Tugas Besar III IF2211 Strategi Algoritma

Pencarian Berita pada News Aggregator dengan Algoritma Pencocokan String



Dipersiapkan oleh :

WTS > Solusi Tubes 3 Stima Minat PM

Adrian Mulyana Nugraha 13515075

Vincent Hendryanto Halim 13515089

Mikhael Artur Darmakesuma 13515099

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

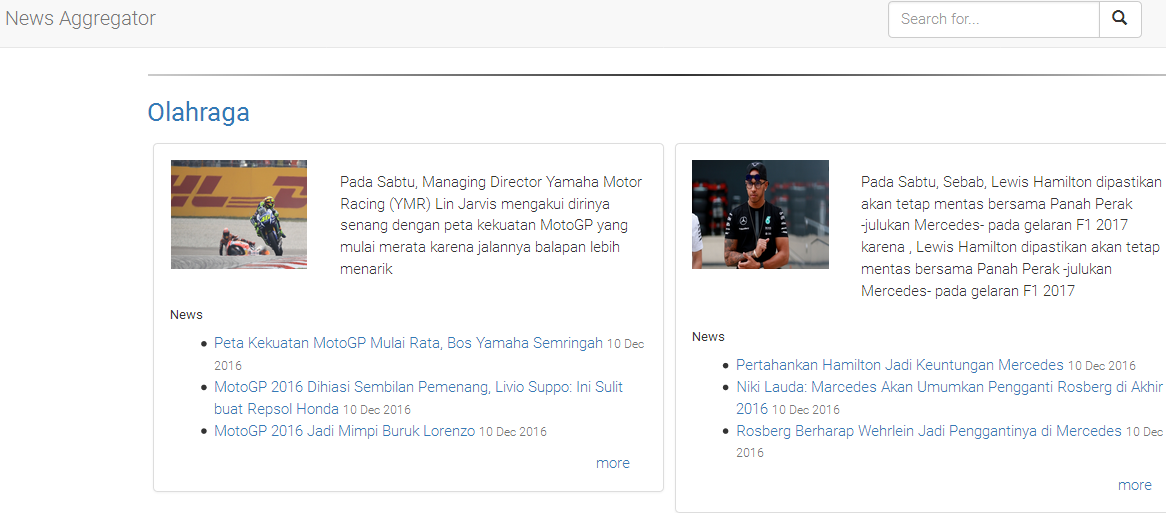
**2017**

# Deskripsi Masalah

Sistem agregasi berita (*news aggregator*) dikembangkan untuk membantu pembaca berita dengan mengumpulkan informasi berita dari berbagai sumber dan menyajikannya dalam satu tempat. Dengan sistem ini, pembaca tidak perlu mencari berita sendiri, dan aplikasi dapat mengambil berita sesuai kebutuhan dari pembaca (Lasica, 2003). Pencarian berita merupakan salah satu fitur yang terdapat pada sistem ini.



Gambar 1. News aggregator dari Google



Gambar 2. Prototipe News aggregator dari Lab GAIB ITB



Gambar 3. News aggregator dari situs www.newslookup.com

Algoritma pencocokan *string* (*pattern*) Knuth-Morris-Pratt (KMP) dan Algoritma Boyer-Moore merupakan algoritma yang lebih baik daripada *brute force*. Pada Tugas Besar III kali ini Anda diminta membuat aplikasi sederhana pencarian berita pada *news aggregator* dengan kedua algoritma tersebut, plus menggunakan *regular expression* (*regex*). Teks yang akan Anda proses adalah kumpulan berita berbahasa Indonesia. Pengguna aplikasi ini akan memberikan masukan berupa keyword pencarian, dan menghasilkan daftar berita yang diurut berdasarkan tanggal berita.

Pencocokan *string* yang anda buat adalah *exact matching* (untuk KMP dan BM) jadi artikel berita yang diproses mengandung string yang tepat sama dengan *keyword* yang dimasukkan oleh pengguna. Sedangkan bila menggunakan *regex* maka tidak selalu *exact matching*. Pencarian juga tidak bersifat *case sensitive*, jadi huruf besar dan huruf kecil dianggap sama (hal ini dapat dilakukan dengan mengganggap seluruh karakter di dalam *pattern* dan teks sebagai huruf kecil semua atau huruf kapital semua).

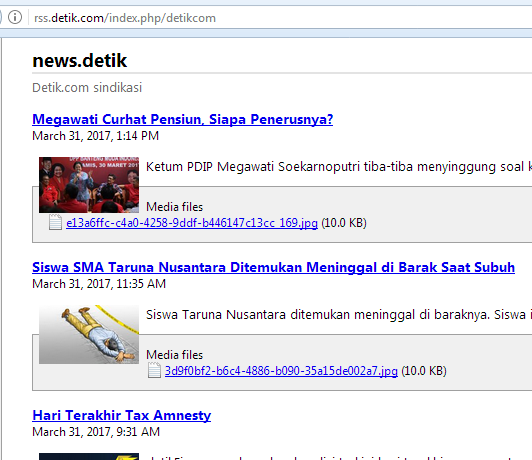
Kumpulan berita diambil secara otomatis menggunakan *crawler* berbasis *RSS* (*rich site summary* atau *really simple syndication*) dari situs berita daring berbahasa Indonesia. Saat membaca *RSS* dengan *XML parser*, informasi yang dibutuhkan berupa judul, tanggal berita, dan *URL* berita. Berikut daftar *RSS* yang dapat digunakan:

<http://rss.detik.com/index.php/detikcom>

<http://tempo.co/rss/terkini>

<http://rss.vivanews.com/get/all>

<http://www.antaranews.com/rss/terkini>



Gambar 4. Isi RSS detik.com jika dilihat dari browser

Untuk setiap *URL*, unduh artikelnya dan lakukan *parsing*. Artikel berupa file HTML dan tidak hanya mengandung konten berita, tetapi masih mengandung *header*, *footer*, iklan, dan tambahan informasi pada situs berita tersebut. Untuk itu, dilakukan *parsing* HTML untuk mendapatkan hanya teks konten berita dan foto yang terkait berita tersebut. Salah satu library *html* *parser* yang dapat digunakan misalnya adalah <https://jsoup.org/> untuk Java, *RSS parser* untuk .NET (<http://stackoverflow.com/questions/684507/rss-parser-in-net>) dan *XML parser* dalam Bahasa C# ( <http://stackoverflow.com/questions/642293/how-do-i-read-and-parse-an-xml-file-in-c> ),

# Dasar Teori

Yang menjadi dasar dari news aggregator adalah algoritma pencocokan *string*. Ada 3 macam algoritma yang kami gunakan, yakni Knuth-Morris-Pratt (KMP), Boyer-Moore, serta *Regular Expression* (*Regex*).

* 1. **Algoritma Knuth-Morris-Pratt**

Algoritma pencocokan *string* ini memulai pencarian dari kiri ke kanan (dari awal *string* hingga akhir). Sepintas algoritma ini menyerupai algoritma *brute force*, tetapi ada perbedaan dalam metode penelusuran *string* tersebut yang menyebabkan algoritma ini jadi lebih efisien.

Algoritma ini akan memulai mencocokkan *pattern* terhadap sebuah teks. Bila terjadi ketidakcocokan pada saat *pattern* sejajar dengan tekst e k s [ i . . i + n − 1 ] {\displaystyle teks[i..i+n-1]} , kita bisa menganggap ketidakcocokan yang pertama terjadi di antara teks t e k s [ i + j ] {\displaystyle teks[i+j]} dan pattern*,* dimana nilai 0 < j < n {\displaystyle 0<j<n} . Berarti, seluruh teks dan *pattern* dari awal hingga sebelum urutan huruf ke- adalah samat e k s [ i . . i + j − 1 ] = p a t t e r n [ 0.. j − 1 ] {\displaystyle teks[i..i+j-1]=pattern[0..j-1]} b = p a t t e r n [ j ] {\displaystyle b=pattern[j]} . Kemudian akan dilakukan penggeseran, dengan syarat ada sebuah awalan v {\displaystyle v} dari *pattern* yang sama dengan sebagian akhiran u {\displaystyle u} dari sebagian teks, sehingga kita bisa menggeser *pattern* agar awalan v {\displaystyle v} tersebut sejajar dengan akhiran dari bagian teks tersebut.u {\displaystyle u} Pencocokan string akan berjalan secara efisien bila kita mempunyai tabel yang menentukan berapa panjang kita seharusnya menggeser seandainya terdeteksi ketidakcocokkan di karakter ke- j {\displaystyle j} dari *pattern*. Tabel itu harus memuat posisi karakter pattern p a t t e r n [ j ] {\displaystyle pattern[j]} setelah digeser, sehingga kita bisa menggeser pattern sebesar (*j* - posisi setelah digeser) j − n e x t [ j ] {\displaystyle j-next[j]} relatif terhadap teks.

Jika ditemukan potongan dari teks yang cocok dengan pattern, maka algoritma akan mengembalikan nilai posisi dimana kecocokan tersebut terjadi. Algoritma ini akan diulang hingga penelusuran teks mencapai akhir dari teks. Dengan panjang teks , dan panjang *pattern*, kompleksitas dari algoritma ini untuk kasus terburuk adalah O(*m+n*).

* 1. **Algoritma Boyer-Moore**

Algoritma pencocokan *string* ini terbalik dengan algoritma KMP di atas, karena algoritma ini memulai penelusuran *pattern* dari kanan ke kiri (dari akhir teks ke awal).

Algoritma ini akan memulai mencocokkan *pattern* dari kanan ke kiri terhadap teks. Bila terjadi ketidakcocokan pada saat *pattern* sejajar dengan tekst e k s [ i . . i + n − 1 ] {\displaystyle teks[i..i+n-1]} , akan dilakukan penggeseran *bad-character* yang menyejajarkan teks dengan kemunculan karakter tersebut di paling kanan pada *pattern*. Bila ada bagian dari teks yang cocok dengan *pattern*, maka algoritma akan mengembalikan posisi dalam teks dimana kecocokan tersebut terjadi. Algoritma ini akan diulang hingga penelusuran teks mencapai akhir dari teks.

Dengan panjang teks , dan panjang *pattern*, kompleksitas dari algoritma ini untuk kasus terburuk adalah O(*mn + A*), dimana A merupakan ukuran ruang alfabet. Algoritma ini lebih efektif jika ukuran A lebih besar, seperti pada teks narasi.

* 1. ***Regular Expression***

*Regular expression* atau *Regex* merupakan kumpulan sintaks / karakter spesial yang mendefinisikan pola pencarian dalam sebuah teks. *Regex* menerapkan *Finite State Machine* (FSM), yakni komputasi yang menerima sebuah *string* sebagai masukan dan keluaran berupa menerima/menolak input tersebut. FSM tersebut akan melakukan pencocokan dari awal teks, jika tidak cocok dengan *pattern* dan aturan *regex* yang digunakan, maka pencarian akan bergeser 1 karakter ke kanan. Pencarian akan terus dilakukan hingga akhir dari *pattern* sejajar dengan akhir dari teks. Berbagai sintaks yang umumnya digunakan dalam *regex* adalah sebagai berikut:

|  |  |  |  |
| --- | --- | --- | --- |
| Sintaks | Arti | Contoh Penggunaan | Contoh pattern |
| [ . ] | Memilih sekelompok karakter | [gt] | good times |
| [ . - .] | Memilih karakter dari range tertentu | /[A - Z]/ | Rest in Peace |
| ^ | Negasi | /[^r]/ | grrrr |
| ? | Memberikan variasi terhadap ekspresi sebelumnya | Dance?s | Dance, Dances |
| . | Memilih karakter apapun | /th.s/ | this, thus, th’s |

# Analisis Pemecahan Masalah

* 1. **Strategi Umum**

Untuk melakukan pattern matching pada file yang sudah diunduh, kami menggunakan tiga algoritma yang sudah dijelaskan di bagian sebelumnya. Ketiga algoritma tersebut memiliki perbedaan pada karakteristik pergeseran pattern. Algoritma-algoritma tersebut diterapkan dengan pattern yang didapat dari input pengguna dan text yang didapat dari hasil *parsing* html file.

Pertama-tama, program menerima input pattern dari pengguna. Kemudian program akan membuka *file* xml untuk mengakses data berita yang urlnya terdapat pada file xml tersebut. Selama parsing file xml, selain mengambil url kami juga mengambil tanggal terbit berita dan judul berita tersebut.

Dari url yang didapat tadi, program akan mengakses html file kemudian mengambil isi dari html file tersebut. Namun, dalam kasus url yang tersedia tidak dapat diakses maka html file tersebut tidak akan diambil isinya. Dari judul dan konten yang sudah didapat, diterapkan algoritma sesuai pilihan pengguna. Kemudian program akan menampilkan hasil pencarian dari pattern yang diberikan pengguna.

* 1. **Solusi Akhir**

Solusi yang didapat dari pencarian dengan menggunakan program news crawler ini adalah tampilan berupa berita yang mengandung pattern yang diberikan pengguna beserta url yang akan mengirim pengguna ke situs asli dari berita tersebut.

* 1. **Struktur Data**

Untuk mempermudah penyimpanan, kami menggunakan struktur data berupa kelas bernama News yang memiliki data member title, url, date dan content yang secara berurutan berguna untuk menyimpan judul, url, tanggal terbit dan konten dari berita yang bersangkutan.

# Implementasi dan Pengujian

* 1. **Spesifikasi Teknis**

public class SearchController : Controller

/\*\*

\* Mencari string pada newsList dengan urutan judul baru kemudian isi.

\*

\* @param newsList list berita dari database

\* @param searchQuery search pattern yang digunakan

\* @return List berita yang lolos search dengan lokasi ditemukannya

\*/

public ActionResult Index(string searchQuery, int searchType)

/\*\*

\* Mencari string pada newsList dengan algoritma KMP judul baru kemudian isi.

\*

\* @param newsList list berita dari database

\* @param searchQuery search pattern yang digunakan

\* @return List berita yang lolos search dengan lokasi ditemukannya

\*/

private List<NewsFound> searchKMP(News[] newsList, string searchQuery)

/\*\*

\* Mencari string pada newsList dengan algoritma BM judul baru kemudian isi.

\*

\* @param newsList list berita dari database

\* @param searchQuery search pattern yang digunakan

\* @return List berita yang lolos search dengan lokasi ditemukannya

\*/

private List<NewsFound> searchBM(News[] newsList, string searchQuery)

/\*\*

\* Mencari string pada newsList dengan regex judul baru kemudian isi.

\*

\* @param newsList list berita dari database

\* @param searchQuery search pattern yang digunakan

\* @return List berita yang lolos search dengan lokasi ditemukannya

\*/

private List<NewsFound> searchRegex(News[] newsList, string searchQuery)

/\*\*

\* Menampilkan newsList

\*

\* @param newsList list berita dari database

\* @return List semua berita

\*/

private List<NewsFound> showAll(News[] newsList)

public class Searcher

/\*\*

\* Mencari string pada text yang cocok dengan pattern dengan algoritma KMP

\*

\* @param text text yang dicari

\* @param pattern search pattern yang digunakan

\* @return indeks pertama text cocok dengan pattern

\*/

public static int KMPSearchFirst(string text, string pattern)

/\*\*

\* Mencari string pada text yang cocok dengan pattern dengan algoritma KMP

\*

\* @param text text yang dicari

\* @param pattern search pattern yang digunakan

\* @return kumpulan indeks pertama text yang cocok dengan pattern

\*/

public static int[] KMPSearchAll(string text, string pattern)

/\*\*

\* Mencari string pada text yang cocok dengan pattern dengan algoritma Boyer-

\* Moore

\*

\* @param text text yang dicari

\* @param pattern search pattern yang digunakan

\* @return indeks pertama text cocok dengan pattern

\*/

public static int BMSearchFirst(string text, string pattern)

/\*\*

\* Mencari string pada text yang cocok dengan pattern dengan algoritma Boyer-

\* Moore

\*

\* @param text text yang dicari

\* @param pattern search pattern yang digunakan

\* @return kumpulan indeks pertama text yang cocok dengan pattern

\*/

public static int[] BMSearchAll(string text, string pattern)

/\*\*

\* Mencari string pada text yang cocok dengan pattern dengan algoritma Regex

\*

\* @param text text yang dicari

\* @param pattern search pattern yang digunakan

\* @return indeks pertama text cocok dengan pattern

\*/

public static int RegexSearchFirst(string text, string pattern)

/\*\*

\* Mencari string pada text yang cocok dengan pattern dengan algoritma Regex

\*

\* @param text text yang akan dicari badCharnya

\* @param size ukuran text

\* @param badChar hasil dari prosedur

\*/

private static void BadCharHeuristic(string text, int size, ref int[] badChar)

/\*\*

\* Membantu pergeseran dalam KMP

\*

\* @param pattern search pattern KMP

\* @return list pergeseran yang perlu dilakukan

\*/

private static int[] computeFail(string pattern)

/\*\*

\* Konversi regex

\*

\* @param text regex

\* @return hasil konversi regex

\*/

public static string regexConvert(string text)

public class Loader

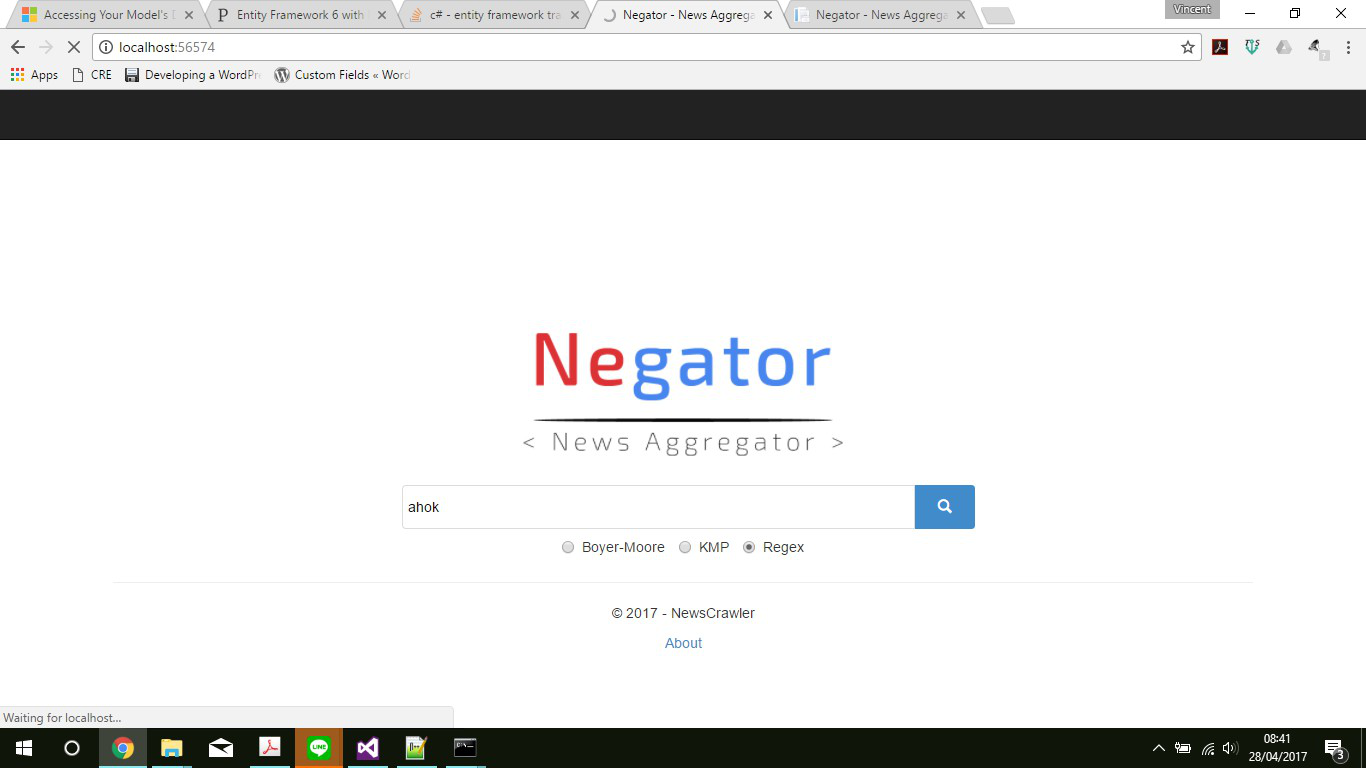
/\*\*

\* Melakukan loadRSS

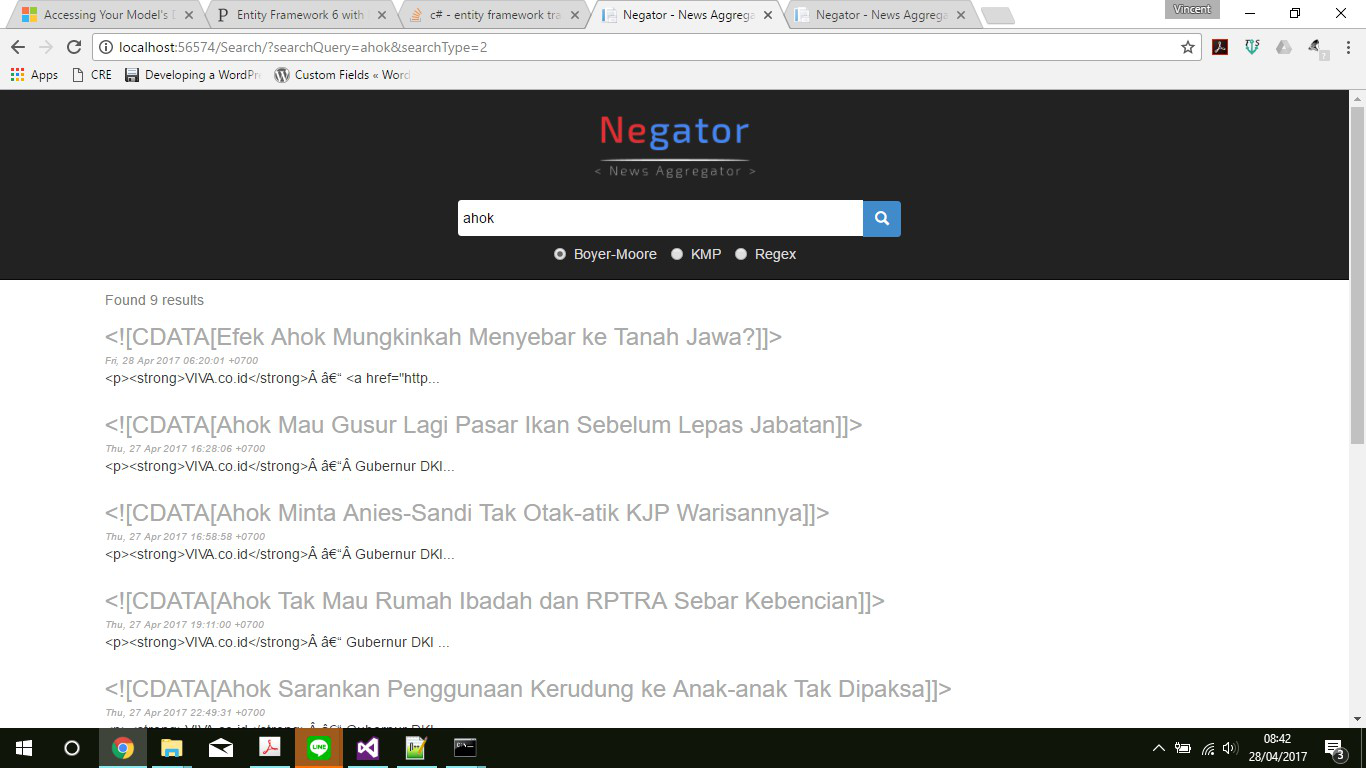
\*/

public static void loadRSS(string url, NewsCrawlerDB db, MySqlConnection connection)

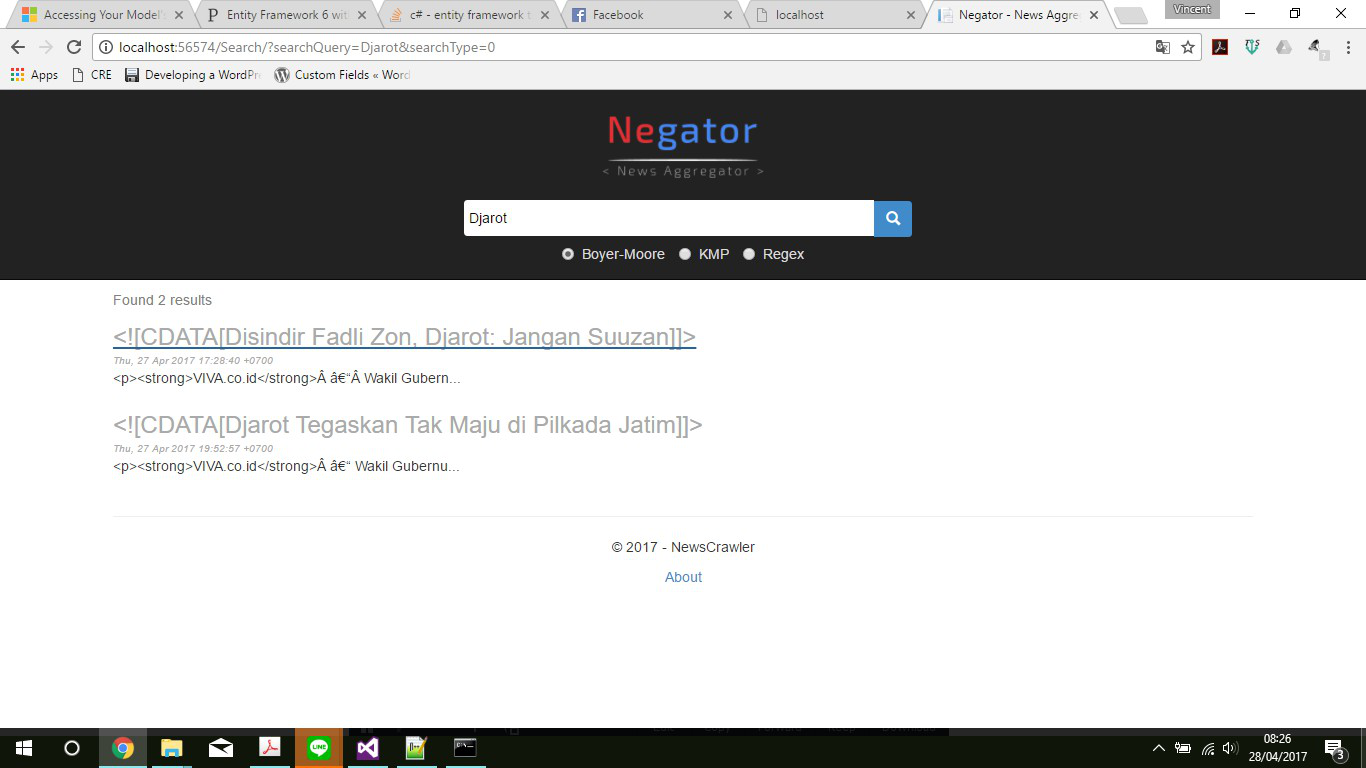
* 1. **Hasil Pengujian**



Gambar 5. Interface utama pencarian



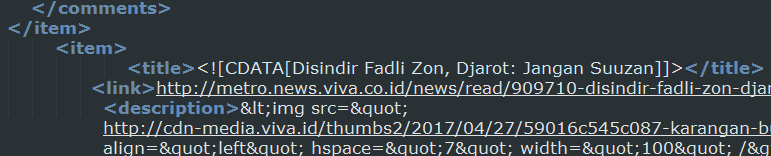
Gambar 5. Contoh pencarian



Gambar 6. Contoh pencarian

* 1. **Analisis**

Pencarian sudah dapat dilakukan dan menghasilkan hasil dari beberapa situs rss yang disediakan. Namun untuk situs viva.co.id masih mengeluarkan judul yang memiliki format. Hal ini disebabkan oleh file xml dari situs tersebut memiliki format demikian.



Gambar 7. File xml dari viva.co.id

Hal tersebut menyebabkan parser kami masih mengambil bagian lain selain judul yang diinginkan. Hal yang sama terjadi pada tampilan konten *preview* dari situs yang masih memiliki *formatting* seperti <p> dan lain lain.

# Kesimpulan dan Saran

* 1. **Kesimpulan**

Algoritma Knuth-Morris-Pratt(KMP) dan Boyer-Moore memiliki kompleksitas yang hampir sama, hanya saja memiliki kelemahan yang berbeda satu sama lain. Algoritma KMP jauh lebih baik daripada Boyer-Moore untuk kasus pattern yang pendek. Sebaliknya, untuk pattern yang panjang algoritma Boyer-Moore lebih baik daripada KMP. Sementara regex memiliki kelebihan daripada KMP dan Boyer-Moore dalam hal fleksibilitas sehingga dapat memperoleh hasil lebih banyak.

* 1. **Saran**

File html dari situs berbeda seringkali menggunakan gaya yang berbeda sehingga sulit untuk membuat algoritma yang fleksibel untuk semua situs. Pengambilan rss dari situs yang bergaya sama akan mempermudah pembuatan html parser sehingga tidak perlu memikirkan gaya-gaya penulisan html tiap situs. Cara lain yang dapat dilakukan adalah menyimpan gaya penulisan html tiap situs beserta pembeda dari rss masing-masing situs.

RSS yang dimiliki juga perlu ditambah agar berita yang tersedia menjadi lebih banyak namun harus ditunjang dengan penulisan kode untuk memungkinkan gaya penulisan baru dari RSS tersebut.

# Referensi

<https://msdn.microsoft.com/en-us/library/system.xml.xmldocument(v=vs.110).aspx>

<http://htmlagilitypack.codeplex.com/>

<https://msdn.microsoft.com/en-us/library/system.text.regularexpressions(v=vs.110).aspx>

Munir, R. (2009). *Diktat Kuliah IF2211 Strategi Algoritma.* Bandung: Teknik Informatika ITB.