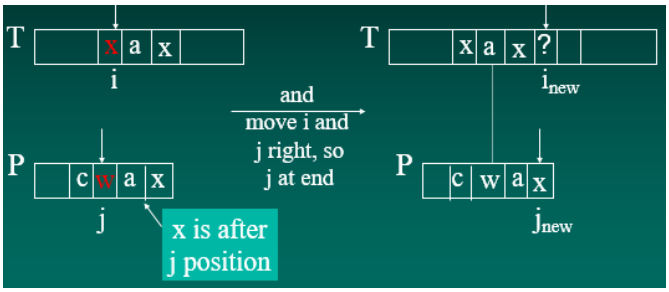
Peraturan *Bad-Character* adalah sebagai berikut. Jika terjadi *mismatch*, geser pattern ke kanan sampai terjadi kejadian pertama *mismatch* karakter di P. Peraturan *good-suffix* digunakan untuk mengetahui karakter yang cocok di *suffix pattern.* Terdapat 3 kasus yang mungkin terjadi dalam hal ini

1. Jika di P terdapat suatu karakter misalnya ‘x’, P di coba digeser ke kanan hingga ke kemunculan terakhir x di P dengan T[i]



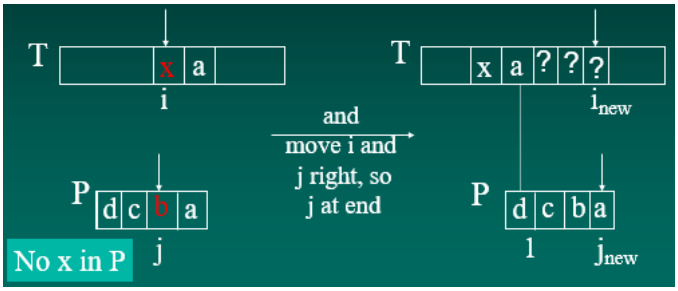
Gambar 6 - kasus 1 dalam algoritma BM

2. Jika P mengandung karakter ‘x’, tetapi saat digeser ke kanan hingga kemunculan terakhir tidak dimungkinkan, maka P di geser ke kanan 1 karakter ke T[i+1].

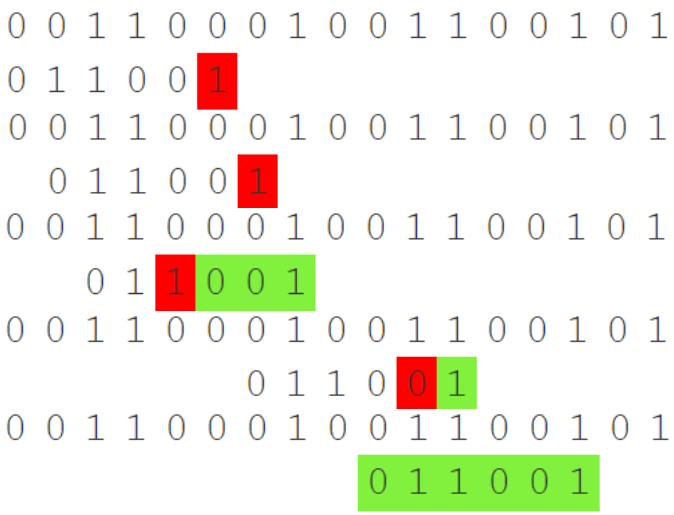


Gambar 7 - kasus 2 dalam algoritma BM

3. Jika kasus pertama dan kasus kedua tidak dapat dilakukan maka geser P hingga P[0] dengan T[i+1] sampai setelah karakter tersebut di T.



Gambar 8 - kasus 3 dalam algoritma BM



Gambar 9 - mekanisme pencocokan dengan algoritma BM

Cara menghitung *occurrence* adalah sebagai berikut
Contoh

P:	a	b	c	a	b	y
pergeseran	2	1	3	2	1	0

1. Lakukan iterasi mulai dari posisi paling kanan *pattern* sampai ke posisi awal. Dimulai dengan 0, catat karakter yang sudah ditemukan
2. Mundur ke posisi sebelumnya (kiri), dan niai pencacah di inkremen atau ditambah 1.

3. Jika karakter pada posisi ini belum pernah muncul sebelumnya, maka nilai pergeserannya sama dengan nilai pencacah.
4. Jika karakter pada posisi ini sudah pernah muncul sebelumnya, maka nilai pergeserannya adalah nilai pergeseran yang sama dengan nilai pergeseran karakter tersebut sebelumnya.
5. Ulangi langkah 2 hingga posisi awal *pattern.*

Kompleksitas waktu pada algoritma BM adalah *O(mn + A)* dengan A adalah jumlah alphabet yang ada. BM akan sangat efektif terhadap text yang memiliki jenis alphabet yang beragam dibandingkan dengan KMP

III. Analisis String Matching Pada Fitur Auto-Correct

Fungsi *auto-correction* adalah fungsi validasi data otomatis yang umum ditemukan pada pengolah kata dan pada text editing di *smartphone* atau tablet. Tujuan utamanya adalah untuk memperbaiki ejaan atau kesalahan umum dalam mengetik. Hal ini tentu akan menghemat waktu bagi pengguna debandingkan jika harus menghapus dan menulis ulang kembali kata.

Pada awalnya, kita memerlukan sebuah *database* yang berisi kata-kata penggantian. Nantinya, kata yang akan diketik oleh pengguna akan dicocokkan 1 per 1 karakter dengan huruf-huruf dari kata yang ada di dalam *database* ini. Pencocokan ini akan menggunakan algoritma *string matching* dengan kata-kata di dalam *database* berperan sebagai text, dan kata yang sedang diketik berperan sebagai *pattern.*

Input	Output
yuor	your
(r)	®
aboutit	about it
where;s	where's
bold	**bold**
italic	*italic*

Gambar 10 - contoh database pada auto correct

Dalam melakukan pengecekan, kita menggunakan algoritma *brute force*. Ini dimaksudkan agar ketika dilakukan pengecekan, kita mendapatkan hasil yang akurat dengan mengecek 1 per 1 karakter. Dengan begitu, jika terdapat suatu kerakter yang tidak sesuai dengan kata-kata yang ada di *database* maka akan langsung keluar *suggestion* dari kata-kata yang benar dan memungkinkan. Pengkoreksian dilakukan di tengah-tengah pengetikan, bukan di akhir. Sehingga kesalahan yang timbul ketika selesai mengetik akan dapat diminimalisir.

Dalam melakukan pengeluaran *word-prediction* kita menggunakan algoritma Boyer-Moore. Hal ini dilakukan karena mengingat algoritma ini melakukan pengecekan dari belakang belakang ke depan, dan memiliki *best case* yaitu jika jumlah alphabet nya beragam. Kebanyakan text editor atau semacamnya pasti digunakan untuk mengetik text yang memiliki jenis huruf yang beragam. Sehingga Algoritma ini cocok digunakan untuk mengeluarkan prediksi-prediksi kata. Bisa saja jika menggunakan algoritma KMP, hanya saja nantinya untuk mengeluarkan prediksi akan memakan waktu yang lebih lama dibanding dengan algoritma BM. Tetapi ada masalah baru yang muncul saat mengeluarkan prediksi kata. Kata yang seperti apa yang menjadi kriteria untuk dimunculkan sebagai prediksi? Dan bagaimana jika *suggestion words* tersebut tidak sesuai dengan yang *user* inginkan. Ini akan dapat memperhambat proses pengetikan kata jika langsung menekan *suggestion word* tadi.

Seiring berkembangnya teknologi, manusia semakin ingin membuat sesuatu yang lebih pintar yang dapat memudahkan pekerjaannya. Masalah-masalah tadi, adalah masalah memasukkan kata-kata baru ke dalam *database*. Tinggal menambahkan kata-kata baru yang sesuai dengan keinginan *user* maka, masalah tersebut teratasi. Tapi untuk memasukkan kata ke *database* cukup repot, maka seringkali *user* jadi malas menggunakan fitur ini. Sehingga muncul solusi yaitu memeriksa setiap kata yang ditulis, dan menghitung frekuensi kemunculan kata tersebut. Jika presentasi kemunculannya tidak tinggi, maka akan dimasukkan ke database sementara. Saat angkanya sudah menunjukan lebih dari batas, maka secara otomatis, kata tersebut akan dipindah ke database utama.

Lalu untuk masalah bagaimana menentukan kriteria kata yang akan menjadi prediksi, digunakanlah algoritma *dynamic programming* (tidak dibahas pada kali ini). Algoritma ini membagi masalah menjadi beberapa langkah. Pada kasus ini, langkah yang diambil adalah minimum jumlah karakter antara *pattern* dan string pada text. Pada setiap tahap, diambil *cost* minimal dengan tujuan ketika tahap terakhir diperoleh solusi optimal. Sehingga algoritma ini merupakan algoritma yang cocok untuk mencarii jumlah perbedaan karakter antara *pattern* yaitu kata yang sedang di ketik, dengan text yaitu kata yang ada dalam database. Kebanyakan fitur ini menggunakan jumlah perbedaan 2-3 huruf.

IV. Kesimpulan

Fitur *auto-correct* dan *word-sugggestion* memanfaatkan algoritma *brute force* dalam melakukan pengecekan terhadap huruf-huruf yang sedang diketik. Hal ini dikarenakan untuk mendapatkan hasil yang akurat.

Algoritma *Boyer-Moore* termasuk algoritma yang paling efektif terhadap text yang jenis alfabetnya beragam dibandingkan algoritma string matching lainnya. Sehingga algoritma ini cocok untuk membuat *word-prediction.*.

Algoritma *Dynamic programming* merupakan algoritma yang cocok untuk mencari jumlah perbedaan karakter antara *pattern* dengan text. Sehingga dynamic programming akan cocok dimanfaatkan untuk membuat *auto-correct.*

REFERENCE

[1] Munir, Rinaldi, "Pencocokan string (string matching)", Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2015

[2] https://en.wikipedia.org/wiki/Autocorrection (diakses tanggal 7 Mei 2016)

[3] Lecroq, Thierry Charras, Christian. 2001. Handbook of Exact String Matching Algorithm. ISBN 0-9543006-4-5.

[4] https://id.wikipedia.org/wiki/Algoritma_pencarian_string. (diakses tanggal 7 Mei 2016)

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, 8 Mei 2016

Johan - 13514026