Mesin Pencari untuk Website Pribadi

Menggunakan Algoritma *Breadth-First Search*, Konsep *Greedy*, dan *Regular Expression*

Turfa Auliarachman / 13515133
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515133@std.stei.itb.ac.id

***Abstrak*—Salah satu permasalahan dalam membuat website adalah membuat sistem pencarian konten dalam website tersebut. Makalah ini memuat alternatif implementasi dari sebuah mesin pencari untuk website pribadi berdasarkan algoritma b*readth-first search*, konsep *greedy*, dan *regular expression*.**

Kata Kunci— BFS; greedy; indeks; mesin pencari; spider

I. Pendahuluan

Salah satu permasalahan yang akan timbul setelah membuat sebuah website adalah membuat sistem pencarian dalam website tersebut. Masalah ini akan muncul terutama pada website-website yang berbasis konten.

Jika website tersebut dibuat menggunakan *Content Management System* (CMS) dinamis yang sudah tersedia, mungkin CMS tersebut sudah menyediakan fitur pencarian. Tetapi jika website dibuat menggunakan CMS hasil pengembangan sendiri, menggunakan *generator* halaman statis, atau bahkan langsung dibuat dalam bentuk halaman statis, maka pencarian konten akan menimbulkan permasalahan tersendiri.

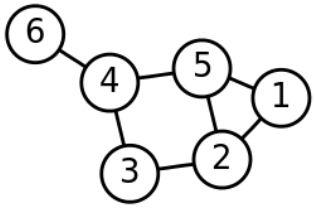
Alternatif solusi yang bisa digunakan adalah menggunakan *Google Custom Search* (GSC). Yang menjadi masalah dari GSC adalah indeks Google terupdate dalam waktu yang tidak sebentar. Harus menunggu beberapa waktu agar suatu halaman baru dapat terindeks oleh Google.

Dengan masalah tersebut, jika ingin menggunakan mesin pencari yang mengindeks sebuah halaman baru pada waktu sesuai arahan user, mesin pencari tersebut harus dikembangkan sendiri.

II. Dasar Teori

A. *Graf*

Graf adalah kumpulan simpul dan sisi yang menghubungkan simpul-simpul tersebut. Secara formal, graf adalah pasangan himpunan (V, E), di mana V adalah himpunan dari simpul dan E adalah himpunan dari sisi yang menghubungkan simpul-simpul tersebut. E adalah *multiset*, jadi bisa saja ada elemen yang muncul lebih dari sekali. [1]



Bagan 1: Contoh Graf. Sumber: https://upload.wikimedia.org/wikipedia/commons/thumb/5/5b/6n-graf.svg/250px-6n-graf.svg.png

Dalam graf, sebuah jalan adalah rangkaian simpul dan sisi sebuah graf yang saling berselang-seling antara simpul dan sisi, yang diawali dan diakhir oleh sebuah simpul. Sebuah jalan dikatakan sebagai jejak jika setiap sisi maksimal dikunjungi sebanyak *sekali*. Sebuah jejak dikatakan lintasan jika setiap simpul maksimal dikunjungi sebanyak sekali, kecuali simpul awal dan simpul akhir, jika keduanya sama. Kasus tersebut dinamakan sebagai lintasan tertutup atau sirkuit. [1]

Dua simpul *u* dan *v* dikatakan terhubung jika ada jalan yang berawal dari *u* dan berakhir di *v*. Jika *u* dan *v* terhubung serta *v* dan *w* terhubung, maka *u* dan *w* pasti terhubung. Sebuah graf dikatakan terhubung jika semua pasangan simpulnya terhubung. [1]

Graf bisa berarah dan tidak berarah. Graf berarah maksudnya setiap sisi memiliki arah dan hanya menyambungkan dua simpul dalam satu arah. [1]

Graf juga bisa berbobot dan tidak berbobot. Dalam graf berbobot, setiap simpul atau setiap sisi memiliki bobotnya masing-masing yang tidak harus unik. [1]

Ada beberapa graf yang dianggap graf berproperti khusus. Salah satunya adalah graf sederhana. Graf sederhana adalah graf yang tidak berarah, tidak berbobot, tidak memiliki sisi yang menghubungkan simpul yang sama, dan tidak memiliki sisi yang muncul lebih dari sekali. Graf sederhana bisa terhubung maupun tidak. [2]

B. Graph Traversal

Pada dasarnya, g*raph traversal* adalah proses pencarian pada suatu graf untuk mencari simpul mana saja yang dapat dicapai dari sebuah kumpulan akar. [3] Perhatikan bahwa di sini, simpul akar bisa lebih dari satu.

Ada banyak kegunaan dari *graph traversal*. Selain untuk pencarian, *graph traversal* juga bisa digunakan untuk mencari strategi terbaik pada beberapa permainan, baik permainan dengan satu pemain maupun dua pemain. Selain itu, *graph traversal* juga bisa digunakan untuk mencari rute optimal, memverifikasi apakah sebuah graf berarah memiliki *cycle* atau tidak, dan melakukan *topological sorting* pada sebuah graf. [3]

Dibagi berdasarkan urutan simpul yang dikunjungi, *graph traversal* dibagi menjadi dua algoritma, yaitu algoritma *depth-first search* (DFS) dan *breadht-first search* (BFS). Seperti namanya, algoritma DFS akan mengunjungi simpul-simpul pada suatu lintasan sampai simpul yang paling dalam, lalu cek lintasan yang berikutnya. Pada algoritma BFS, simpul yang paling dekat dengan akar akan dikunjungi terlebih dahulu. [3]

C. *Algoritma* Breadth-First Search *(BFS)*

Algoritma BFS adalah salah satu algoritma untuk melakukan *graph traversal*. Algoritma ini akan megunjungi simpul-simpul pada graf tidak berbobot terurut sesuai jaraknya terhadap akar. [3] Dengan kata lain, algoritma BFS akan mengunjungi simpul-simpul pada graf tersebut sesuai urutan jarak penemuan simpulnya.

Algoritma BFS dapat diimplementasikan menggunakan struktur data antrian. [3] Setiap menemukan simpul baru yang belum dikunjungi (ketika sedang memproses sebuah simpul), simpul tersebut dimasukkan ke dalam antrian. Simpul diproses menurut antrian.

Salah satu kelebihan algoritma BFS dibandingkan dengan algoritma DFS adalah algoritma BFS dapat melihat jarak terpendek dari akar ke simpul tersebut. [3]

D. Greedy

Meskipun sering disebut algoritma *greedy*, namun sejatinya *greedy* adalah sebuah teknik pengambilan keputusan. Seperti namanya (*greedy:* serakah, KBBI), sebuah algoritma *greedy* akan selalu mengambil pilihan terbaik pada suatu waktu, meskipun pada akhirnya mungkin saja pilihan tersebut bukan pilihan terbaik sesungguhnya. Algoritma *greedy* tidak akan kembali ke tahap sebelumnya untuk memperbaiki pilihan. Dalam bahasa yang lebih formal, algoritma *greedy* akan mengusahakan mendapatkan hasil optimum lokal dengan harapan pilihan tersebut akan mengarahkana kepada optimum global. [4]

Jika dilihat dari penjelasan di atas, untuk mengimplementasikan algoritma *greedy*, pada dasarnya dibutuhkan dua komponen. Pertama adalah daftar pilihan yang dapat dipilih pada suatu saat. Kedua adalah fungsi yang digunakan untuk mengevaluasi daftar pilihan tersebut, untuk kemudian hasil fungsi tersebut dibandingkan untuk menentukan pilihan mana yang kemudian akan dipilih. [4]

Yang paling sulit dari menggunakan algoritma *greedy* adalah membuktikan bahwa strategi *greedy* yang digunakan memang akan menghasilkan optimum global. [4]

E. Regular Expression

Regular Expression adalah teks string spesial yang mewakili sebuah *pattern* pencarian pada teks. [5]

Berbeda dengan algoritma pencarian string seperti *Knuth-Morris-Patt* ataupun *Boyer-Moore* yang mencari string pada teks yang benar-benar sama dengan pola pencarian, pada *regular expression*, pola pencarian tersebut memang berbentuk pola. Contohnya,

$$\backslash b[A-Z0-9.\_\%+-]+@[A-Z0-9.-]+\backslash.[A-Z]\{2,6\}\backslash b$$

akan *match* dengan semua alamat email yang ada pada teks. Hal ini bisa dimanfaatkan untuk melakukan validasi apakah alamat email yang dimasukkan pada sbuah formulir memang merupakan sebuah alamat email atau bukan. [5]

Berikut beberapa pola *regular expression* yang paling banyak digunakan.

Character	Meaning
\	general escape character with several uses
^	assert start of string (or line, in multiline mode)
$	assert end of string (or line, in multiline mode)
.	match any character except newline (by default)
[	start character class definition
|	start of alternative branch
(	start subpattern
)	end subpattern
?	extends the meaning of (, or 0/1 quantifier, or quantifier minimizer
*	0 or more quantifier
+	1 or more quantifier, also "possessive quantifier"
{	start min/max quantifier

Tabel 1: Pola Regular Expression *di Luar Kurung Siku. Sumber http://kb.4d.com/assetid=77249*

Character	Meaning
\	general escape character
^	negate the class, but only if the first character
-	indicates character range
[	POSIX character class (only if followed by POSIX syntax)
]	terminates the character class

Tabel 2: Pola Regular Expression *di Dalam Kurung Siku. Sumber http://kb.4d.com/assetid=77249*

Escape	Meaning
\a	alarm, that is, the BEL character (hex 07)
\cx	"control-x", where x is any character
\e	escape (hex 1B)
\f	formfeed (hex 0C)
\n	newline (hex 0A)
\r	carriage return (hex 0D)
\t	tab (hex 09)
\ddd	character with octal code ddd, or backreference
\xhh	character with hex code hh
\x{hhh..}	character with hex code hhh..

Tabel 3: Karakter Non-Printing *pada* Regular Expression. *Sumber http://kb.4d.com/assetid=77249*

F. Mesin Pencari

Mesin pencari adalah sebuah program yang berguna untuk mencari dokumen yang berisi kata kunci tertentu. Mesin pencari ini akan mengembalikan daftar dokumen yang mengandung kata kunci tersebut. Salah satu kelompok mesin pencari adalah mesin pencari web, yaitu mesin pencari dengan domain dokumen halaman-halaman yang tersedia di internet. [6]

Pada umumnya, mesin pencari web memiliki sebuah robot yang dinamakan spider. Spider ini akan mengunduh sebanyak-banyaknya halaman web yang dapat diakses. Sebuah program yang bernama pengindeks akan memproses halaman-halaman web tersebut dan membuat indeksnya berdasarkan kata-kata yang terdapat pada setiap halaman. Setiap mesin pencari memiliki algoritma khusus bagimana mencari dan memilih dokumen yang relevan berdasarkan kata kunci pencarian dan indeks yang tersedia. [6]

G. Hashing

Hashing adalah proses mengubah sesuatu menjadi sesuatu lain dengan harapan memudahkan dan mengoptimalkan proses selanjutnya. Biasanya hasil dari *hashing* berbentuk bilangan bulat, karena operasi pada bilangan bulat sangat mudah dan cepat. [7]

Idealnya, fungsi *hash* harus memiliki sifat injektif. Artinya, setiap nilai hasil pemetaan unik terhadap nilai sebelum pemetaan. Namun, karena keterbatasan dari algoritma *hashing* itu sendiri, tidak jarang fungsi *hash* tidak memiliki sifat injektif. Artinya, ada dua nilai awal berbeda yang dipetakan ke nilai *hash* yang sama. Fenomena ini dinamakan *collision*. [7]

Untuk meminimalisasi kemungkinan terjadinya *collision*, dapat digunakan dua algoritma *hashing* yang berbeda. Jika dua nilai awal memiliki hasil *hash* di salah satu algoritma sama, tetapi di algoritma *hashing* lain berbeda, maka dapat dipastikan nilai awal tersebut berbeda. Dengan implementasi seperti itu, memang masih ada kemungkinan terjadinya *collision*. Tetapi kemungkinan itu jauh lebih kecil dibandingan sebelum menggunakan dua algoritma *hashing*.

III. Rancangan Mesin Pencari

Mesin pencari yang akan dibuat hanya akan mengindeks halaman dari satu website saja. Definisi dari satu website adalah halaman yang memiliki alamat dengan prefiks yang sama, yang ditentukan oleh pengguna. Prefiks ini akan dinamakan *root*.

Akan dibuat tiga program terpisah, yaitu spider dan pengindeks, pemroses pencarian, serta antarmuka pencarian. Spider dan pengindeks digabungkan karena tidak akan dibuat indeks yang rumit dalam pembuatan mesin pencari ini. Indeks dibuat tidak terlalu rumit karena halaman web yang akan diproses tidak akan terlalu banyak, mengingat mesin pencari hanya akan mengindeks halaman web dari satu website. Selain itu, algoritma pencocokan string yang digunakan tidak membutuhkan struktur indeks yang rumit. Supaya lebih mudah, program spider dan pengindeks ini dinamakan Spindeks.

A. Indeks

Hal yang paling penting dari indeks adalah alamat halaman web yang dituju indeks tersebut dan isi dari halaman web. Alamat yang disimpan akan dimulai dari domain. Protokol yang digunakan tidak akan disimpan karena diasumsikan bahwa halaman web dengan alamat yang sama, walaupun menggunakan protokol berbeda (http/https), akan memiliki isi yang sama.

Ketika program spider dan pengindeks bekerja, program akan melakukan banyak pencarian pada indeks berdasarkan alamat dari halaman web. Hal ini dilakukan untuk memastikan tidak ada duplikasi data pada indeks. Maksudnya, satu halaman web cukup dibuat satu indeks. Pencarian tersebut akan dilakukan berkali-kali pada alamat halaman web yang sama. Padahal, pencarian berdasarkan string memakan waktu yang tidak sebentar. Oleh karena itu, akan disimpan juga *hash* dari alamat halaman web yang disimpan dalam bentuk bilangan bulat. Pencarian berdasarkan bilangan bulat memakan waktu yang lebih cepat daripada pencarian berdasarkan string.

Salah satu resiko dari penggunaan *hash* adalah terjadi *collision* sehingga dua alamat yang berbeda disimpan dengan *hash* yang sama. Oleh karena itu, akan digunakan dua buah *hash* dengan pemangkat dan pengali yang berbeda. Dengan digunakannya dua buah *hash* seperti ini, diharapkan *collision* tidak terjadi.

Supaya pencarian lebih cepat lagi, akan dibuat indeks pada basis data berdasarkan kedua *hash* ini. Perhatikan istilah "indeks" pada basis data merujuk pada hal yang berbeda dari istilah "indeks" pada mesin pencari pada website pribadi yang sedang dirancang.

B. Spindeks

Spider dan pengindeks adalah komponen utama dari sebuah mesin pencari. Pada pembuatan mesin pencari kali ini, spider dan program spider dan pengindeks akan digabungkan. Seperti yang sudah dijelaskan, program spider dan pengindeks digabungkan karena tidak akan dibuat indeks yang rumit dalam pembuatan mesin pencari ini. Agar memudahkan pemanggilan, rogram gabungan spider dan pengindeks ini dinamakan Spindeks.

Seperti yang sudah dibahas di bagian sebelumnya, indeks dibuat tidak terlalu rumit karena halaman web yang akan diproses tidak akan terlalu banyak, mengingat mesin pencari hanya akan mengindeks halaman web dari satu website. Selain itu, algoritma pencocokan string yang digunakan tidak membutuhkan struktur indeks yang rumit.

Program Spindeks akan memproses halaman web satu per satu. Artinya, dalam sekali pemrosesan, hanya satu halaman web yang informasinya diekstrak.

Setiap melakukan pemrosesan halaman web, Spindeks akan mengambil semua teks yang diprediksi sebagai konten halaman tersebut. Teks yang diprediksi sebagai konten halaman adalah teks yang diapit oleh tagar *<h1></h1>*, *<h2></h2>*, *<h3></h3>*, atau * *. Ini dilakukan dengan mengasumsikan bahwa halaman web dibuat dengan mengikuti standar yang ada, karena tagar *<h1></h1>*, *<h2></h2>*, *dan <h3></h3>* digunakan untuk *header* sebuah konten dan * * digunakan untuk membuat sebuah paragraf. Spindeks mengasumsikan bahwa konten sebuah halaman web adalah kumpulan dari *header-header* dan paragraf-paragraf.

Spindeks lalu akan melakukan *hashing* terhadap alamat halaman web tersebut. Kemudian informasi *hash* ini, bersama dengan alamat halaman web dan teks halaman web, disimpan sebagai indeks halaman tersebut. Jika sebelumnya sudah ada indeks dengan alamat tersebut (memiliki *hash* yang sama), isi teks indeks yang lama akan diperbarui dengan isi teks yang baru saja didapatkan.

Setelah selesai melakukan pengindeksan, Spindeks mencari alamat-alamat halaman web lain yang belum pernah ditemukan pada proses Spindeks kali ini dalam halaman web tersebut. Alamat dari halaman web baru yang ditemukan disimpan untuk kemudian diproses oleh Spindeks pada gilirannya, sesuai mekanisme antrian.

Sebagai catatan, halaman web yang akan dimasukkan ke antrian hanya halaman web dalam satu website. Root alamat dari website yang tercakup dalam mesin pencari ini diberikan kepada program Spindeks sebelum proses *crawling* dan pengindeksan dilakukan.

C. *Pemroses Pencarian*

Program pemroses pencarian akan dapat diakses menggunakan *request* HTTP. Akan disediakan *Restful* API agar komunikasi antar program pemroses pencarian dan program penggunanya terstandardisasi dengan standar yang sudah umum digunakan.

Dalam mesin pencari yang sedang dirancang, tidak digunakan algoritma yang rumit untuk mencari dokumen mana saja yang cocok dengan apa yang pengguna cari.

Program ini akan menerima input berupa string yang merupakan kata kunci pencarian. Kata kunci akan dikonversi menjadi sebuah pola *regular expression*, sedemikian sehingga program nantinya akan mencari halaman yang memuat kata kunci, dan jika ada beberapa kata dalam kata kunci tersebut, kata-kata itu boleh terpisah asalkan urutannya sama.

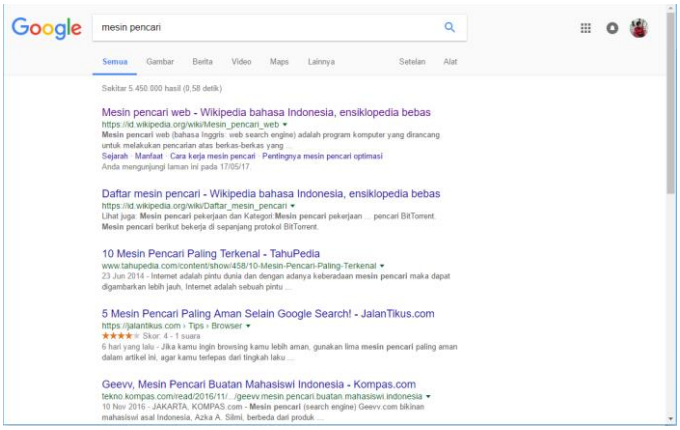
Program lalu mencari pada indeks yang sudah disimpan sebelumnya, pada halaman web mana saja pola *regular expression* hasil konversi dari kata kunci *match* dengan isi teksnya. Halaman-halaman web tersebut diberikan kepada pengguna.

Urutan halaman web yang dikembalikan diurutkan berdasarkan relevansi dengan kata kunci yang diberikan. Program menganggap semakin mirip string yang *match* dengan kata kunci, maka semakin relevan lah halaman web tersebut. Dari penjelasan halaman web mana saja yang dikembalikan, dapat disimpulkan bahwa semakin pendek string yang *match* dengan pola *regular expression*, maka relevansi halaman web tersebut semakin tinggi.

D. *Antarmuka Pencarian*

Antarmuka pencarian tentu akan berupa aplikasi website. Akan ada sebuah kotak teks yang dipasang di halaman website bersangkutan. Jika ada pengguna yang melakukan pencarian, akan ditampilkan halaman-halaman web yang sesuai dengan paginasi.

Desain antarmuka hasil pencariannya sendiri tidak perlu terlalu muluk-muluk, cukup sederhana saja. Desain antarmuka pencarian dari mesin pencari ini akan mengadopsi desain dari mesin pencari yang sudah ada, yaitu Google.



Bagan 2: Desain Antarmuka Google

IV. IMPLEMENTASI MESIN PENCARI

Setelah selesai melakukan perancangan, masalah selanjutnya adalah bagaimana detail implementasi dari rancangan tersebut.

Yang dibahas pada bab ini bukan tentang *tools* apa yang digunakan dalam pembuatan mesin pencari ini. Yang dibahas adalah bagaimana merepresentasikan komponen-komponen yang sudah dibahas pada bab *III. Rancangan Mesin Pencari* dalam bentuk teknis.

A. *Indeks*

Dibutuhkan sebuah basis data ntuk menyimpan semua indeks yang dihasilkan dari mesin pencari ini. Mengingat struktur dari indeks sudah diketahui, sistem yang cocok adalah sistem basis data relasional yang berdasarkan *Structured Query Language* (SQL), karena akan banyak dilakukan pencarian, pembaruan, penambahan, dan interasi *record* dari basis data indeks.

Dengan asumsi panjang alamat halaman web tidak akan lebih dari 256 karakter dan besar teks halaman web tidak akan lebih dari 16 MiB, maka akan dibuat struktur data indeks yang memiliki struktur seperti berikut.

```
struct indeks {
    int hash_1;
    int hash_2;
    char url[256];
    string teks;
};
```

Bagan 3: Struktur Data Indeks

Selain itu, untuk penyimpanan dalam basis data, akan dibuat tabel indeks pada basis data dengan struktur sebagai berikut.

```
CREATE TABLE ‘indeks’ (
    hash_1 INT NOT NULL,
    hash_2 INT NOT NULL,
    url VARCHAR(256) NOT NULL,
    teks TEXT,
    CONSTRAINT hash PRIMARY KEY (hash_1,
hash_2));
```

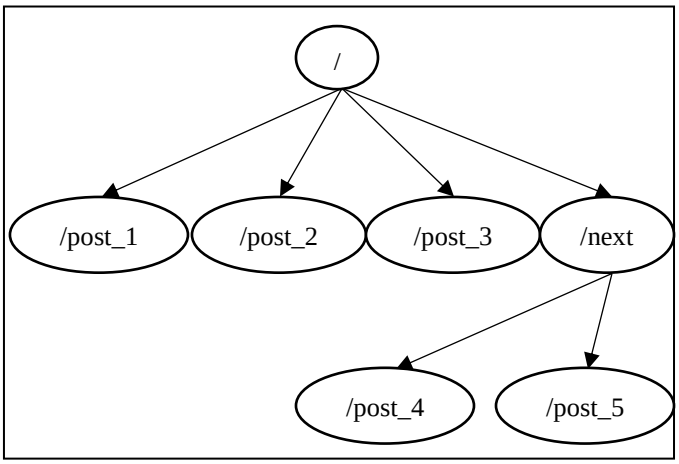
Bagan 4: Struktur Tabel Indeks

B. Spindeks 1 – Spider

Program Spindeks sejatinya memiliki dua unsur, yaitu spider dan pengindeks. Pada mesin pencari yang sedang dibuat, spider berfungsi untuk menjelajahi semua halaman yang ada pada website.

Untuk menjalankan tugasnya sebagai spider, Spindeks akan menganggap sebuah website sebagai sebuah graf berarah yang tidak berbobot. Dari sana, Spindeks akan menganggap setiap halaman web sebagai simpul. Jika ada tautan dari satu halaman web ke halaman web lain, dibuat sebuah sisi berarah dari halaman awal ke halaman tujuan tautan. Spindeks akan menggunakan algoritma *Breadth-First Search* (BFS) untuk menjelajahi semua halaman web yang terdapat pada website yang sedang diproses.

Sebagai contoh, anggap sebuah blog menampilkan tiga tulisan per halaman. Jika ada lima tulisan pada blog tersebut, maka grafnya berbentuk seperti berikut.



Bagan 5: Representasi Website dalam Indeks

Untuk mengimplementasikan algoritma BFS, dibutuhkan juga struktur data antrian.

C. Spindeks 2 – Pengindeks

Bagian kedua dari program Spindeks adalah pengindeks. Setiap sebuah halaman diproses, ada dua kemungkinan operasi pada indeks:

1. Jika belum ada indeks dengan alamat yang sama dengan halaman tersebut, buat indeks baru.
2. Jika sudah ada indeks dengan alamat yang sama dengan halaman tersebut, perbarui teks indeks tersebut.

Spindeks lalu melakukan *hashing* dua kali terhadap alamat halaman web. Algoritma *hashing* yang digunakan adalah sebagai berikut.

```
int hash(string alamat, int pengali, int
modulo) {
    long long hasil_hash = 0;

    for (char c : alamat) {
        hasil_hash *= pengali;
        hasil_hash += c;
        hasil_hash %= modulo;
    }

    return hasil_hash;
}
```

Bagan 6: Algoritma Hashing

Dari dua kali melakukan *hashing*, akan digunakan pengali dan modulo yang berbeda. Kedua bilangan ini akan dipilih dari bilangan prima.

Karena tidak ada batasan untuk alamat halaman web, diasumsikan bahwa alamat untuk halaman web dapat menggunakan seluruh karakter ASCII yang berjumlah 256 karakter. Oleh karena itu, bilangan pengali harus lebih dari atau sama dengan 256. Jika tidak, kemungkinan *collision* akan lebih besar.

Untuk bilangan modulo, akan digunakan bilangan prima yang besar, untuk memperkecil kemungkinan *collision*.

Untuk *hashing* pertama, akan digunakan bilangan pengali 257 dan modulo 100.000.007. Sedangkan untuk *hashing* kedua, akan digunakan bilangan pengali 353 dan modulo 1.000.000.009.

Perhatikan bahwa variabel lokal *hasil_hash* memiliki tipe data *long long* sedangkan keluaran fungsi memiliki tipe data *int*. Hal ini dikarenakan ketika melakukan pengalian, variabel *hasil_hash* bisa saja melebihi batas atas tipe data *int*. Tetapi setelah dilakukan modulo, *hasil_hash* akan tetap berada pada *range* tipe data *int.*

D. Pemroses Pencarian

Dalam program pemroses pencarian, semua indeks akan terlebih dahulu disimpan di memori atau di penyimpanan lokal. Hal ini bertujuan untuk melakukan optimasi kecepatan, karena pembacaan dari basis data lebih lama dari pembacaan berkas di penyimpanan lokal, apalagi di memori. Program baru akan memperbarui indeks lokal ketika diberitahu oleh Spindeks bahwa indeks di basis data sudah diperbarui.

Seperti sudah dijelaskan sebelumnya, program pemroses pencarian akan melakukan konversi kata kunci input dari pengguna menjadi sebuah *regular expression*. Untuk mencapai tujuan yang sudah dijelaskan pada bab *III. Rancangan Mesin Pencari*, program cukup mengubah setiap spasi yang ada pada kata kunci menjadi ".*". Artinya, boleh ada karakter apapun di antara setiap kata yang ada pada kata kunci.

Program akan mengecek kecocokan kata kunci pada teks yang ada di setiap indeks menggunakan *regular expression* tersebut.

Yang harus diperhatikan juga adalah daftar halaman web yang dikembalikan program harus terurut berdasarkan relevansinya, dengan definisi relevansi sudah dijelaskan pada bab *III. Rancangan Mesin Pencari*. Ada dua alternatif yang dapat dilakukan untuk mencapai tujuan ini.

1. Melakukan pengurutan di akhir ketika daftar halaman web sudah tersedia. Agar tidak melakukan pencocokan *regular expression* setiap melakukan perbandingan, panjang string yang *match* juga harus disimpan.
2. Me-*maintain* daftar halaman web yang sudah terurut selama program berjalan. Hal ini dapat dilakukan menggunakan *priority queue*. Tingkat prioritas sebuah halaman web ditulis sebagai negatif dari panjang string yang *match*. Hal ini dikarenakan semakin panjang stringnya, prioritasnya semakin kecil.

Tidak ada perbedaan yang berarti dari kedua alternatif tersebut, karena kedua alternatif tersebut akan sama-sama bekerja dalam kompleksitas *O(n log n)*. Dalam program yang dibuat, digunakan alternatif kedua.

E. Antarmuka Pencarian

Tidak ada hal khusus yang diperlukan untuk implementasi antarmuka pencarian.

Antarmuka ini akan menggunakan *Restful API* yang disediakan oleh program pemroses pencarian. Setelah mendapatkan daftar halaman web dari program pemroses pencarian, program antarmuka pencarian melakukan paginasi secara *offline*. Hal ini dikarenakan kemungkinan halaman web yang didapat tidak terlalu banyak, mengingat hanya melakukan pengindeksan dalam satu website.

PERSANTUNAN

Penulis menghaturkan banyak terima kasih kepada Ibu Masayu Leylia Khodra, ST., MT., Ibu Dr. Nur Ulfa Maulidevi, ST., MSc., dan Bapak Dr. Ir. Rinaldi Munir, MT. Yang telah mengampu mata kuliah IF2211 Strategi Algoritma selama keberjalanan perkuliahan satu semester ini. Tanpa bimbingan dan arahan beliau, makalah ini tidak akan terselesaikan dengan baik. Tak lupa juga penulis mengucapkan terima kasih kepada rekan-rekan dari Himpunan Mahasiswa Informatika ITB (HMIF ITB) atas kesempatan, kerja sama, dan bantuannya.

REFERENSI

[1] K. Ruohonen, *Graph Theory*. Dipublikasikan secara online, bab 1-2. Diambil dari *http://world.mathigon.org/Graph_Theory*, diakses tanggal 17 Mei 2017.

[2] E. W. Weisstein, *Simple Graph.* Dari *MathWorld--A Wolfram Web Resource.* Diambil dari *http://mathworld.wolfram.com/SimpleGraph.html*, diakses tanggal 17 Mei 2017.

[3] Andrew Myers, *Graph Traversal*. Bahan kuliah *Object-Oriented Design and Data Structures* di *Department of Computer Science Cornell University*. Diambil dari *http://www.cs.cornell.edu/courses/cs2112/2012sp/lectures/lec24/lec24-12sp.html*, diakses tanggal 17 Mei 2017.

[4] Akash Sharma, *Basics of Greedy Algorithm.* Diambil dari *https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/*, diakses tanggal 18 Mei 2017.

[5] Tim Regexbuddy.com, *What is a Regular Expression, Regexp, or Regex?* Diambil dari *https://www.regexbuddy.com/regex.html*, diakses tanggal 18 Mei 2017.

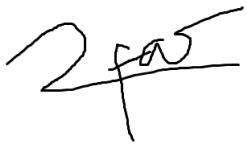
[6] Webopedia, *Search Engine*. Diambil dari *http://www.webopedia.com/TERM/S/search_engine.html*, diakses tanggal 18 Mei 2017.

[7] L. A. Siswanto, *Fast Pattern Matching Algorithm on Two-Dimensional String*. Sekolah Teknik Elektro ITB, 2015. Dipublikasikan secara online. Diambil dari *http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF221_Strategi_Algoritma_2015_031.pdf*, diakses tanggal 19 Mei 2017.

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, 19 Mei 2017



Turfa Auliarachman
13515133