DESIGN IMPROVEMENT OF (CRAWLING ENGINE) WITH (RUST) BENCHMARKED AGAINST THE ORIGINAL CRAWLING MODULE IN PYTHON

Skripsi

Disusun untuk memenuhi salah satu syarat memperoleh gelar Sarjana Komputer

Oleh:
Muhammad Daffa Haryadi Putra
1313619034

PROGRAM STUDI ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI JAKARTA

2021

LEMBAR PENGESAHAN

Dengan ini saya mahasiswa Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta

Nama	: Muhammad Fathan Qoriiba
No. Registrasi	: 3145161299
Program Studi	: Ilmu Komputer
Judul	: Desain Perancangan *Crawler* Sebagai Pendukung pada *Search Engine*

Menyatakan bahwa skripsi ini telah siap diajukan untuk skripsi.

Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
Med Irzal, M.Kom.	**Muhammad Eka Suryana, M. Kom.**
NIP. 19770615 200312 1 001	NIP. 19851223 201212 1 002

Mengetahui,

Koordinator Program Studi Ilmu Komputer

Ir. Fariani Hermin Indiyah, M.T.

NIP. 19600211 198703 2 001

LEMBAR PERSETUJUAN HASIL SIDANG SKRIPSI

Desain Perancangan *Crawler* Sebagai Pendukung pada *Search Engine*

Nama : Muhammad Fathan Qoriiba

No. Registrasi : 3145161299

	Nama	**Tanda Tangan**	**Tanggal**
Penanggung Jawab			
Dekan	: Prof. Dr. Muktiningsih N, M.Si.	.....	.....
	NIP. 19640511 198903 2 001		
Wakil Penanggung Jawab			
Wakil Dekan Bidang Akademik	: Dr. Esmar Budi, S.Si., MT.	.....	.....
	NIP. 19720728 199903 1 002		
Ketua	: Drs. Mulyono, M.Kom.	.....	.....
	NIP. 19660517 199403 1 003		
Sekretaris	: Ari Hendarno, S.Pd, M.Kom	.....	.....
	NIDK. 8857650017		
Penguji Ahli	: Ria Arafiyah, M.Si.	.....	.....
	NIP. 19751121 200501 2 004		
Pembimbing I	: Med Irzal, M.Kom.	.....	.....
	NIP. 19770615 200312 1 001		
Pembimbing II	: Muhammad Eka Suryana, M.Kom.	.....	.....
	NIP. 19851223 201212 1 002		

Dinyatakan lulus ujian skripsi tanggal: 04 Agustus 2021

LEMBAR PERNYATAAN

Saya menyatakan dengan sesungguhnya bahwa skripsi dengan judul **Desain Perancangan *Crawler* Sebagai Pendukung pada *Search Engine*** yang disusun sebagai syarat untuk memperoleh gelar Sarjana komputer dari Program Studi Ilmu Komputer Universitas Negeri Jakarta adalah karya ilmiah saya dengan arahan dari dosen pembimbing.

Sumber informasi yang diperoleh dari penulis lain yang telah dipublikasikan yang disebutkan dalam teks skripsi ini, telah dicantumkan dalam Daftar Pustaka sesuai dengan norma, kaidah dan etika penulisan ilmiah.

Jika dikemudian hari ditemukan sebagian besar skripsi ini bukan hasil karya saya sendiri dalam bagian-bagian tertentu, saya bersedia menerima sanksi pencabutan gelar akademik yang saya sanding dan sanksi-sanksi lainnya sesuai dengan peraturan perundang-undangan yang berlaku.

Jakarta, 26 Julii 2021

Muhammad Fathan Qoriiba

HALAMAN PERSEMBAHAN

Untuk Abi dan Umi.

KATA PENGANTAR

Ungkapan Puji dan Syukur penulis panjatkan kehadirat Allah SWT, atas segala rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan proposal skripsi ini dengan baik. Adapun jenis penelitian yang dipilih yaitu *information retrieval* dengan judul Desain Perancangan *Crawler* Sebagai Pendukung pada *Search Engine*.

Dalam menyelesaikan proposal ini, penulis selalu mendapat dorongan dan bantuan. Oleh karena itu, penulis menyampaikan terima kasih kepada Bapak Med Irzal, M. Kom. selaku Dosen Pembimbing I dan Bapak Muhammad Eka Suryana, M. Kom. selaku Dosen Pembimbing II yang telah membimbing, mengarahkan, serta memberikan saran dan koreksi terhadap proposal ini.

Dalam penulisan proposal skripsi ini, penulis menyadari bahwa dengan keterbatasan ilmu dan pengetahuan penulis, proposal ini masih jauh dari sempurna, baik dari segi penulisan, penyajian materi, maupun bahasa. Oleh karena itu, penulis sangat membutuhkan kritik dan saran yang dapat dijadikan sebagai pembelajaran serta dapat membangun penulis agar lebih baik lagi kedepannya.

Akhir kata, penulis berharap ini bermanfaat bagi semua pihak khususnya penulis sendiri, serta menjadi semangat dan motivasi bagi rekan-rekan yang akan melaksanakan skripsi berikutnya. Semoga Allah SWT senantiasa membalas kebaikan semua pihak yang telah membantu penulis dalam menyelesaikan proposal ini.

Terima kasih,

Jakarta, 8 Januari 2021

Penulis

ABSTRAK

MUHAMMAD FATHAN QORIIBA. Desain Perancangan *Crawler* Sebagai Pendukung pada *Search Engine*. Skripsi. Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta. 2021. Di bawah bimbingan Med Irzal, M. Kom dan Muhammad Eka Suryana, M. Kom.

Koperasi Mahasiswa Universitas Negeri Jakarta (KOPMA UNJ) merupakan organisasi kemahasiswaan tingkat universitas yang bergerak dalam bidang perkoperasian. KOPMA UNJ merupakan koperasi serba usaha, yang tujuan utamanya adalah menyejahterakan anggota koperasi. Aspek perkoperasian yang dijalankan KOPMA UNJ adalah simpanan dan usaha. Pada prosesnya, pengelolaan dana simpanan akan digunakan untuk pembelian barang usaha yang pada akhirnya akan menghasilkan keuntungan untuk anggota yang disebut sisa hasil usaha (shu). Skripsi ini bertujuan untuk membangun sistem informasi KOPMA UNJ berbasis *website* agar memudahkan pengurus KOPMA UNJ dalam mengelola dana simpanan dan usaha koperasi, serta agar anggota KOPMA UNJ dapat mengetahui transparansi dana simpanan yang mereka bayarkan kapanpun dan dimanapun. Sistem informasi KOPMA UNJ berbasis *website* dikembangkan menggunakan metode pengembangan perangkat lunak (*system development life cycle*) model *spiral*. Model *spiral* memiliki 4 tahapan pengembangan, yaitu analisis kebutuhan, desain sistem, implementasi, dan *maintenance*. Sistem informasi KOPMA UNJ dibangun menggunakan *framework codeigniter* untuk bagian *back-end* dan *framework bootstrap* untuk tampilan *user front-end*. *User* yang ada pada sistem informasi KOPMA UNJ berjumlah 3, yaitu *admin* yang dapat mengelola seluruh data simpanan dan usaha, anggota yang dapat mengetahui transparansi seluruh data, dan pengawas yang dapat mengelola data penilaian.

Kata kunci : *Webite*, sistem informasi, model *spiral*, *framework codeigniter*, *framework bootstrap*, koperasi, simpanan, usaha.

ABSTRACT

MUHAMMAD FATHAN QORIIBA. *Crawler Design as Support for Search Engine. Thesis. Faculty of Mathematics and Natural Sciences, State University of Jakarta. 2021. Supervised by* Med Irzal, M. Kom *and* Muhammad Eka Suryana, M. Kom.

Koperasi Mahasiswa Universitas Negeri Jakarta (KOPMA UNJ) is a university-level student organization engaged in cooperatives. KOPMA UNJ is a multi-business cooperative, where the main goal is for cooperative member's prosperity. The cooperative aspects carried out by KOPMA UNJ are savings and business. For the process, the management of deposit funds will be used for purchasing business goods where will give benefits for members called sisa hasil usaha (shu). This thesis aims to build a website-based KOPMA UNJ information system in order to make it easier administrators of KOPMA UNJ to manage savings and cooperative business funds, and also for members of KOPMA UNJ will know the transparency of their deposit funds they have paid. Website-based KOPMA UNJ information system have been developed using a spiral model software development (system development life cycle) method. The spiral model have 4 stages of development, there are needs analysis, system design, implementation, and maintenance. The KOPMA UNJ information system was built using the framework codeigniter for the back-end and framework bootstrap for user front-end display. There are three users in the KOPMA UNJ information system, those are 'admin' where can manage all deposit and business data, 'members' where can find out the transparency of all data, and 'supervisors' where can manage the assesment data.

Keywords: Website, information system, spiral model, framework codeigniter, framework bootstrap, cooperative, savings, business.

DAFTAR ISI

HALAMAN PERSEMBAHAN **v**

KATA PENGANTAR **vi**

ABSTRAK **vii**

ABSTRACT **viii**

DAFTAR ISI **x**

DAFTAR GAMBAR **xi**

DAFTAR TABEL **xii**

I PENDAHULUAN **1**

1.1 Latar Belakang Masalah 1

1.2 Rumusan Masalah 3

1.3 Pembatasan Masalah 3

1.4 Tujuan Penelitian 3

1.5 Manfaat Penelitian 4

II KAJIAN PUSTAKA **5**

2.1 Sejarah *Search Engine* 5

2.2 *URL* 9

2.3 *HTML* 11

2.4 *Graph* 11

2.5 *Breadth First Search* 14

2.6 Definisi *Search Engine* 17

2.7 Arsitektur *Search Engine* 18
2.8 *Web Crawler* 20
2.8.1 *Importance Metrics* 20
2.8.2 Model *Crawler* 23
2.9 Arsitektur *Web Crawler* 25
2.10 Algoritma *Crawling* 26

III DESAIN MODEL **29**
3.1 Desain *Crawler* 29
3.2 Desain Eksperimen 29
3.3 Arsitektur Diagram 30
3.4 *Flowchart* Algoritma 31
3.5 Desain *Entity Relationship Diagram* 32
3.6 Domain Pencarian 33
3.7 Parameter Keberhasilan 34

IV UJI COBA DAN HASIL UJI COBA **36**

V KESIMPULAN DAN SARAN **37**
5.1 Kesimpulan 37
5.2 Saran 38

DAFTAR PUSTAKA **39**

DAFTAR GAMBAR

Gambar 2.1 *market cap search engine* tahun 2010 hingga 2020 (Statista, 2020) 9

Gambar 2.2 Ilustrasi *Graph* 12

Gambar 2.3 *Adjacent* 13

Gambar 2.4 Pengoperasian BFS pada *undirected graph* (Cormen et al., 2009) 17

Gambar 2.5 *High Level Google Architecture* (Brin & Page, 1998) 19

Gambar 2.6 *High Level Architecture of Web Crawler* (Castillo, 2005) . . . 25

Gambar 3.1 Arsitektur Diagram 30

Gambar 3.2 *Flowchart* Algoritma 31

Gambar 3.3 *entity relationship diagram* 32

Gambar 3.4 Ilustrasi *Page Map Graph* 35

Gambar 3.5 Ilustrasi *Ilustrasi Site Map Graph* 35

DAFTAR TABEL

Tabel 2.1 *Base* dan *Output* pada *URL* (WHATWG, 2020) 10

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

Search engine merupakan sebuah program yang digunakan untuk menjelajahi dan mencari informasi dari web. Terdapat beberapa komponen algoritma yang membangun arsitektur *Search engine* seperti *Web crawler*, *Page rank*, dan *indexer*. Dalam proses pencarian web yang dilakukan *Search engine* tahap pertama yang di lakukan adalah *Web crawler* menjelajahi dan mengekstraksi data-data dari list *url* lalu menyimpan data tersebut dan data lain yang terkait ke dalam database. Data yang disimpan akan di-index, diberikan skor dan di urutkan melalui algoritma *pagerank* (Brin & Page, 1998).

Dari penjelasan diatas dapat disimpulkan bahwa *Web Crawler* merupakan komponen penting dalam pembuatan arsitektur *Search engine* secara keseluruhan. Penelitian yang telah dilakukan oleh Lazuardy Khatulisitwa telah berhasil mengimplementasikan *Web crawler* yang dibuat oleh Muhammad Fathan Qorriba kedalam arsitektur *Search engine* yang berjalan. *Web crawler* yang di buat oleh Fathan mengimplentasikan algoritma *Breadth First Search* dengan modifikasi algoritma *similiarity based* untuk meningkatkan akurasi dari crawling dan parsing suatu halaman. *Web crawler*.

Algoritma *Modified Similarity-Based* yang digunakan oleh fathan untuk memperbaiki akurasi dari *Breadth First Search* memanfaatkan konsep penyimpanan *queue* dalam melakukan proses *crawling*. Dalam proses tersebut *crawler* akan menyimpan 2 jenis *queue* yaitu, *hot queue* untuk menyimpan *url* yang mengandung kata *anchor* sedangkan *url queue* digunakan untuk menyimpan *url* lain (Cho et al.,

1998). Proses ini dapat membantu *crawler* untuk mengunjungi dan melakukan *crawling* ke dalam *page* yang terdapat di *hot queue* terlebih dahulu, bila *page* yang berkaitan dengan kata *anchor* di kunjungi terlebih dahulu maka *child page*-nya kemungkinan besar akan memiliki konten yang berkaitan dengan kata *anchor* tersebut (Cho et al., 1998). Untuk arsitektur dari *crawler* yang di kembangkan oleh Fathan, menggunakan *python* sebagai bahasa pemograman, untuk *library* pendukung yang digunakan adalah *beautifulsoup4* untuk melakukan *parsing* dari halaman *website*, *request* untuk mengirimkan request kepada halaman *website* yang ingin di *scrape*, dan *regex*.

Selain test menilai efisiensi algoritma yang digunakan, dalam proses pembuatan skripsi ini juga dilakukan proses *profiling* untuk menemukan faktor apa saja yang menghambat performa *crawler*. Proses *profiling* yang dilakukan menggunakan *pycharm* sebagai tools pembantu dan target proses yang dijalankan *profiling* merupakan proses *crawler* dalam file *crawl.py*.

Hasil Profiling	
Name	Time (ms)
GET Request	27429110.4
BeautifulSoup 4	21103605.3
Get Queue urls	601870632.6
Opening DB Connection	497664.1
Inserting page linking into DB	84778953.0

Dari hasil *profiling* dapat dilihat bahwa salah satu *bottleneck* terbesar adalah proses pemanggilan *request method* terhadap *page*, proses ini memakan waktu 27429110.4 ms yaitu sekitar 96 persen dari keseluruhan waktu dari pemanggilan function BFS. Salah satu rekomendasi perbaikan yang dapat digunakan adalah

dengan memanfaatkan protokol lain dalam proses pemanggian tersebut. Dalam arsitektur *crawler* yang ada sekarang pemanggilan *page* dilakukan menggunakan protokol *HTTP/1.1*, protokol ini memiliki penerus yang lebih cepat yaitu *HTTP/2.0*. Hal lain yang dapat di simpulkan dari hasil profiling diatas adalah blocker lain terdapat di pemanggilan dan penggunaan *library beautifulsoup4* dalam melakukan parsing HTML dari halaman Website tersebut, hal yang mungkin dilakukan untuk menangani hal tersebut adalah dengan menggunakan library parsing yang lebih cepat dan metode parsing yang lebi hemat waktu dan memori dibanding dengan metode tersebut.

1.2 Rumusan Masalah

Berdasarkan uraian pada latar belakang yang di utarakan di atas, maka perumusan masalah pada penelitian ini adalah Bagaimana cara mendesain perancangan *crawler* sebagai pendukung pada *search engine*?

1.3 Pembatasan Masalah

Adapun batasan-batasan masalah yang digunakan agar lebih terarah dan sesuai dengan yang diharapkan serta terorganisasi dengan baik adalah:

1. pembuatan sebagian arsitektur *search engine* yaitu *crawling*. Algoritma *crawler* yang akan dibuat mengacu pada model algoritma *Google* awal.
2. *Web crawler* hanya akan *crawling* pada *website statis*.

1.4 Tujuan Penelitian

1. Membuat *crawler* yang dipakai untuk kebutuhan *search engine*.

2. Untuk mengetahui arsitektur *search engine*.

3. Untuk mempelajari cara kerja *crawling*.

1.5 Manfaat Penelitian

1. Bagi penulis

   Menambah pengetahuan dibidang *information retrieval* khususnya mengenai *search engine* dan *crawling*, mengasah kemampuan *programming*, dan memperoleh gelar sarjana dibidang Ilmu Komputer. Selain itu, penulisan ini juga merupakan media bagi penulis untuk mengaplikasikan ilmu yang didapat di kampus ke kehidupan masyarakat.

2. Bagi Program Studi Ilmu Komputer

   Penelitian ini menjadi langkah awal penelitian *search engine* berikutnya, dan dapat memberikan gambaran bagi seluruh mahasiswa khususnya bagi mahasiswa program studi Ilmu Komputer Universitas Negeri Jakarta tentang proses pembuatan desain perancangan *crawler* sebagai pendukung pada *search engine*.

3. Bagi Universitas Negeri Jakarta

   Menjadi pertimbangan dan evaluasi akademik khususnya Program Studi Ilmu Komputer dalam penyusunan skripsi sehingga dapat meningkatkan kualitas akademik di program studi Ilmu Komputer Universitas Negeri Jakarta serta meningkatkan kualitas lulusannya.

BAB II

KAJIAN PUSTAKA

2.1 Sejarah *Search Engine*

Mesin pencari pertama bernama *Archie* atau disebut "*archive*" tanpa menyebut huruf "v" yang dibuat 1990. J.Peter Deutsch, Bill Heelan, dan Alan Emtage merupakan mahasiswa Universitas McGill jurusan ilmu komputer di Kanada, mereka bersama-sama membuat mesin pencari ini. Database *Archie* terdiri dari direktori file yang berisi ratusan sistem. *Archie* tidak dapat mengindeks isi konten dari sebuah situs. Melainkan, *Archie* secara berkala menjangkau semua situs *File Transfer Protocol* (FTP) yang tersedia, kemudian membuat daftar-daftar file dan membuat indeks yang dapat dicari. Perintah untuk mencari di *Archie* menggunakan perintah *UNIX*, dan dibutuhkan pengetahuan tentang *UNIX* untuk menggunakannya secara maksimal (Seymour et al., 2011).

Tahun 1991 muncul aplikasi bernama *Gopher*. Kemunculan *Gopher* juga membawa program pencarian yang bernama *Jughead* dan *Veronica*. Cara kerjanya mirip dengan *Archie*, *Jughead* dan *Veronica* mencari nama judul dan file yang disimpan dalam sistem indeks *Gopher*. *Gopher* dibuat oleh Mark McCahill di Universitas Minnesota pada tahun 1991. Dibuatnya *Gopher* bertujuan untuk mencari, mengambil, dan mendistribusikan file lewat Internet. *Gopher* menerapkan tingkatan yang lebih kuat pada informasi yang disimpan di dalamnya dan mempunyai sejumlah fitur yang tidak didukung oleh Web. *Gopher* merupakan sistem protokol dan sebelumnya berupa *World Wide Web*, memungkinkan *file* teks berbasis server diatur secara hierarki, dan nyaman terlihat oleh pengguna yang menggunakan aplikasi *Gopher* dengan akses server di komputer jarak jauh. Awalnya

Browser *Gopher* hanya dapat menampilkan *file* berbasis teks, kemudian terdapat pengembangan seperti *Hyper Gopher* yang mampu menangani format grafik sederhana (Seymour et al., 2011).

Setelah mesin pencari pertama muncul di tahun 1990, tiga tahun berikutnya di tahun 1993 pada bulan November, muncul mesin pencari bernama *Aliweb*. Mesin pencari kedua ini tidak menggunakan *web robot*, dan mengizinkan pengguna menuliskan alamat indeks file di situs pengguna, sehingga mesin pencari dapat memasukkan Halaman Web dan menambahkan kata kunci dan deskripsi halaman yang dituliskan pengguna. *Aliweb* tidak secara otomatis mengindeks situs, Jika pengguna ingin sebuah situs terindeks pada mesin pencari ini, maka pengguna harus menulis file khusus dan mendaftarkannya ke *Aliweb Server* (Seymour et al., 2011).

Jumpstation dirilis tahun 1993 pada bulan Desember. *Jumpstation* mencari halaman-halaman web dan membangun indeksnya dengan menggunakan *web robot*. *Jumpstation* adalah tool pertama yang menggunakan tiga fitur penting dari *web search engine*, yaitu *searching*, *indexing*, dan *crawling*. Karena sumberdaya pada platformnya sedikit terbatas, pengindeksan dan pencariannya dibatasi pada *headings* dan judul yang ditemukan di *website* oleh *crawler* (Seymour et al., 2011).

Search engine pertama yang menyediakan pencarian teks lengkap adalah *WebCrawler*. *Search engine* berbasis *crawler* ini muncul pada tanggal 20 April 1994 dan dibuat oleh Brian Pinkerton di Universitas Washington (Seymour et al., 2011). Para pengguna dapat mencari setiap kata yang terdapat pada halaman web, dan ini menjadi standar utama pada search engine sejak saat itu.

Banyak *search engine* yang kemudian muncul dan bersaing untuk memperebutkan popularitas. Beberapa diantaranya *Magelan search engine*, *AltaVista*, *Northern Light*, *Inktomi*, *Infoseek*, dan *Excite*. Orang-orang menganggap *Yahoo!* sebagai cara yang paling favorit untuk mencari halaman web, tetapi fungsi

pencariannya beroperasi di direktori website *Yahoo!*, bukan teks lengkap yang disalin dari *website*.

AltaVista pernah menjadi salah satu *search engine* paling populer, muncul pada tahun 1995. Michael Burrows yang menulis pengindeksnya dan Louis Monier yang memegang *crawler*. *AltaVista* memiliki server komputasi yang paling kuat, dan membuat *AltaVista* menjadi *search engine* tercepat dan dapat menangani jutaan hit dalam sehari. Satu fitur pada *AltaVista* yang menarik dibandingkan *search engine* sebelumnya, yaitu pengguna dapat mengetikkan frasa atau pertanyaan, dan mendapatkan respon yang sesuai. Tetapi popularitas *AltaVista* berkurang dengan munculnya *Google*.

Ask Jeeves (*Ask*) adalah *searh engine* yang dibuat oleh David Warthen dan Garrett Gruener, didirikan tahun 1996 di Barkeley, California (Seymour et al., 2011). Ide awal *Ask* muncul supaya memudahkan pengguna menemukan jawaban dari pertanyaan yang ditanyakan menggunakan bahasa yang umum digunakan. Saat ini *Ask.com* masih tersedia, terdaoat fitur tambahan, yaitu dapat menerima pertanyaan matematika, kamus, dan pertanyaan konversi.

Perusahaan *Netscape* di tahun 1996 ingin memberikan kesepakatan ekslusif kepada satu *search engine* untuk menjadi *search engine* utama di *web browser Netscape*. Keputusan yang di ambil *Netscape* saat itu terdapat lima *search engine* utama, dan setiap *search engine* mendapat bayaran lima juta dolar per tahun. Lima *search engine* tersebut adalah *Yahoo!*, *Magellan*, *Lycos*, *Infoseek*, dan *Excite* (Seymour et al., 2011).

Google resmi muncul pada tahun 1998 oleh Sergey Brin dan Larry Page, mahasiswa Universitas Stanford di California. Untuk meningkatkan hasil kualitas penelusuran, Brin dan Page memperkenalkan PageRank, sebuah metode untuk menghitung peringkat setiap halaman web (Page et al., 1999). *Google search engine*

telah meminimalisir hasil penelusuran sampah di hasil penelusuran teratas, memudahkan pengguna untuk mencari informasi yang ada di web. Mulai sekitar tahun 2000, *Google search engine* menjadi terkenal hingga saat ini.

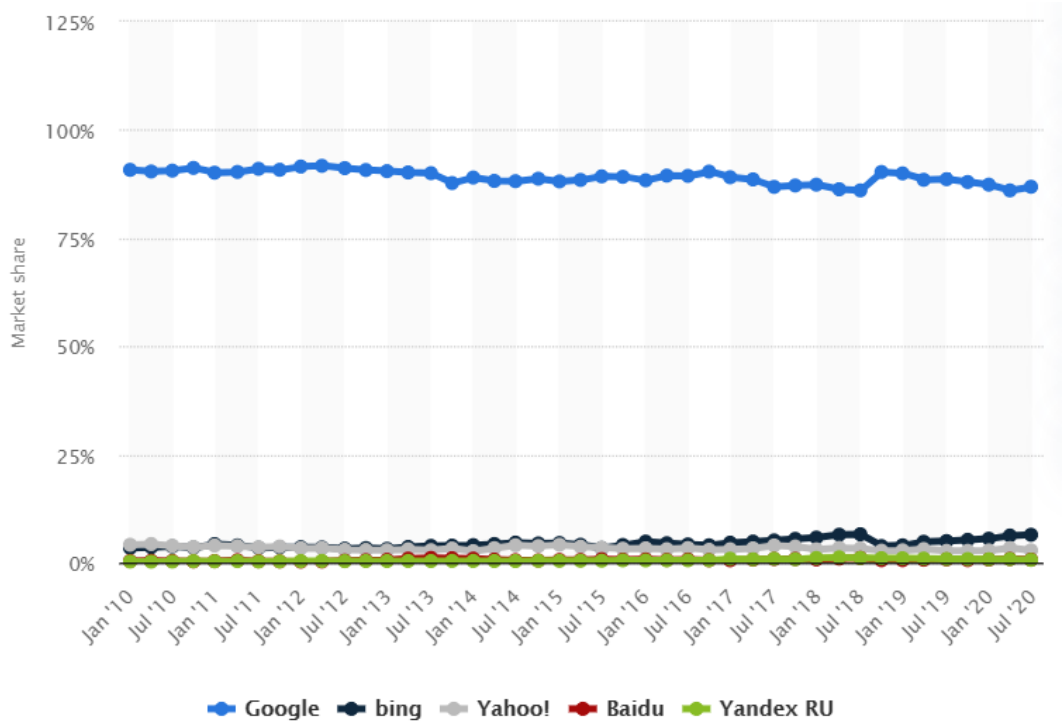
Diawal abad ke 20 pada tanggal 18 Januari 2000, muncul *search engine* bernama *Baidu*. Baidu berpusat di Beijing, RRC. *Baidu* merupakan *search engine* untuk *website*, file audio dan gambar berbahasa Cina dan Jepang. *Baidu* merupakan perusahaan Cina yang masuk dalam indeks NASDAQ-100 pertama kali. Robin Li dan Eric Xu merupakan pendiri perusahaan *Baidu*.

Yahoo! menggunakan *Google search* hingga tahun 2004, dan setelah itu membangun *search engine* sendiri. Sebelumnya, *Yahoo!* membeli *Overture* yang memiliki *search engine AlthaWeb* dan *AltaVista*, walaupun memiliki banyak *search engine*, *Yahoo!* tidak menggunakan *search engine* tersebut untuk situs web utama mereka, dan *Google search* digunakan untuk hasil penelusurannya. *Yahoo! search* menggabungkan kemampuan semua perusahaan *search engine* yang mereka peroleh, dan dengan penelitian yang ada, kemudian menggabungkannya kedalam satu *search engine*.

MSN search diluncurkan pertama kali oleh *Microsoft* pada tahun 1998, menggunakan *Inktomi* sebagai hasil penelusurannya (Seymour et al., 2011). *Microsoft* memiliki *web crawler* sendiri bernama *msnbot*, dan perlahan akan mulai menciptakan teknologi *search engine* sendiri di tahun 2004. *Bing* merupakan *Search engine Microsoft*, resmi muncul pada tahun 2009 di awal bulan Juni. Kemudian satu bulan lebih setelahnya tanggal 29 Juli 2009, *Microsoft* dan *Yahoo!* membuat kesepakatan, teknologi *Microsoft Bing* akan mendukung hasil pencarian *Yahoo! search*.

DuckDuckGo didirikan pada Februari 2008 di Valley Forge, PA (USA). *Search engine* ini dibuat sendiri oleh Gabriel Weinberg. Resmi diluncurkan 25 September

2008. Situs komunitas *DuckDuckGo* dibangun oleh Weinberg pada bulan Juli 2010. Komunitas ini dibentuk supaya pengguna dapat membahas, melaporkan masalah dan mendiskusikan tentang "*open sourcing the code*". Tighe Caine merupakan karyawan pertama *DuckDuckGo* pada bulan september 2011.



Gambar 2.1: *market cap search engine* tahun 2010 hingga 2020 (Statista, 2020)

Dari data pada gambar 2.1 dapat disimpulkan bahwa, tiga *search engine* yang paling banyak digunakan pada tahun 2010 hingga 2020, yaitu *Google*, *Yahoo*, dan *Bing*. *Google* selama sepuluh tahun selalu berada di peringkat pertama dengan persentase diatas 85%. Kemudian diikuti Bing yang pada bulan Juli 2020 dengan presentase 6,43%. Dan Yahoo dengan presentase 2,84% pada bulan Juli 2020.

2.2 *URL*

URL adalah singkatan dari *Uniform Resource Locator*. *URL* juga sering dikenal dengan istilah "alamat web". Fungsi *URL* adalah menentukan lokasi sumber

daya (seperti *website*) di internet. Selain itu, *URL* juga menentukan cara mengambil sumber daya, atau dikenal sebagai "*protocol*", seperti *HTTP*, *HTTPS*, *FTP*, dan sebagainya. *URL* adalah teks yang dapat dibaca manusia "*human-readable*" yang dirancang untuk menggantikan angka (alamat IP). *URL* digunakan komputer untuk berkomunikasi dengan server. *URL* juga mengidentifikasi struktur file di situs web tertentu (Moz, 2020).

Tabel 2.1: *Base* dan *Output* pada *URL* (WHATWG, 2020)

Input	**Base**	**Valid**	**Output**
https:contoh.org		No	https://contoh.org/
hello:world	https://contoh.com/	Yes	hello:world
\contoh\..\data/.\	https://contoh.com/	No	https://contoh.com/data/
file:///C|/data		No	file:///C:/data
..	file:///C:/data	Yes	file:///C:/
https://CNTH.com/../y		Yes	https://cnth.com/y
https://co ntoh.org/		No	*Failure*
contoh	https://contoh.com/data	Yes	https://contoh.com/contoh
contoh		No	*Failure*
http://[www.cth.com]/		No	*Failure*
https://cth.com//		Yes	https://cth.com//

URL terdiri dari protokol, nama domain, dan *path*. *URL* memiliki format dasar berikut: *protocol*://nama-domain.*top-level-domain*/*path*. *Path* merupakan struktur subfolder spesifik tempat halaman web berada.

Protocol menunjukkan bagaimana *browser* harus mengambil informasi dari sumber daya. Standar web adalah "http://" atau "https://" ("s" adalah singkatan dari "*secure*"), tetapi juga dapat mencakup hal-hal seperti "mailto:" (untuk membuka klien email default Anda) atau "ftp:" (untuk menangani transfer file).

Nama domain (atau *hostname*) adalah nama yang dapat dibaca manusia

"*human-readable*" dari lokasi spesifik tempat sumber daya berada. Top-level domain merupakan kategori untuk *website*. Seperti .com, .edu untuk situs pendidikan, .gov untuk situs pemerintah, dan lain sebagainya.

URL juga berisi hal-hal seperti folder atau subfolder tertentu yang ada di *website* tertentu, parameter apa pun (seperti *click tracking* atau *session ID*) yang mungkin disimpan di URL, dan *anchor* yang mengizinkan pengunjung *website* untuk melompat ke titik tertentu di sumber. Tabel 2.1 merupakan contoh *base* dan *output* pada *URL* (WHATWG, 2020).

2.3 *HTML*

HTML merupakan singkatan dari *Hypertext Markup Language*, adalah bahasa yang mendeskripsikan struktur *website*. Dokumen *HTML* memiliki dua bagian utama:

1. *Head*. Elemen *head* berisi *title* dan *meta data* dari dokumen web.
2. *Body*. Elemen *body* berisi informasi yang berupa tampilan di halaman web.

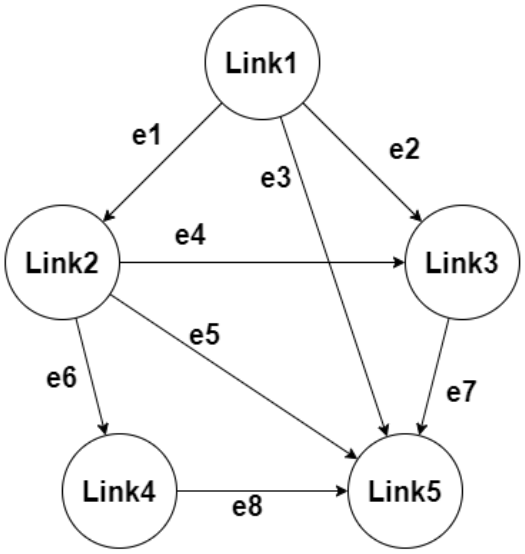
Pada sebuah halaman web, tag pertama (khususnya, <html>) menunjukkan penanda bahasa yang digunakan untuk dokumen. Tag <head> berisi informasi tentang halaman web. Terakhir, konten muncul di tag <body> (TechnaCenterLLC, 2020).

2.4 *Graph*

Suatu graph didefinisikan dengan himpunan verteks & himpunan sisi (*edge*). Verteks menyatakan entitas-entitas data & sisi menyatakan keterhubungan antara verteks. Biasanya melambangkan suatu *graph* G dipakai notasi matematis.

$$G = (V, E) \tag{2.1}$$

Dimana, G adalah *graph*. V merupakan himpunan verteks atau merepresentasikan sebuah *link* & E himpunan sisi yang terdefinisi antara pasangan-pasangan verteks. Sebuah sisi antara verteks x & y ditulis x, y. Suatu *graph* $P = (V_1, E_1)$ dianggap *subgraph* berdasarkan *graph* G bila V_1 merupakan himpunan bagian berdasarkan V & E_1 himpunan bagian berdasarkan E.



Gambar 2.2: Ilustrasi *Graph*

Graph pada gambar 2.2 dapat dinyatakan sebagai *graph* $G = (V_1, E_1)$ dimana $V = \{Link1, Link2, Link3, Link4, Link5\}$ dan $E = \{\{Link1, Link2\}, \{Link1, Link3\}, \{Link1, Link5\}, \{Link2, Link3\}, \{Link2, Link5\}, \{Link2, Link4\}, \{Link3, Link5\}, \{Link4, Link5\}\}$.

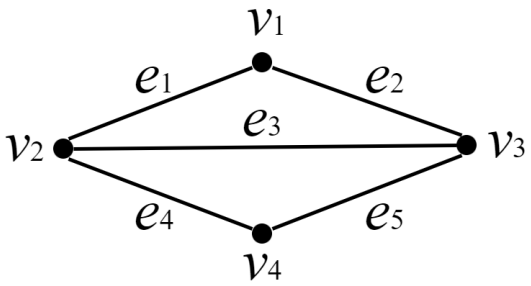
Terdapat beberapa istilah yang berkaitan menggunakan *graph*, yaitu:

1. *Edge*. Himpunan garis yang menghubungkan tiap *node* / *vertex* / *link*.
2. *Vertex*. Himpunan *node* / titik dalam sebuah *graph*. Merepresentasikan sebuah *link*.
3. *Weight*. Sebuah *graph* $G = (V, E)$ dianggap sebuah *graph* berbobot (*weight graph*), bila masih ada sebuah fungsi bobot bernilai real W dalam himpunan

E. nilai $W(e)$ disebut bobot untuk sisi $e, \forall e \in E$. Graph berbobot tersebut juga dinyatakan sebagai $G = (V, E, W)$.

$$W : E \rightarrow R \tag{2.2}$$

4. *Adjacent*. Merupakan 2 buah titik dikatakan berdekatan (adjacent) bila 2 buah titik tadi terhubung menggunakan sebuah sisi. Pada gambar 2.3, Sisi $e_3 = v_2v_3$ *insident* menggunakan titik v_2 & titik v_3, namun sisi $e_3 = v_2v_3$ tidak *insident* menggunakan titik v_1 & titik v_4.



Gambar 2.3: *Adjacent*

5. *Walk*. Merupakan barisan simpul dan ruas yang berganti-ganti. Banyaknya ruas disebut Panjang *Walk*. *Walk* bisa ditulis lebih singkat menggunakan hanya menulis gugusan ruas.

6. *Trail*. *Walk* yang menggunakan seluruh ruas pada barisannya berbeda.

7. *Path*. Jalur *walk* yang seluruh simpul pada barisannya berbeda, jadi suatu path pasti merupakan sebuah *trail*.

8. *Cycle* atau sirkuit merupakan suatu *trail* tertutup menggunakan derajat setiap simpulnya = 2, sebuah trail menggunakan awal & akhir dalam simpul yang sama.

2.5 *Breadth First Search*

Breadth first search merupakan salah satu algoritma paling sederhana untuk mencari *graph*. Jika diberikan *graph* $G = (V, E)$ dan dimulai dari *vertex* s, *breadth first search* secara sistematis mengeksplorasi tepi G untuk menemukan setiap *vertex* yang dapat dijangkau dari s. Dapat menghitung jarak dari s ke setiap simpul yang dapat dijangkau. Dan juga menghasilkan "*breadth first tree*" dengan *root* s yang berisi semua simpul yang bisa dijangkau. Untuk setiap *vertex* v yang dapat dijangkau dari s, *simple path* dalam *breadth first tree* dari s ke v sesuai dengan *shortest path* dari s ke v dalam G, yaitu *path* yang berisi jumlah sisi terkecil.

Untuk melacak progres, *breadth first search* mewarnai setiap *vertex* dengan warna putih, abu-abu, dan hitam. Semua *vertex* dimulai dengan warna putih, kemudian menjadi abu-abu dan kemudian hitam. Sebuah *vertex* saat pertama kali ditemukan selama pencarian akan menjadi warna selain putih. *Vertex* abu-abu dan hitam adalah *vertex* yang telah ditemukan, tetapi *breadth first search* membedakan keduanya. Jika $(u, v) \in E$ dan simpul u hitam, maka *vertex* v antara abu-abu atau hitam. artinya, semua *vertex* yang *adjacent* dengan *vertex* hitam telah ditemukan. *Vertex* abu-abu mungkin memiliki beberapa *vertex* putih yang *adjacent*, mewakili perbatasan antara *vertex* yang ditemukan dan yang belum ditemukan.

Algorithm 1 *BFS(G, s)* (Cormen et al., 2009)

1: **for** each vertex $u \in G.V - \{s\}$ **do**
2: $u.color =$ WHITE
3: $u.d = \infty$
4: $u.\pi =$ NIL
5: **end for**
6: $s.color =$ GRAY
7: $s.d = 0$
8: $s.\pi = NIL$
9: $Q = \infty$
10: ENQUEUE(Q, s)
11: **while** $Q \neq \emptyset$ **do**
12: $u =$ DEQUEUE(Q)
13: **for** each $v \in G.Adj[u]$ **do**
14: **if** $v.color ==$ WHITE **then**
15: $v.color ==$ GRAY
16: $v.d = u.d + 1$
17: $v.\pi = u$
18: ENQUEUE(Q, v)
19: $v.color ==$ BLACK
20: **end if**
21: **end for**
22: **end while**

Breadth first search membangun breadth first tree, awalnya hanya berisi *root*, yang merupakan sumber *vertex* s. Kapanpun pencarian menemukan *vertex* putih dalam proses *scanning* daftar *adjacency* dari *vertex* u yang sudah ditemukan, *vertex*

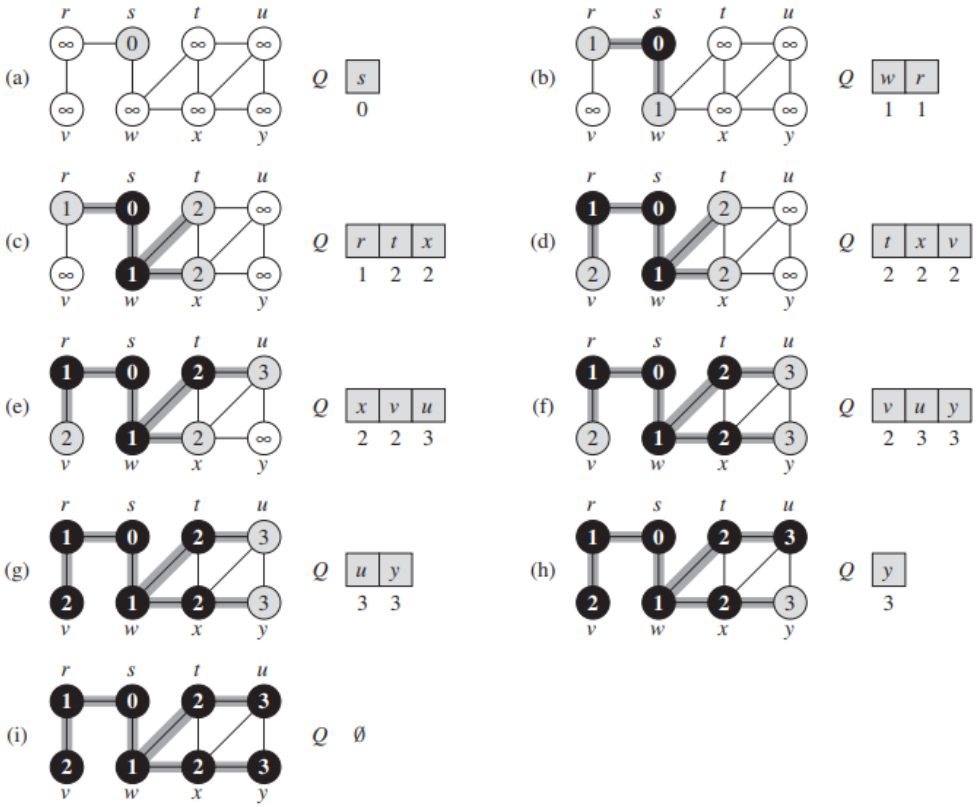
v dan sisi (u, v) ditambahkan ke *tree*. u adalah *predecessor* atau *parent* dari *breadth first tree*. Karena *Vertex* ditemukan paling banyak dalam satu waktu, maka *vertex* memiliki paling banyak satu *parent*. Hubungan *ancestor* dan *descendant* di *breadth first tree* didefinisikan relatif terhadap root s. jika u berada di *simple path* dalam *tree* dari *root* s ke *vertex* v, maka u adalah *ancestor* v, dan V dan merupakan *descendant* dari u.

Prosedur *breadth first search* (BFS) pada algoritma 1 mengasumsikan bahwa graph input $G = (V, E)$ diwakili menggunakan *adjacency lists*. Melampirkan beberapa atribut tambahan ke setiap *vertex* dalam *graph*. Warna setiap simpul disimpan u/inV di atribut *u.color* dan *predecessor* u di atribut $u.\pi$. Jika u tidak memiliki *predecessor* (misalnya, jika $u = s$ atau u belum ditemukan), maka $u.\pi = NIL$. Atribut $u.d$ menahan jarak dari sumber s ke *vertex* u yang dihitung oleh algoritma. Algoritma ini juga menggunakan *first-in, first-out queue* Q untuk mengatur himpunan simpul abu-abu (Cormen et al., 2009).

Prosedur algoritma BFS (algoritma 1) dijelaskan sebagai berikut. Dengan pengecualian sumber *vertex* s, baris pertama sampai baris 5 mewarnai setiap *vertex* putih, *set* $u.d$ menjadi tak terhingga untuk setiap *vertex* u, dan *set parent* dari setiap *vertex* menjadi NIL. baris 6 mewarnai abu-abu, karena akan ditemukan saat prosedur dimulai. baris 7 menginisialisasi $s.d$ ke 0, dan baris 8 menetapkan *predecessor* sumber menjadi NIL. Baris 910 menginisialisasi Q ke *queue* yang hanya berisi *vertex* s.

Loop pada baris 11-22 berulang selama masih ada *vertex* abu-abu. Pada baris 10, *queue* Q terdiri dari himpunan *vertex* abu-abu. Sebelum iterasi pertama, satu-satunya *vertex* abu-abu, dan satu-satunya *vertex* di Q, adalah sumber *vertex* s. Baris 11 menentukan *vertex* abu-abu u di kepala *queue* Q dan menghilangkannya dari Q. *for loop* dari baris 1321 mempertimbangkan setiap *vertex* dalam daftar *adjacency*

u. Jika berwarna putih, maka belum ditemukan, dan prosedur untuk menemukannya dengan mengeksekusi baris 15-19. Prosedur mewarnai *vertex* abu-abu, menyetel jarak $v.d$ ke $u.d + 1$, mencatat u sebagai induknya, dan menempatkannya di ujung queue Q. Setelah prosedur memeriksa semua *vertex* pada daftar adjacency u, kemudian menghitamkan u pada baris 19. Gambar 2.4 merupakan progres BFS pada *sample undirected graph*.



Gambar 2.4: Pengoperasian BFS pada *undirected graph* (Cormen et al., 2009)

2.6 Definisi *Search Engine*

Menurut Seymour et al. (2011), *Search engine* atau mesin pencari adalah sebuah program yang dibuat untuk melakukan pencarian pada situs web. Menurut William dan Sawyer (2001), *Search engine* adalah program yang memungkinkan pengguna buat mengajukan pertanyaan atau memakai kata kunci untuk membantu

mencari informasi pada web.

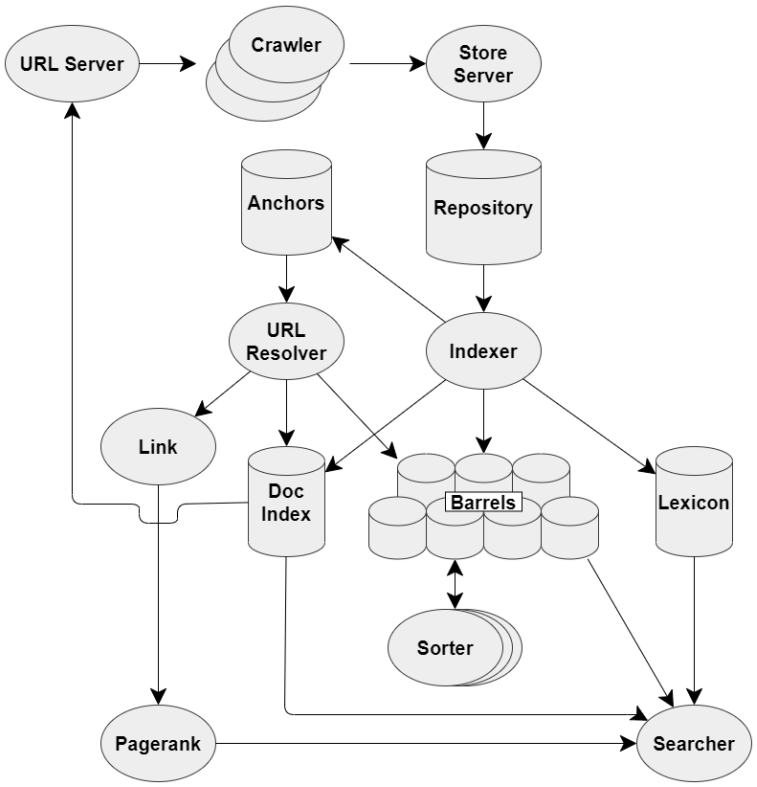
Search engine merupakan sebuah program yang bisa diakses melalui internet yang kegunaannya untuk membantu pengguna internet mencari aneka macam yang ingin diketahuinya. *Search engine* pada dasarnya merupakan sebuah halaman web, tetapi perannya berfokus untuk mengumpulkan dan mengorganisir berbagai informasi di internet. Salah satu contoh *search engine* yang sangat populer saat ini adalah *Google*.

2.7 Arsitektur *Search Engine*

Search engine bekerja dengan menggunakan cara menyimpan seluruh informasi dari berbagai *website*. Halaman-halaman tersebut didapatkan dengan menggunakan program khusus yang disebut *Web Crawler*, isi di setiap halaman lalu dianalisis buat memilih bagaimana halaman tadi akan diindeks. Gambar 2.5 merupakan *High Level Google Architecture* (Brin & Page, 1998).

Web crawling (mengunduh halaman web) dilakukan oleh beberapa *crawler* yang terdistribusi. Terdapat *URL server* yang mengirimkan daftar *URL* untuk diambil ke *crawler*. Halaman web yang sudah diambil setelah itu dipindahkan ke *store server* dan *store server* menyimpan halaman web ke dalam *Repository*. Semua halaman web memiliki nomor ID yang dinamakan *docID*, ditetapkan setiap kali *URL* baru diurai dari halaman web. Pengindeksan dilakukan oleh *indexer* dan *sorter*. Fungsi dari *indexer* yaitu: membaca repositori, membuka kompresi dokumen, dan mengurainya. Setiap dokumen kemudian diubah menjadi sekumpulan kejadian kata yang disebut *hits*. *Hits* berfungsi untuk merekam kata, menyimpan posisi dalam dokumen, menyimpan perkiraan ukuran *font* dan kapitalisasi. *Indexer* mendistribusikan *hits* ke dalam satu set "*barrels*", membuat *forward index* yang diurutkan sebagian. *Indexer* melakukan fungsi penting lainnya, yaitu mengurai

semua *link* di setiap halaman web dan menyimpan informasi penting ke dalam *anchors file*. *File* ini berisi informasi yang cukup untuk menentukan darimana dan kemana setiap *link point*, dan berisi teks dari setiap *link*.



Gambar 2.5: *High Level Google Architecture* (Brin & Page, 1998)

URL resolver membaca *anchors file* dan mengubah *URL* relatif menjadi *URL* absolut, serta merubahnya menjadi *docID*. Kemudian menempatkan *anchors text* ke *forward index*, terkait dengan *docID* yang ditunjukkan oleh *anchor*. Dan juga menghasilkan *database of links* yang merupakan pasangan *docID*. *Database link* digunakan untuk menghitung *PageRank* untuk semua dokumen.

Sorter mengambil *barrels* yang telah diurutkan berdasarkan *docID* dan menggunakan *barrels* dengan *wordID* untuk menghasilkan *inverted index*. Hal ini dilakukan di dalam sebuah tempat, sehingga sedikit ruang sementara yang dibutuhkan untuk operasi ini. *Sorter* juga menghasilkan daftar *wordID* dan *offset* ke

dalam *inverted index*. Sebuah program bernama *DumpLexicon* mengambil daftar *wordID* bersama dengan *leksicon* yang dihasilkan oleh *indexer* dan menghasilkan *leksicon* baru yang dipakai oleh *searcher*. *Searcher* dijalankan oleh *web server* & memakai *leksicon* yang dibentuk oleh *DumpLexicon* beserta menggunakan *inverted index* & *PageRank* buat menjawab pertanyaan.

Pada tahun 1998, *Google* menjalankan proses tersebut membutuhkan waktu kurang lebih 9 hari untuk menngunduh data dari 26 juta halaman web. Jika dihitung dalam detik, maka membutuhkan waktu rata-rata sekitar 33,44 halaman per detik. Proses ini selalu diulang untuk mendapat perubahan informasi atau data yang ada di seluruh halaman web. Diimplementasikan hampir seluruhnya menggunakan *C* atau *C++*, tetapi sebagian lainnya, seperti *URL server* dan *crawler* diimplementasikan menggunakan *python*.

2.8 *Web Crawler*

Crawler adalah program yang mengambil halaman Web, biasanya untuk digunakan oleh mesin pencari (Pinkerton, 1994). Secara kasar, *crawler* memulai dengan *URL* untuk halaman awal P_0. Kemudian *Crawler* mengambil P_0, mengekstrak semua *URL* yang terdapat di dalam P_0, dan menambahkannya ke antrian *URL* untuk dipindai. Kemudian *crawler* mengambil *URL* dari antrian (dalam urutan tertentu), dan mengulangi prosesnya (Cho et al., 1998).

2.8.1 *Importance Metrics*

Tidak semua halaman memiliki minat yang sama dengan *crawler's client*. Misalnya, jika *client* sedang membangun database khusus tentang topik tertentu, maka halaman yang merujuk ke topik tersebut lebih penting, dan harus dikunjungi sesegera mungkin. Demikian pula, *search engine* dapat menggunakan jumlah *URL*

Web yang mengarah ke sebuah halaman, disebut *backlink count*, untuk menentukan peringkat hasil *query* pengguna.

Sebuah *Web page* P, dapat ditentukan *importance of the page* $I(P)$ dengan salah satu cara berikut (Cara tersebut dapat dikombinasikan satu sama lain):

1. *Similarity to a Driving Query* Q. Query Q mendorong proses *crawling*, dan $I(P)$ didefinisikan sebagai *textual similarity* antara P dan Q. *Similarity* telah dipelajari dengan baik di komunitas *Information Retrieval* (IR) (Salton, 1989). *Importance metrics* ini disebut $IS(P)$. Untuk menghitung *similarities* dapat dilihat pada setiap dokumen (P atau Q) sebagai vektor berdimensi-m $(w_1, ..., w_n)$. Istilah w_i dalam vektor ini merepresentasikan pentingnya kata ke-i dalam kosakata. Salah satu cara umum untuk menghitung signifikansi w_i adalah mengalikan berapa kali kata ke-i muncul dalam dokumen dengan *inverse document frequency* (*idf*) dari kata ke-i. Faktor *idf* adalah satu dibagi dengan berapa kali kata tersebut muncul di seluruh "*document collection*". Persamaan antara P dan Q kemudian dapat didefinisikan sebagai *inner product* dari vektor P dan Q.

   Jika menggunakan istilah *idf* dalam perhitungan *similarity*, maka diperlukan informasi global untuk menghitung *importance of a page*. Selama proses *crawling*, Tidak dapat melihat seluruh *colection*, dan harus memperkirakan faktor *idf*. Perlu menggunakan $IS'(P)$ untuk mengacu pada estimasi *importance of a page* P, yang berbeda dari *importance of a page* sebenarnya $IS(P)$. Jika faktor *idf* tidak digunakan, maka $IS'(P) = IS(P)$.

2. *Backlink Count*. Nilai $I(P)$ adalah jumlah *link* ke P yang muncul di seluruh Web. *Importance metrics* ini disebut $IB(P)$. Halaman P yang dirujuk oleh banyak halaman lain, lebih penting daripada halaman yang jarang dirujuk. Di

Web, *Backlink Count* atau $IB(P)$ berguna untuk menentukan peringkat hasil *query*, memberikan halaman *end-users* yang lebih mungkin menjadi minat umum.

Untuk mengevaluasi $IB(P)$ membutuhkan penghitungan *backlink* di seluruh Web. *Crawler* dapat memperkirakan nilainya dengan $IB'(P)$, yaitu jumlah *link* ke P yang sudah terlihat saat ini.

3. *PageRank*. $IB(P)$ memperlakukan semua *link* secara setara. Karenanya, *link* dari halaman utama *Yahoo!* dihitung sama dengan *link* dari halaman lain. Namun, karena halaman utama *Yahoo!* lebih penting (memiliki jumlah IB yang jauh lebih tinggi), masuk akal untuk menghargai tautan itu dengan lebih tinggi. *PageRank backlink metric* atau disebut $IR(P)$, secara rekursif mendefinisikan *importance of a page* sebagai jumlah yang terhitung dari *backlink* ke halaman tersebut. *Metric* ini terbukti sangat berguna dalam memberi peringkat hasil *query* pengguna (Page et al., 1999). $IR'(P)$ berfunsi untuk estimasi nilai $IR(P)$ saat hanya memiliki sebagian halaman yang tersedia.

   Secara lebih formal, jika halaman tidak memiliki *outgoing link*, maka diasumsikan bahwa halaman tersebut memiliki *outgoing link* ke setiap halaman Web. Selanjutnya, pertimbangkan halaman P yang dirujuk oleh halaman $T_1, ..., T_n$. Misalkan c_i adalah jumlah *outlink* dari halaman T_i. Dan misalkan d menjadi *damping factor* (yang intuisinya diberikan di bawah). Kemudian, jumlah *backlink* yang terhitung dari halaman P dapat dilihat dari persamaan berikut:

$$IR(P) = (1 - d) + d\left(\frac{IR(T_1)}{c_1} + ... + \frac{IR(T_n)}{c_n}\right) \tag{2.3}$$

Persamaan ini mengarah ke satu persamaan per halaman Web, dengan jumlah yang sama tidak diketahui. Persamaan dapat diselesaikan untuk nilai IR. Dihitung secara iteratif, dimulai dengan semua nilai IR sama dengan 1. Pada setiap langkah, nilai $IR(P)$ baru dihitung dari nilai $IR(T_i)$ lama (dengan menggunakan persamaan 2.3), hingga nilainya didapat.

Salah satu model intuitif untuk *PageRank* adalah dengan membayangkan pengguna "menjelajahi" Web, mulai dari halaman mana pun, dan secara acak memilih link untuk diikuti dari halaman tersebut. Saat pengguna mencapai halaman yang tidak terdapat *outlink*, pengguna akan menuju ke halaman acak. Juga, ketika pengguna berada di sebuah halaman, ada beberapa kemungkinan, d, bahwa halaman yang dikunjungi berikutnya akan benar-benar acak. Nilai $IR(P)$ yang dihitung di persamaan 2.3 memberikan probabilitas bahwa *surfer* acaknya berada di P pada waktu tertentu.

4. *Location Metric*. *importance of a page* P, $IL(P)$ atau *Location Metric* dari halaman P berfungsi dari lokasinya, bukan isinya. Jika *URL* u mengarah ke P, maka $IL(P)$ adalah fungsi dari u. Misalnya, *URL* yang diakhiri dengan "*.com*" mungkin dianggap lebih berguna daripada *URL* dengan akhiran lain, atau *URL* yang berisi string "*home*" mungkin lebih menarik daripada *URL* lainnya. *Location Metric* lain yang terkadang digunakan, menganggap *URL* dengan garis miring (*slash*) yang lebih sedikit, lebih berguna daripada *URL* dengan garis miring lebih banyak. Semua contoh ini adalah *local metric*, karena dapat dievaluasi hanya dengan melihat *URL* u.

2.8.2 Model *Crawler*

Secara umum terdapat tiga model crawler yang dirancang agar *crawler* dapat mengunjungi halaman $I(P)$ tinggi sebelum mengunjungi halaman yang berperingkat

lebih rendah. Tentu saja, *crawler* hanya akan memiliki nilai $I'(P)$ yang tersedia, jadi berdasarkan ini crawler harus menebak halaman $I(P)$ tinggi yang akan diambil selanjutnya.

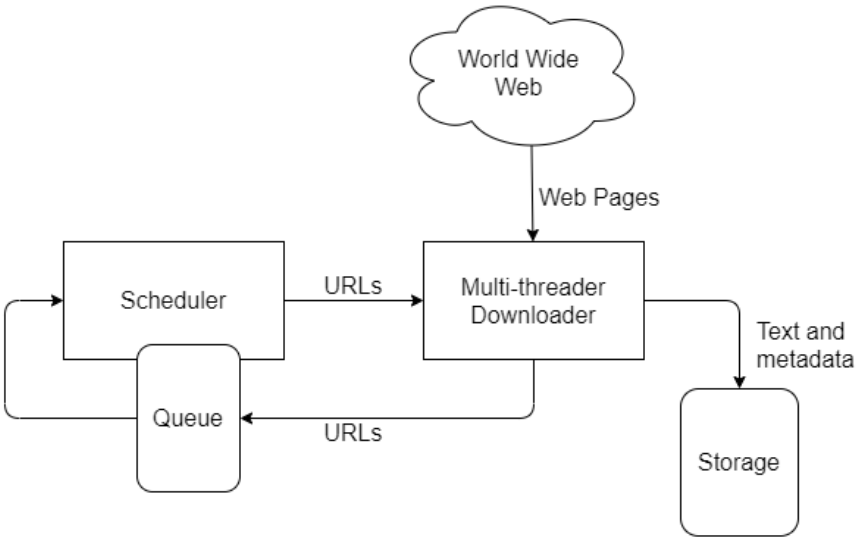
Crawl & Stop. Pada model ini, crawler C memulai halaman awalnya P_0 dan berhenti setelah mengunjungi halaman K. Pada titik ini crawler yang sempurna akan mengunjungi halaman $R_1, ..., R_K$, dengan R_1 adalah halaman dengan nilai kepentingan tertinggi, R_2 adalah halaman tertinggi berikutnya, dan seterusnya. Halaman R_1 sampai R_K disebut sebagai *hot pages*. Halaman K yang dikunjungi oleh *crawler* sebenarnya hanya akan berisi halaman M dengan peringkat lebih tinggi dari atau sama dengan $I(R_K)$. Didefinisikan kinerja *crawler* C menjadi $P_{CS}(C) = (M \cdot 100)/K$. Kinerja *crawler* yang ideal tentu saja 100%. Sebuah crawler yang entah bagaimana berhasil mengunjungi halaman secara acak akan memiliki kinerja $(K \cdot 100)/T$, dimana T merupakan jumlah total halaman di Web.

Crawl & Stop with Threshold. Diasumsikan bahwa *crawler* mengunjungi halaman K. Namun, sekarang diberi target kepentingan G, dan halaman mana pun dengan $I(P) \geq G$ dianggap *hot*. Maka diasumsikan bahwa jumlah total *hot pages* adalah H. Kinerja *crawler*, $P_{ST}(C)$, adalah persentase dari H *hot pages* yang telah dikunjungi ketika *crawler* berhenti. Jika $K < H$, maka *crawler* yang ideal akan memiliki performa $(K \cdot 100)/H$. Jika $K \leq H$, maka *crawler* yang ideal memiliki performa 100%. *Crawler* acak murni yang mengunjungi kembali halaman diharapkan untuk mengunjungi $(H/T) \cdot K$ *hot pages* saat berhenti. Jadi performanya adalah $(K \cdot 100)/T$.

Limited Buffer Crawl. Model ini mempertimbangkan dampak dari penyimpanan terbatas pada proses *crawling*. Diasumsikan bahwa *crawler* hanya dapat menyimpan halaman B dalam *buffer*. Jadi, setelah *buffer* terisi, *crawler* harus memutuskan halaman mana yang akan di-*flush* untuk memberi ruang bagi halaman

baru. *Crawler* yang ideal dapat dengan mudah menjatuhkan halaman dengan nilai $I(P)$ terendah, tetapi *crawler* yang sebenarnya harus menebak halaman mana dalam *buffer* yang pada akhirnya akan memiliki nilai $I(P)$ yang rendah. *Crawler* dibolehkan untuk mengunjungi total T halaman, sama dengan jumlah total halaman Web. Pada akhir proses ini, persentase halaman *buffer* B yang *hot* memberikan kinerja $P_{BC}(C)$. Performa *crawler* ideal dan *crawler* acak serupa dengan performa di kasus sebelumnya.

2.9 Arsitektur *Web Crawler*



Gambar 2.6: *High Level Architecture of Web Crawler* (Castillo, 2005)

Web crawler adalah bagian utama dari *search engine*. *Crawler* harus memiliki strategi *crawling* yang baik dan juga membutuhkan arsitektur yang sangat dioptimalkan. Desain sistem harus optimal agar dapat membentuk sistem berkinerja tinggi yang bisa mengunduh ratusan juta *page* selama beberapa minggu. *High level architecture of Web crawler* ditunjukkan pada Gambar 2.6, membutuhkan *scheduler* dan *multi-threader downloader*. Dua struktur data utama adalah *Web page* (*text*) dan

URL queue.

2.10 Algoritma *Crawling*

Algorithm 2 *Typical Crawling Model* (Castillo, 2005)

Require: $p_1, p_2, ..., p_n$ starting URLs

1: $Q = p_1, p_2, ..., p_n$, queue of URLs to visit.
2: $V = \emptyset$, visited URLs.
3: **while** $Q \neq \emptyset$ **do**
4: Dequeue $p \in Q$, select p according to some criteria.
5: Do an asynchronous network fetch for p.
6: $V = V \cup \{p\}$
7: Parse p to extract text and extract outgoing links
8: $\Gamma^+(p) \leftarrow$ pages pointed by p
9: **for** each $p' \in \Gamma^+(p)$ **do**
10: **if** $p' \notin V \wedge p' \notin Q$ **then**
11: $Q = Q \cup \{p'\}$
12: **end if**
13: **end for**
14: **end while**

Pada umumnya, algoritma yang digunakan untuk *crawling* menggunakan *breadth first search*. Algoritma ini merupakan bentuk algoritma *crawling* yang paling sederhana. Idenya, mulai dari sebuah *link*, kemudian terus mengikuti *link* lain yang terhubung. Algoritma *typical crawling model* dapat dilihat pada 2 (Castillo, 2005).

URL page $p_1, p_2,$ hingga p_n dikumpulkan dalam variabel Q. kemudian, satu persatu *page* dilakukan proses *fetch*. Didapatkan *text* dan juga *outgoing link* pada

page yang sudah diambil. *Outgoing link* yang terambil dari proses sebelumnya akan dimasukkan ke dalam variabel Q jika *link* tersebut belum masuk proses *fetch*. Proses ini terus berulang hingga variabel Q kosong.

Desain *crawler* yang baik harus menghindari *overloading website* saat sedang menjalankan prosesnya. *Crawler* juga harus menangani volume data yang sangat besar. Maka, *crawler* harus menentukan urutan *URL* yang akan dipindai terlebih dahulu, karena sumber daya dan waktu yang terbatas. Muncul algoritma *modified similarity-based crawling* untuk mengatasi masalah tersebut. *Modified similarity-based crawling algorithm* (Cho et al., 1998) dapat dilihat pada algoritma 3.

Terdapat perbedaan pada algoritma 2 *typical crawling model* dan algoritma 3 *Modified similarity-based crawling*. Pada algoritma 3 *starting_url* dimasukkan ke *url_queue*. Selain *url_queue*, ada juga *hot_queue*. *hot_queue* merupakan kumpulan *URL hot page*. Suatu *page* dikatakan *hot*, karena memiliki salah satu dari kriteria berikut:

1. Page yang berisi lebih dari 10 kata spesifik (di contohkan kata "*computer*") pada *body* atau 1 kata *computer* di *title*.
2. Kata *computer* terdapat pada *anchor* atau *URL*.
3. *URL* yang memiliki hubungan *outlink* atau *inlink* dengan *hot_page*, masuk kedalam kumpulan *URL* di *hot_queue*, jika jaraknya kurang dari 3 *link/node*.

URL baru yang didapat pada proses *crawling*, dimasukkan ke dalam *url_queue* ataupun *hot_queue*. Kemudian, kumpulan *URL* tersebut diurutkan lagi pada *function reorder_queue*. Algoritma *reorder_queue* menggunakan *backlink count*. Nilai *backlink count* didapat dari banyaknya *URL* yang ditautkan pada *URL* lain. Kemudian *URL* diurutkan berdasarkan nilai *backlink count* tertinggi.

Algorithm 3 *Modified similarity-based crawling* (Cho et al., 1998)

```
1: enqueue(url_queue, starting_url);
2: while (not empty(hot_queue)) and not empty(url_queue) do
3:     url = dequeue2(hot_queue, url_queue);
4:     page = crawl_page(url);
5:     if [page contains 10 or more computer in body or one computer in title] then
6:         hot[url] = TRUE;
7:     end if
8:     enqueue(crawled_pages, (url, pages));
9:     url_list = extract_urls(page);
10:    for each u in url_list do
11:        enqueue(links, (url, u));
12:        if [u not in url_queue] and [u not in hot_queue] and [(u,-) not in
       crawled_pages] then
13:            if [u contains computer in anchor or url] then
14:                enqueue(hot_queue, u);
15:            else if [distance_from_hotpage(u) < 3] then
16:                enqueue(hot_queue, u);
17:            else
18:                enqueue(url_queue, u);
19:            end if
20:        end if
21:        reorder_queue(url_queue);
22:        reorder_queue(hot_queue);
23:    end for
24: end while
```

BAB III

DESAIN MODEL

3.1 Desain *Crawler*

Pada proses pembuatan *crawler* di penelitian ini, penulis akan membuat dua versi *crawler* yang sudah dijelaskan pada bab sebelumnya. Pertama adalah *typical crawling model* yang menggunakan algoritma *breadth first search* dasar, *crawler* ini akan selalu bekerja dengan titik awal yang sudah ditentukan. Kedua adalah *modified similarity-based crawling* yang menggunakan algoritma *breadth first search* yang sudah dikembangkan.

Penelitian ini pada dasarnya akan menggunakan *modified similarity-based crawling*, tetapi dalam penggunaannya *crawler* tersebut membutuhkan *crawler* lain yang sudah berjalan, terutama untuk *keyword hot link*. Sementara *hot link* hanya akan berjalan *efficient* dari *crawling cache* yang sudah berjalan sebelumnya.

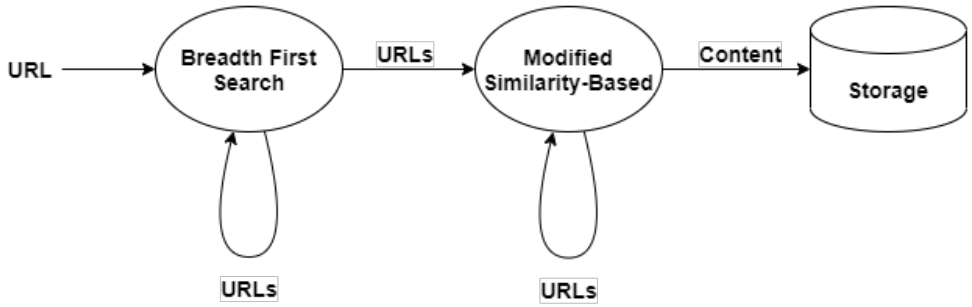
Crawler ini berbasis *terminal windows*, bahasa yang digunakan menggunakan bahasa pemrograman *python*. *Input crawler* ini berupa sebuah *link* dan *query* yang diujikan. Kemudian *output* dari *crawler* ini berupa data yang sudah dikumpulkan dari berbagai halaman web, dan juga akan di tampilkan peta halaman *graph* dan peta situs *graph* dari *link* yang sudah didapat. Saya sebagai penulis akan membuat dan mengoperasikan *crawler* ini.

3.2 Desain Eksperimen

Untuk desain eksperimennya, penelitian ini akan menggunakan sejumlah *query* yang telah ditentukan, dan juga pada bidang yang sudah ditentukan. Rancangannya sebagai berikut:

1. Perancangan *breadth first search crawler*
2. Perancangan *modified similarity-based crawler*.
3. Menentukan domain pencarian tertentu.
4. Menentukan sejumlah *query* pencarian yang akan diuji untuk domain yang telah ditentukan.
5. Menjalankan *breadth first search crawler*.
6. Menjalankan *modified similarity-based crawler*.
7. Menguji performa pada *query* yang diujikan.

3.3 Arsitektur Diagram



Gambar 3.1: Arsitektur Diagram

Terdapat dua komponen utama, yaitu *Breadth first search* dan *modified similarity-based*. *Crawler* ini berbasis terminal, inputnya berupa sebuah *URL* yang akan menjadi node pertama. Sebagian besar program akan berjalan menggunakan bahasa *python*.

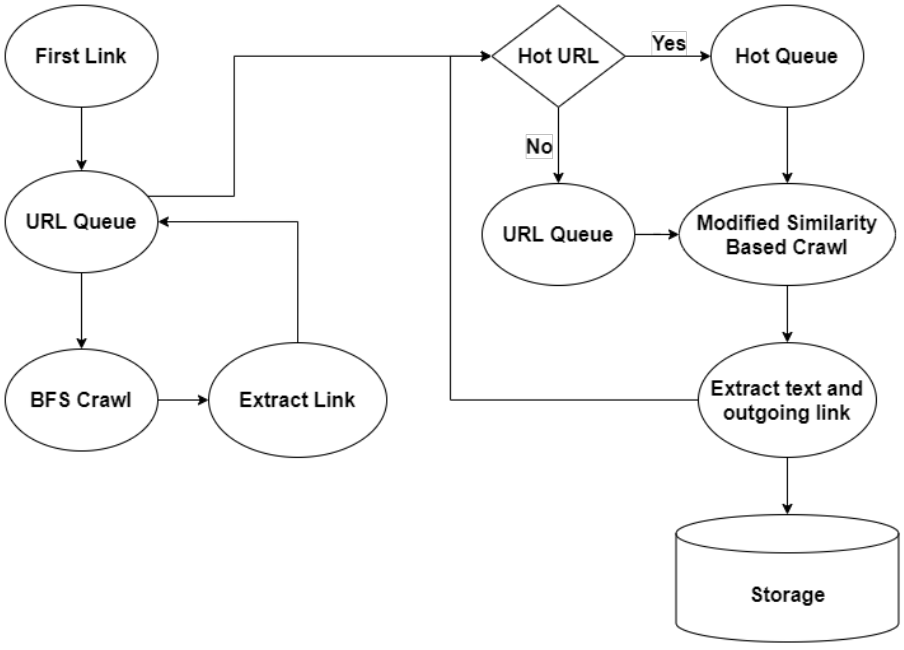
Breadth first search perlu sebuah *input* berupa *URL*, Algoritma ini akan mengunjungi node pertama berupa *URL*, kemudian akan mengunjungi semua *URL* yang berdekatan dengan *URL* yang sedang dikunjungi. *Breadth first search* juga

memerlukan sebuah *url_queue* untuk menyimpan *URL* yang belum dikunjungi, sedangkan yang sudah dikunjungi akan ditempatkan di *visited_url*. *Output* programnya berupa kumpulan *URL* yang nantinya akan diproses oleh *modified similarity-based*.

Modified similarity-based merupakan algoritma *breadth first search* yang sudah dikembangkan. Algoritma ini memerlukan *hot_queue* yang berisi kumpulan *URL* dan memerlukan sebuah *keyword*. *URL* yang dimasukkan kedalam *hot_queue* akan dipilih berdasarkan *keyword* yang telah ditentukan.

3.4 *Flowchart* Algoritma

Flowchart Algoritma pada penelitian ini dapat dilihat di gambar 3.2. Terdapat dua *crawler*, yaitu *breadth first search crawler* dan *modified similarity based crawler*.



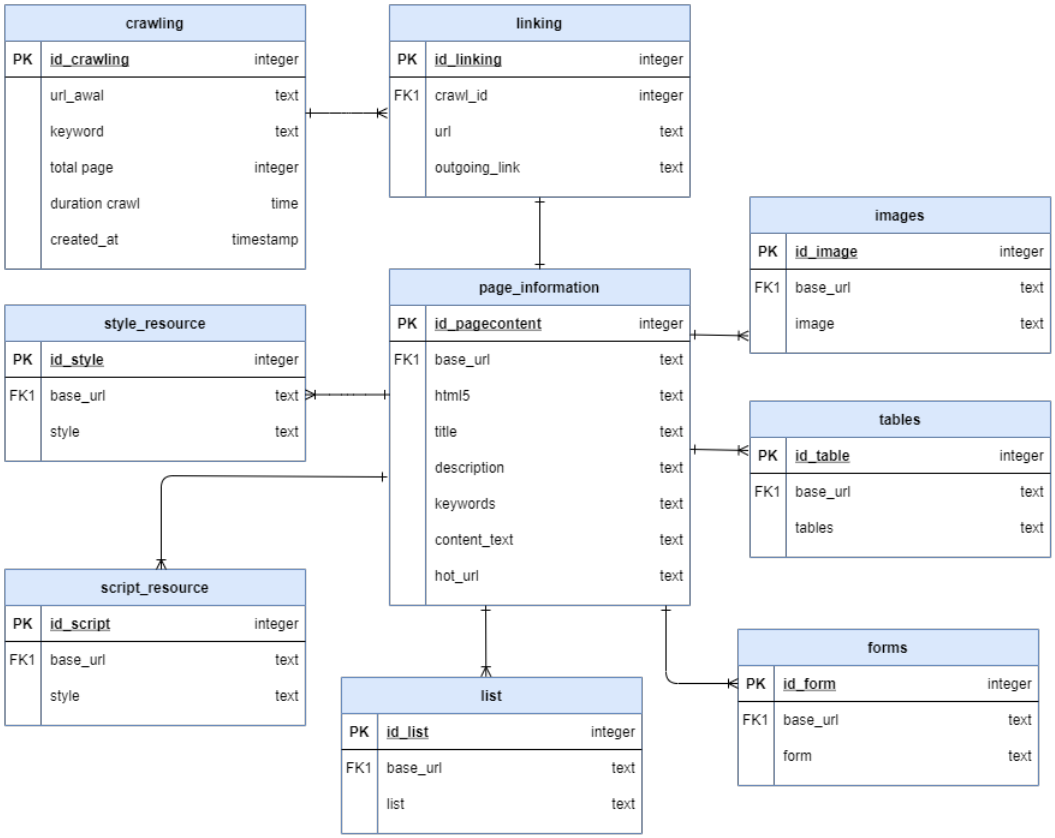
Gambar 3.2: *Flowchart* Algoritma

Situs pertama ditunjuk sebagai *node* awal pada penelitian. *Breadth first search crawler* akan terus melakukan *crawling*. *Link* yang didapat dari hasil

crawling tersebut akan tersimpan di *URL Queue*. *URL Queue* yang sudah berisi sekumpulan *link*, akan dipilih untuk dimasukkan kedalam *Hot Queue*.

Hot Queue meyimpan *url* yang menyebutkan *keyword* atau memiliki *keyword* di dalamnya. Sedangkan *URL Queue* akan menyimpan sisa dari *url* yang tidak termasuk dalam *Hot Queue*. *Crawler* akan mengambil *url* dari *Hot Queue* terlebih dahulu. Kemudian, hasil dari *crawling* tersebut akan disimpan dalam *Storage*.

3.5 Desain *Entity Relationship Diagram*



Gambar 3.3: *entity relationship diagram*

Pemodelan *entity relationship diagram* (ERD) pada *crawler* ini terlihat pada gambar 3.3. Terdapat sembilan entitas atau tabel yang masing-masing menyimpan data yang dibutuhkan dalam *crawler*. Setiap kali *crawler* dijalankan, maka akan

membuat satu data yang akan disimpan pada tabel *crawling*. Tabel *crawling* menyimpan data *url* awal yang dijalankan, *keyword* untuk kebutuhan *hot_url*, total halaman yang sudah didapat, berapa lama waktu yang dibutuhkan untuk menjalankan *crawler*, dan data kapan *crawler* ini berjalan. Pada penelitian ini, *crawler* hanya akan berjalan sekali, tetapi pada penerapannya nanti, *crawler* akan rutin dijalankan ulang.

Pada tabel *linking* menyimpan nama *url* dan *outgoing link* dari *url* tersebut. Sehingga relasi yang dimiliki tabel *linking* dengan tabel *crawling* adalah *one-to-many*, karena setiap sekali *crawling* bisa memiliki banyak *linking*. Kemudian untuk informasi halaman disimpan pada tabel *page_information*, tabel ini menyimpan *meta data* seperti *description* dan *keywords* pada situs, *title*, isi konten berupa *text* yang seluruh *tag html* dihilangkan, kemudian pada tabel ini juga disimpan informasi mengenai *html5* atau *html* versi sebelumnya, dan juga menyimpan jika situs ini merupakan *hot url* atau *url* biasa.

Semua data yang diperoleh saat proses *crawling* tidak dibuang begitu saja, selain menyimpan data penting yang ada di tabel *page_information*, seperti data *css*, *js*, *list*, *forms*, *tables*, dan *images* juga akan disimpan pada tabelnya masing-masing. Relasi yang dimiliki tabel *page_information* dengan tabel lainnya adalah *one-to-many*. Relasi ini juga akan mendukung untuk menyimpan data *multiple item*.

3.6 Domain Pencarian

Bidang pada penelitian ini adalah olahraga, dan akan menunjuk salah satu portal olahraga Indonesia. Tema olahraga ini sangat umum untuk dibahas, dan cakupannya juga luas. Diantaranya terdapat sepakbola, basket, badminton, otomotif, dan *e-sport*. Maka, penulis memilih tema olahraga yang sudah umum dikenal masyarakat Indonesia.

Situs yang menjadi *node* awal penelitian ini adalah *Indosport*, dengan nama situs: https://www.indosport.com. Portal tersebut berisi tentang berita olahraga yang menggunakan bahasa Indonesia. Kemudian, untuk *query* yang akan diujikan nantinya adalah:

1. Klub bola *Barcelona*.
2. Sirkuit Mandalika *MotoGP*.
3. Pemain bulu tangkis Kevin Sanjaya.
4. *Mobile Legends Profesional League*.

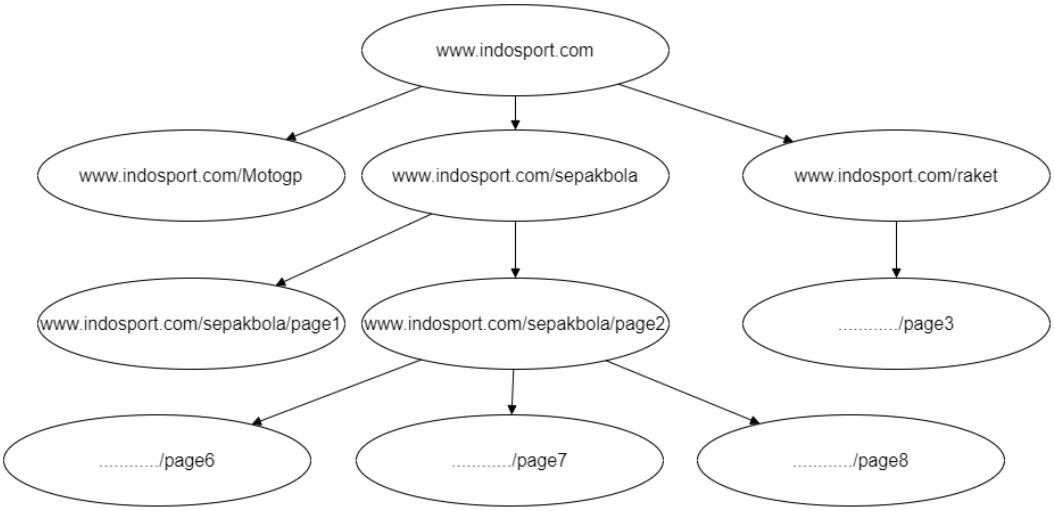
Query yang disebutkan di atas mencakup sepak bola, *MotoGP*, badminton, dan *e-sport*. Hasil utama yang diharapkan adalah, *crawler* dapat mengutamakan *url* yang terdapat *keyword* pada *query* yang disebutkan.

3.7 Parameter Keberhasilan

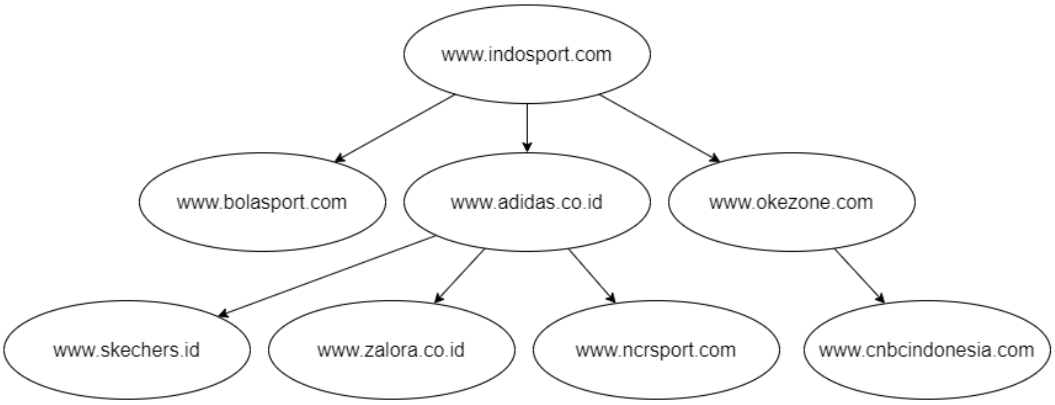
Breadth first search crawler dan *modified similarity-based crawler* yang sudah berjalan akan diuji untuk setiap *query*-nya. Untuk mengukur keberhasilan *breadth first search crawler* dan *modified similarity-based crawler* diperlukan dua situasi yang berbeda, yaitu:

1. *Crawler* hanya akan berjalan pada sebuah *website*.
2. *Crawler* akan berjalan pada sebuah *website*, tetapi akan terus melakukan *crawling* hingga berhenti.

Hasil yang diharapkan untuk situasi pertama yaitu menampilkan *page map graph* seperti pada gambar 3.4. Kemudian, hasil untuk situasi kedua menampilkan *site map graph* seperti pada gambar 3.5.



Gambar 3.4: Ilustrasi *Page Map Graph*



Gambar 3.5: Ilustrasi *Ilustrasi Site Map Graph*

BAB IV

UJI COBA DAN HASIL UJI COBA

Bagian ini terdiri dari dua bagian. Bagian pertama adalah pemaparan tahap-tahap yang dilakukan dalam percobaan beserta data-data yang digunakan. Bagian kedua memuat data dan keterangan tentang hasil yang diperoleh dari percobaan tersebut. Data tersebut dapat disajikan dalam tabel, grafik, gambar ataupun bentuk lain disesuaikan dengan produk yang dihasilkan. Hasil percobaan harus dapat memperlihatkan bahwa produk yang dihasilkan dapat bekerja dengan baik sesuai dengan rujukan.

Spesifikasi Laptop yang digunakan. Implementasi code. Hasil berupa (database, dan isi database. peta graph).

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil implementasi dan pengujian fitur sistem informasi yang telah dirancang, maka diperoleh kesimpulan sebagai berikut:

1. Perancangan sistem informasi operasi serba usaha berbasis *website* pada lembaga Koperasi Mahasiswa Universitas Negeri Jakarta menggunakan metode pengembangan perangkat lunak *System Development Life Cycle* dengan *spiral model* yang memiliki beberapa tahapan, yaitu analisis kebutuhan, perancangan desain sistem *(prototype)*, pengimplementasian (*coding & testing-unit*), dan *maintenance* (umpan balik dan tanggapan).

2. Sistem informasi Koperasi Mahasiswa Universitas Negeri Jakarta dibangun dengan menggunakan *framework codeigniter* dan *framework bootstrap*. *Codeigniter* untuk membangun sisi dalam *back-end* dan *Bootstrap* yang mempercantik bagian luar *front-end*. Berdasarkan pengolahan data hasil kuesioner *user acceptance test*, didapatkan rata-rata persentase mencapai angka 4,27 dari 5 atau sekitar 85% untuk *admin* yang berarti sistem sudah sangat sesuai, 3,96 dari 5 atau sekitar 79% untuk anggota yang berarti sistem sudah sesuai, dan 4,53 dari 5 atau sekitar 91% pada sistem pengawas yang berarti sistem sudah sangat sesuai.

3. Sistem informasi Koperasi Mahasiswa Universitas Negeri Jakarta berbasis *website* dibangun agar sistem pengelolaan data di KOPMA UNJ dapat menjadi lebih efektif dan efisian, serta agar kemungkinan adanya *human error*

dapat diatasi. Data yang dikelola berupa data anggota, barang, stok (pergudangan), simpanan, transaksi, penilaian, dan keuangan.

4. Sistem informasi Koperasi Mahasiswa Universitas Negeri Jakarta berbasis *website* mempermudah *admin* dalam melakukan pengelolaan data yang ada di KOPMA UNJ, mempermudah pengawas dalam melakukan pengelolaan data penilan, dan membuat anggota dapat mengetahui transparansi simpanan yang mereka bayarkan hingga mengetahui transparansi sisa hasil usaha yang mereka dapatkan.

5.2 Saran

Adapun saran untuk penelitian selanjutnya adalah:

1. Mengintegrasikan sistem informasi KOPMA UNJ dengan aplikasi dan sistem *scanning barcode* agar proses transaksi semakin efektif dan efisien.
2. Menambahkan dan mengintegrasikan sistem informasi KOPMA UNJ dengan aplikasi untuk *scanning barcode* pada kartu anggota agar proses kebutuhan simpanan dan transaksi untuk anggota dapat semakin efektif dan efisien.
3. Menambahkan sistem pengelolaan kegiatan keorganisasian KOPMA UNJ yang merupakan bagian dari peran KOPMA UNJ sebagai Organisasi Kemahasiswaan di UNJ.
4. Membuat pengembangan sistem informasi KOPMA UNJ berbasis *mobile* (*android*).

DAFTAR PUSTAKA

Brin, S. & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine.

Castillo, C. (2005). Effective web crawling. In *Acm sigir forum*, volume 39, pages 55–56. Acm New York, NY, USA.

Cho, J., Garcia-Molina, H., & Page, L. (1998). Efficient crawling through url ordering.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. MIT press.

Moz (2020). Urls.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Pinkerton, B. (1994). Finding what people want: Experiences with the webcrawler. In *Proc. 2nd WWW Conf., 1994*.

Salton, G. (1989). Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley*, 169.

Seymour, T., Frantsvog, D., & Kumar, S. (2011). History of search engines. *International Journal of Management & Information Systems (IJMIS)*, 15(4):47–58.

Statista (2020). Worldwide market share of search engine.

TechnaCenterLLC (2020). Basic structure of an html document.

WHATWG (2020). Url standard.