Jurnal Riset Komputer (JURIKOM), Vol. 3 No. 4, Agustus 2016
ISSN 2407-389X (Media Cetak)
Hal : 16-21

PERANCANGAN APLIKASI TEXT EDITOR DENGAN MENERAPKAN ALGORITMA KNUTH-MORRIS-PRATT

Firman Matondang[1], Nelly Astuti Hasibuan[2], Imam Saputra[3], Suginam[4]

[1] Mahasiswa Teknik Informatika STMIK Budi Darma
[2,3,4] Dosen Tetap STMIK Budi Darma
[1,2,3,4] Jl. Sisingamangaraja No.338 Simpang Limun Medan

ABSTRAK

Text Editor adalah sebuah software aplikasi atau program komputer yang memungkinkan penggunanya membuat, mengubah, atau mengedit file teks. sebuah aplikasi Text Editor juga harus dilengkapi dengan fitur pencarian. Fitur pencarian pada aplikasi Text Editor merupakan fitur yang sangat penting karena fitur ini akan membantu pengguna untuk menemukan kata (string) yang dicari dengan mudah dan cepat Jika pencarian kata (string) pada aplikasi Text Editor dilakukan secara manual maka membutuhkan waktu yang relatif lama. Masalah utama dalam pencarian kata pada aplikasi Text Editor adalah semakin banyak data yang terdapat pada aplikasi Text Editor, maka semakin bertambah waktu untuk menemukan kata (string) tersebut. Oleh sebab itu dibutuhkan algoritma untuk menyelesaikan masalah pencarian kata (string) yang secara cepat dan akurat yang dapat memangkas waktu seminimal mungkin. Algoritma pencarian string merupakan salah satu bagian terpenting dalam berbagai proses yang berkaitan dengan data tipe teks Algoritma. Knutt-Morris-Pratt menyimpan sebuah informasi yang digunakan untuk melakukan jumlah pergeseran, sehingga algoritma ini melakukan pergeseran lebih jauh (tidak hanya bergeser satu karakter seperti dalam brute force). Dengan ini penggunaan algoritma Knutt-Morris-Pratt dapat mempersingkat waktu pencocokan string.

Kata Kunci: Pencarian, String, Text Editor, Algoritma Knutt-Morris-Pratt.

I. PENDAHULUAN

Saat ini telah tersedia banyak perangkat lunak yang berguna untuk membantu pengguna dalam membuat aplikasi komputer. *Text Editor* adalah sebuah *software* aplikasi atau program komputer yang memungkinkan penggunanya membuat, mengubah, atau mengedit file teks. Aplikasi *Text Editor* dapat digunakan untuk membuat program komputer, mengedit *source code* bahasa pemograman, serta membuat halaman *web*. Jika pencarian kata (*string*) pada aplikasi *Text Editor* dilakukan secara manual maka membutuhkan waktu yang relatif lama. Masalah utama dalam pencarian kata pada aplikasi *Text Editor* adalah semakin banyak data yang terdapat pada aplikasi *Text Editor*, maka semakin bertambah waktu untuk menemukan kata (*string*) tersebut. Oleh sebab itu dibutuhkan algoritma untuk menyelesaikan masalah pencarian kata (*string*) yang secara cepat dan akurat yang dapat memangkas waktu seminimal mungkin. Algoritma pencarian *string* merupakan salah satu bagian terpenting dalam berbagai proses yang berkaitan dengan data tipe teks.

Algoritma *Knuth-Morris-Pratt* salah satu algoritma pencarian *string*, dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966, namun keduanya mempublikasikannya secara bersamaan pada tahun 1977. Algoritma *Knuth-Morris-Pratt* merupakan jenis *Exact String Matching Algorithm* yang merupakan pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama. Langkah kerja Algoritma *Knuth-Morris-Pratt* yaitu melakukan proses awal pada *panttern P* dengan menghitung fungsi pinggiran. Tujuan dari fungsi pinggiran adalah untuk melakukan pergeseran *panttern* terjauh pada teks (*text*), kemudian melakukan perbandingan karakter di teks dan karakter di pola dari kiri ke kanan.

II. TEORITIS

A. Perancangan

Perancangan adalah langkah pertama dalam fase pengembangan rekayasa produk atau sistem. Perancangan itu adalah proses penerapan berbagai teknik dan prinsip yang bertujuan untuk mendefinisikan sebuah peralatan, satu proses atau satu sistem secara detail yang membolehkan dilakukan realisasi fisik (Pressman, 2001).

B. Text Editor

Sebuah *software* aplikasi atau program komputer yang memungkinkan penggunanya membuat, mengubah, atau mengedit file teks. *Text Editor* dapat digunakan untuk membuat program komputer, mengubah *source code* bahasa pemograman, serta membuat halaman *web* atau *template web design* (Sampurna, 1996 : 6).

C. String Matching (Pencocokan String/Kata)

Pencocokan pola merupakan teknik yang bisa digunakan untuk menentukan apakah suatu *string* sesuai dengan pola yang telah dispesifikasi sebelumnya. Pola dapat dibuat dengan menggunakan kombinasi karakter biasa. selama pencocokan pola karakter biasa harus benar-benar cocok dengan apa yang dispesifikasi pada *string* (Adi Nugroho, 2010).

D. Algoritma

Algoritma adalah urutan langkah-langkah untuk memecahkan suatu masalah. Terdapat macam-macam definisi algoritma, berikut ini merupakan beberapa definisi lain dari algoritma, antara lain :

1. Algoritma adalah deretan langkah-langkah komputasi yang *mentransformasikan* data masukan menjadi data keluaran.
2. Algoritma adalah deretan intruksi yang jelas untuk memecahkan masalah, yaitu untuk memperoleh keluaran yang diinginkan dari suatu masukan dalam jumlah waktu yang terbatas.

Prosedur komputasi yang terdefinisi dengan baik yang menggunakan beberapa nilai sebagai masukan dan menghasilkan beberapa nilai yang disebut keluaran. Jadi Algoritma adalah deretan langkah komputasi yang masukan menjadi keluaran (Rinaldi Munir, 2007).

E. Algoritma *Knuth-Morris-Pratt*

Algoritma *Knuth-Morris-Pratt* salah satu algoritma pencarian *string*, dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966, namun keduanya mempublikasikannya secara bersamaan pada tahun 1977. Algoritma *Knuth-Morris-Pratt* melakukan perbandingan karakter di teks dan karakter di pola dari kiri ke kanan seperti algoritma *Brute Force*. Hampir setiap perangkat lunak mengandung algoritma pencarian *string*. Contoh yang sederhana yaitu membaca masukan dari pengguna. Dalam bahasa pemrograman tingkat tinggi bisa menemukan fungsi atau *method* untuk mencocokkan Algoritma *string*.

Knuth-Morris-Pratt merupakan salah satu algoritma yang sering digunakan untuk menyelesaikan masalah pencocokan *string*. Algoritma ini adalah penyempurnaan dari algoritma pencocokan *string* dengan menggunakan algoritma *brute force*. Pada algoritma *brute force*, setiap kali ditemukan ketidak cocokan *pattern* dengan teks, maka *pattern* akan digeser satu ke kanan. Sedangkan pada algoritma *Knuth-Morris-Pratt* memelihara informasi yang digunakan untuk melakukan jumlah pergeseran. Algoritma menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karakter seperti pada algoritma *brute force*. Dengan algoritma *Knuth-Morris-Pratt* ini, waktu pencarian dapat dikurangi secara signifikan (Kukuh Nasrul Wicaksono, 2007).

Fungsi Pinggiran (*Border Function)*

Fungsi pinggiran dihitung hanya berdasarkan kepada karakter-karakter dalam *pattern*, tidak menyertakan karakter-karakter dalam teks (*string* target). Fungsi pinggiran *b(j)* didefinisikan sebagai ukuran awalan terpanjang dari *pattern P* yang merupakan akhiran dari *P[1..j]*. Untuk lebih jelasnya, berikut ini diberikan sebuah contoh untuk mengitung fungsi piggiran dari sebuah *pattern P* = xlnxls. Sebagai catatan, penulis menggunakan nilai 0 (nol) sebagai indeks awal *string* pada permasalahan ini.

Awalan dari *P* adalah □, x, xl, xln, xlnx, xlnxl
Akhiran dari *P* adalah □, s, ls, xls, nxls, lnxls
Keterangan : □ = *string* kosong

Nilai fungsi pinggiran *b(j)* untuk setiap karakter dalam *P* adalah :

Tabel 1 Fungsi Pinggiran

J	0	1	2	3	4	5
P[*j*]	x	l	n	x	l	s
b(*j*)	0	0	0	1	2	0

Secara sistematis, langkah-langkah yang dilakukan algoritma *Knuth-Morris-Pratt* pada saat mencocokkan *string* adalah sebagai berikut:.

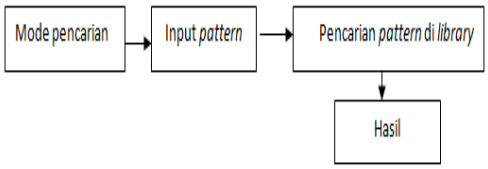
1. Algoritma Knuth-Morris-Pratt mulai mencocokkan pattern pada awal teks.
2. Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
   a. Karakter di pattern dan di teks yang dibandingkan tidak cocok *(mismatch)*.
   b. Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.

Algoritma kemudian menggeser pattern berdasarkan tabel, lalu mengulangi-langkah 3. sampai pattern berada diujung teks(Ana Ervana dan Asri Pertiwi, 2012).

III. ANALISA dan PEMBAHASAN

Tujuan dari proses pencarian adalah mencari sebuah elemen dari sebuah himpunan dengan suatu kunci (kemungkinan memuat informasi yang terkait dengan kunci). Adanya penggunaan proses *searching* pada aplikasi *Text Editor* adalah untuk kepentingan kecepatan pada saat melakukan proses pencarian teks. Fitur pencarian pada Aplikasi *Text Editor* akan melakukan pengecekan terhadap *pattern* atau kata yang ingin dicari, dimasukkan dalam *form* pencarian dan melakukan proses pencarian atau pencocokan pola pada *library* (himpunan pustaka teks). Proses pencarian akan terus berulang hingga ditemukan pantern yang cocok dengan yang ada pada *library* atau tidak ditemukan *pattern* (pola) yang cocok pada *library* hingga akhir teks. Kemudian memberitahukan hasil penemuan *pattern* (pola) tersebut.

Gambaran secara umum dari proses pencarian dapat dilihat pada gambar dibawah ini



Gambar 1 Proses Pencarian

Penulis akan menjelaskan lebih lanjut tentang bagaimana algoritma *Knuth-Morris-Pratt* bekerja pada aplikasi *Text Editor* yang akan dirancang, berdasarkan sistem aplikasi yang akan dibangun menggunakan algoritma *Knuth-Morris-Pratt*. Algoritma *Knuth-Morris-Pratt* memelihara informasi dan menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karakter.

Berikut ini merupakan penerapan pencarian *string* menggunakan algoritma *Knuth-Morris-Pratt :*

Contoh :
Teks : include('template/template.php');
Pattern : template.php

Langkah pertama pecahkan *string* teks dan *string pattern* menjadi *array* kemudian menghitung fungsi pinggiran untuk *pattern* tersebut. Dapat dilihat pada tabel 1 array teks dan tabel 2 fungsi pinggiran.

Tabel 2. Array Teks

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	i	n	c	l	u	d	e	(	‘	t	e	m	p	l	a	T	e	/	t	e	m

Lanjutan tabel 3 Array Teks

I	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
T	P	l	a	t	e	/	t	e	m	p	l	a	t	E	.	p	h	p	‘

Lanjutan tabel 3 Array Teks

I	40	41
T	)	;

Tabel 4 Fungsi Pinggiran

J	0	1	2	3	4	5	6	7	8	9	10	11
P[j]	t	e	M	p	l	a	t	e	.	p	H	p
B[j]	0	0	0	0	0	0	1	2	0	0	0	0

Tabel 5 Pencocokan *Pattern*

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	i	n	c	l	u	d	e	(	‘	t	e	m	P	l	a	t	e	/	t	e	m
P	t	e	m	p	l	a	t	e	.	p	h	p									

Langkah ke-dua, bandingkan sisi paling kiri *pattern* (*P)* dengan sisi paling kiri teks (*T)*. Dapat dilihat pada tabel 5 diatas, terjadi ketidak cocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter, dapat dilihat pada tabel 6 dibawah.

Tabel 6 Pencocokan *Pattern*2

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	i	n	c	l	u	d	e	(	‘	t	e	m	P	l	a	t	e	/	t	e	m
P		t	e	m	p	l	a	T	e	.	p	h	P								

Langkah ke-tiga, melakukan kembali perbandingan karakter dari *i*(1), yaitu bandingkan sisi kiri *pattern* (*P)* dengan sisi kiri teks (*T)* diawali dari *i*(1), terlihat pada tabel 6 diatas. Terjadi ketidak cocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter. terlihat pada tabel 7

Tabel 7 Pencocokan *Pattern*3

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	i	n	c	l	u	d	e	(	‘	t	e	m	p	l	a	t	e	/	t	e	m
P			t	e	m	p	l	a	t	e	.	p	h	p							

Langkah ke-empat, melakukan kembali perbandingan karakter dari *i*(2), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i*(2), terlihat pada tabel 7 diatas. Terjadi ketidak cocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter, terlihat pada tabel 8 dibawah.

Tabel 8 Pencocokan *Pattern*4

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	I	n	c	l	u	d	e	(	‘	t	e	m	p	l	a	t	e	/	t	e	m
P				t	e	m	p	l	a	t	e	.	p	h	p						

Langkah ke-lima, melakukan kembali perbandingan karakter dari *i*(3), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i*(3), terlihat pada tabel 8 diatas. Terjadi ketidak cocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter, terlihat pada tabel 9 dibawah..

Tabel 9 Pencocokan *Pattern*5

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	i	n	c	l	u	d	e	(	‘	t	e	m	p	l	a	t	e	/	t	e	m
P					t	e	m	p	l	a	t	e	.	p	h	p					

Langkah ke-enam, melakukan kembali perbandingan karakter dari *i*(4), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i*(4), terlihat pada tabel 9 diatas. Terjadi ketidak cocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter, terlihat pada tabel 10 dibawah.

Tabel 10 Pencocokan *Pattern*6

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	i	n	c	l	u	d	e	(	‘	t	e	m	p	l	a	T	e	/	t	e	m
P						t	e	m	p	l	a	t	e	.	p	H	p				

Langkah ke-tujuh, melakukan kembali perbandingan karakter dari *i*(5), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i*(5), terlihat pada tabel 10 diatas. Terjadi ketidak cocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter, terlihat pada tabel 11 dibawah.

Tabel 11 Pencocokan *Pattern*7

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	i	n	c	l	u	d	e	(	‘	t	e	m	p	l	a	T	e	/	t	e	m
P							t	e	m	p	l	a	t	e	.	P	h	p			

Langkah ke-delapan, melakukan kembali perbandingan karakter dari *i*(6), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i*(6), terlihat pada tabel 11 diatas. Terjadi ketidak cocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter, terlihat pada tabel 12 dibawah.

Tabel 12 Pencocokan *Pattern*8

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	i	n	c	l	u	d	e	(	‘	t	e	m	p	l	a	T	e	/	t	e	m
P								t	e	m	p	l	a	t	e	.	p	h	p		

Langkah ke-sembilan, melakukan kembali perbandingan karakter dari *i*(7), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i*(7), terlihat pada tabel 12 diatas. Terjadi ketidak cocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter, terlihat pada tabel 13 dibawah.

Tabel 13 Pencocokan *Pattern*9

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	i	n	c	l	u	d	E	(	‘	t	e	m	p	l	a	T	e	/	t	e	m
P									t	e	m	p	l	a	t	E	.	p	h	p	

Langkah ke-sepuluh, melakukan kembali perbandingan karakter dari *i*(8), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i*(8), terlihat pada tabel 13 diatas. Terjadi ketidak cocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter, terlihat pada tabel 14 dibawah.

Tabel 14 Pencocokan *Pattern*10

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	i	n	c	l	u	d	E	(	‘	t	e	m	p	l	a	t	e	/	t	e	m
P										t	e	m	p	l	a	t	e	.	p	h	p

Langkah ke-sebelas, melakukan kembali perbandingan karakter dari *i*(9), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i*(9), terlihat pada tabel 14 diatas. Karakter pada posisi *i(*9) sampai dengan *i(*16) memiliki pola yang sama, tetapi *pattern* tidak cocok dengan teks pada posisi i(17). Karena terjadi ketidak cocokan karakter, maka dilakukan pergeseran *pattern* ditentukan dari pinggiran awalan *P* yang bersesuaian, pada contoh diatas, awalan yang bersesuaian adalah ‘template’, dengan panjang *i*=8. Nilai pinggiran

terpanjang untuk *string P*[0..1] adalah *B*(1)=2. Maka jumlah pergeseran *I-B*=8-2=6. Jadi *pattern* digeser sebanyak 6 karakter ke kanan dihitung dari awal *pattern*, terlihat pada tabel 15 dibawah.

Tabel 15 Pencocokan *Pattern*11

Index	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
T	t	e	m	p	i	a	T	e	/	t	e	m	p	l	a	t	e	.	p	h	p
P							T	e	m	p	l	a	t	e	.	p	h	p			

Langkah ke-duabelas, melakukan kembali perbandingan karakter dari *i*(15), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i*(15), terlihat pada tabel 15 diatas. Karakter pada posisi *i(*15) sampai dengan *i(*16) memiliki pola yang sama, tetapi *pattern* tidak cocok dengan teks pada posisi *i*(17). Karena terjadi ketidak cocokan karakter, maka dilakukan pergeseran *pattern* ditentukan dari pinggiran awalan *P* yang bersesuaian, pada contoh diatas, awalan yang bersesuaian adalah template dengan panjang *i*=2. Nilai pinggiran terpanjang untuk *string P*[0..1] adalah *B*(1)=0. Maka jumlah pergeseran *I-B*=2-0=2. Jadi *pattern* digeser sebanyak 2 karakter ke kanan dihitung dari awal *pattern*, terlihat pada tabel 3.14 dibawah.

Tabel 16 Pencocokan *Pattern*12

Index	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
T	t	e	m	p	l	a	t	e	/	t	e	m	p	l	a	t	e	.	p	h	p
P									T	e	m	p	l	a	t	e	.	p	h	p	

Langkah ke-tigabelas, melakukan kembali perbandingan karakter dari *i*(17), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i*(17), dapat terlihat pada tabel 16 diatas. Terjadi ketidak cocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter, seperti terlihat pada tabel 17 dibawah

Tabel 17 Pencocokan *Pattern*13

Index	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
T	t	e	m	p	l	a	t	e	/	t	e	m	p	l	a	t	e	.	p	h	p
P										t	e	m	p	l	a	t	e	.	p	h	p

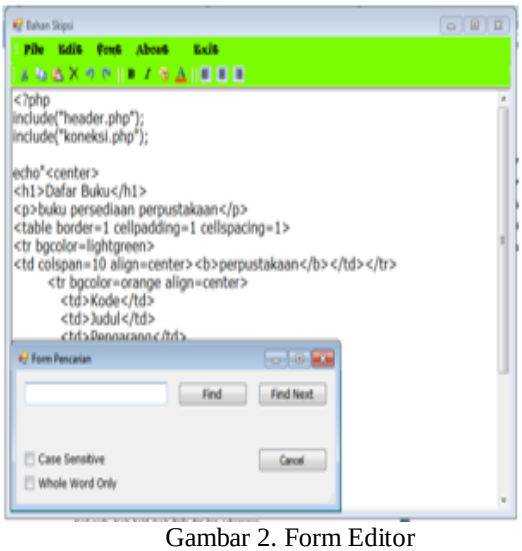
Langkah ke-empatbelas, selanjutnya melakukan kembali perbandingan karakter dari *i*(8), yaitu bandingkan ujung kiri *pattern* (*P)* dengan ujung kiri teks (*T)* diawali dari *i(8),* terlihat pada tabel 3.15 diatas. Ditemukan pola *Pattern* (*P*) yang cocok pada karakter teks.

V. IMPLEMENTASI

1. Tampilan Form Editor

Gambar 1. *Form* Editor

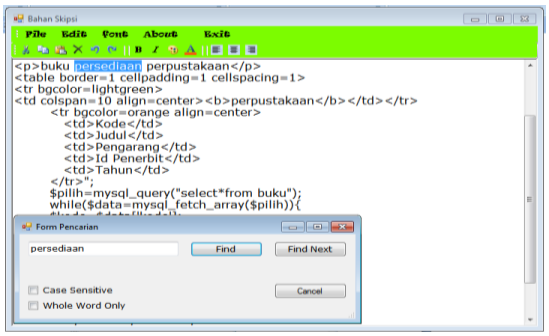
2. Form Pencarian.



Gambar 2. Form Editor

3. Hasil Pengujian Program

Adapun hasil dari pengujian program yang telah dibuat dan hasil pengujian kata atau teks yang dicari adalah sebagai berikut :



Gambar 4. Tampilan Hasil Pegujian

Adapun tahapan yang harus dilakukan untuk mencari kata yang dicari adalah sebagai berikut :

1. Tekan crt+f (pilih edit *find*) maka akan muncul *form* percarian
2. Ketikkan kata yang dicari pada form pencarian.
3. Kemudian klik tombol *find*, jika kata yang dicari ditemukan makan pada halaman *form* editor tulisan yang ditemukan akan ditampilkan.

Setelah dilakukan pengujian program seperti yang terlihat pada gambar 4.5 diatas, kata persediaan yang dicari dan diketikkan pada form pencarian kemudian pada editor akan ditampilkan kata yang ditemukan tersebut.

V. KESIMPULAN

Berdasarkan pembahasan dan evaluasi dari bab-bab sebelumnya serta teori yang ada, maka dapat diambil kesimpulan sebagai berikut:

1. Proses pencarian pada Aplikasi *Text Editor* akan melakukan pengecekan terhadap *pattern* atau kata yang ingin dicari, dimasukkan dalam *form* pencarian dan melakukan proses pencarian atau pencocokan pola pada *library* (himpunan pustaka teks).
2. Penerapan algoritma *Knutt-Morris-Pratt* pada aplikasi *Text Editor* ini dapat memangkas waktu pencarian kata yang dicari, algoritma ini melakukan pergeseran lebih jauh (tidak hanya bergeser satu karakter seperti dalam *brute force*). Dengan ini penggunaan algoritma *Knutt-Morris-Pratt* dapat mempersingkat waktu pencocokan *string*, serta dapat menyajikan kata yang ingin dicari dengan tepat dan akurat.
3. Aplikasi *Text Editor* dengan menerapkan algoritma *Knutt-Morris-Pratt* telah selesai dirancang dan dapat dijadikan salah satu alternatif *Text Editor*.

VI. DAFTAR PUSTAKA

[1] Abdul Kadir. (2013). "Pengertian Algoritma, Pendekatan Secara Visual dan Interaktif Menggunakan Raptor". Yogyakarta : ANDI.
[2] Adi nugroho. (2010). "Rekayasa Perangkat Lunak Berorientasi Objek Dengan Metode USDP (unified Software Development Process)". Yogyakarta : ANDI.
[3] Hafni Syaeful Sulun. (2007). "Penerapan Algoritma *Knuth-Morris-P*ratt pada Aplikasi Pencarian Berkas di Komputer".
[4] Hendayudi. (2009). "VB 2008 untuk Berbagai Keperluan Pemrograman". Jakarta : PT. Alex Media Komputindo.
[5] http://edukasi.kompasiana.com/2010/05/03/perancangan-sistem-132346.html tanggal akses : 06-Mei-2015.
[6] http://id.wikipedia.org/wiki/Algoritma_ pencarian_string, tanggal akses : 24 april 2015.
[7] Kukuh Nasrul Wicaksono. (2007). "Penerapan Algoritma Pencocokan String *Knuth-Morris-Pratt* Sebagai Algoritma Pencocokan DNA".
[8] Rinaldi Munir. (2007). "Algoritma dan Pemograman Dalam Bahasa Pascal dan C." Bandung : INFORMATIKA.
[9] F. T. Waruwu and Mesran, "IMPLEMENTASI ALGORITMA KNUTH MORRIS PRATT PADA APLIKASI KAMUS ISTILAH LATIN FLORA DAN FAUNA BERBASIS ANDROID," *Maj. Ilm. INTI*, vol. 4, no. 1, pp. 96–102, 2014.
[10] Mesran, "IMPLEMENTASI ALGORITMA BRUTE FORCE DALAM PENCARIAN DATA KATALOG BUKU PERPUSTAKAAN," *Maj. Ilm. INTI*, vol. 3, no. 1, pp. 100–104, 2014.