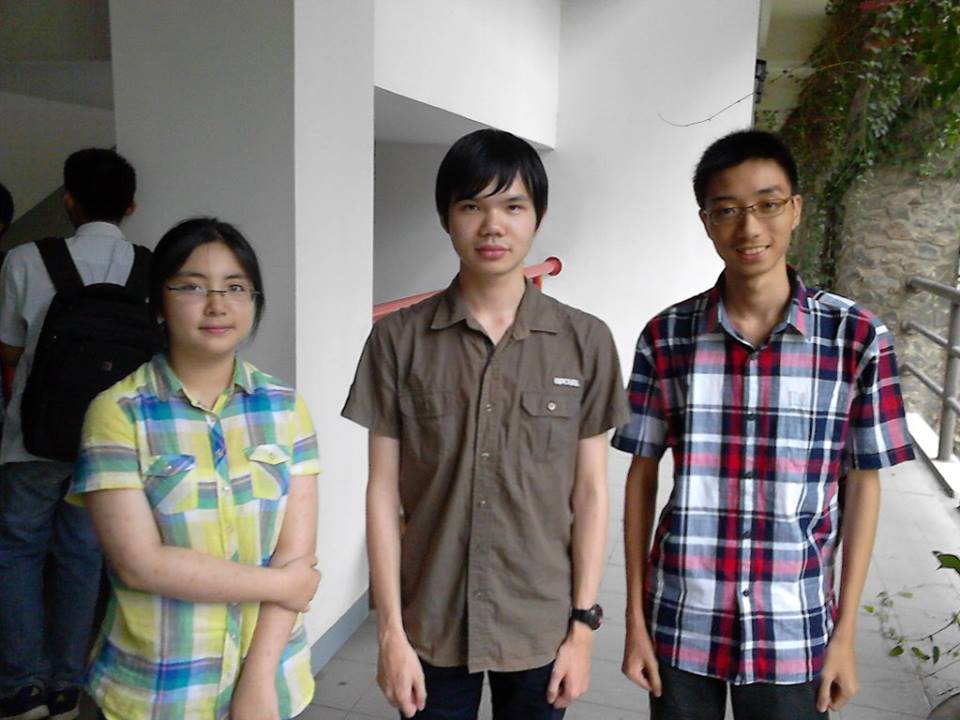
Tugas Besar II IF 2211

Strategi Algoritma

Aplikasi Depth First Search dan Breadth First Search pada Web Crawler dalam Mesin Pencari

**Disusun Oleh:**

Felicia Christie – 13512039

Michael Alexander Wangsa – 13512046

Winson Waisakurnia – 13512071

Sekolah Teknik Elektro dan Informatika

Jurusan Teknik Informatika

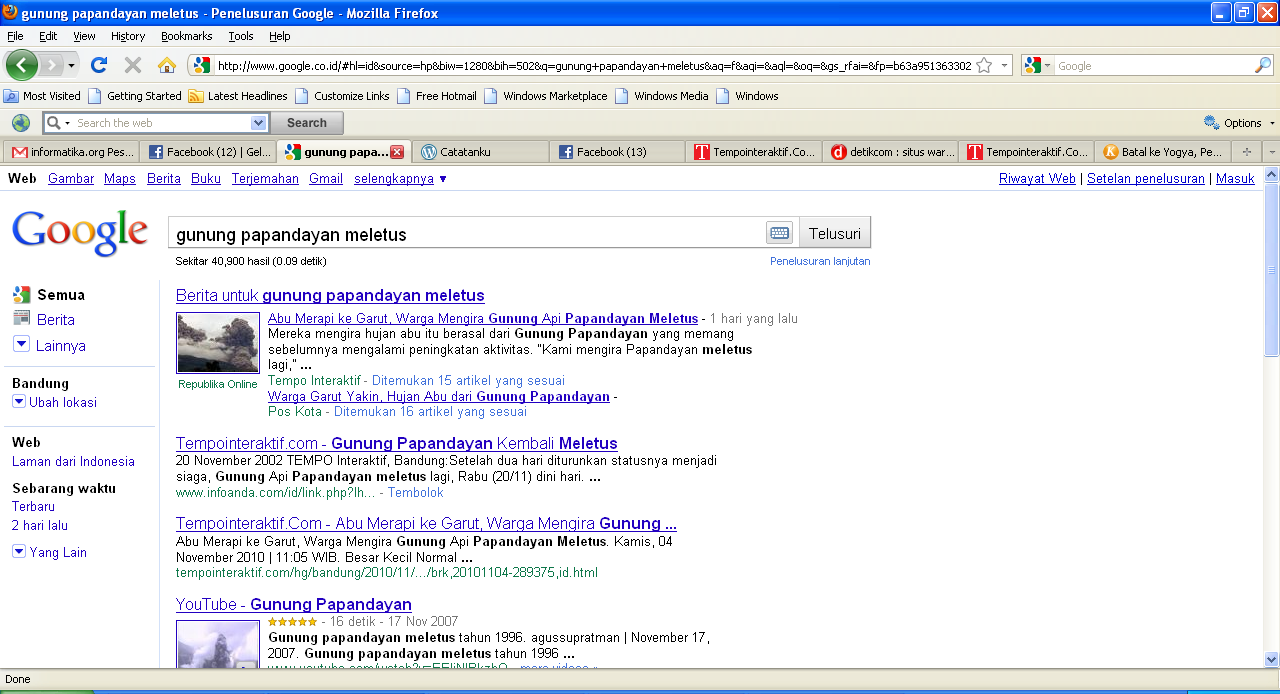
Institut Teknologi Bandung

2014

1. **Deskripsi Permasalahan**

Mesin pencari (*search engine*) adalah kakas yang sangat berguna untuk mencari informasi yang dibutuhkan di internet. Mesin pencari yang popular adalah *Google*, selain itu ada juga *Yahoo!*, *Altavista*, dan lain-lain meskipun tidak sepopuler *Google*.

Tinjau mesin pencari yang bernama *Google*. Seringkali orang awam bertanya, mengapa *Google* begitu cepat menampilkan laman web (*web pages*) yang mengandung *query* yang dimasukkan pengguna? Jawabannya adalah karena laman-laman *web* tersebut sudah dijelajah terlebih dahulu oleh *Web Crawler*.

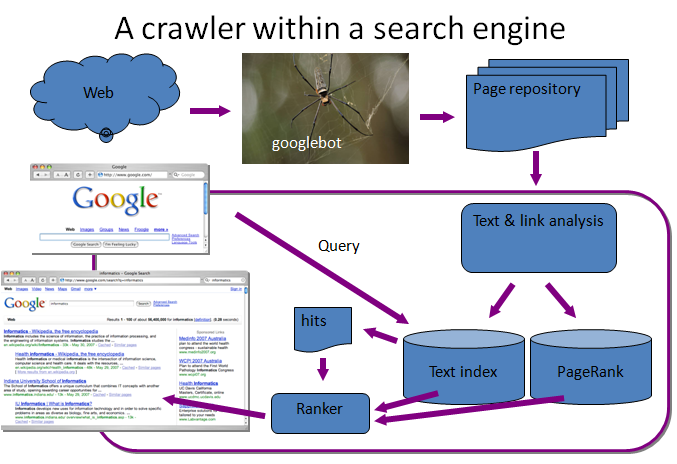


Laman-laman web yang mengandung kata ‘gunung papandayan meletus’

**Gambar 1**. Hasil pencarian Google untuk *query* “gunung papandayan meletus”

*Web crawler* adalah komponen mesin pencari yang terpenting. Tugasnya adalah menjelajah seluruh laman-laman *web* yang ada di dunia maya secara periodik (misalnya setiap 1 jam, setiap hari, dan sebagainya), karena laman-laman web selalu tumbuh dan berubah (bertambah, menyusut, atau sudah dimutakhirkan isinya). Nama lainnya adalah *Spider*, *Robot*, *Web Agent*, dan instans seperti *googlebot*, *msnbot*, dan lain-lain.

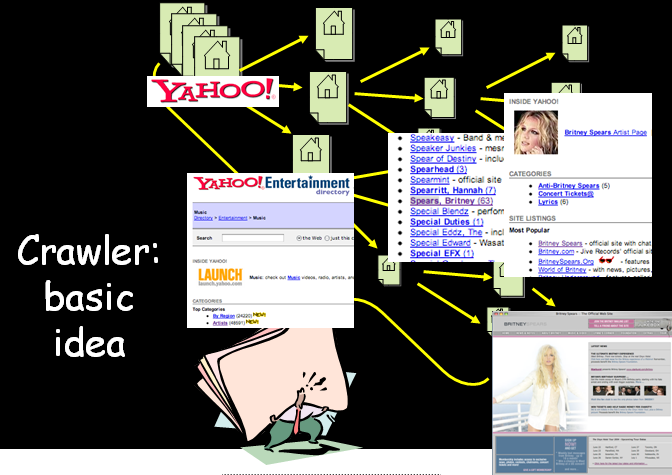
Setiap kali menjelajah internet, *web crawler* mencatat alamat laman tersebut (*URL*), beberapa kata penting (*keywords*) di dalam laman (nama laman web, judul di dalam laman, upa judul, dll), dan atribut lain seperti tanggal dan jam. Semua informasi tersebut disimpan di dalam sebuah file *index* (*text index*). Ketika pengguna (*user*) memasukkan *query* pada antaramuka mesin pencari, maka yang terjadi sesungguhnya adalah mesin melakukan pencarian *query* di dalam file *index* tersebut lalu menampilkan hasil pencarian sesuai dengan *rangking* laman. Oleh karena itu tidak heran jika pencarian berlangsung sangat cepat (Gambar 2).



**Gambar 2**. *Web Crawler* di dalam mesin pencari

(Sumber: Filippo Menczer @Indiana University School of Informatics in *Web Data Mining*)

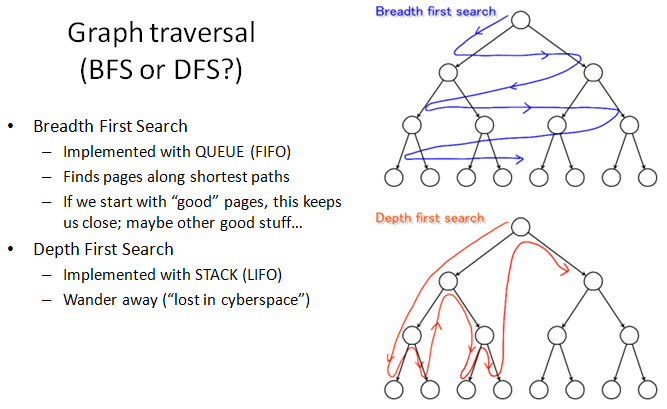
Bagaimana *crawler* melakukan penjelajahan? Perhatikan bahwa setiap lama *web* umumnya memiliki pranala atau tautan (*link*) ke laman web lainnya. *Crawler* memulai penjelajahan dari sebuah laman *web* awal, lalu ia mengakses setiap pranala di dalam laman tersebut, mengunjungi laman *web* yang ditunjuk pranala tersebut, mencatat informasi di dalam laman tersebut, mengunjungi pranala berikutnya, dan seterusnya. Rangkaian pranala tersebut dapat dipandang sebagai sebuah struktur pohon, yang dalam hal ini simpul menyatakan laman *web* sedangkan busur menyatakan pranala ke laman tersebut (lihat Gambar 3).



**Gambar 3**. Struktur laman-laman web membentuk pohon

(Sumber: Filippo Menczer @Indiana University School of Informatics in *Web Data Mining*)

Penjelahan laman-laman *web* tersebut dapat menggunakan salah satu dari dua algoritma pencarian di dalam graf, yaitu DFS dan BFS (lihat Gambar 4).



**Gambar 4**. Prinsip BFS dan DFS dalam penjelajahan laman laman web.

Tugas anda adalah membuat *web crawler* sederhana (tanpa perangkingan laman *web*) yang diterapkan untuk menjelajah laman-laman *web* hanya di dalam situs-situs tertentu saja (tidak ke semua laman *web* di dunia maya). Selanjutnya *web crawler* tersebut diaplikasikan di dalam sebuah mesin pencari sederhana yang hanya melakukan pencarian pada situs *web* tersebut. Situs *web* yang dijelajahi sebagai uji coba adalah situs-situs kecil dan sedang seperti:

1. Situs rinaldimunir (<http://informatika.stei.itb.ac.id/~rinaldi.munir>)
2. Situs [www.itb.ac.id](http://www.itb.ac.id)
3. Situs Wikipedia ([www.wikipedia.org](http://www.wikipedia.org))
4. Situs lainnya yang tidak terlalu besar

**Spesifikasi program :**

1. Program *search engine* yang anda buat terdiri dari tiga bagian: Program *query*, *web crawler*, dan *explorer*. Program *query* adalah berupa antarmuka pengguna-komputer, program *query* dibuat berbasis *web* (*web-base*). Tampilan antarmuka pengguna-komputer kira-kira seperti Gambar 5 di bawah ini:

|  |
| --- |
| **My Crawler and Search Engine**  Cari    Crawler Algoritma traversal Perihal |

**Gambar 5**. Antarmuka pengguna-kumputer

Crawler: pengguna mengaktifkan menu ini untuk memulai melakukan penjelajahan laman-laman web pada situs yang dispesifdapat menspesifikasikan

Algoritma: pengguna dapat memilih algoritma traversal (DFS atau BFS). Default: BFS

Perihal: keterangan tentang program dan pembuatnya

Anda dapat menambahkan menu lainnya, gambar, logo, dan sebagainya. Susunan menu di dalam antarmuka tidak harus seperti di atas, silakan dikreasikan sendiri.

1. Hasil penjelajahan *Crawler* disimpan di dalam sebuah file *index* dan dapat ditampilkan ke layar jika pengguna ingin melihatnya. Pengguna dapat mengklik URL pada setiap *record* di dalam file dan *web browser* mengakses laman tersebut dan menampilkannya ke layar. File *index* boleh menggunakan file database (dbf).
2. *Keyword* pada setiap lama web yang disimpan oleh *crawler* dapat berupa judul pada laman tersebut, kata-kata di dalam tag, dan lain-lain (silakan dipikirkan). Jumlah keyword tidak dibatasi.
3. Modul *explorer* melakukan pencocokan *query* pada file *index* menggunakan fungsi *string matching* yang sudah tersedia di dalam bahasa pemrogaman.
4. Seluruh laman *web* yang mengandung *query* yang cocok ditampilkan daftarnya di halaman yang baru. Misalnya jika *query* yang dimasukkan pengguna adalah “merapi”, maka luaran yang dihasilkan adalah seluruh laman yang judulnya mengandung kata “merapi” seperti contoh berikut:

Gunung **Merapi** Meletus pada Jumat Dinihari

Jalan-jalan ke Gunung **Merapi**

Mbah Maridjan, Juru Kunci **Merapi**

Antara Keraton, **Merapi**, dan Laut Kidul

Pada setiap judul yang ditampilkan terdapat *hyperlink* (pranala) sehingga dengan mengklik pranala tersebut maka lama neb tersebut diakses oleh *web browser* dan ditampilkan ke layar.

1. **Dasar Teori**

DFS (*Depth First Search*) dan BFS (*Breath First Search*) adalah algoritma penelusuran pada graf. Kedua algortima ini memulai pencarian pada graf dimulai dari sebuah simpul. Dari simpul awal tersebut akan diakses data pada simpul tersebut dan kemudian mengakses simpul selanjutnya yang bersebelahan dengan simpul yang sudah pernah dikunjungi.

Perbedaan dari DFS dan BFS adalah pada urutan pengaksesan. Seperti tergambar pada namanya, DFS mengutamakan pencarian pada simpul yang lebih dalam. DFS berhenti mengakses simpul yang lebih dalam atau lebih jauh jika simpul tersebut tidak memiliki anak atau semua simpul anak sudah pernah dikunjungi. BFS mengutamakan pencarian pada seluruh simpul yang lebih dekat dengan simpul awal terlebih dahulu. DFS biasanya diimplementasikan secara rekursif atau dengan memanfaatkan struktur data *stack*. BFS biasanya diimplementasikan dengan memanfaatkan struktur data *queue*.

Pseudocode DFS:

1. **procedure** DFS(G,v)
2. tandai v sudah dilalui
3. **for** semua simpul w yang merupakan tetangga simpul v **do**
4. **if** simpul v belum pernah dilalui **then**
5. **call** DFS(G,w)

Pseudocode BFS:

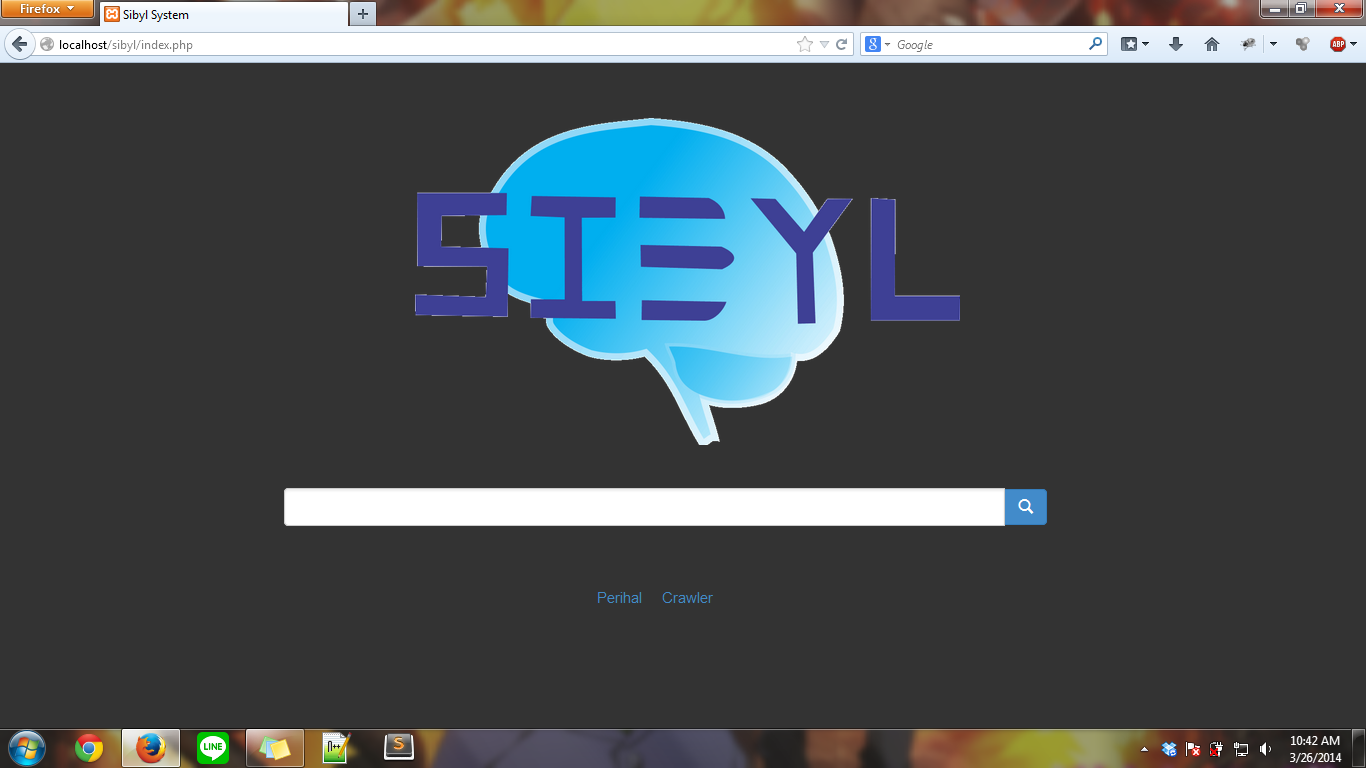
1. **procedure** BFS(G,v)
2. buat queue Q
3. tandai v sudah pernah dikunjungi
4. tambahkan simpul v ke dalam queue
5. **while** Q tidak kosong **do**
6. w adalah elemen yang dikeluarkan dari queue
7. **if** w adalah yang dicari **then**
8. **return** w
9. **for** semua simpul x yang bersebelahan dengan simpul w **do**
10. **if** x tidak pernah dikunjungi **then**
11. tandai x sudah pernah dikunjungi
12. tambahkan simpul x ke dalam queue

Karena BFS dan DFS kadang digunakan untuk pencarian pada graf yang sangat besar dan tidak diketahi ujungnya, seringkali dibuat batasan kedalaman pencarian. DLS (*Depth Limited Search*) dikenal sebagai modifikasi dari DFS yang kedalaman pencariannya pada graf dibatasi sampai kedalaman tertentu.

1. **Analisis Pemecahan Masalah**
2. Crawling dengan algoritma BFS
3. Crawler membaca suatu alamat awal untuk penelusuran yang dimasukkan ke dalam prosedur BFS.
4. Dari alamat awal tersebut, bentuk simpul-simpul penelusuran yang terdiri dari alamat pranala yang terdapat pada alamat awal tersebut. *Depth* dari simpul awal yang teratas tersebut adalah 0, sementara simpul-simpul anak dari simpul awal memiliki *depth* bernilai *depth* awal ditambah 1.
5. Jika semua simpul dari alamat pertama sudah terbentuk, maka penelusuran alamat pranala dilakukan dari simpul paling kiri, kemudian dilanjutkan sampai pada tingkat kedalaman tersebut, setiap simpul dari simpul-simpul penelusuran yang terbentuk dari alamat pertama telah didata pranalanya. Saat penelusuran pada halaman dilakukan, prosedur BFS juga akan memanggil fungsi untuk *indexing*.
6. Pranala yang sebelumnya telah ditambahkan pada pohon penelusuran tidak akan dimasukkan lagi.
7. Penelusuran dilakukan secara iteratif, mengikuti urutan tingkat dari pohon penelusuran.
8. Penelusuran dilakukan sampai semua halaman yang berada dalam *depth*  yang diinginkan selesai ditelusuri dan dimasukkan dalam indeks.
9. Crawling dengan DFS
10. DFS (Depth First Search) yang digunakan adalah DFS yang menggunakan cara rekursif
11. Parameter dari fungsi DFS tersebut adalah sisa kedalaman yang akan ditelusuri dan URL yang akan ditelusuri. Variasi DFS yang dipakai ini disebut DLS (Depth Limited Search), yaitu variasi dari DFS yang dengan kedalaman maksimum penelusuran sudah ditentukan.
12. Dalam fungsi DLS pertama dilakukan penecekan apakah sisa kedalaman yang akan ditelusuri masih ≥0 dan URL sudah berada di dalam daftar URL yang pernah ditelusuri
13. Jika sisa kedalaman <0 atau URL tersebut sudah berada pada daftar URL, maka penelusuran pada URL tersebut dihentikan
14. Saat ditelusuri, pertama dilakukan penambahan URL pada daftar URL yang sudah pernah dikunjungi
15. Kemudian dilakukan pembacaan seluruh teks html yang terdapat pada URL tersebut
16. Selanjutnya seluruh kata ini diserahkan kepada pemroses teks untuk dimasukkan ke dalam file indeks
17. Dari teks html tersebut juga dilakukan pembacaan seluruh URL yang terdapat pada URL tersebut yang disimpan sebagai daftar URL lokal
18. Untuk setiap {x | x ‌Є daftar URL lokal} dilakukan pemanggilan fungsi DFS(sisa kedalaman sekarang-1, x)
19. **Implementasi dan Pengujian**

Pada implementasi dalam program, kami membuat sebuah prosedur untuk BFS dan dua buah prosedur untuk DFS (satu untuk rekursif dan satu untuk memanggil prosedur rekursif). Meninjau kondisi internet yang sekarang sudah semakin luas dan saling terhubung, maka setiap prosedur tersebut memiliki parameter kedalaman (*depth*) tersendiri. Selain parameter depth tersebut, prosedur crawling juga memiliki parameter String, yaitu masukan alamat awal untuk penelusuran.

1. Spesifikasi teknis program



**Gambar 4.1 Tampilan halaman awal mesin pencari SIBYL**

1. Pada halamanawal yang ditunjukkan oleh gambar 4.1 di atas, pengguna dapat memasukkan kata kunci (*keyword*) pada *textbox* kemudian meng-klik tombol pencarian yang terletak di sebelah *textbox*. Kemudian, pengguna akan diarahkan ke halaman yang menunjukkan hasil-hasil pencarian pada basis data yang sesuai dengan kata kunci tersebut.
2. Halaman *web* SIBYLdibuat sesuai dengan spesifikasi soal yaitu dengan menggunakan PHP dan HTML, serta mengimplementasikan program-program dasar yang sebelumnya telah dibuat dengan bahasa C# yang mengimplementasikan *library* tambahan yaitu HTMLAgilityPack.
3. Hasil penelusuran tersebut mengambil dari file hasil *indexing* yang sebelumnya telah dilakukan oleh *Crawler*.
4. Pengguna dapat melakukan *crawl* kapanpun ia mau dari halaman awal tersebut dengan meng-klik pranala *Crawler* yang berada tepat dibawah *textbox* untuk kata kunci.
5. Setelah metode untuk *crawling*, kedalaman, dan halaman yang akan di-*crawl* dipilih, maka halaman akan memanggil prosedur *crawl* yang sesuai dengan permintaan pengguna.
6. Proses yang dilakukan saat *crawling* terbagi menjadi dua tahap yang berlangsung hampir bersamaan, yaitu penelusuran pranala dan pemasukan (*dumping*) pada basis data.
7. Prosedur dan fungsi yang digunakan:

|  |
| --- |
| /\* Nama File : SibylCrawler.cs \*/  /\* Deskripsi : File program untuk Crawling dan operasi-operasi pada alamat.\*/ |
| public static String ReadUrl(String url);  /\* Fungsi melakukan *request* terhadap alamat url kemudian mengembalikan kode HTML dari halaman yang dituju oleh url \*/  public static ArrayList LinkSearchAgility(String htmlText, String url);  /\* Fungsi melakukan pencarian terhadap pranala-pranala yang terdapat dalam alamat url dan terdiri dari kode HTML htmlText dengan bantuan *library* HTMLAgilityPack.\*/  public static String GetTitle(String htmlText);  /\* Fungsi mengembalikan string judul dari sebuah halaman web.\*/  public static void BFS(int depth, String url);  /\* Fungsi melakukan pencarian dan peng-indeks-an secara BFS pada alamat url dengan kedalaman depth.\*/  private static void DFSrecursive(int depth\_left, String url);  /\* Prosedur melakukan pencarian secara rekursif yang kemudian disimpan pada variabel global untuk menampung pranala.\*/  public static void DFS(int depth, String url);  /\* Fungsi melakukan pencarian dengan memanggil fungsi DFSrecursive, kemudian\*/ |

|  |
| --- |
| /\* Nama File : SibylCore.cs \*/  /\* Deskripsi : File program untuk operasi-operasi pada basis data dan pengelolaan lanjut untuk teks yang masih mengandung kode-kode html\*/ |
| public void Dump(string title, string url, string keywords);  /\* I.S: SibylDB.dat dalam keadaan sembarang, keywords masih dalam keadaan mengandung kode-kode HTML.  F.S: SibylDB.dat ditambah dengan data judul, alamat dan keywords yang telah difilter agar tidak muncul lebih dari sekali dan tidak mengandung kode HTML ataupun kode lainnya\*/  private bool isUrlExists(string url);  /\* Fungsi mengembalikan nilai true jika url telah ada dalam SibylDB.dat.\*/  public string Retrieve(string what)  /\* Fungsi mengembalikan sebuah string yang terdiri dari <title> [^SEPARATOR&] <url>, yaitu judul dan alamat tempat ditemukannya string what. \*/ |

|  |
| --- |
| /\* Nama File : SibylSearcher.cs \*/  /\* Deskripsi : Driver yang akan menerima kata kunci pencarian, kemudian memanfaatkan fungsi Retrieve dari kelas SibylCore untuk mendapatkan pranala yang mengandung kata kunci yang telah dimasukkan. Hasilnya kemudian diimplementasi pada tampilan halaman hasil pencarian.\*/ |
| public static void Main(String[] args); |

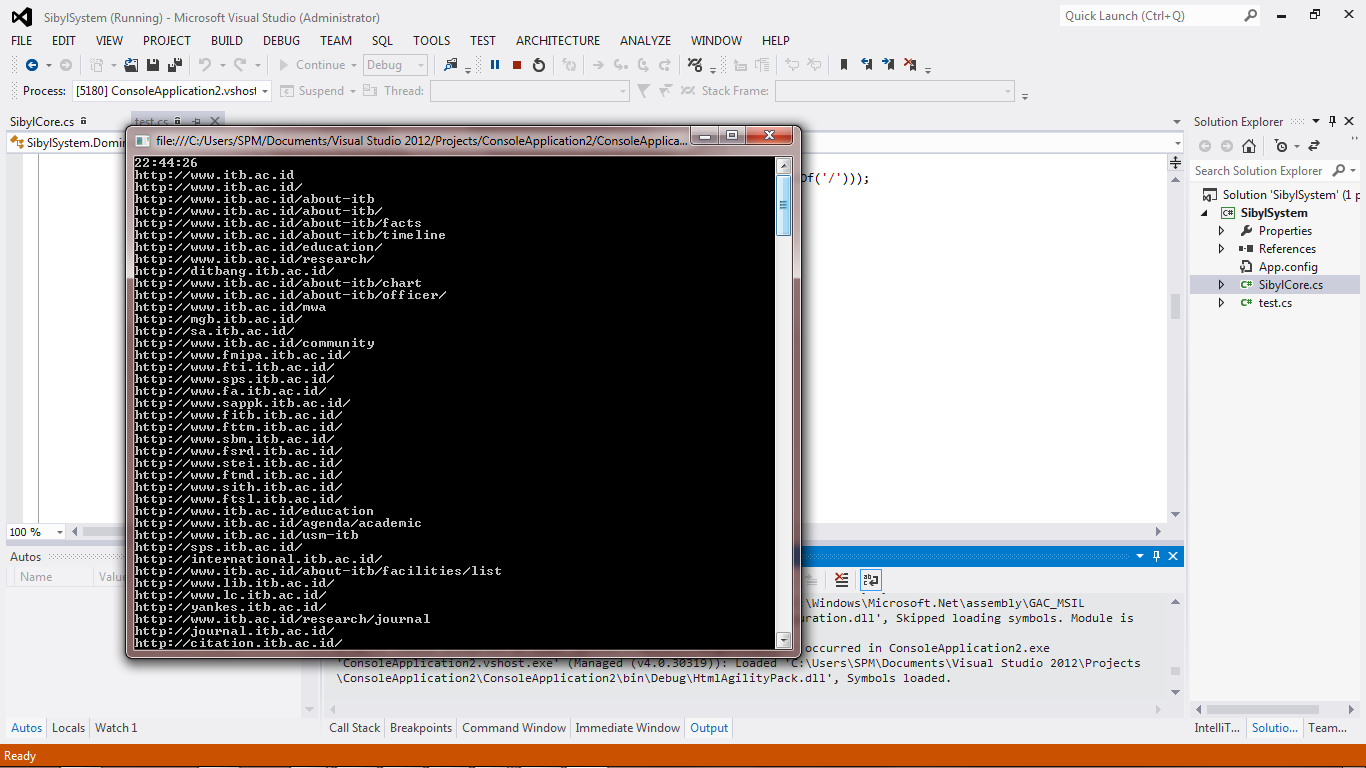
|  |
| --- |
| /\* Nama File : SibylSpider.cs \*/  /\* Deskripsi : Driver untuk menerima pilihan pencarian (DFS atau BFS), kedalaman, dan alamat awal pencarian. Driver ini memanggil prosedur yang sesuai dengan pilihan user pada kelas Dominator untuk melakukan pencarian\*/ |
| public static void Main(String[] args); |

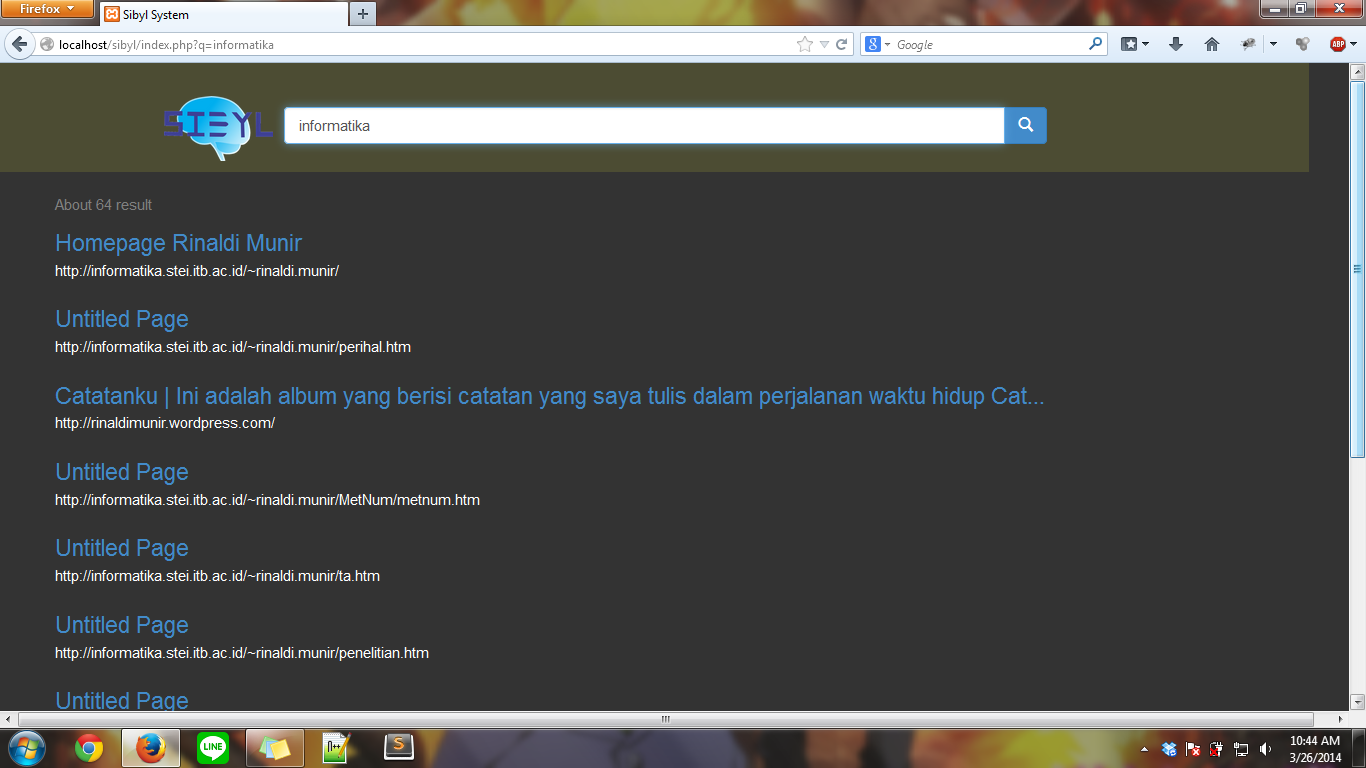
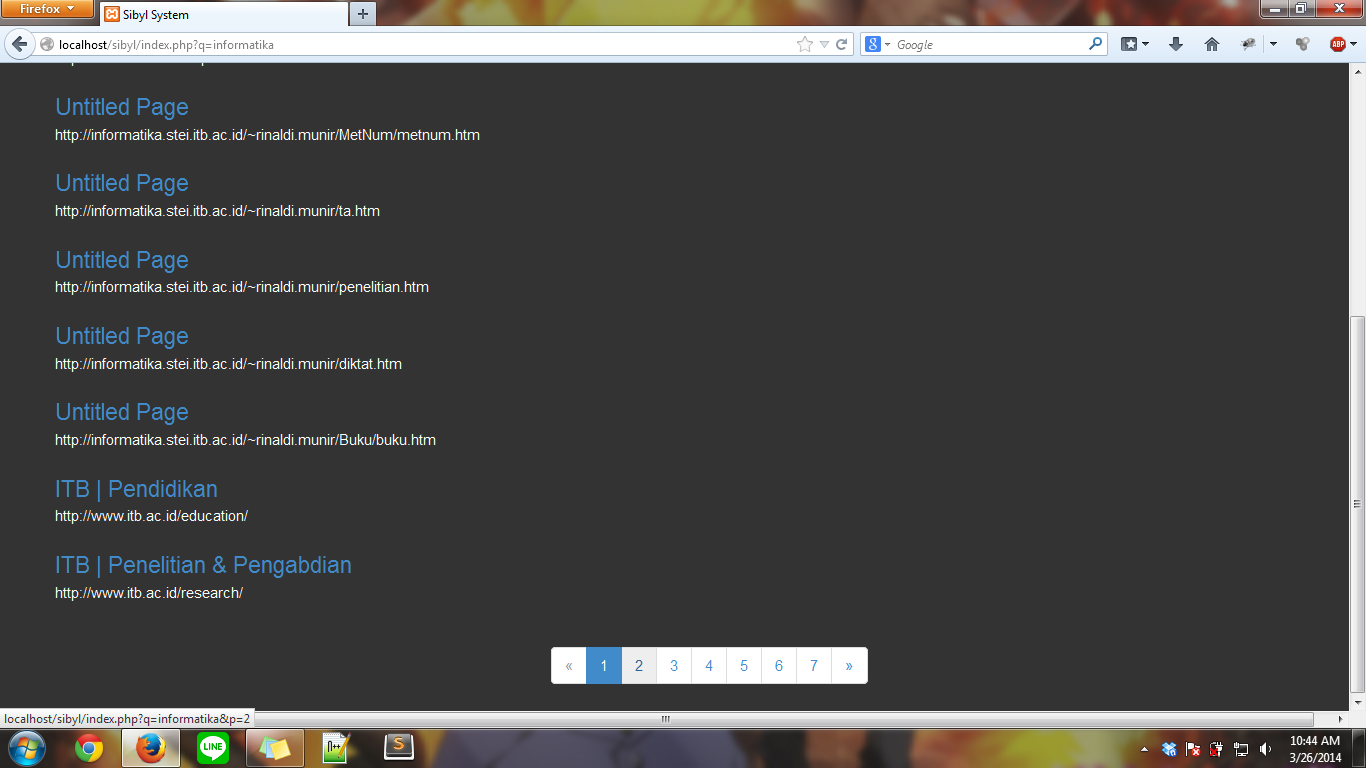
Untuk bagian *interface*, program ini mengimplementasi:

1. HTML
2. CSS
3. Javascript
4. PHP
5. Bootstrap (Framework CSS)
6. XAMPP
7. Format basis data penyimpanan hasil crawling

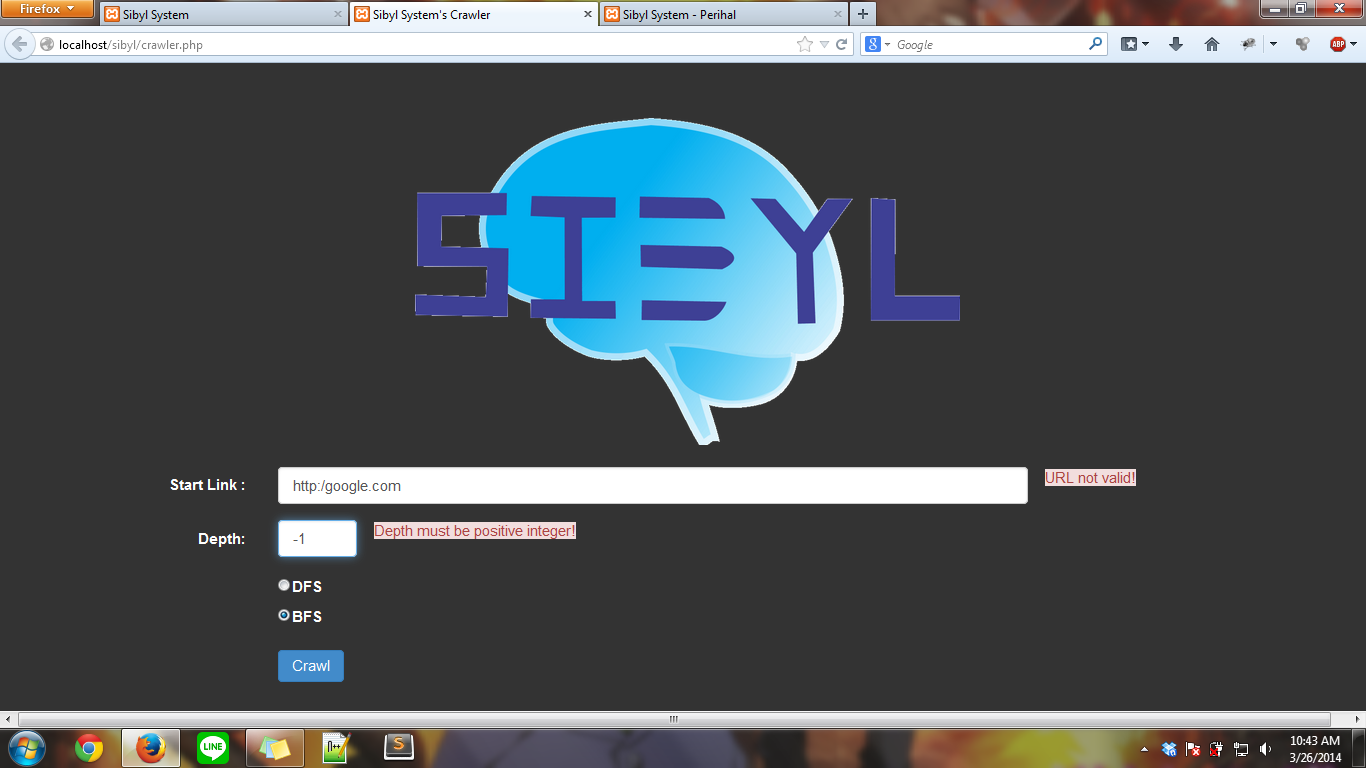
|  |  |
| --- | --- |
| /\* Nama File : SibylDB.dat \*/  /\* Deskripsi : File tempat penyimpanan seluruh hasil crawling. Format dalam bentuk :  <judul1> [^SEPARATOR&] <url1> [^SEPARATOR&]<keyword1\_1> ... <keyword1\_n>  <judul2> [^SEPARATOR&] <url2> [^SEPARATOR&]<keyword2\_1> ... <keyword2\_m>  ...  \*/ | |
| Baris | Text dalam file |
| 1 | Homepage Rinaldi Munir[^SEPARATOR&]http://informatika.stei.itb.ac.id/~rinaldi.munir/[^SEPARATOR&] homepage rinaldi munir sekolah teknik elektro dan informatika itb.... |
| 2 | Untitled Page[^SEPARATOR&]http://informatika.stei.itb.ac.id/~rinaldi.munir/perihal.htm[^SEPARATOR&] perihal saya klik foto lahir di kota padang yang eksotis sumatera barat tempat siti nurbaya... |

1. Eksperimen / pengujian program
2. Misalkan kata kunci yang dimasukkan adalah “informatika”. Maka setelah meng-klik tombol pencarian, pengguna akan diarahkan ke halaman hasil pencarian yang berisi judul laman dan alamat laman yang mengandung kata kunci tersebut. Judul dari setiap halaman hasil pencarian tersebut dapat mengarahkan pengguna ke laman yang mengandung kata kunci tersebut. Tampilan halaman hasi pencarian dapat dilihat pada gambar 4.2 di bawah ini.
3. Penelusuran awal dilakukan dengan program console application pada C#. Saat testing, kami memunculkan alamat-alamat yang ditelusuri. Contoh tampilan dapat dilihat pada gambar 4.2.
4. Pada implementasi *crawling* dalam halaman web, hasil *crawling* dari alamat web yang dimasukkan berhasil muncul pada file index SibylDB.dat

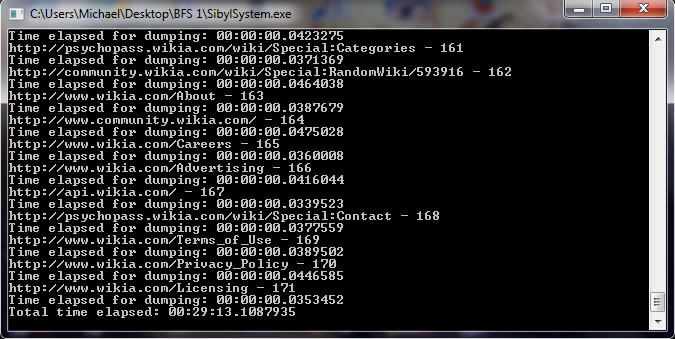


**Gambar 4.2 Output alamat yang ditelusuri**

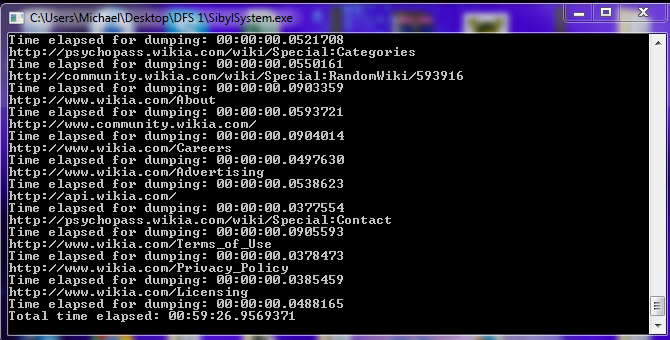
**Gambar 4.3 Halaman hasil penelusuran dengan kata kunci ‘informatika’**

****

**Gambar 4.4 Bagian *crawler* yang menerima masukan pengguna dari web**



**Gambar 4.5 Total waktu untuk *crawling* dan *dumping* data dengan BFS, depth 1**



**Gambar 4.6 Total waktu untuk *crawling* dan *dumping* data dengan DFS, depth 1**

1. Analisis hasil pengujian

* Pada *depth* bernilai antara 0 dan 1, terlihat bahwa perbedaan waktu antara DFS dan BFS sedikit, dengan BFS memakan waktu lebih sedikit dibanding DFS. Muncul selisih waktu yang semakin melebar saat depth diganti menjadi 2. DFS memakan waktu lebih sedikit dibanding BFS. Hal ini dikarenakan DFS menggunakan *stack* sehingga walaupun *depth* cukup dalam, alamat yang disimpan jauh lebih sedikit dibandingkan BFS yang menggunakan *queue*. Dalam BFS ini, seluruh alamat yang sebelumnya telah muncul masih disimpan dalam memori sehingga pemrosesan berlangsung semakin lama. Selain itu, isi variabel bertipe ArrayLinks yaitu Links yang dipakai untuk menampung alamat pada BFS akan bertambah terus secara eksponensial, sehingga saat pengecekan alamat agar tidak terjadi duplikasi, pengecekan akan berlangsung makin lama. *Array* ini juga memakan banyak memori karena tidak dilakukan dealokasi / penghapusan terhadap data yang sebelumnya (alamat-alamat yang sudah diakses tetap perlu dimasukkan dalam larik Links untuk pengecekan)
* Dapat muncul kesalahan apabila URL masukan pengguna pada web SIBYL tidak memenuhi standar kelas HttpWebRequest. Contohnya: tidak diawali dengan http:// atau https://

1. **Kesimpulan dan Saran**

* Kesimpulan:
  + Untuk kasus depth kecil, maka tidak terdapat perbedaan yang signifikan antara BFS dan DFS. Secara teori, untuk depth lebih besar maka performansi DFS akan menurun, namun karena struktur penyimpanan data yang kurang optimal (*array* dinamis tanpa *deletion* pada BFS dan *stack* dengan *deallocation* pada BFS), maka pada kasus *depth* banyak, DFS akan menggunakan lebih sedikit memori dari pada BFS sehingga DFS berjalan lebih mulus dan cepat dibandingkan BFS.
  + Karena DFS bersifat rekursif, maka kedalaman yang terlalu tinggi dapat menyebabkan *stack overflow* sehingga tidak disarankan menggunakan DFS, atau jika bisa, tidak menggunakan rekursif untuk DFS.
  + Penelusuran secara BFS dan DFS akan menghasilkan pohon penelusuran yang sama namun urutan kemunculan simpul yang berbeda.
* Saran:
  + Untuk tipe data penyimpanan, lebih baik menggunakan *queue* untuk BFS karena *backtracking* tidak akan dilakukan. Namun ada kemungkinan terdapat duplikasi jika tidak disimpan terlebih dahulu.
  + Pada kasus penelusuran pranala dari sebuah website, jumlah pranala tiap *depth* dapat bertambah secara eksponensial (setiap *depth* lebih lebar secara eksponensial daripada *depth* sebelumnya), oleh karena itu, DFS lebih efektif dari BFS untuk penelusuran pranala. BFS akan lebih efektif daripada DFS jika jumlah simpul dari tiap *depth* lebih kecil dari jumlah *depth.*
  + Diusahakan saat *crawling*, kecepatan dan sinyal internet jelas dan konstan sehingga data indeks dan waktu pemrosesan yang dihasilkan bisa dibandingkan dengan benar.

**DAFTAR REFERENSI**

1. Munir, Rinaldi. 2009. “Diktat Kuliah IF2211 Strategi Algoritma”. Bandung
2. Knuth, Donald E. 1997. “The Art Of Computer Programming”.3rd. Boston: Addison-Wesley
3. Heaton, Jeff. 2007. “HTTP Programming Recipes for C# Bots”. Chesterfield: Heaton Research Inc.