

**PERANCANGAN CRAWLER SEBAGAI PENDUKUNG PADA
SEARCH ENGINE**

Skripsi

**Disusun untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer**



**Oleh:
Muhammad Fathan Qoriiba
3145161299**

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

2021

LEMBAR PERSETUJUAN HASIL SIDANG SKRIPSI

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



27-8-2021

Wakil Penanggung Jawab

Wakil Dekan Bidang Akademik : Dr. Esmar Budi, S.Si., MT.

NIP. 19720728 199903 1 002

27-8-2021

Ketua

: Drs. Mulyono, M.Kom.

13-8-2021

Sekretaris

: Ari Hendarno, S.Pd, M.Kom

13-8-2021

Penguji Ahli

: Ria Arafiyah, M.Si.

16-8-2021

Pembimbing I

: Med Irzal, M.Kom.

18-8-2021

Pembimbing II

: Muhammad Eka Suryana, M.Kom.

20-08-2021

NIP. 19851223 201212 1 002

Dinyatakan lulus ujian skripsi tanggal: 04 Agustus 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

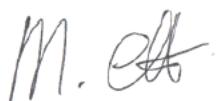
Judul : Perancangan *Crawler* Sebagai Pendukung
pada *Search Engine*

Menyatakan bahwa skripsi ini telah siap diajukan untuk skripsi.

Menyetujui,

Dosen Pembimbing I

Dosen Pembimbing II



Med Irzal, M.Kom.

NIP. 19770615 200312 1 001

Muhammad Eka Suryana, M. Kom.

NIP. 19851223 201212 1 002

Mengetahui,

Koordinator Program Studi Ilmu Komputer



Ir. Fariani Hermin Indiyah, M.T.

NIP. 19600211 198703 2 001

LEMBAR PERNYATAAN

Saya menyatakan dengan sesungguhnya bahwa skripsi dengan judul **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, 29 Juli 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 skripsi ini dengan baik. Adapun jenis penelitian yang dipilih yaitu *information retrieval* dengan judul Desain Perancangan *Crawler* Sebagai Pendukung pada *Search Engine*.

Selama penyusunan skripsi ini, penulis sedikit banyak menghadapi hambatan. Namun dengan bantuan berbagai pihak, hambatan tersebut dapat diatasi sehingga penulis dapat menyusun skripsi ini dengan baik. Untuk itu penulis mengucapkan terima kasih yang sebesar-besarnya kepada semua pihak yang telah membantu penulis dalam menyelesaikan skripsi ini khususnya kepada:

1. Bapak Med Irzal, M. Kom. selaku Dosen Pembimbing I yang telah membimbing, mengarahkan, serta memberikan dorongan untuk menyelesaikan skripsi ini.
2. Muhammad Eka Suryana, M. Kom. selaku Dosen Pembimbing II yang telah membimbing, mengarahkan, serta memberikan dorongan untuk menyelesaikan skripsi ini.
3. Ibu Ir. Fariani Hermin Indiyah M.T. selaku Koordinator Program Studi Ilmu Komputer FMIPA UNJ sekaligus Dosen Pembimbing Akademik penulis.
4. Teman-teman Progam Studi Ilmu Komputer 2016 yang telah membantu dan mendukung, sehingga skripsi ini dapat diselesaikan.
5. Keluarga dan sahabat tercinta yang telah memberikan dorongan serta bantuan yang besar kepada penulis.

6. Dan semua pihak yang juga telah membantu dengan tidak mengurangi rasa hormat penulis yang tidak dapat disebutkan satu persatu.

Dalam penulisan skripsi ini, penulis menyadari bahwa dengan keterbatasan ilmu dan pengetahuan penulis, skripsi 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 skripsi ini.

Terima kasih,

Jakarta, 29 Juli 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 Irvan, M. Kom dan Muhammad Eka Suryana, M. Kom.

Search engine berkembang mulai dari tahun 1990 hingga hari ini. Google mulai membangun search engine pada tahun 1998, hingga saat ini Google masih sangat populer. Fungsi dari search engine ini sangat banyak, bahkan data yang diperoleh dari search engine dapat dimanfaatkan pada aplikasi atau program lain. Untuk membuat search engine ini, mulanya dibutuhkan sebuah crawler. Crawler merupakan sebuah program untuk mengambil informasi yang ada di halaman web. Dibutuhkan perancangan crawler yang baik untuk mendapatkan hasil yang baik. Algoritma yang digunakan mengikuti algoritma awal perkembangan Google, yaitu modified similarity based crawling. Seluruh program yang dibuat menggunakan bahasa pemrograman python. Hasil akhir dari crawler ini, data dapat disimpan baik kedalam database MySQL.

Kata kunci : *Search engine, crawler, information retrieval, breadth first search, modified similarity based, python, Google.*

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.*

Search engines have been growing from 1990 to the present. Google started building search engines in 1998, until now Google is still very popular. The functions of search engines are very numerous, even the data obtained from search engines can be used on other applications or programs. The first step needed to make a search engine is a crawler. Crawler is a program to retrieve information from the web page. Good crawler design is necessary to achieve good result. The algorithm used to follow early Google development algorithms, which is called modified similarity based crawling. The entire program was made using Python programming language. As the result of this crawler, the data can be stored well into MySQL database.

Keywords : *Search engine, crawler, information retrieval, breadth first search, modified similarity based, python, Google.*

DAFTAR ISI

HALAMAN PERSEMBAHAN	v
KATA PENGANTAR	vi
ABSTRAK	viii
ABSTRACT	ix
DAFTAR ISI	xii
DAFTAR GAMBAR	xiv
DAFTAR TABEL	xv
I PENDAHULUAN	1
1.1 Latar Belakang Masalah	1
1.2 Rumusan Masalah	5
1.3 Pembatasan Masalah	5
1.4 Tujuan Penelitian	6
1.5 Manfaat Penelitian	6
II KAJIAN PUSTAKA	8
2.1 Sejarah <i>Search Engine</i>	8
2.2 <i>URL</i>	12
2.3 <i>HTML</i>	14
2.4 <i>Graph</i>	14
2.5 <i>Breadth First Search</i>	17
2.6 Definisi <i>Search Engine</i>	20

2.7	Arsitektur <i>Search Engine</i>	21
2.8	<i>Web Crawler</i>	23
2.8.1	<i>Importance Metrics</i>	23
2.8.2	Model <i>Crawler</i>	26
2.9	Arsitektur <i>Web Crawler</i>	28
2.10	Algoritma <i>Crawling</i>	28
III DESAIN MODEL		32
3.1	Desain <i>Crawler</i>	32
3.2	Desain Eksperimen	33
3.3	Arsitektur Diagram	33
3.4	<i>Flowchart</i> Algoritma	34
3.5	Desain <i>Entity Relationship Diagram</i>	35
3.6	Domain Pencarian	37
3.7	Parameter Keberhasilan	37
IV UJI COBA DAN HASIL UJI COBA		39
4.1	Implementasi	39
4.1.1	<i>Library Python</i>	39
4.1.2	Implementasi Algoritma <i>Breadth First Search Crawling</i>	40
4.1.3	Implementasi Algoritma <i>Modified Similarity Based Crawling</i>	43
4.1.4	Koneksi <i>Database</i>	44
4.2	Referensi <i>Code</i>	46
4.3	Uji Coba	47
4.4	Hasil Uji Coba	49
V KESIMPULAN DAN SARAN		54
5.1	Kesimpulan	54

5.2 Saran	54
DAFTAR PUSTAKA	57
LAMPIRAN	58
A Sampel Kode <i>Breadth First Search Crawling</i>	58
B Sampel Kode <i>Modified Similarity Based Crawling</i>	70

DAFTAR GAMBAR

Gambar 2.1	<i>Market cap search engine</i> tahun 2010 hingga 2020 (Statista, 2020)	12
Gambar 2.2	Ilustrasi <i>graph</i>	15
Gambar 2.3	<i>Adjacent</i>	16
Gambar 2.4	Pengoperasian BFS pada <i>undirected graph</i> (Cormen et al., 2009)	19
Gambar 2.5	<i>High level google architecture</i> (Brin and Page, 1998)	21
Gambar 2.6	<i>High level architecture of web crawler</i> (Castillo, 2005)	28
Gambar 3.1	Arsitektur diagram	33
Gambar 3.2	<i>Flowchart</i> algoritma	34
Gambar 3.3	<i>Entity relationship diagram</i>	36
Gambar 3.4	Ilustrasi <i>page map graph</i>	38
Gambar 3.5	Ilustrasi <i>Ilustrasi site map graph</i>	38
Gambar 4.1	<i>Library python</i>	39
Gambar 4.2	Implementasi algoritma <i>breadth first search crawling</i> 1	40
Gambar 4.3	Implementasi algoritma <i>breadth first search crawling</i> 2	40
Gambar 4.4	Implementasi algoritma <i>breadth first search crawling</i> 3	41
Gambar 4.5	Implementasi algoritma <i>breadth first search crawling</i> 4	41
Gambar 4.6	Implementasi algoritma <i>breadth first search crawling</i> 5	41
Gambar 4.7	Implementasi algoritma <i>breadth first search crawling</i> 6	42
Gambar 4.8	Implementasi algoritma <i>breadth first search crawling</i> 7	42
Gambar 4.9	Implementasi algoritma <i>breadth first search crawling</i> 8	43
Gambar 4.10	Implementasi algoritma <i>modified similarity based crawling</i> 1	43
Gambar 4.11	Implementasi algoritma <i>modified similarity based crawling</i> 2	44

Gambar 4.12 Implementasi algoritma <i>modified similarity based crawling</i> 3	44
Gambar 4.13 Implementasi algoritma <i>modified similarity based crawling</i> 4	44
Gambar 4.14 Koneksi <i>database</i> 1	44
Gambar 4.15 Koneksi <i>database</i> 2	45
Gambar 4.16 Koneksi <i>database</i> 3	45
Gambar 4.17 Koneksi <i>database</i> 4	46
Gambar 4.18 Uji coba 1	47
Gambar 4.19 Uji coba 2	47
Gambar 4.20 Uji coba 3	48
Gambar 4.21 Uji coba 4	48
Gambar 4.22 Hasil uji coba 1	49
Gambar 4.23 Hasil uji coba 2	49
Gambar 4.24 Hasil uji coba 3	50
Gambar 4.25 Hasil uji coba 4	51
Gambar 4.26 Hasil uji coba 5	51
Gambar 4.27 Hasil uji coba 6	52
Gambar 4.28 Hasil uji coba 7	52
Gambar 4.29 Hasil uji coba 8	52
Gambar 4.30 Hasil uji coba 9	53
Gambar 4.31 Hasil uji coba 10	53
Gambar 4.32 Hasil uji coba 11	53

DAFTAR TABEL

Tabel 1.1	Penggunaan <i>global search engine</i> Agustus 2020 (NetMarketShare, 2020)	2
Tabel 2.1	<i>Base</i> dan <i>Output</i> pada <i>URL</i> (WebHypertextApplicationTechnologyWorkingGroup, 2020)	13
Tabel 4.1	Referensi <i>code</i>	46

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

Search engine atau mesin pencari adalah sebuah program yang dibuat untuk melakukan pencarian pada situs web. *Search engine* pertama kali diperkenalkan oleh Alan Emtage pada tahun 1990 dengan nama *Archie*. Prinsip kerja *Archie* adalah melakukan pengindeksan semua *file* pada web. Lokasi file yang dicari akan lebih mudah ditemukan dengan *Archie* oleh para pengguna internet (Seymour et al., 2011).

Archie dapat menampilkan daftar nama situs, tetapi tidak dapat menampilkan isi atau konten. Pada tahun berikutnya mulai bermunculan *search engine* baru. *Aliweb* muncul pada tahun 1993, para pengguna diberi kesempatan untuk mengunggah halaman situs yang ingin terindeks di internet, dan dapat mengisi deskripsi untuk halaman tersebut. Kemudian, muncul *AltaVista* yang populer di internet pada tahun 1995. *AltaVista* memberikan *unlimited bandwith* untuk penggunanya, teknik dan sistem algoritmanya juga sudah lebih maju. Sembilan tahun setelahnya, muncul *Yahoo*. Data dari situs-situs yang sudah dikenal banyak orang tersedia di *Yahoo*. Pengelola situs dan pemilik situs dapat menambah informasi tanpa mengeluarkan biaya, terdapat juga fitur yang sudah lengkap dengan dikenakan sejumlah biaya. Perusahaan *Yahoo* sangat lambat dalam pengembangan mesin pencarian dan para pengguna internet perlahan mulai mencari *search engine* lain (Seymour et al., 2011).

Search engine yang saat ini masih populer yaitu *Google*. *Google* diluncurkan pada tahun 1998, *Google* mengembangkan sistem yang berbeda dengan *search engine* sebelumnya. Sistem yang dikembangkan bernama *BackRub*, sistem ini

menggunakan *backlink* untuk memberi peringkat pada setiap halaman. *Ranking* untuk setiap halaman situs diurutkan berdasarkan penyebutan situs terbanyak di situs-situs lain (Brin et al., 1998). Selain *Google*, *search engine* yang masih populer saat ini yaitu *Bing*. *Bing* muncul di pertengahan 2009. *Bing* merupakan salah satu *search engine* favorit di Amerika Serikat, karena terdapat beberapa fitur yang pas digunakan masyarakat Amerika. Namun, selain di Amerika, *Google* lebih populer dan sangat favorit dibandingkan dengan *Bing* (Seymour et al., 2011).

Tabel 1.1: Penggunaan *global search engine* Agustus 2020 (NetMarketShare, 2020)

Search engine	Global share (%)
Google	83.46
Baidu	7.35
Bing	6.15
Yahoo!	1.42
Yandex	0.87
DuckDuckGo	0.31

Apple adalah perusahaan yang saat ini menggunakan *Google* sebagai mesin pencarinya. Selama beberapa tahun ini *Google* membayar *Apple* lebih dari satu miliar dolar untuk tetap menjadi *search engine* utama di *safari* untuk *iOS*, *iPadOS*, dan *macOS*. *Apple* sudah membuat *web crawler* bernama *Applebot* untuk *search engine* *Apple* dan kemungkinan *search engine* tersebut akan rilis di tahun ini (Henshaw, 2020).

Fungsi dari *search engine* selain mencari informasi adalah sebagai tempat untuk memasangkan iklan. Karena jumlah pengguna yang terus bertambah, menjadikan *search engine* sebagai media pemasaran saat ini. Melalui *search engine*, pengguna internet dapat mencari apa saja, termasuk juga barang yang mereka ingin

beli. *Search engine* membuat pencarian informasi semakin mudah. Banyaknya pengguna dapat menjadikan *search engine* sebagai peluang bagi pemilik usaha atau perusahaan untuk memasarkan produk dan jasa mereka.

Google memasangkan iklan pada mesin pencarinya untuk mendapatkan keuntungan, layanan iklan ini bernama *AdWords*. *AdWords* memungkinkan pengiklan menjangkau pengguna *online* dengan beriklan di platform *Google*. Jika mencari sebuah informasi suatu barang, maka beberapa baris hasil penelusuran teratas yang disematkan dengan tulisan “Ad” akan terlihat. Informasi tersebut dipasang oleh pengiklan yang menggunakan fitur *AdWords Google*.

Google pun juga mendapatkan banyak data melalui mesin pencarinya. Selain itu, *Google* juga membuat layanan lain seperti *Gmail*, *YouTube*, *Search*, *Drive*, *Maps*, dan *PlayStore*. Setiap layanan memiliki fungsi tersendiri untuk membantu aktivitas masyarakat. Dan dari setiap layanan *Google* terdapat data-data pengguna yang dapat dipakai untuk keperluan lainnya.

Proses pembuatan arsitektur *search engine* ini dimulai dengan membuat salah satu bagiannya, yaitu *crawler*. Sebelum akhirnya *search engine* dapat menampilkan data-data yang diperlukan *user*, *search engine* memerlukan sebuah bagian yang berfungsi untuk mengumpulkan data-data tersebut. Maka dibutuhkan sebuah *crawler* yang akan dirancang dengan sebaik mungkin.

Web crawler merupakan *software* yang bekerja otomatis untuk menjelajahi *World Wide Web* secara terorganisir. Web crawler pertama muncul pada tahun 1944 bernama *RBSE* (Eichmann, 1994). Web crawler ini didasarkan pada dua program, yaitu:

1. *Spider* yang berfungsi memelihara *queue* dalam *relational database*.
2. *Mite* untuk mendownload halaman dari web.

Fungsi dari *web crawler* adalah untuk mengumpulkan data. Jika tidak ada *web crawler*, maka *search engine* tidak akan mendapatkan data dan melakukan *indexing*. Data-data yang dihasilkan oleh *web crawler* merupakan data terbaru dan akurat. *Google* memiliki *web crawler* bernama *Googlebot*. Selain *Googlebot*, terdapat *web crawler* lain yaitu: *Bingbot* (*web crawler Bing*), *Slurp bot* (*web crawler Yahoo!*), *DuckDuckBot* (*web crawler DuckDuckGo*), *Baiduspider* (*web crawler Baidu*), *Yandex Bot* (*web crawler Yandex*).

Kaur dan Geetha (2020) dalam jurnalnya, telah membuat *web crawler* untuk *hidden web* dengan menggunakan *SIM+HASH* dan *Redis Server*. *Focused web crawler* dapat memberikan hasil halaman relevan yang maksimum. Akan tetapi, *crawler* ini sudah terlalu rumit untuk pembuatannya. Komponen pada *focused web crawler* hampir sama dengan *web crawler* pada umumnya.

Skripsi ini dikerjakan bersama dengan Savira Rahmayanti dari Ilmu Komputer angkatan 2015 Universitas Negeri Jakarta, akan mengembangkan *server based search engine* yang bersifat *open source* dan menjelaskan seperti apa arsitektur *search engine* tersebut. Penulis mendapatkan bagian untuk mengimplementasi *web crawling* pada *search engine*, sedangkan Savira fokus untuk mengerjakan bagian *searching* dan *indexing*.

Pada skripsi ini tidak mengimplementasikan apa yang telah dikembangkan oleh Sawroop Kaur dan Geetha (2020), karena *web crawler* tersebut sudah terlalu rumit dan komponennya tidak jauh berbeda dengan *crawler* biasa. Akan tetapi, skripsi ini akan mengembangkan *web crawler* dasar yang merujuk pada awal perkembangan *search engine Google* (Brin and Page, 1998). Penelitian ini akan menjadi dasar pada penelitian *search engine* berikutnya.

Metode algoritma *crawler* pada penelitian ini, akan mengacu pada awal perkembangan *Google*, yaitu *modified similarity based crawling* (Cho et al., 1998).

Karena algoritma ini sangat pas untuk merancang sebuah *crawler* yang membangun database dengan topik tertentu. Sebagai contoh, jika ingin membangun *crawler* dengan topik *Barcelona*, maka *crawler* akan *crawling* halaman yang terdapat kata *Barcelona* terlebih dahulu. Algoritma *modified similarity based crawling* sangat menarik untuk diimplementasikan pada penelitian *search engine* ini.

Crawler sangat dibutuhkan *search engine* untuk dapat mencari informasi dengan lebih efisien. Skripsi ini juga untuk memperdalam ilmu mengenai *information retrieval* khususnya *web crawling*. Oleh karena itu, perlu dibuat perancangan *web crawler* pada *search engine*. Dan ini tertuang pada penelitian yang berjudul “**Perancangan Crawler Sebagai Pendukung pada Search Engine**”.

1.2 Rumusan Masalah

Berdasarkan uraian pada latar belakang yang diutarakan di atas, maka perumusan masalah pada penelitian ini adalah “Bagaimana cara membuat rancangan *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. Penelitian ini hanya membuat sebagian arsitektur *search engine*, yaitu *crawler*. Algoritma *crawler* yang akan dibuat mengacu pada model algoritma *Google* awal.
2. *Crawler* tidak mengambil data dari isi database web, melainkan dari struktur data *HTML* web. Maka *crawler* hanya akan berjalan pada *website statis*.

3. Penelitian ini akan menargetkan dua situs. Situs pertama menggunakan *HTML* versi 4, dan situs kedua menggunakan *HTML* versi 5.

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 perancangan *crawler*.

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*

Search engine 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 *search engine* 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 *search engine* pertama muncul di tahun 1990, tiga tahun berikutnya di tahun 1993 pada bulan November, muncul *search engine* bernama *Aliweb*. *Search engine* kedua ini tidak menggunakan *web robot*, dan mengizinkan pengguna menuliskan alamat indeks file di situs pengguna, sehingga *search engine* 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 *search engine* 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 sumber daya pada platformnya sedikit terbatas, pengindeksan dan pencarinya 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 di antaranya *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. Akan 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, terdapat 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 diambil *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.

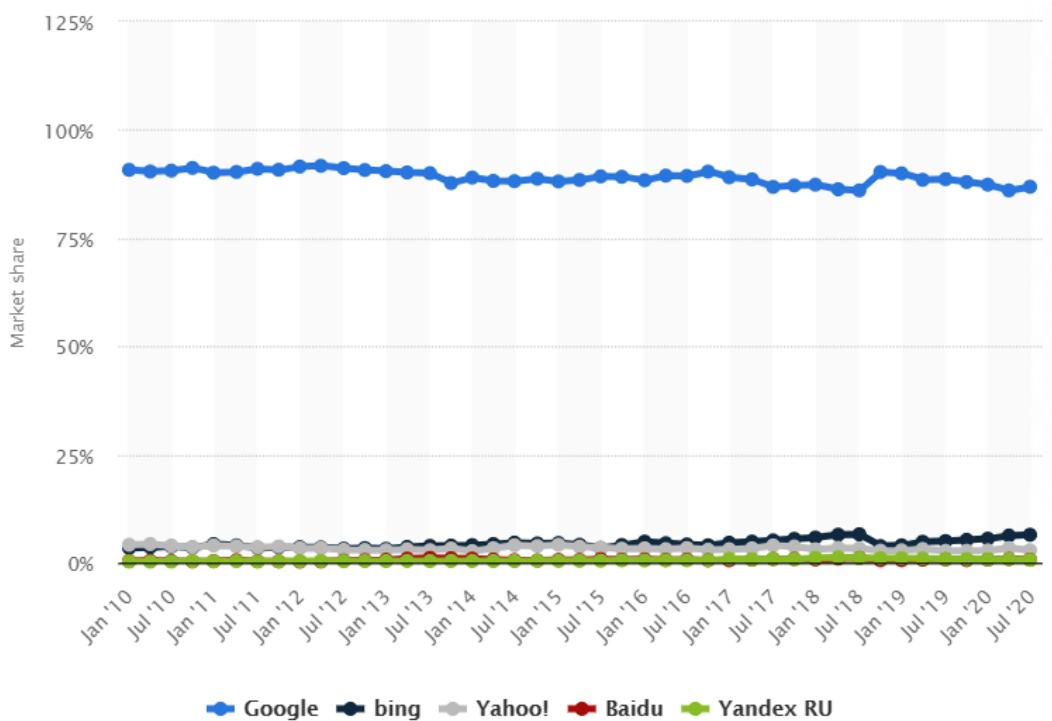
Diawal abad ke 20 pada tanggal 18 Januari 2000, muncul *search engine* bernama *Baidu*. *Baidu* berpusat di Beijing, RRC. *Baidu* merupakan *search engine* untuk *website*, file audio dan gambar berbahasa Cina dan Jepang. *Baidu* merupakan perusahaan Cina yang masuk dalam indeks NASDAQ-100 pertama kali. 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* (WebHypertextApplicationTechnologyWorkingGroup, 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	<i>Failure</i>
contoh	https://contoh.com/data	Yes	https://contoh.com/contoh
contoh		No	<i>Failure</i>
http://[www.cth.com]/		No	<i>Failure</i>
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* (WebHypertextApplicationTechnologyWorkingGroup, 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.

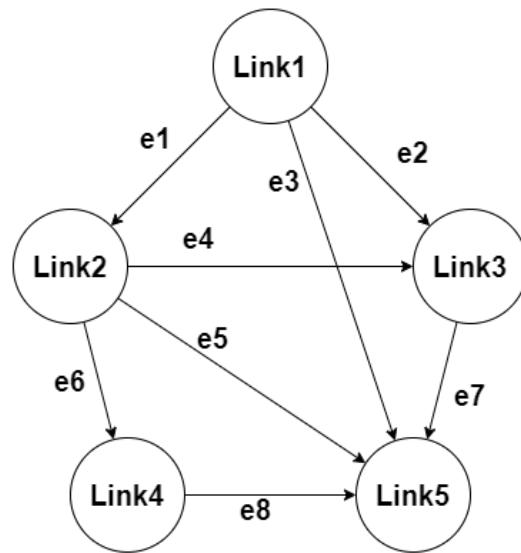
Pada sebuah halaman web, tag pertama (khususnya, <html>) menunjukkan penanda bahasa yang digunakan untuk dokumen. Tag <head> berisi informasi tentang halaman web. Terakhir, konten muncul di tag <body> (TechnaCenterLLC, 2020).

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 $E = \{\{Link1, Link2\}, \{Link1, Link3\}, \{Link1, Link5\}, \{Link2, Link3\}, \{Link2, Link5\}, \{Link2, Link4\}, \{Link3, Link5\}, \{Link4, Link5\}\}$.

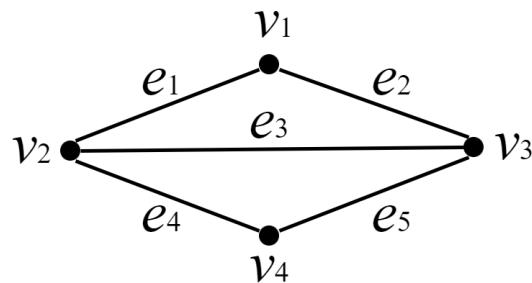
Terdapat beberapa istilah yang berkaitan menggunakan *graph*, yaitu:

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 $G = (V, E, W)$.

$$W : E \rightarrow R \quad (2.2)$$

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

Breadth first search membangun *breadth first tree*, awalnya hanya berisi *root*, yang merupakan sumber *vertex* s . Kapan pun 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 merupakan *descendant*

dari u .

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

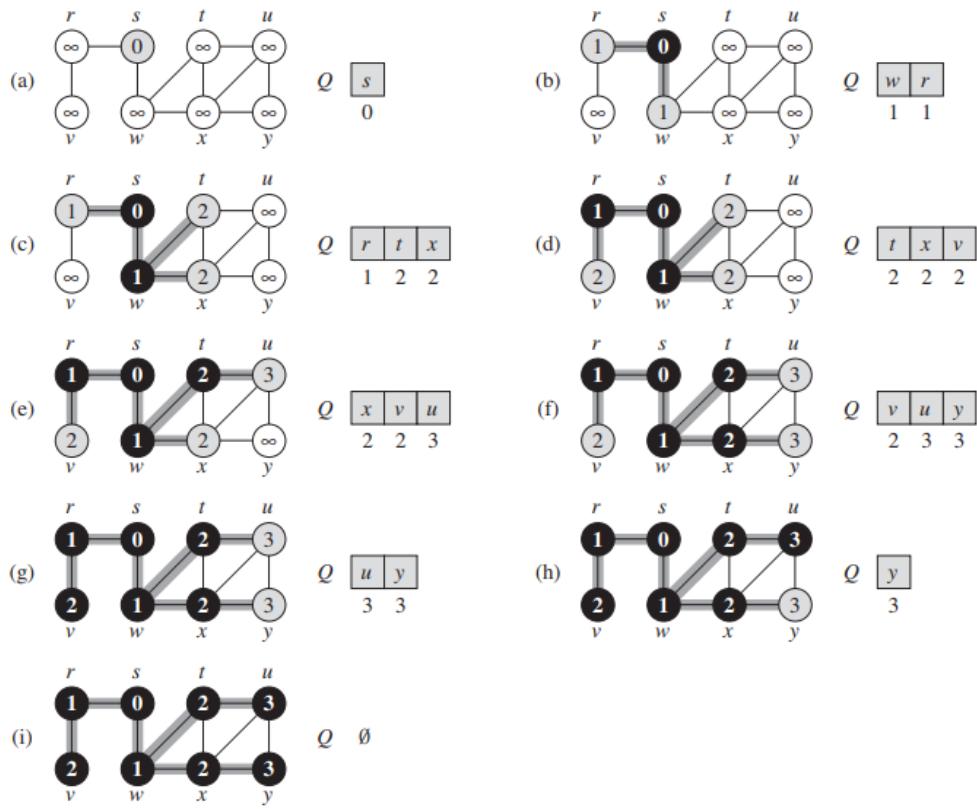
```

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

```

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).



Gambar 2.4: Pengoperasian BFS pada *undirected graph* (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 9–10 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 13–21 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*.

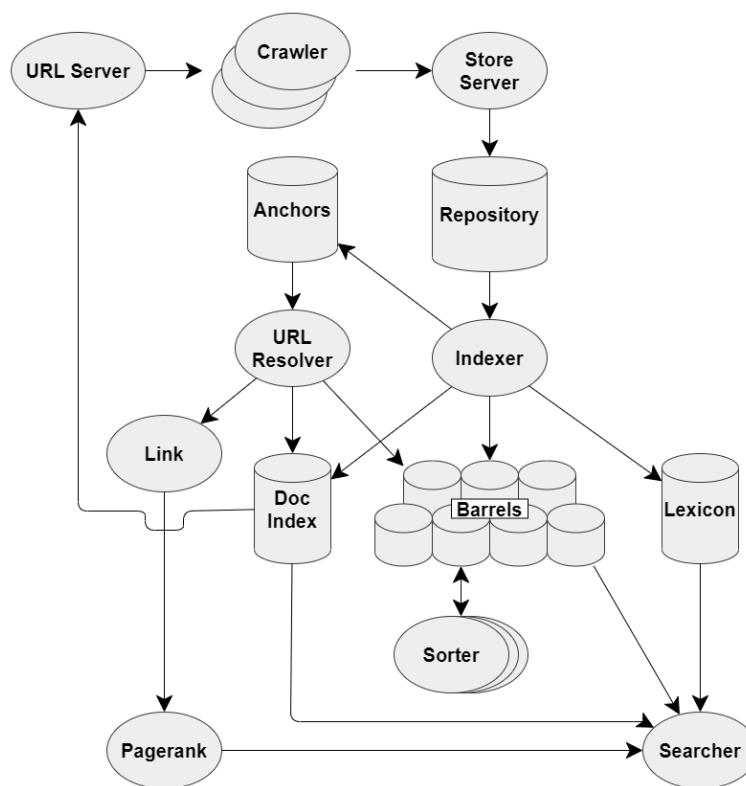
2.6 Definisi *Search Engine*

Menurut Seymour et al. (2011), *Search engine* 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.

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 untuk memilih bagaimana halaman tadi akan diindeks. Gambar 2.5 merupakan *High Level Google Architecture* (Brin and Page, 1998).



Gambar 2.5: High level google architecture (Brin and 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*.

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 *lexicon* yang dihasilkan oleh *indexer* dan menghasilkan *lexicon* baru yang dipakai oleh *searcher*. *Searcher* dijalankan oleh *web server* & memakai *lexicon* 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 mengunduh 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 *search engine* (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, \dots, 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 *collection*, 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). $IR'(P)$ berfungsi 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, \dots, 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:

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

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.

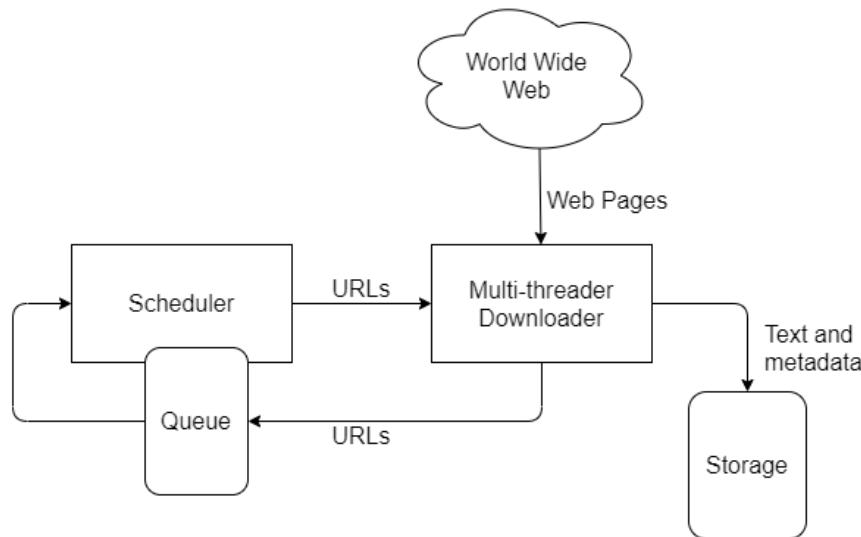
Crawl & Stop. Pada model ini, *crawler* C memulai halaman awalnya P_0 dan berhenti setelah mengunjungi halaman K . Pada titik ini *crawler* yang sempurna akan mengunjungi halaman R_1, \dots, R_K , dengan R_1 adalah halaman dengan nilai kepentingan tertinggi, R_2 adalah halaman tertinggi berikutnya, dan 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

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 algoritma 2 (Castillo, 2005).

Algorithm 2 Typical Crawling Model (Castillo, 2005)

Require: p_1, p_2, \dots, p_n starting URLs

```

1:  $Q = p_1, p_2, \dots, p_n$ , 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
  
```

URL page p_1, p_2, \dots, 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.

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
  
```

Algoritma 3 adalah *Modified similarity-based crawling algorithm*, algoritma ini merupakan algoritma yang dikembangkan oleh *Google* (Cho et al., 1998). 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.

BAB III

DESAIN MODEL

3.1 Desain *Crawler*

Pada proses pembuatan *crawler* di penelitian ini, penulis akan membuat crawler dengan model *modified similarity-based crawling* yang menggunakan algoritma *breadth first search* yang sudah dikembangkan. Akan 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. Maka pada model *modified similarity-based crawling* didalamnya juga terdapat model *typical crawling* yang menggunakan algoritma *breadth first search* dasar.

Crawler tidak mengambil data dari isi *database* web. Akan tetapi, *crawler* mengambil data dari struktur *HTML* web. Maka, *crawler* hanya akan berjalan pada *website statis*. Kemudian, struktur *HTML* dibedakan antara *HTML5* dan *HTML* versi sebelumnya. Pada penelitian ini ditargetkan 2 versi *HTML*, yaitu *HTML5* dan *HTML4*. Isi konten pada *HTML5* terletak pada bagian *tag article*, sedangkan isi konten *HTML4* diambil pada bagian *tag body* seluruhnya, dengan menghilangkan seluruh *tag* dan merapikannya dalam bentuk teks bersih.

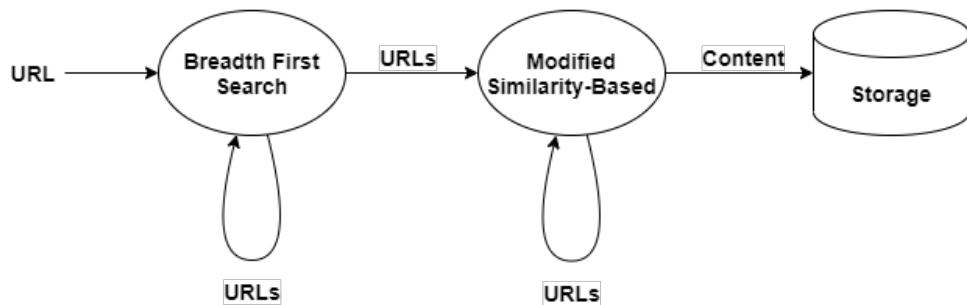
Crawler ini berbasis *terminal windows*, bahasa yang digunakan menggunakan bahasa pemrograman *python*. *Input crawler* ini berupa sebuah *link* dan *query* yang diujikan. Kemudian *output* dari *crawler* ini berupa data yang sudah dikumpulkan dari berbagai halaman web yang akan disimpan pada *database*. 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. Melihat data yang tersimpan pada database.

3.3 Arsitektur Diagram



Gambar 3.1: Arsitektur diagram

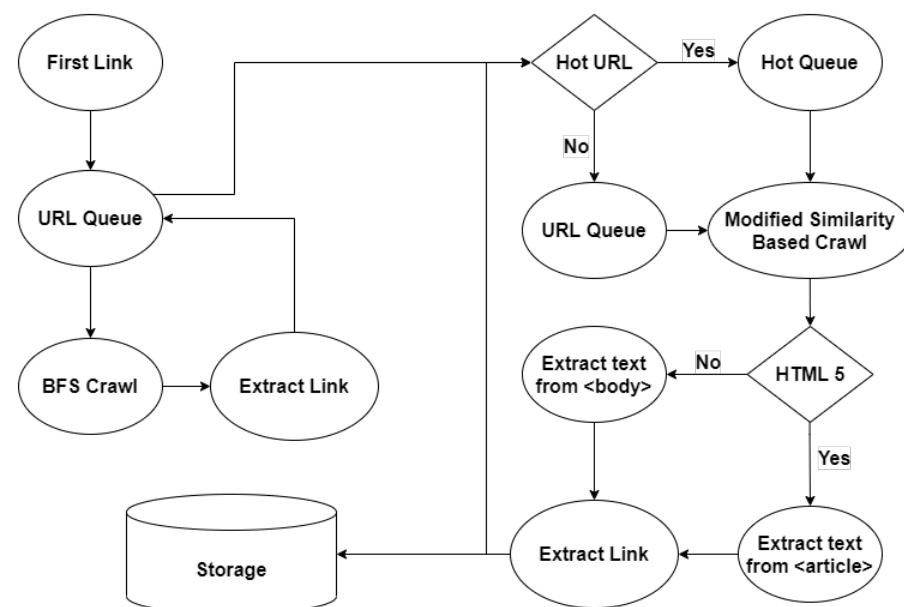
Terdapat dua komponen utama, yaitu *Breadth first search* dan *modified similarity-based*. *Crawler* ini berbasis terminal, inputnya berupa sebuah *URL* yang

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



Gambar 3.2: *Flowchart* algoritma

Flowchart Algoritma pada penelitian ini dapat dilihat di gambar 3.2. 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 untuk dilakukan proses *crawling*.

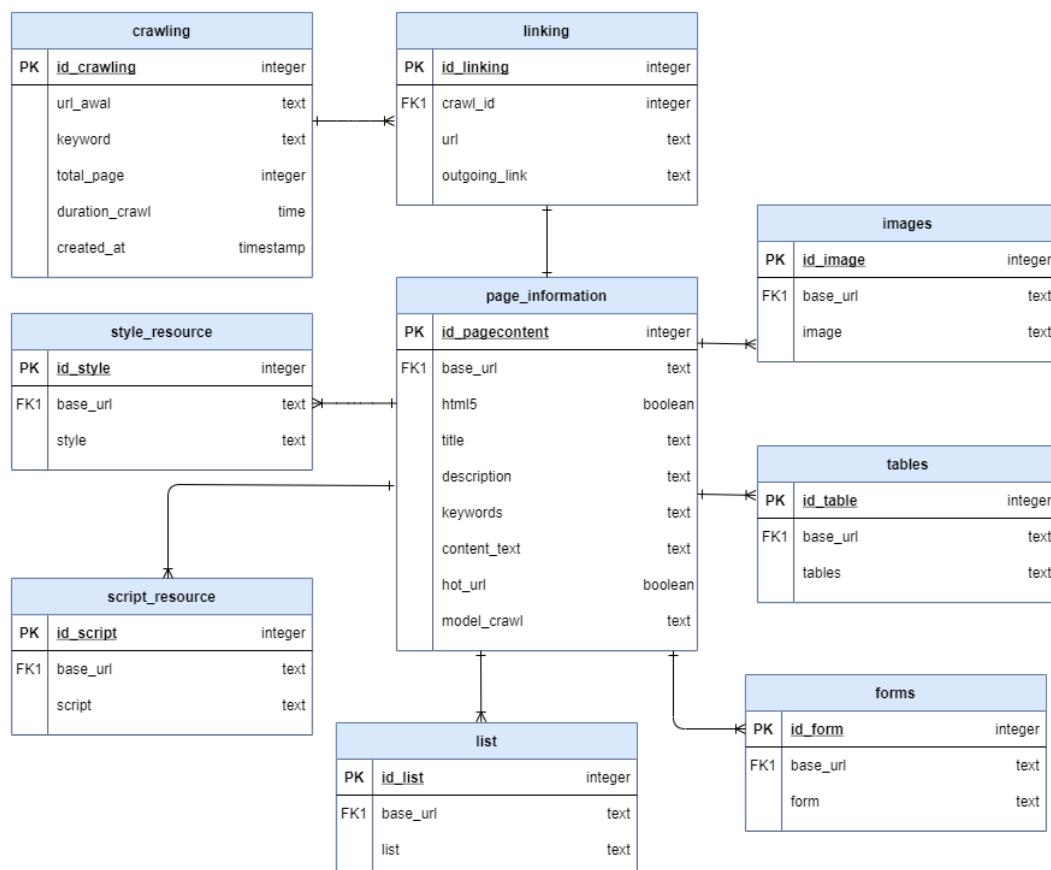
Crawler akan membedakan situs dengan *HTML* versi 5 atau *HTML* versi sebelumnya. *HTML5* ini dikhususkan karena informasi atau isi kontennya terletak pada bagian *tag article*. Selain *HTML5*, maka isi konten dianggap terletak pada *tag body*. Kemudian, hasil dari *crawling* tersebut akan disimpan dalam *Storage*.

3.5 Desain *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. Algoritma yang digunakan untuk crawling tersimpan di *model_crawl*.



Gambar 3.3: *Entity relationship diagram*

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 mencakup 2 bidang, yaitu dengan tema olahraga dan makanan. Tema olahraga dan makanan ini sangat umum untuk dibahas, dan cakupannya juga luas. Diantaranya terdapat sepakbola, basket, badminton, otomotif, makanan khas Asia, bahan dan resep makanan. Maka, penulis memilih tema olahraga dan makanan yang sudah umum dikenal masyarakat Indonesia.

Situs yang menjadi *node* awal penelitian ini adalah *Indosport* dengan nama situs: <https://www.indosport.com>, dan *Curious Cuisiniere* dengan nama situs: <https://www.curiouscuisiniere.com>. Portal web *Indosport* akan menjadi contoh situs yang menggunakan html versi 4, sedangkan *Curious Cuisiniere* akan menjadi contoh situs yang menggunakan html versi 5. Kemudian, untuk *query* yang akan diujikan pada algoritma *modified similarity based crawling* nantinya adalah:

1. *Barcelona*. Diujikan dengan *url* awal <https://www.indosport.com>.
2. *Rice*. Diujikan dengan *url* awal <https://www.curiouscuisiniere.com>.

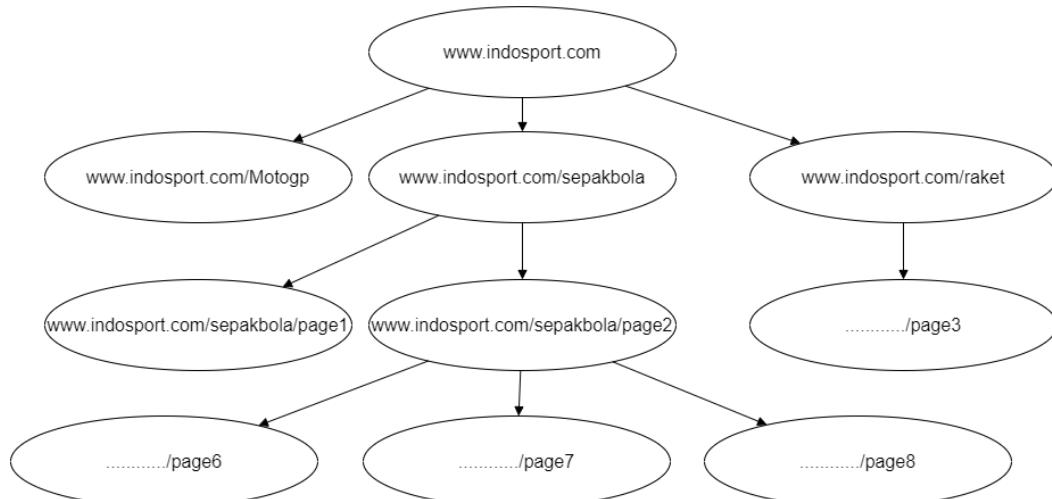
Hasil yang diharapkan adalah, *crawler* dapat mengutamakan *url* yang terdapat *keyword* pada *query* yang disebutkan.

3.7 Parameter Keberhasilan

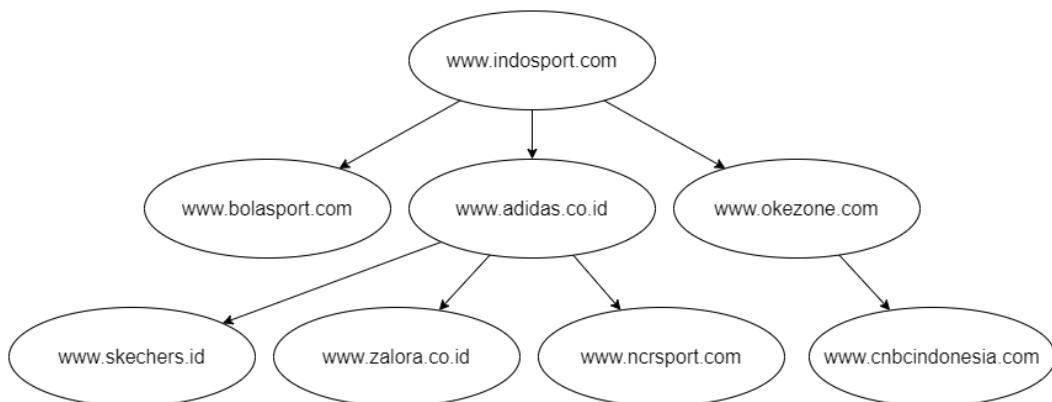
Breadth first search crawler dan *modified similarity-based crawler* yang sudah berjalan akan diuji untuk setiap *query*-nya. Untuk mengukur keberhasilan *breadth first search crawler* dan *modified similarity-based crawler* diperlukan dua situasi yang berbeda, yaitu:

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

4.1 Implementasi

Pada tahap implementasi, penulis memulai dari implementasi algoritma *Breadth First Search Crawling*, kemudian implementasi algoritma *Modified Similarity Based Crawling*, dan terakhir memasukan hasilnya kedalam *database*. Seluruh *code* diimplementasikan menggunakan bahasa *python* dan dijalankan melalui terminal *command prompt windows*. Semua *resource code* yang ada pada penelitian ini dapat dilihat di <https://github.com/fathang/web-crawler> tiap *code* tercatat sebagai *branch* pada github tersebut. Berikut merupakan hasil seluruh implementasi programnya.

4.1.1 *Library Python*

```
1 import os
2 import bs4
3 import requests
4 import re
5 import pandas as pd
6 from collections import deque, Counter
7 import urllib.request
8 from urllib.parse import urljoin
9 import networkx as nx
10 import matplotlib.pyplot as plt
11 import pydot
12 from networkx.drawing.nx_pydot import graphviz_layout
13 import time
14 import pymysql
```

Gambar 4.1: *Library python*

Gambar 4.1 adalah *library python* yang digunakan pada penelitian ini. Diantaranya yaitu: *os*, *bs4* atau *beautifulsoup4*, *requests*, *re*, *pandas*, *deque* dan *counter* diambil dari *collections*, *urljoin*, *networkx*, *matplotlib*, *pydot*, *graphviz*, *time*, dan *pymysql* untuk koneksi ke *database*.

4.1.2 Implementasi Algoritma *Breadth First Search Crawling*

Proses pertama terlihat pada gambar 4.2. *Crawler* membuat dua *queue*. Pertama untuk menyimpan seluruh *url* yang akan diproses, kedua untuk menyimpan seluruh *url* yang sudah diproses. *Crawler* juga membutuhkan *input url* awal dan sebuah *keyword* yang nantinya digunakan pada tahap selanjutnya.

Gambar 4.3 sampai gambar 4.8 adalah *function* bernama *function crawl*. *Function crawl* merupakan inti dari *code* ini, masukannya berupa sebuah *url*, dan algoritma ini merupakan algoritma rekursif. Terdapat juga *function tag_visible* (gambar 4.9), berguna untuk merapihkan *content text* yang didapat dari proses *crawling*.

```

21 # daftar url yang akan dicrawl
22 url_queue = deque([])
23
24 # daftar url yang sudah di crawl
25 visited_url = []
26
27 # titik awal: 1 situs
28 url_awal = input("URL Awal: ")

```

Gambar 4.2: Implementasi algoritma *breadth first search crawling* 1

```

46 def crawl(url):
47     """Function untuk crawling page.
48     """
49     try:
50         # kondisi berhenti
51         time_now = time.time() - start_time
52         time_now_int = int(time_now)
53         if time_now_int >= 3600:
54             return
55
56         # memasukan url kedalam visited_url
57         visited_url.append(url)
58
59         # crawl page
60         print("page yang sedang di crawl:", url)
61         page = requests.get(url)
62         request = page.content
63         soup = bs4.BeautifulSoup(request, 'html.parser')
64
65         # extract title
66         title = soup.title.string

```

Gambar 4.3: Implementasi algoritma *breadth first search crawling* 2

```

68     # check version html
69     article_html5 = soup.find('article')
70     if article_html5 is None:
71         # extract text content from html4
72         html5 = "no"
73         texts = soup.find('body').findAll(text=True)
74         visible_texts = filter(tag_visible, texts)
75         text = u" ".join(t.strip() for t in visible_texts)
76         text = text.lstrip().rstrip()
77         text = text.split(',')
78         clean_text = ''
79         for sen in text:
80             if sen:
81                 sen = sen.rstrip().lstrip()
82                 clean_text += sen+','
83         complete_text = clean_text
84         # print(complete_text)

```

Gambar 4.4: Implementasi algoritma *breadth first search crawling* 3

```

85     else:
86         # extract text content from html5
87         html5 = "yes"
88         texts = article_html5.findAll(text=True)
89         visible_texts = filter(tag_visible, texts)
90         text = u" ".join(t.strip() for t in visible_texts)
91         text = text.lstrip().rstrip()
92         text = text.split(',')
93         clean_text = ''
94         for sen in text:
95             if sen:
96                 sen = sen.rstrip().lstrip()
97                 clean_text += sen+','
98         complete_text = clean_text
99         # print(complete_text)
100
101     # get meta description
102     description = soup.find("meta", attrs={"name": "description"})
103     if description is None:
104         description = "_"
105     else:
106         description = description.get("content")

```

Gambar 4.5: Implementasi algoritma *breadth first search crawling* 4

```

108     # get meta keywords
109     keywords = soup.find("meta", attrs={"name": "keywords"})
110     if keywords is None:
111         keywords = "_"
112     else:
113         keywords = keywords.get("content")
114
115     # isHotURL
116     hot_link = "no"

```

Gambar 4.6: Implementasi algoritma *breadth first search crawling* 5

```

179     # extract outgoing link
180     links = soup.findAll("a", href=True)
181
182     # memasukan outgoing link kedalam queue
183     for i in links:
184         flag = 0
185
186         # Complete relative URLs and strip trailing slash
187         complete_url = urljoin(url, i["href"]).rstrip("/")
188
189         # create graph
190         # G.add_edges_from([(url, complete_url)])
191
192         # create list graph
193         branch = []
194         # remove https://
195         new_url = url.replace('https://', '')
196         new_url = new_url.replace('http://', '')
197         new_complete = complete_url.replace('https://', '')
198         new_complete = new_complete.replace('http://', '')
199         branch.append(new_url)
200         branch.append(new_complete)
201         list_g.append(branch)

```

Gambar 4.7: Implementasi algoritma *breadth first search crawling* 6

```

210     # Check if the URL already exists in the url_queue
211     for j in url_queue:
212         if j == complete_url:
213             flag = 1
214             break
215
216     # Check if the URL already exists in the visited_url
217     for j in visited_url:
218         if (j == complete_url):
219             flag = 1
220             break
221
222     # If not found in queue
223     if flag == 0:
224         if (visited_url.count(complete_url)) == 0:
225             url_queue.append(complete_url)
226
227     except (AttributeError, KeyError, requests.exceptions.InvalidSchema):
228         title = "no-title"
229         complete_text = "no-text"
230
231     # crawl url selanjutnya
232     if len(url_queue) == 0:
233         return
234     current = url_queue.popleft()
235     crawl(current)

```

Gambar 4.8: Implementasi algoritma *breadth first search crawling* 7

```

34 def tag_visible(element):
35     """Function untuk merapikan content text.
36     """
37     if element.parent.name in ['style', 'script', 'head', 'title', 'meta', '[document]']:
38         return False
39     if isinstance(element, bs4.element.Comment):
40         return False
41     if re.match(r"[\n]+", str(element)):
42         return False
43     return True

```

Gambar 4.9: Implementasi algoritma *breadth first search crawling* 8

4.1.3 Implementasi Algoritma *Modified Similarity Based Crawling*

Proses *crawling* menggunakan algoritma *Modified Similarity Based Crawling* membutuhkan *crawler* lain. Maka *crawler* ini berjalan setelah menjalankan *crawler* model sebelumnya. Implementasinya tidak banyak perubahan, yang menjadi tambahan pada *code* ini adalah terdapat *hot url*, *keyword*, dan juga *function reorder_queue*. Gambar 4.10 sampai gambar 4.13 adalah *code* tambahan yang hanya terdapat pada algoritma *modified similarity based crawling*.

```

4 def reorder_queue(queue):
5     """Function untuk reorder queue
6     """
7     value_backlink = []
8     for u in queue:
9         # menentukan nilai backlink_count
10        backlink_count = list_g.count(u)
11        # memasukan backlink_count ke array value_backlink
12        value_backlink.append(backlink_count)
13
14    # membuat dictionary backlink untuk proses sorting
15    backlink_dictionary = dict(zip(queue, value_backlink))
16    # sorting backlink_dictionary
17    sort_orders = sorted(backlink_dictionary.items(),
18                          key=lambda x: x[1], reverse=True)
19
20    # mengkosongkan queue
21    queue.clear()
22    # membuat queue yang sudah di sort
23    for i in sort_orders:
24        queue.append(i[0])
25
26    return queue

```

Gambar 4.10: Implementasi algoritma *modified similarity based crawling* 1

```

32  # membuat hot_queue
33  hot_queue = deque([])
34
35  # new start_time_MSB
36  start_time_MSB = time.time()

```

Gambar 4.11: Implementasi algoritma *modified similarity based crawling 2*

```

39  def jumlah_key_body(complete_text):
40      """Function untuk menghitung jumlah key di text
41      """
42      keyword = hot_key
43      jumlah_keyword = complete_text.count(keyword)
44      return jumlah_keyword
45
46
47  def jumlah_key_title(title):
48      """Function untuk menghitung jumlah key di title
49      """
50      keyword = hot_key
51      jumlah_keyword = title.count(keyword)
52      return jumlah_keyword

```

Gambar 4.12: Implementasi algoritma *modified similarity based crawling 3*

```

130  # check hot_url
131  hot_url = False
132  hot_link = "no"
133  if (jumlah_key_body(complete_text) >= 10) or (jumlah_key_title(title) >= 1):
134      hot_url = True
135      hot_link = "yes"

```

Gambar 4.13: Implementasi algoritma *modified similarity based crawling 4*

4.1.4 Koneksi Database

Database yang digunakan berupa versi 10.4.9.19-MariaDB. Cara menghubungkan antara *database* dengan *python*, yaitu dengan menggunakan *library* bernama *pymysql*. Gambar 4.14 sampai gambar 4.17 merupakan *code* untuk mengkoneksikan data ke *database*.

```

14  import pymysql
15
16  #connecting mysql database
17  db = pymysql.connect( host = 'localhost', user = 'root', passwd = '', db='dbcrawl')
18  # prepare a cursor object using cursor() method
19  cursor = db.cursor()

```

Gambar 4.14: Koneksi *database* 1

Data pada tabel *page_information* sifatnya dapat diperbarui. Jika *crawler* dijalankan dari awal, kemudian sudah tedapat *url* yang tersimpan, maka datanya akan tersimpan di dalam *row* yang sudah ada. Jika belum ada informasi mengenai *url* yang sedang dipindai, maka datanya akan ditambahkan.

```

122     # check table if exist at crawldb
123     cursor.execute(
124         "SELECT base_url, COUNT(*) FROM page_information WHERE base_url = %s GROUP BY base_url"
125         (url,))
126     )
127     results = cursor.fetchall()
128     # gets the number of rows affected by the command executed
129     row_count = cursor.rowcount
130     if row_count == 0:
131         # Create a new record
132         sql = "INSERT INTO `page_information` (`base_url`, `html5`, `title`, `description`, `keywords`, `complete_text`, `hot_link`)"
133         # Execute the query
134         cursor.execute(sql, (url, html5, title, description, keywords, complete_text, hot_link))
135         # commit to save our changes
136         db.commit()
137     else:
138         # update database
139         sql = "UPDATE page_information SET hot_url = %s WHERE base_url = %s"
140         # Execute the query
141         cursor.execute(sql, (hot_url, url))
142         # commit to save our changes
143         db.commit()

```

Gambar 4.15: Koneksi *database* 2

```

144     # extract style
145     for style in soup.findAll('style'):
146         # Create a new record
147         sql = "INSERT INTO `style_resource` (`base_url`, `style`) VALUES (%s, %s)"
148         # Execute the query
149         cursor.execute(sql, (url, style))
150         # commit to save our changes
151         db.commit()
152
153     # extract script
154     for script in soup.findAll('script'):
155         # Create a new record
156         sql = "INSERT INTO `script_resource` (`base_url`, `script`) VALUES (%s, %s)"
157         # Execute the query
158         cursor.execute(sql, (url, script))
159         # commit to save our changes
160         db.commit()
161
162     # extract lists
163     for lists in soup.findAll('li'):
164         # Create a new record
165         sql = "INSERT INTO `list` (`base_url`, `list`) VALUES (%s, %s)"
166         # Execute the query
167         cursor.execute(sql, (url, lists))
168         # commit to save our changes
169         db.commit()

```

Gambar 4.16: Koneksi *database* 3

```

171      # extract forms
172  for form in soup.findAll('form'):
173      # Create a new record
174      sql = "INSERT INTO `forms` (`base_url`, `form`) VALUES (%s, %s)"
175      # Execute the query
176      cursor.execute(sql, (url, form))
177      # commit to save our changes
178      db.commit()
179
180      # extract tables
181  for table in soup.findAll('table'):
182      # Create a new record
183      sql = "INSERT INTO `tables` (`base_url`, `tables`) VALUES (%s, %s)"
184      # Execute the query
185      cursor.execute(sql, (url, table))
186      # commit to save our changes
187      db.commit()
188
189      # extract images
190  for image in soup.findAll('img'):
191      # Create a new record
192      sql = "INSERT INTO `images` (`base_url`, `image`) VALUES (%s, %s)"
193      # Execute the query
194      cursor.execute(sql, (url, image))
195      # commit to save our changes
196      db.commit()

```

Gambar 4.17: Koneksi database 4

4.2 Referensi *Code*

Pada tahap pembuatan program. Penulis tidak sepenuhnya menulis isi seluruh *code*. Akan tetapi, penulis mendapatkan beberapa referensi *code* dari berbagai sumber. Tabel 4.1 merupakan penjelasan darimana sumber *code* didapat untuk pembuatan *crawler* dalam penelitian ini.

Tabel 4.1: Referensi *code*

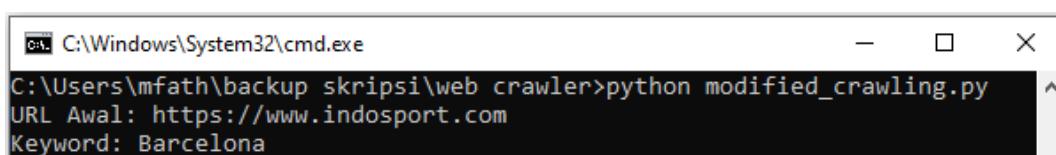
Code	Reference
<i>Simple BFS web crawler</i>	https://github.com/matharp/bfs-crawler
Koneksi pymysql	https://belajarpython.com/tutorial/akses-database-python
Merapihkan konten teks	https://stackoverflow.com/questions/1936466/beautifulsoup-grab-visible-webpage-text

4.3 Uji Coba

Uji coba dilakukan dengan menargetkan 2 situs awal yaitu:

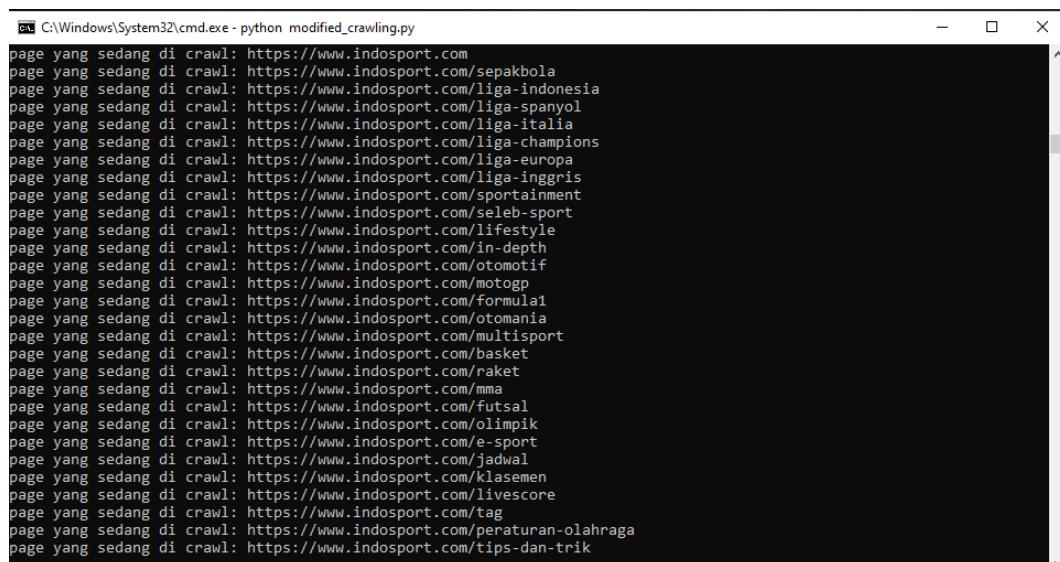
1. <https://www.indosport.com>. Website dengan *HTML* versi 4.
2. <https://www.curiouscuisiniere.com>. Website dengan *HTML* versi 5.

Keyword yang dimasukkan ada dua. Untuk *Indosport* yaitu "*Barcelona*" dan *Curious Cuisiniere* dengan *keyword* "*Rice*". Masing-masing *crawler* akan berjalan selama setengah jam dengan total waktu 1 jam. Menggunakan laptop tipe *Asus VivoBook 14 A442UR*.



```
C:\Windows\System32\cmd.exe
C:\Users\mfath\backup skripsi\web crawler>python modified_crawling.py
URL Awal: https://www.indosport.com
Keyword: Barcelona
```

Gambar 4.18: Uji coba 1



```
C:\Windows\System32\cmd.exe - python modified_crawling.py
page yang sedang di crawl: https://www.indosport.com
page yang sedang di crawl: https://www.indosport.com/sepakbola
page yang sedang di crawl: https://www.indosport.com/liga-indonesia
page yang sedang di crawl: https://www.indosport.com/liga-spanyol
page yang sedang di crawl: https://www.indosport.com/liga-italia
page yang sedang di crawl: https://www.indosport.com/liga-champions
page yang sedang di crawl: https://www.indosport.com/liga-eropa
page yang sedang di crawl: https://www.indosport.com/liga-inggris
page yang sedang di crawl: https://www.indosport.com/sportainment
page yang sedang di crawl: https://www.indosport.com/seleb-sport
page yang sedang di crawl: https://www.indosport.com/lifestyle
page yang sedang di crawl: https://www.indosport.com/in-depth
page yang sedang di crawl: https://www.indosport.com/otomotif
page yang sedang di crawl: https://www.indosport.com/motogp
page yang sedang di crawl: https://www.indosport.com/formula1
page yang sedang di crawl: https://www.indosport.com/otomania
page yang sedang di crawl: https://www.indosport.com/multisport
page yang sedang di crawl: https://www.indosport.com/basket
page yang sedang di crawl: https://www.indosport.com/raket
page yang sedang di crawl: https://www.indosport.com/mma
page yang sedang di crawl: https://www.indosport.com/futsal
page yang sedang di crawl: https://www.indosport.com/olimpik
page yang sedang di crawl: https://www.indosport.com/e-sport
page yang sedang di crawl: https://www.indosport.com/jadwal
page yang sedang di crawl: https://www.indosport.com/klasemen
page yang sedang di crawl: https://www.indosport.com/livescore
page yang sedang di crawl: https://www.indosport.com/tag
page yang sedang di crawl: https://www.indosport.com/peraturan-olahraga
page yang sedang di crawl: https://www.indosport.com/tips-dan-trik
```

Gambar 4.19: Uji coba 2

Gambar 4.18 adalah tampilan *crawler* yang akan berjalan pada terminal *command prompt windows*. *Crawler* yang sedang berjalan dapat terlihat pada

gambar 4.19, *crawler* menampilkan halaman yang saat ini sedang *crawling*. Kemudian, ketika *crawler* berhenti, *crawler* menunjukkan jumlah *url* yang sudah diproses, jumlah *url* yang masih dalam *queue*, dan waktu yang dibutuhkan untuk proses *crawling*.

```

C:\Windows\System32\cmd.exe
page yang sedang di crawl: http://portuguesediner.com/tiamaria/visiting-soon
page yang sedang di crawl: http://portuguesediner.com/tiamaria/category/photo-gallery
page yang sedang di crawl: http://portuguesediner.com/tiamaria/category/recipes
page yang sedang di crawl: http://portuguesediner.com/tiamaria/category/rice-2
page yang sedang di crawl: http://portuguesediner.com/tiamaria/author/administrator
page yang sedang di crawl: http://portuguesediner.com/tiamaria/arroz-de-tomate-tomato-flavored-rice/dsc04467
page yang sedang di crawl: http://portuguesediner.com/tiamaria/arroz-de-tomate-tomato-flavored-rice/dsc04467
page yang sedang di crawl: http://catavino.net/portuguese-rice-arroz-carolino
page yang sedang di crawl: http://portuguesediner.com/tiamaria/arroz-de-tomate-tomato-flavored-rice/dsc04455
page yang sedang di crawl: http://portuguesediner.com/tiamaria/arroz-de-tomate-tomato-flavored-rice/dsc04457
page yang sedang di crawl: http://portuguesediner.com/tiamaria/arroz-de-tomate-tomato-flavored-rice/dsc04461
page yang sedang di crawl: http://portuguesediner.com/tiamaria/tag/arroz
page yang sedang di crawl: http://portuguesediner.com/tiamaria/tag/rice
page yang sedang di crawl: http://portuguesediner.com/tiamaria/arroz-de-camarao-portuguese-shrimp-rice
page yang sedang di crawl: http://portuguesediner.com/tiamaria/arroz-de-bacalhau
page yang sedang di crawl: http://portuguesediner.com/tiamaria/chick-pea-rice
page yang sedang di crawl: http://portuguesediner.com/tiamaria/index.php/arroz-de-tomate-tomato-flavored-rice#respond
page yang sedang di crawl: https://www.facebook.com/pages/Tia-Marias-Blog/463406760067
page yang sedang di crawl: http://www.pinterest.com/tiamariablog/tia-maria-s-blog
page yang sedang di crawl: https://twitter.com/tiamariablog
page yang sedang di crawl: https://www.youtube.com/channel/UcvTPMD8CqSrZFL7NUs0RBWQ
page yang sedang di crawl: http://instagram.com/tiamariablog
page yang sedang di crawl: http://www.amazon.ca/Taste-Portugal-Easy-Portuguese-Recipes/dp/1500978663/ref=cm_cr_pr_product_top?ie=UTF8
page yang sedang di crawl: https://www.facebook.com/463406760067
Jumlah url yg sudah dilihat: 604
Jumlah url dalam queue: 5634
Jumlah url dalam hot queue: 87
Waktu yang dibutuhkan: 1809.7991724014282 detik
C:\Users\mfath\backup skripsi\web_crawler>

```

Gambar 4.20: Uji coba 3

```

C:\Windows\System32\cmd.exe
page yang sedang di crawl: https://www.indosport.com/futsal/20210127/afp-sulsel-akan-rampungkan-2-program-besar-dalam-waktu-dekat
page yang sedang di crawl: https://www.indosport.com/futsal/20210112/pelatih-futsal-jawa-barat-siapkan-progam-untuk-persaingan-pon-xx-papua
page yang sedang di crawl: https://www.indosport.com/futsal/20201111/batal-mentas-tahun-ini-timnas-futsal-indonesia-tetapi-semangat-ikut-tc
page yang sedang di crawl: https://www.indosport.com/futsal/20201019/jelang-kejuaraan-afc-timnas-futsal-indonesia-kejarnas-level-tertinggi
page yang sedang di crawl: https://www.indosport.com/futsal/20201016/baru-saja-potong-tumpeng-bagaimana-nasib-tc-timnas-futsal-indonesia
page yang sedang di crawl: https://www.indosport.com/futsal/20201015/resmi-afc-futsal-championship-ditunda-tahun-depan
page yang sedang di crawl: https://www.indosport.com/futsal/20201015/berkumpul-dalam-kondisi-sehat-timnas-futsal-indonesia-gelar-syukuran
page yang sedang di crawl: https://www.indosport.com/futsal/20200721/ketua-afp-sulsel-akui-pembinaan-futsal-terhambat-akibat-virus-corona
page yang sedang di crawl: https://www.indosport.com/futsal/20200715/ffi-hapus-liga-futsal-nusantara-2020-afp-sulsel-beri-komentar
page yang sedang di crawl: https://www.indosport.com/futsal/20200608/pfl-2020-terhenti-pemain-cosmo-fc-latihan-mandiri
page yang sedang di crawl: https://www.indosport.com/futsal/20200515/unik-metode-latihan-kiper-di-klub-futsal-asal-jepang-selama-corona
page yang sedang di crawl: https://www.indosport.com/futsal/20200513/resmi-fifa-usulkan-jadwal-baru-piala-dunia-futsal-2020
page yang sedang di crawl: https://www.indosport.com/esports/20210727/resmi-dibatalkan-etimnas-indonesia-batal-tampil-di-piala-dunia
Jumlah url yg sudah dilihat: 476
Jumlah url dalam queue: 3509
Jumlah url dalam hot queue: 0
Waktu yang dibutuhkan: 1812.9706456661224 detik
C:\Users\mfath\backup skripsi\web_crawler>

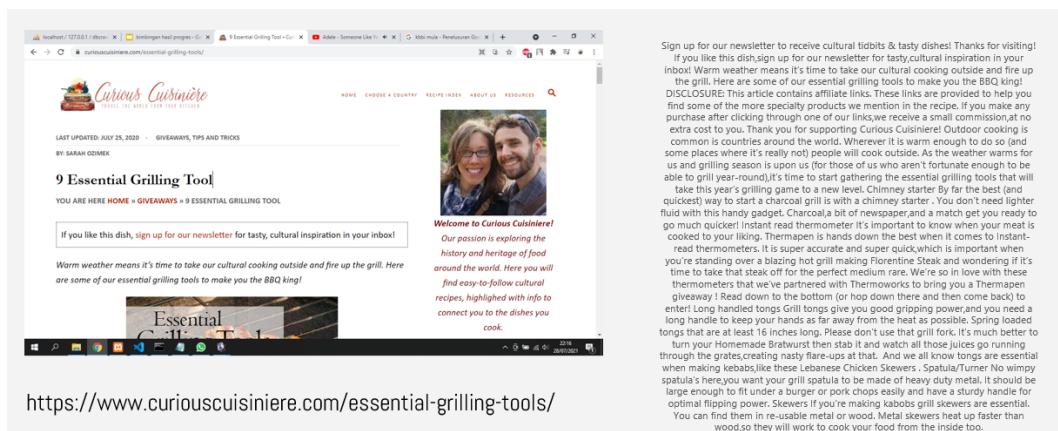
```

Gambar 4.21: Uji coba 4

Gambar 4.20 adalah tampilan *crawler* yang sudah berhenti untuk *html5*.

Gambar 4.21 adalah tampilan *crawler* yang sudah berhenti untuk *html4*.

4.4 Hasil Uji Coba



Gambar 4.22: Hasil uji coba 1

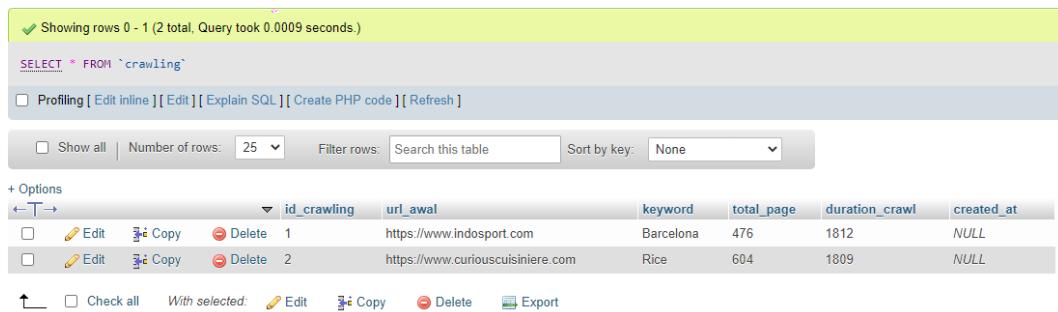
Hasil uji coba pada penelitian ini akan menunjukkan tampilan *database* yang sudah terisi data selama proses *crawling*. Untuk hasil ekstrak teks yang didapat dari tiap halaman terlihat pada gambar 4.22. Gambar 4.23 adalah *screenshot database dbcrawl* yang menunjukkan total *size* data yang disimpan sebanyak 176 MiB atau sekitar 185 Mb.

Table	Action	Rows	Type	Collation	Size	Overhead
crawling		2	InnoDB	utf8mb4_general_ci	16.0 Kib	-
forms		2,476	InnoDB	utf8mb4_general_ci	6.5 MiB	-
images		38,644	InnoDB	utf8mb4_general_ci	21.5 MiB	-
linking		~131,581	InnoDB	utf8mb4_general_ci	21.5 MiB	-
list		~68,341	InnoDB	utf8mb4_general_ci	35.6 MiB	-
page_information		1,060	InnoDB	utf8mb4_general_ci	8.5 MiB	-
script_resource		29,732	InnoDB	utf8mb4_general_ci	49.5 MiB	-
style_resource		7,540	InnoDB	utf8mb4_general_ci	32.5 MiB	-
tables		75	InnoDB	utf8mb4_general_ci	272.0 Kib	-
9 tables	Sum	~279,451				176.0 MiB

Gambar 4.23: Hasil uji coba 2

Hasil dari kinerja *crawler* dapat dilihat pada tabel *crawling* (gambar 4.24), menunjukkan hasil *crawling* 2 situs. Untuk situs awal

<https://www.indosport.com>, didapat *total page* sebanyak 476 halaman, dengan durasi 1812 detik, maka *crawler* membutuhkan waktu sekitar 3,8 detik per halaman. Sedangkan situs awal <https://www.curiouscuisiniere.com>, didapat *total page* sebanyak 664, dengan durasi 1809 detik, maka *crawler* membutuhkan waktu sekitar 2,7 detik per halaman.



The screenshot shows a MySQL query results page. At the top, it says "Showing rows 0 - 1 (2 total, Query took 0.0009 seconds)". Below that is the SQL query: "SELECT * FROM `crawling`". There is a checkbox for "Profiling" followed by links for "Edit inline", "Edit", "Explain SQL", "Create PHP code", and "Refresh". Below the query are buttons for "Show all" (unchecked), "Number of rows: 25", "Filter rows: Search this table", and "Sort by key: None". A "None" dropdown menu is open. The main area displays a table with the following data:

		id_crawling	url_awal	keyword	total_page	duration_crawl	created_at
<input type="checkbox"/>	Edit Copy Delete	1	https://www.indosport.com	Barcelona	476	1812	NULL
<input type="checkbox"/>	Edit Copy Delete	2	https://www.curiouscuisiniere.com	Rice	604	1809	NULL

At the bottom, there are buttons for "Check all", "With selected: Edit Copy Delete", and "Export".

Gambar 4.24: Hasil uji coba 3

Gambar 4.25 sampai gambar 4.32 adalah hasil *screenshot* setiap tabel pada *database* *dbcrawl*. Data yang diperoleh untuk setiap tabelnya ditampilkan sebagai berikut:

1. Pada tabel *page_information* terdapat total 1060 halaman web.
2. tabel *linking*, sebanyak 131.581 total *linking*.
3. tabel *style_resouce*, sebanyak 7.530 total *css*.
4. tabel *script_resource*, sebanyak 29.732 total *js*.
5. tabel *forms*, sebanyak 2.476 total *form*.
6. tabel *images*, sebanyak 38.644 total *image*.
7. tabel *list*, sebanyak 68.341 total *list*.
8. tabel *tables*, sebanyak 75 total *table*.

Image pada hasil *crawling* disimpan sebagai *text*, karena data yg diambil sangat banyak. Terdapat total 38.644 *image*, jika disimpan dalam bentuk *png* atau *jpg*, maka dibutuhkan ukuran *memory* yang cukup besar. Pada penelitian ini hanya disimpan dalam bentuk *text* saja.

Showing rows 0 - 24 (1060 total, Query took 0.0049 seconds.)							
SELECT * FROM `page_information`							
<input type="checkbox"/> Profiling Edit inline Edit Explain SQL Create PHP code Refresh							
1	>	>>	Number of rows:	25	Filter rows: <input type="text" value="Search this table"/>	Sort by key:	<input type="button" value="None"/>
+ Options	← →	▼	id_pagecontent	base_url	html5	title	description
<input type="checkbox"/>	Edit	Copy	Delete 1	https://www.indosport.com	no	INDOSPORT - Berita Olahraga - Portal Berita Olahraga dan Terkini dan Sepak Bola...	Indosport.com Berita Olahraga, Berita Sepak Bola, Sepak Bola, Berita Bola, Be...
<input type="checkbox"/>	Edit	Copy	Delete 2	https://www.indosport.com/sepakbola	no	Berita Olahraga Sepak Bola - INDOSPORT	Berita Sepak Bola, Liga Indonesia, Liga Inggris, L...

Gambar 4.25: Hasil uji coba 4

Showing rows 0 - 24 (131581 total, Query took 0.0075 seconds.)					
SELECT * FROM `linking`					
<input type="checkbox"/> Profiling Edit inline Edit Explain SQL Create PHP code Refresh					
1	>	>>	Number of rows:	25	Filter rows: <input type="text" value="Search this table"/>
+ Options	← →	▼	id_linking	crawl_id	url
<input type="checkbox"/>	Edit	Copy	Delete 1	1	https://www.indosport.com https://www.indosport.com
<input type="checkbox"/>	Edit	Copy	Delete 2	1	https://www.indosport.com https://www.indosport.com/sepakbola
<input type="checkbox"/>	Edit	Copy	Delete 3	1	https://www.indosport.com https://www.indosport.com/sepakbola
<input type="checkbox"/>	Edit	Copy	Delete 4	1	https://www.indosport.com https://www.indosport.com/liga-indonesia

Gambar 4.26: Hasil uji coba 5

Showing rows 0 - 24 (7540 total. Query took 0.0036 seconds.)

SELECT * FROM `style_resource`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

	<input type="button" value="1"/>	<input type="button" value=">"/>	<input type="button" value=">>"/>	Number of rows:	25	Filter rows:	Search this table	Sort by key:	None
+ Options									
<input type="button" value="←"/> <input type="button" value="→"/> <input type="button" value="id_style"/> <input type="button" value="base_url"/> <input type="button" value="style"/>									
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1	https://www.facebook.com/indosportdotcom	<style nonce="JzYXsUel"></style>			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	2	https://www.youtube.com/c/indosportdotcom	<style class="global_styles" nonce="RVqArWIU4lfbT7...>			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	3	https://www.youtube.com/c/indosportdotcom	<style class="masthead_shell" nonce="RVqArWIU4lfbT...>			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	4	https://www.youtube.com/c/indosportdotcom	<style class="masthead_custom_styles" id="ext-styl...>			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	5	https://www.youtube.com/c/indosportdotcom	<style class="searchbox" nonce="RVqArWIU4lfbT7w89N...>			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	6	https://www.youtube.com/c/indosportdotcom	<style class="kevlar_global_styles" nonce="RVqArWI...>			

Gambar 4.27: Hasil uji coba 6

Showing rows 0 - 24 (29732 total. Query took 0.0024 seconds.)

SELECT * FROM `script_resource`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

	<input type="button" value="1"/>	<input type="button" value=">"/>	<input type="button" value=">>"/>	Number of rows:	25	Filter rows:	Search this table	Sort by key:	None
+ Options									
<input type="button" value="←"/> <input type="button" value="→"/> <input type="button" value="id_script"/> <input type="button" value="base_url"/> <input type="button" value="script"/>									
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1	https://www.indosport.com	<script type="text/javascript"> window.ga=windo...			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	2	https://www.indosport.com	<script async="" src="https://www.google-analytics...>			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	3	https://www.indosport.com	<script type="text/javascript"> _atrk_opt...			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	4	https://www.indosport.com	<script> var _comscore = _comscore [];			

Gambar 4.28: Hasil uji coba 7

Showing rows 0 - 24 (2476 total. Query took 0.0019 seconds.)

SELECT * FROM `forms`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

	<input type="button" value="1"/>	<input type="button" value=">"/>	<input type="button" value=">>"/>	Number of rows:	25	Filter rows:	Search this table	Sort by key:	None
+ Options									
<input type="button" value="←"/> <input type="button" value="→"/> <input type="button" value="id_form"/> <input type="button" value="base_url"/> <input type="button" value="form"/>									
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1	https://www.indosport.com	<form action="https://www.indosport.com/search" me...			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	2	https://www.indosport.com	<form action="https://www.indosport.com/search" me...			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	3	https://www.indosport.com	<form action="https://www.indosport.com/search" me...			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	4	https://www.indosport.com/sepkbola	<form action="https://www.indosport.com/search" me...			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	5	https://www.indosport.com/sepkbola	<form action="https://www.indosport.com/search" me...			
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	6	https://www.indosport.com/sepkbola	<form action="https://www.indosport.com/search" me...			

Gambar 4.29: Hasil uji coba 8

Showing rows 0 - 24 (38644 total, Query took 0.0077 seconds.)

```
SELECT * FROM `images`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

		id_image	base_url	image								
<input type="checkbox"/>	Edit	Copy	Delete	1 https://www.indosport.com	Edit	Copy	Delete	2 https://www.indosport.com	Edit	Copy	Delete	3 https://www.indosport.com <img alt="Logo Indosport" class="img-responsive" s...
<input type="checkbox"/>	Edit	Copy	Delete	4 https://www.indosport.com <img alt="Logo Indosport" class="img-responsive" s...								
<input type="checkbox"/>	Edit	Copy	Delete	5 https://www.indosport.com <img alt="Sudah Deall 3 Bintang yang Jadi Korban K..."								
<input type="checkbox"/>	Edit	Copy	Delete	6 https://www.indosport.com <img alt="Bedah Peluang All Indonesian Final di Ga..."								

Gambar 4.30: Hasil uji coba 9

Showing rows 0 - 24 (68341 total, Query took 0.0021 seconds.)

```
SELECT * FROM `list`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

		id_list	base_url	list
<input type="checkbox"/>	Edit	Copy	Delete	1 https://www.indosport.com <li class="bc_home"><a href="https://www.indospo...
<input type="checkbox"/>	Edit	Copy	Delete	2 https://www.indosport.com <li class="bc_sepakbola"><a class="openslidemenu ...
<input type="checkbox"/>	Edit	Copy	Delete	3 https://www.indosport.com <a href="https://www.indosport.com/sepakbola"...
<input type="checkbox"/>	Edit	Copy	Delete	4 https://www.indosport.com <a href="https://www.indosport.com/liga-indon...

Gambar 4.31: Hasil uji coba 10

Showing rows 0 - 24 (75 total, Query took 0.0091 seconds.)

```
SELECT * FROM `tables`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

		id_table	base_url	tables
<input type="checkbox"/>	Edit	Copy	Delete	1 https://www.indosport.com <table class="table-klasemen"><thead><tr><th>#<...>
<input type="checkbox"/>	Edit	Copy	Delete	2 https://www.indosport.com <table class="table-klasemen"><thead><tr><th>#<...>

Gambar 4.32: Hasil uji coba 11

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil implementasi dan pengujian *crawler* yang telah dirancang, maka diperoleh kesimpulan sebagai berikut:

1. *Crawler* merupakan sebuah program yang berjalan untuk mengumpulkan informasi yang ada di halaman sebuah *website*.
2. Algoritma *crawler* yang digunakan pada penelitian ini adalah algoritma *breadth first search crawling* dan *algoritma modified similarity based crawling* yang dikembangkan oleh *Google*. Bahasa pemrograman yang digunakan seluruhnya menggunakan bahasa *python*, kemudian seluruh datanya disimpan pada *database MySQL*.
3. *Crawler* belum bisa menemukan seluruh informasi pada situs yang sudah memiliki banyak halaman, seperti situs *indosport*. Proses *crawling* yang cukup lama dan juga kapasitas memori yang terbatas merupakan masalah utama pada saat menjalankan *crawler*. Tetapi untuk situs yang masih kecil, *crawler* mampu menemukan seluruh informasinya.
4. *Crawler* dapat berjalan dengan baik di situs yang menggunakan *html* versi 4 maupun *html* versi 5. Akan tetapi, *crawler* hanya bekerja pada *website statis*.

5.2 Saran

Adapun saran untuk penelitian selanjutnya adalah:

1. Sebaiknya dapat crawling pada *website dinamis*, kemudian juga dapat *crawling multiple* situs.
2. Mengembangkan algoritma yang telah ada, supaya *crawler* dapat bekerja lebih efisien.
3. Meningkatkan durasi *crawling*, hingga *crawler* dapat menemukan seluruh informasi di *website*.
4. Menyambungkan *crawler* ke tahap pengembangan *search engine* berikutnya, agar dapat menjadi sebuah *search engine* yang utuh.

DAFTAR PUSTAKA

- Brin, S., Motwani, R., Page, L., and Winograd, T. (1998). What can you do with a web in your pocket? *IEEE Data Eng. Bull.*, 21(2):37–47.
- Brin, S. and 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., and Page, L. (1998). Efficient crawling through url ordering.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.
- Eichmann, D. (1994). The rbse spider-balancing effective search against web load. In *Proc. 1st WWW Conf.* Citeseer.
- Henshaw, J. (2020). Apple showing signs it may soon launch a search engine to compete against google search. *Coywolf News*.
- Kaur, S. and Geetha, G. (2020). Simhar-smart distributed web crawler for the hidden web using sim+ hash and redis server. *IEEE Access*, 8:117582–117592.
- Moz (2020). Urls.
- NetMarketShare (2020). Search engine market share.
- Page, L., Brin, S., Motwani, R., and 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., and 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.
- WebHypertextApplicationTechnologyWorkingGroup (2020). Url standard.
- William, B. K. and Sawyer, S. C. (2001). Using information technology: A practical introduction to computers and communications.

LAMPIRAN A

Sampel Kode *Breadth First Search Crawling*

```
import os
import bs4
import requests
import re
import pandas as pd
from collections import deque, Counter
import urllib.request
from urllib.parse import urljoin
import networkx as nx
import matplotlib.pyplot as plt
import pydot
from networkx.drawing.nx_pydot import graphviz_layout
import time
import sys
import pymysql

# Using sys.setrecursionlimit() method
sys.setrecursionlimit(100000000)

#conneting mysql database
db = pymysql.connect( host = 'localhost',
user = 'root', passwd = '', db='dbcrawl')
# prepare a cursor object using cursor() method
```

```
cursor = db.cursor()

# daftar url yang akan dicrawl
url_queue = deque([])

# daftar url yang sudah di crawl
visited_url = []

# titik awal: 1 situs
url_awal = input("URL Awal: ")

# keyword
hot_key = input("Keyword: ")

def tag_visible(element):
    """Function untuk merapihkan content text.

    """
    if element.parent.name in ['style', 'script',
        'head', 'title', 'meta', '[document]']:
        return False
    if isinstance(element, bs4.element.Comment):
        return False
    if re.match(r"\n+", str(element)):
        return False
    return True
```

```
def crawl(url):  
    """Function untuk crawling page.  
    """  
  
    try:  
  
        # kondisi berhenti  
        time_now = time.time() - start_time  
        time_now_int = int(time_now)  
        if time_now_int >= 900:  
            return  
  
        # memasukan url kedalam visited_url  
        visited_url.append(url)  
  
        # crawl page  
        print("page yang sedang di crawl:", url)  
        page = requests.get(url)  
        request = page.content  
        soup = bs4.BeautifulSoup(request,  
                               'html.parser')  
  
        # extract title  
        title = soup.title.string  
  
        # check version html
```

```
article_html5 = soup.find('article')

if article_html5 is None:

    # extract text content from html4

    html5 = "no"

    texts = soup.find('body').findAll(text=True)

    visible_texts = filter(tag_visible, texts)

    text = u" ".join(t.strip() for t in

                      visible_texts)

    text = text.lstrip().rstrip()

    text = text.split(',')

    clean_text = ''

    for sen in text:

        if sen:

            sen = sen.rstrip().lstrip()

            clean_text += sen+','

    complete_text = clean_text

    # print(complete_text)

else:

    # extract text content from html5

    html5 = "yes"

    texts = article_html5.findAll(text=True)

    visible_texts = filter(tag_visible, texts)

    text = u" ".join(t.strip() for t in

                      visible_texts)

    text = text.lstrip().rstrip()

    text = text.split(',')  

```

```
clean_text = ''  
  
for sen in text:  
  
    if sen:  
  
        sen = sen.rstrip().lstrip()  
        clean_text += sen+','  
  
complete_text = clean_text  
  
# print(complete_text)  
  
  
# get meta description  
  
description = soup.find("meta",  
attrs={"name": "description"})  
  
if description is None:  
  
    description = "--"  
  
else:  
  
    description = description.get("content")  
  
  
# get meta keywords  
  
keywords = soup.find("meta",  
attrs={"name": "keywords"})  
  
if keywords is None:  
  
    keywords = "--"  
  
else:  
  
    keywords = keywords.get("content")  
  
  
# isHotURL  
  
hot_link = "no"
```

```
# check table if exist at crawldb
cursor.execute(
    "SELECT base_url, COUNT(*) FROM
     page_information WHERE
     base_url = %s GROUP BY base_url",
    (url,))
results = cursor.fetchall()
# gets the number of rows affected by
the command executed
row_count = cursor.rowcount
if row_count == 0:
    # Create a new record
    sql = "INSERT INTO `page_information` (
        `base_url`, `html5`, `title`,
        `description`, `keywords`,
        `content_text`, `hot_url`,
        `model_crawl`) VALUES (%s, %s, %s, %s,
        %s, %s, %s, %s)"
    # Execute the query
    cursor.execute(sql, (url, html5, title,
        description, keywords, complete_text,
        hot_link, "BFS crawling"))
    # commit to save our changes
    db.commit()
```

```
else:

    # update database

    sql = "UPDATE page_information SET
        hot_url = %s WHERE base_url = %s"
        # Execute the query

    cursor.execute(sql, (hot_url, url))

    # commit to save our changes

    db.commit()

# extract style

for style in soup.findAll('style'):

    # Create a new record

    sql = "INSERT INTO `style_resource` (
        `base_url`, `style`) VALUES (%s, %s)"
        # Execute the query

    cursor.execute(sql, (url, style))

    # commit to save our changes

    db.commit()

# extract script

for script in soup.findAll('script'):

    # Create a new record

    sql = "INSERT INTO `script_resource` (
        `base_url`, `script`) VALUES (%s, %s)"
        # Execute the query

    cursor.execute(sql, (url, script))
```

```
# commit to save our changes
db.commit()

# extract lists
for lists in soup.findAll('li'):
    # Create a new record
    sql = "INSERT INTO `list` (`base_url`,
    `list`) VALUES (%s, %s)"
    # Execute the query
    cursor.execute(sql, (url, lists))
    # commit to save our changes
    db.commit()

# extract forms
for form in soup.findAll('form'):
    # Create a new record
    sql = "INSERT INTO `forms` (`base_url`,
    `form`) VALUES (%s, %s)"
    # Execute the query
    cursor.execute(sql, (url, form))
    # commit to save our changes
    db.commit()

# extract tables
for table in soup.findAll('table'):
    # Create a new record
```

```
sql = "INSERT INTO `tables` ('base_url',  
    `tables`) VALUES (%s, %s)"  
  
# Execute the query  
cursor.execute(sql, (url, table))  
  
# commit to save our changes  
db.commit()  
  
  
# extract images  
for image in soup.findAll('img'):  
  
    # Create a new record  
    sql = "INSERT INTO `images` ('base_url',  
        `image`) VALUES (%s, %s)"  
  
    # Execute the query  
    cursor.execute(sql, (url, image))  
  
    # commit to save our changes  
    db.commit()  
  
  
# extract outgoing link  
links = soup.findAll("a", href=True)  
  
  
# memasukan outgoing link kedalam queue  
for i in links:  
  
    flag = 0  
  
  
    # Complete relative URLs  
    # and strip trailing slash
```

```
complete_url = urljoin(
    url, i["href"]).rstrip('/')

# create graph
# G.add_edges_from([(url, complete_url)])


# create list graph
branch = []
# remove https://
new_url = url.replace('https://', '')
new_url = new_url.replace('http://', '')
new_complete = complete_url.replace(
    'https://', '')
new_complete = new_complete.replace(
    'http://', '')
branch.append(new_url)
branch.append(new_complete)
list_g.append(branch)

# Create a new record
sql = "INSERT INTO `linking` (`crawl_id`,
    `url`, `outgoing_link`) VALUES (%s, %s, %s)"
# Execute the query
cursor.execute(sql, (1, url, complete_url))
# commit to save our changes
db.commit()
```

```
# Check if the URL already exists
# in the url_queue
for j in url_queue:
    if j == complete_url:
        flag = 1
        break

# Check if the URL already exists
# in the visited_url
for j in visited_url:
    if (j == complete_url):
        flag = 1
        break

# If not found in queue
if flag == 0:
    if (visited_url.count(
        complete_url)) == 0:
        url_queue.append(complete_url)

except (AttributeError,
        KeyError, requests.exceptions.InvalidSchema,
        requests.exceptions.ConnectionError):
    title = "no-title"
    complete_text = "no-text"
```

```
# crawl url selanjutnya
if len(url_queue) == 0:
    return
current = url_queue.popleft()

crawl(current)

# time
start_time = time.time()

# graph
G = nx.DiGraph()
list_g = []

# crawl begin
crawl(url_awal)

print("Jumlah url yg sudah dicrawl:", len(visited_url))
print("Jumlah url dalam queue:", len(url_queue))
```

LAMPIRAN B

Sampel Kode *Modified Similarity Based Crawling*

```
from BFS_crawling import *

def reorder_queue(queue):
    """Function untuk reorder queue
    """
    value_backlink = []
    for u in queue:
        # menentukan nilai backlink_count
        backlink_count = list_g.count(u)
        value_backlink.append(backlink_count)

    # create dictionary backlink for sorting
    backlink_dictionary = dict(zip(queue,
                                    value_backlink))
    # sorting backlink_dictionary
    sort_orders = sorted(backlink_dictionary.items(),
                         key=lambda x: x[1], reverse=True)

    # mengkosongkan queue
    queue.clear()
    # membuat queue yang sudah di sort
    for i in sort_orders:
```

```
queue.append(i[0])

return queue

# reorder url_queue dari crawl sebelumnya
reorder_queue(url_queue)

# membuat hot_queue
hot_queue = deque([])

# new start_time_MSB
start_time_MSB = time.time()

def jumlah_key_body(complete_text):
    """Function untuk menghitung jumlah key di text
    """
    keyword = hot_key
    jumlah_keyword = complete_text.count(keyword)
    return jumlah_keyword

def jumlah_key_title(title):
    """Function untuk menghitung jumlah key di title
    """
```

```
keyword = hot_key
jumlah_keyword = title.count(keyword)
return jumlah_keyword

def modified_crawl(url):
    """Function untuk modified crawling page.
    """
    try:
        # kondisi berhenti
        time_now = time.time() - start_time_MSB
        time_now_int = int(time_now)
        if time_now_int >= 900:
            return
        # if len(visited_url) >= 1800:
        #     return
        # kondisi berhenti dari algoritma
        if (len(hot_queue) == 0) and
           (len(url_queue) == 0):
            return
        # memasukan url kedalam visited_url
        visited_url.append(url)

        # crawl page
        print("page yang sedang di crawl:", url)
```

```
page = requests.get(url)
request = page.content
soup = bs4.BeautifulSoup(request,
'html.parser')

# extract title
title = soup.title.string
# print("judul:", title)

# check version html
article_html5 = soup.find('article')
if article_html5 is None:
    # extract text content from html4
    html5 = "no"
    texts = soup.find('body').findAll(
        text=True)
    visible_texts = filter(tag_visible, texts)
    text = u" ".join(
        t.strip() for t in visible_texts)
    text = text.lstrip().rstrip()
    text = text.split(',')
    clean_text = ''
    for sen in text:
        if sen:
            sen = sen.rstrip().lstrip()
            clean_text += sen+','
```

```
complete_text = clean_text

# print(complete_text)

else:

    # extract text content from html5

    html5 = "yes"

    texts = article_html5.findAll(text=True)

    visible_texts = filter(tag_visible, texts)

    text = u" ".join(

        t.strip() for t in visible_texts)

    text = text.lstrip().rstrip()

    text = text.split(',')

    clean_text = ''

    for sen in text:

        if sen:

            sen = sen.rstrip().lstrip()

            clean_text += sen+','

    complete_text = clean_text

    # print(complete_text)

# get meta description

description = soup.find("meta",

    attrs={"name": "description"})

if description is None:

    description = "-"

else:

    description = description.get("content")
```

```
# get meta keywords
keywords = soup.find("meta",
attrs={"name": "keywords"})
if keywords is None:
    keywords = "-"
else:
    keywords = keywords.get("content")

# check hot_url
hot_url = False
hot_link = "no"
if (jumlah_key_body(complete_text) >= 10)
or (jumlah_key_title(title) >= 1):
    hot_url = True
    hot_link = "yes"

# Create a new record
sql = "INSERT INTO `page_information` (
`base_url`, `html5`, `title`, `description`,
`keywords`, `content_text`, `hot_url`,
`model_crawl`) VALUES (
%s, %s, %s, %s, %s, %s, %s)"
# Execute the query
cursor.execute(sql, (url, html5, title,
description, keywords, complete_text,
```

```
hot_link, "modified similarity-based crawling"))

# commit to save our changes
db.commit()

# extract style
for style in soup.findAll('style'):

    # Create a new record
    sql = "INSERT INTO `style_resource` (
        `base_url`, `style`) VALUES (%s, %s)"

    # Execute the query
    cursor.execute(sql, (url, style))

    # commit to save our changes
    db.commit()

# extract script
for script in soup.findAll('script'):

    # Create a new record
    sql = "INSERT INTO `script_resource` (
        `base_url`, `script`) VALUES (%s, %s)"

    # Execute the query
    cursor.execute(sql, (url, script))

    # commit to save our changes
    db.commit()

# extract lists
for lists in soup.findAll('li'):
```

```
# Create a new record
sql = "INSERT INTO `list` (
    `base_url`, `list`) VALUES (%s, %s)"

# Execute the query
cursor.execute(sql, (url, lists))

# commit to save our changes
db.commit()

# extract forms
for form in soup.findAll('form'):

    # Create a new record
    sql = "INSERT INTO `forms` (
        `base_url`, `form`) VALUES (%s, %s)"

    # Execute the query
    cursor.execute(sql, (url, form))

    # commit to save our changes
    db.commit()

# extract tables
for table in soup.findAll('table'):

    # Create a new record
    sql = "INSERT INTO `tables` (
        `base_url`, `tables`) VALUES (%s, %s)"

    # Execute the query
    cursor.execute(sql, (url, table))

    # commit to save our changes
```

```
    db.commit()

    # extract images
    for image in soup.findAll('img'):
        # Create a new record
        sql = "INSERT INTO `images` (
            `base_url`, `image`) VALUES (%s, %s)"
        # Execute the query
        cursor.execute(sql, (url, image))
        # commit to save our changes
        db.commit()

    # extract outgoing link
    links = soup.findAll("a", href=True)

    # memasukan outgoing link kedalam queue
    for i in links:
        flag = 0

        # Complete relative URLs
        # and strip trailing slash
        complete_url = urljoin(
            url, i["href"]).rstrip('/')

        # create list graph
        branch = [ ]
```

```
# remove https://
new_url = url.replace('https://', '')
new_url = new_url.replace('http://', '')
new_complete = complete_url.replace(
    'https://', '')
new_complete = new_complete.replace(
    'http://', '')
branch.append(new_url)
branch.append(new_complete)
list_g.append(branch)

# Create a new record
sql = "INSERT INTO `linking` (`crawl_id`,
    `url`, `outgoing_link`) VALUES (
    %s, %s, %s)"
# Execute the query
cursor.execute(sql, (1, url, complete_url))
# commit to save our changes
db.commit()

# Check if the URL already exists in
# the url_queue
for j in url_queue:
    if (j == complete_url):
        flag = 1
        break
```

```
# Check if the URL already exists in
# the hot_queue
for j in hot_queue:
    if (j == complete_url):
        flag = 1
        break

# Check if the URL already exists in
# the visited_url
for j in visited_url:
    if (j == complete_url):
        flag = 1
        break

# If not found in queue
if flag == 0:
    if (visited_url.count(
        complete_url)) == 0:
        if (hot_url == True) or (
            (complete_url.count(
                "klub-bola-barcelona") >= 1)):
            hot_queue.append(complete_url)
    else:
        url_queue.append(complete_url)
```

```
    reorder_queue(hot_queue)
    reorder_queue(url_queue)

except (AttributeError, KeyError,
        requests.exceptions.InvalidSchema,
        requests.exceptions.ConnectionError):
    title = "no-title"
    complete_text = "no-text"

# crawl url selanjutnya
if len(hot_queue) > 0:
    current = hot_queue.popleft()
else:
    current = url_queue.popleft()

modified_crawl(current)

url = url_queue.popleft()
modified_crawl(url)

# Create a new record
sql = "INSERT INTO `crawling` (`url_awal`, `keyword`,
    `total_page`, `duration_crawl`) VALUES (
    %s, %s, %s, %s)"

# Execute the query
time_now = time.time() - start_time
time_now_int = int(time_now)
```

```
cursor.execute(sql, (
    url_awal, hot_key, len(visited_url), time_now_int))

# commit to save our changes
db.commit()

print("Jumlah url yg sudah dilihat:", len(visited_url))
print("Jumlah url dalam queue:", len(url_queue))
print("Jumlah url dalam hot queue:", len(hot_queue))
print("Waktu yang dibutuhkan: %s detik" % (
    time.time() - start_time))

# Close the connection
db.close()
```

DAFTAR RIWAYAT HIDUP



MUHAMMAD FATHAN QORIIBA. Lahir di Cileungsi, 28 Februari 1998. Anak kedua dari empat bersaudara yaitu dari pasangan Bapak Sutara dan Ibu Sri Sumartini. Saat ini beralamatkan di Jl. Musholla Al-Arifiah RT/RW 06/07 No.76 Kp.Kemang, Jatiwaringin, Pondok Gede, Kota Bekasi, 17411.

No. Ponsel : 082112926885

Email : mfathanqoriiba@gmail.com

Riwayat Pendidikan : Penulis mengenyam pendidikan dan lulus dari sekolah dasar di SDIT Raudhatul Muttaqin pada tahun 2010, sekolah menengah pertama di SMPIT IQRO pada tahun 2013 dan sekolah menengah atas di SMAN 5 Kota Bekasi pada tahun 2016. Kemudian penulis melanjutkan pendidikan di Universitas Negeri Jakarta (UNJ) Fakultas Matematika dan Ilmu Pengetahuan Alam (FMIPA) dengan program studi Ilmu Komputer dengan jalur masuk melalui SNMPTN.

Riwayat Organisasi : Selama di bangku perkuliahan, penulis aktif di kelas dan diluar kelas dengan mengikuti organisasi DEFAULT periode 2018-2019, kemudian terpilih menjadi Wakil Ketua DEFAULT yang mengurus bagian birokrasi.