Analasis Komparatif dari Algortima Traversing untuk Web Crawlers: BFS, DFS, and IDS

Raden Rafly Hanggaraksa Budiarto - 13522014
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): ajarafly422@gmail.com

***Abstract*—Dalam era digital saat ini, web crawlers memainkan peran penting dalam pengindeksan dan pengumpulan data dari internet. Program otomatis ini menavigasi web secara sistematis, mengumpulkan informasi untuk berbagai tujuan seperti pengindeksan mesin pencari, analisis data, dan pemantauan konten. Studi ini mengevaluasi dan membandingkan performa algoritma Breadth-First Search (BFS), Depth-First Search (DFS), dan Iterative Deepening Search (IDS) dalam melakukan penelusuran web berdasarkan efisiensi waktu, penggunaan sumber daya, dan kedalaman pencarian. Hasil implementasi dan pengujian pada beberapa entry point menunjukkan bahwa DFS unggul dalam cakupan halaman yang luas namun menggunakan memori tinggi, BFS lebih efisien dalam penggunaan memori dan IDS menawarkan keseimbangan antara efisiensi dan penggunaan sumber daya dengan jaminan solusi optimal. Penelitian ini memberikan wawasan dalam memilih algoritma yang sesuai untuk kebutuhan spesifik dalam penjelajahan web dan berkontribusi pada teknik web crawling yang lebih efektif dan efisien.**

Keywords—Web Crawlers; Breadth-First Search; Depth-First Search; Iterative Deepening Search; Traversal Algorithms; Efisiensi; Penggunaan Sumber Daya;

I. PENDAHULUAN

Dalam era digital saat ini, web crawlers atau perayap web memainkan peran penting dalam pengindeksan dan pengumpulan data dari internet. Web crawlers adalah program atau skrip otomatis yang menavigasi web secara sistematis, mengumpulkan informasi dari halaman web untuk berbagai tujuan seperti pengindeksan mesin pencari, analisis data, dan pemantauan konten. Di antara berbagai algoritma yang digunakan untuk traversal web, Breadth-First Search (BFS), Depth-First Search (DFS), dan Iterative Deepening Search (IDS) adalah beberapa yang paling dikenal dan banyak diterapkan.

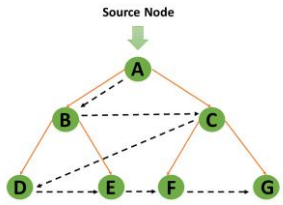
Meskipun BFS, DFS, dan IDS sering digunakan dalam implementasi web crawlers, masing-masing memiliki karakteristik dan performa yang berbeda dalam hal efisiensi waktu, penggunaan sumber daya, dan kedalaman pencarian.

Dalam penelitian ini, penulis mengimplementasikan ketiga algoritma tersebut dan membandingkan performanya berdasarkan beberapa parameter pengujian. Dengan memahami kekuatan dan kelemahan dari masing-masing algoritma, diharapkan bahwa pembaca dapat memilih algoritma yang paling sesuai untuk kebutuhan spesifik dalam penjelajahan web.

II. TEORI DASAR

A. Algoritma Breadth First Search

Breadth First Search merupakan algoritma traversal graf. Algoritma ini mengunjungi semua simpul pada kedalaman tertentu sebelum mengunjungi simpul pada kedalaman selanjutnya. Pencarian dimulai pada suatu simpul spesifik kemudian mengunjungi seluruh tetangganya sebelum melanjutkan ke kedalaman selanjutnya. Untuk mencegah terjadinya kunjungan simpul yang sama, algoritma dapat membagi suatu simpul menjadi *visited* atau *not visited*. Algoritma ini umumnya digunakan pada algoritma *pathfinding*, *connected components*, dan *shortest path*.



Gambar 1 Ilustrasi Breadth First Search
(https://www.simplilearn.com.cach3.com/tutorials/data-structure-tutorial/bfs-algorithm.html)

Isi dari algoritma ini adalah sebagai berikut: Buat sebuah queue **q** yang berisi semua simpul yang akan diproses dan sebuah larik boolean **used** yang menandakan bahwa untuk setiap simpul, apakah sudah dikunjungi atau belum.

Pada awalnya, masukan sumber **s** ke dalam queue dan pasang nilai **`used[s]` = `true`**, dan untuk semua simpul **v** jadikan **`used[v] = false`**. Lalu, iterasi hingga queue menjadi kosong. Pada setiap iterasi, pop sebuah simpul dari depan queue. Ekspansikan semua sisi yang keluar dari simpul tersebut dan apabila simpul tersebut belum dikunjungi, tandai pada larik **used**.

Sebagai hasil, ketika queue tersebut sudah kosong, seluruh simpul pada graf tersebut sudah dikunjungi dengan langkah paling optimal. Program juga dapat menghitung panjang dari

jarak terpendek serta menyimpan semua simpul yang menjadi langkah pada simpul tersebut.

Kompleksitas waktu BFS tergantung pada bagaimana graf diwakili:

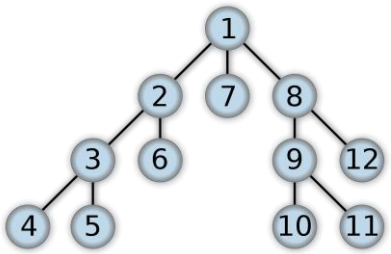
1. Graf menggunakan adjacency list:
    a. Untuk setiap simpul, kita memproses semua tetangganya.
    b. Jika V adalah jumlah simpul (vertices) dan E adalah jumlah sisi (edges), setiap simpul akan diproses sekali dan setiap sisi akan diperiksa sekali.
    c. Kompleksitas waktu adalah O (V + E)
2. Graf menggunakan adjacency matrix:
    a. Mengunjungi semua simpul membutuhkan waktu O(V^2), karena untuk setiap simpul kita memeriksa semua simpul lainnya untuk melihat apakah ada sisi yang menghubungkannya.
    b. Kompleksitas Waktu adalah O(V^2)

Kompleksitas ruang BFS terutama dipengaruhi oleh penggunaan struktur data untuk menyimpan simpul yang harus diproses berikutnya (yaitu, antrean) dan struktur data tambahan untuk melacak simpul yang sudah dikunjungi.

1. Graf menggunakan adjacency list:
    a. Algoritma membutuhkan ruang untuk menyimpan daftar adjacency, yang memakan ruang O(V+E).
    b. Antrean BFS akan menyimpan simpul maksimal setingkat terluas dari pohon BFS, yang dalam kasus terburuk bisa menjadi O(V).
    c. Selain itu, kita membutuhkan ruang O(V) untuk menyimpan status kunjungan setiap simpul.
    d. Kompleksitas ruang total adalah O(V+E)
2. Graf menggunakan adjacency matrix:
    a. Matriks adjacency memakan ruang O(V)
    b. Antrean dan larik kunjungan membutuhkan O(V)
    c. Kompleksitas ruang total adalah O(V^2)

B. Algoritma Depth First Search

Depth First Search (DFS) adalah algoritma lain untuk menjelajahi graf. Algoritma ini dimulai dari satu titik dan menjelajah sejauh mungkin di sepanjang setiap cabang sebelum kembali dan mencoba jalur lain. Analoginya seperti menjelajahi labirin, selalu mengikuti satu jalur hingga menemui jalan buntu, lalu mundur untuk mencoba rute lain. Dengan DFS, kita dapat secara sistematis mengungkap semua koneksi dan jalur dalam sebuah graf.



Gambar 2 Ilustrasi Depth First Search
(https://velog.io/@xpeed96/series/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)

DFS dimulai dari simpul terpilih dan menjelajah sejauh mungkin di sepanjang setiap cabang sebelum kembali ke simpul sebelumnya. DFS dapat diimplementasikan menggunakan rekursi atau struktur data stack. Berikut adalah garis besar dasar algoritma DFS:

1. Pilih simpul awal dan tandai sebagai telah dikunjungi.
2. Kunjungi simpul awal dan jelajahi simpul-simpul yang berdekatan.
3. Untuk setiap simpul berdekatan yang belum dikunjungi, terapkan DFS secara rekursif.
4. Lanjutkan menjelajahi simpul-simpul hingga semua simpul yang dapat dijangkau telah dikunjungi atau tidak ada simpul yang belum dikunjungi.
5. Jika semua simpul yang dapat dijangkau telah dikunjungi, mundur ke simpul sebelumnya dan jelajahi cabang lainnya.

DFS memiliki banyak aplikasi di berbagai bidang karena keserbagunaan dan kesederhanaannya. Berikut adalah beberapa aplikasi umum dari DFS:

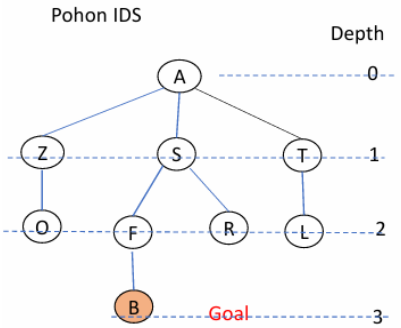
1. DFS terutama digunakan untuk menjelajahi graf, mengunjungi setiap simpul dan sisi secara sistematis. Ini membantu mengidentifikasi komponen yang terhubung, siklus, dan jalur antar simpul.
2. DFS digunakan untuk menavigasi labirin, menjelajahi setiap jalur secara sistematis hingga mencapai pintu keluar. Ini memastikan eksplorasi menyeluruh dari semua rute yang mungkin.
3. DFS dapat menganalisis jaringan, seperti jaringan sosial atau jaringan komputer, dengan menjelajahi koneksi antar simpul dan mengidentifikasi pola atau kelompok dalam jaringan.

Kompleksitas waktu dari DFS adalah O(V + E), dengan V merupakan jumlah simpul (vertices) dalam graf, dan E adalah jumlah sisi (edges) dalam graf. Hal ini karena dalam proses penelusuran, setiap simpul dikunjungi satu kali dan setiap sisi juga dikunjungi satu kali. Dengan kata lain, algoritma DFS akan menjelajahi setiap simpul dan setiap sisi tepat satu kali selama proses penelusuran, sehingga total waktu yang dibutuhkan untuk menyelesaikan penelusuran adalah sebanding dengan jumlah simpul ditambah dengan jumlah sisi dalam graf. Ini berlaku baik untuk representasi graf menggunakan larik ketetanggaan (adjacency list) maupun matriks ketetanggaan

(adjacency matrix), meskipun dalam praktiknya, DFS lebih efisien jika digunakan dengan larik ketetanggaan.

Kompleksitas ruang dari DFS bergantung pada struktur data yang digunakan dan kedalaman maksimum dari rekursi atau tumpukan eksplisit yang digunakan untuk menyimpan status penelusuran. Pada umumnya, kompleksitas ruang DFS adalah O(V), dengan V adalah jumlah simpul dalam graf. Ini karena dalam implementasi rekursif, setiap simpul yang dikunjungi akan ditambahkan ke dalam stack rekursi yang memerlukan ruang memori. Kedalaman maksimum dari stack ini adalah setara dengan jumlah simpul dalam kasus terburuk, yaitu ketika graf berbentuk garis lurus (graf linear) atau ketika seluruh simpul harus dikunjungi sebelum mencapai simpul tujuan.

C. Algoritma Iterative Deepening Search



Gambar 3 Ilustrasi Iterative Deepening Search
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf)

Algoritma Iterative Deepening Search (IDS) adalah kombinasi dari kelebihan metode Depth-First Search (DFS) dan Breadth-First Search (BFS). IDS menggunakan pendekatan eksplorasi mendalam seperti DFS, tetapi dengan batas kedalaman yang terus meningkat hingga solusi ditemukan. Algoritma ini sangat efektif dalam menemukan solusi optimal di berbagai skenario pencarian, termasuk aplikasi web crawling.

IDS bekerja dengan menjalankan DFS berulang kali dengan batas kedalaman (Depth Limited Search) yang meningkat pada setiap iterasi. Pada iterasi pertama, pencarian dilakukan hingga kedalaman 1. Jika solusi belum ditemukan, algoritma meningkatkan batas kedalaman menjadi 2 dan menjalankan DFS lagi. Proses ini terus berulang dengan batas kedalaman yang bertambah hingga solusi ditemukan atau hingga batas tercapai.

Kelebihan IDS adalah kemampuannya untuk menjamin penemuan solusi jika ada dan memastikan solusi tersebut adalah yang terpendek atau optimal. IDS menghindari kelemahan utama dari DFS, yaitu kemungkinan terjebak pada jalur yang sangat dalam tanpa menemukan solusi, dan juga mengatasi kelemahan BFS yang menggunakan banyak memori ketika menjelajahi tingkat-tingkat yang lebih dalam.

Secara praktis, IDS dapat digambarkan sebagai kombinasi dari dua sifat penting:

1. Kedalaman Bertahap

   Algoritma ini menggunakan DFS dengan batas kedalaman tertentu. Jika solusi tidak ditemukan dalam batas kedalaman ini, kedalaman ditingkatkan dan DFS dijalankan lagi. Ini memastikan bahwa solusi paling dangkal ditemukan terlebih dahulu.

2. Efisiensi Memori

   Berbeda dengan BFS yang membutuhkan memori besar untuk menyimpan semua simpul pada satu level, IDS hanya menyimpan simpul yang sedang berada dalam jalur pencarian saat ini, sehingga penggunaan memori menjadi lebih efisien.

D. Web Crawler

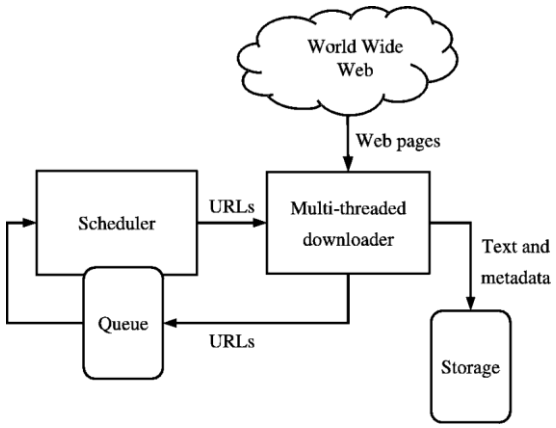
Sebuah web crawler, laba-laba, atau bot mesin pencari mengunduh dan mengindeks konten dari seluruh Internet. Tujuan dari bot semacam itu adalah untuk mempelajari apa informasi yang ada untuk setiap halaman web di web ini, sehingga informasi tersebut dapat diambil ketika dibutuhkan. Mereka disebut "web crawler" karena crawling adalah istilah teknis untuk secara otomatis mengakses sebuah situs web dan mendapatkan data melalui program perangkat lunak.

Bot-bot ini hampir selalu dioperasikan oleh mesin pencari. Dengan menerapkan algoritma pencarian pada data yang dikumpulkan oleh web crawler, mesin pencari dapat memberikan tautan-tautan yang relevan sebagai respons terhadap kueri pencarian pengguna, menghasilkan daftar halaman web yang muncul setelah pengguna mengetik pencarian ke Google atau Bing (atau mesin pencari lainnya).

Sebuah bot web crawler ibarat seseorang yang mengurai semua buku dalam sebuah perpustakaan yang berantakan dan menyusun katalog kartu sehingga siapa pun yang mengunjungi perpustakaan dapat dengan cepat dan mudah menemukan informasi yang mereka butuh kan. Untuk membantu mengategorikan dan menyortir buku-buku perpustakaan berdasarkan topik, pengatur akan membaca judul, ringkasan, dan beberapa teks internal dari setiap buku untuk mencari tahu tentang apa itu.

Namun, berbeda dengan perpustakaan, Internet tidak terdiri dari tumpukan buku fisik, dan hal itu membuat sulit untuk mengetahui apakah semua informasi yang diperlukan telah diindeks dengan benar atau jika sejumlah besar informasi diabaikan. Untuk mencoba menemukan semua informasi relevan yang ditawarkan Internet, bot web crawler akan memulai dengan seperangkat halaman web yang diketahui (*entry point*) dan kemudian mengikuti tautan dari halaman-halaman itu ke halaman-halaman lain, mengikuti tautan dari halaman-halaman lain itu ke halaman-halaman tambahan, dan seterusnya.

Tidak diketahui seberapa besar dari Internet yang tersedia untuk umum sebenarnya di-crawl oleh bot mesin pencari. Beberapa sumber memperkirakan bahwa hanya 40-70% dari Internet yang diindeks untuk pencarian - dan dari itu miliaran halaman web.



Gambar 4 Arsitektur Web Crawler
(https://www.researchgate.net/figure/Typical-high-level-architecture-of-a-Web-crawler-involving-a-scheduler-and-a_fig11_220466525)

Arsitektur dari web crawler mencakup beberapa komponen utama yang bekerja secara sinergis untuk mengumpulkan, mengolah, dan menyimpan data dari web. Berikut adalah deskripsi dari komponen-komponen tersebut:

1) URL Frontier

URL Frontier adalah komponen yang menyimpan daftar URL yang akan dijelajahi oleh web crawler. Daftar ini biasanya diimplementasikan sebagai struktur data seperti queue atau priority queue, tergantung pada algoritma penjelajahan yang digunakan (misalnya, BFS, DFS, dan IDS). URL Frontier harus mengelola URL dengan efisien, memastikan URL yang sama tidak dijelajahi lebih dari sekali, dan memprioritaskan URL yang relevan untuk dijelajahi lebih dulu.

2) Downloader

Downloader bertugas untuk mengambil konten halaman web dari URL yang diberikan. Komponen ini mengirim permintaan HTTP ke server web dan menerima respons berupa HTML, CSS, JavaScript, dan file lainnya. Downloader harus menangani berbagai jenis respons HTTP, seperti pengalihan (*redirects*), kesalahan (*errors*), dan *throttling* yang diberlakukan oleh server untuk membatasi laju permintaan.

3) Parser

Parser bertugas mengolah konten yang diperoleh oleh Downloader. Biasanya, parser menggunakan pustaka seperti BeautifulSoup atau lxml untuk mengekstrak informasi dari HTML dan XML. Parser mencari elemen-elemen seperti tautan (hyperlinks), teks, metadata, dan elemen lainnya yang relevan. Informasi yang diolah akan kemudian dikirim ke komponen lain untuk diproses lebih lanjut.

4) Data Storage

Data Storage adalah tempat penyimpanan sementara atau permanen untuk data yang diekstrak oleh parser. Penyimpanan ini bisa berupa database relasional (misalnya, MySQL, PostgreSQL), database NoSQL (misalnya, MongoDB, Cassandra), atau sistem file biasa. Data Storage harus dirancang untuk mendukung kueri dan analisis data yang cepat dan efisien.

5) Indexer

Indexer bertanggung jawab untuk mengindeks data yang dikumpulkan sehingga dapat dicari dan diakses dengan cepat. Indexer membuat struktur data yang memungkinkan pencarian cepat berdasarkan kata kunci atau atribut lainnya. Algoritma pengindeksan bisa sangat kompleks dan biasanya melibatkan teknik seperti *inverted indexing*, *hashing*, dan *compressing* untuk mengoptimalkan ruang dan waktu pencarian.

6) Scheduler

Scheduler mengatur urutan dan prioritas penjelajahan halaman web berdasarkan berbagai faktor seperti kebaruan konten, relevansi, dan kebijakan robot (robots.txt). Scheduler juga harus memastikan crawler tidak membebani server target dengan terlalu banyak permintaan dalam waktu singkat, dengan menerapkan throttling dan delay antara permintaan.

7) Politeness dan Deduplication

Web crawler harus mematuhi aturan yang ditentukan oleh file robots.txt pada setiap server web, yang menetapkan halaman mana yang boleh dan tidak boleh dijelajahi. Selain itu, crawler harus menghindari mengunjungi URL yang sama lebih dari sekali, yang memerlukan mekanisme untuk mendeteksi dan menghilangkan duplikasi URL.

8) Controller

Controller adalah komponen utama yang mengoordinasikan semua aktivitas web crawler. Controller mengarahkan operasi dari URL Frontier, Downloader, Parser, dan komponen lainnya. Controller memastikan bahwa semua komponen bekerja bersama secara sinkron untuk mencapai tujuan penjelajahan dan pengindeksan.

III. IMPLEMENTASI

A. Program Pengujian

Kode yang digunakan untuk pengujian adalah program Python yang melakukan penjelajahan web (web crawling) menggunakan tiga metode berbeda: BFS (Breadth-First Search), DFS (Depth-First Search), dan IDS (Iterative Deepening Search). Program ini memanfaatkan beberapa pustaka penting seperti BeautifulSoup untuk parsing HTML, requests untuk melakukan permintaan HTTP, dan psutil untuk memantau penggunaan sumber daya sistem.

Program ini dimulai dengan mendefinisikan kelas dasar **Scraper** yang berfungsi sebagai kerangka untuk penjelajahan web. Kelas ini memiliki beberapa atribut penting seperti **_entrypoint** untuk menyimpan URL awal, **_visitedpage** untuk menghitung jumlah halaman yang telah dikunjungi, **_timelimit** untuk batas waktu penjelajahan, serta **_cpu_usage** dan **_memory_usage** untuk mencatat penggunaan CPU dan memori selama penjelajahan. Metode utama dalam kelas ini meliputi inisialisasi objek, pengambilan halaman web dan

parsing dengan BeautifulSoup, ekstraksi tautan dari halaman, serta pencatatan dan pencetakan statistik penggunaan sumber daya.

B. Parameter Pengujian

Dalam membandingkan performa dari metode BFS, DFS, dan IDS, digunakan beberapa parameter pengujian. Parameter pengujian ini mencakup beberapa metrik evaluasi yang penting untuk menentukan efektivitas dan efisiensi dari algoritma penjelajahan web yang digunakan.

1) Coverage

Coverage merupakan metrik yang menunjukkan total halaman yang berhasil ditemukan dan diindeks oleh web crawler. Metrik ini penting untuk menilai seberapa luas cakupan penjelajahan web yang dilakukan oleh crawler, serta kemampuan crawler dalam menemukan sebanyak mungkin halaman yang tersedia di internet.

2) Efficiency

Efficiency mengukur jumlah halaman yang berhasil dijelajahi per satuan waktu. Metrik ini memberikan gambaran mengenai kecepatan dan kinerja dari web crawler dalam menjelajahi dan mengindeks halaman web. Semakin tinggi efisiensinya, semakin baik kinerja crawler dalam memanfaatkan waktu yang tersedia.

3) Depth

Depth mengacu pada kedalaman rata-rata halaman yang berhasil dijangkau dalam struktur situs. Metrik ini menunjukkan sejauh mana crawler mampu menjelajahi halaman-halaman yang lebih dalam dari suatu situs. Semakin dalam halaman yang dijangkau, semakin menyeluruh penjelajahan yang dilakukan oleh crawler.

4) Resource Usage

Resource Usage mencakup penggunaan CPU, dan memori pada proses penjelajahan. Metrik ini penting untuk menilai efisiensi penggunaan sumber daya oleh web crawler. Dengan memantau penggunaan sumber daya, kita dapat memastikan bahwa crawler beroperasi dengan cara yang tidak membebani sistem secara berlebihan dan tetap efisien.

C. Hasil Pengujian

Hasil Pengujian dilakukan kepada Entry Point Pranala berikut:

TABEL I. Entry Point

ID	Entry Point
1.	https://en.wikipedia.org/wiki/Badminton_World_Federation
2.	https://en.wikipedia.org/wiki/Microsoft_Windows
3.	https://www.apple.com/id/
4.	https://en.wikipedia.org/wiki/Steve_Jobs
5.	https://en.wikipedia.org/wiki/Bill_Gates
6.	https://en.wikipedia.org/wiki/Indonesia
7.	https://en.wikipedia.org/wiki/Mars

Dalam melakukan pencarian, batas dari keberlanjutan pencarian adalah durasi sebanyak satu menit. Hal ini menyebabkan algoritma akan berhenti pencariannya apabila telah melewati batas waktu tertentu. Oleh karena itu, dapat dipastikan bahwa durasi dari seluruh pengujian bernilai 60 detik.

TABEL II. Hasil Penelusuran Breadth First Search

Entry Point	Total Page	Memori (MB)	CPU (%)	Efisiensi (Page / Durasi)	Kedalaman
1.	55	7004.70	1.212	0.916	1
2.	58	7135.76	1.28	0.967	1
3.	87	6783.56	4.29	1.45	3
4.	65	6645.20	2.12	1.08	1
5.	62	6601.42	1.80	1.03	1
6.	53	7068.80	1.41	0.88	1
7.	55	7057.87	1.34	0.92	1

TABEL III. Hasil Penelusuran Depth First Search

Entry Point	Total Page	Memori (MB)	CPU (%)	Efisiensi (Page / Durasi)	Kedalaman
1.	4257	6772.54	1.21	70.95	91
2.	5029	6848.90	1.303	83.82	104
3.	2060	6885.78	1.41	34.33	63
4.	4945	7106.20	5.11	82.41	93
5.	5342	6890.77	1.56	89.03	101
6.	5552	6883.11	1.58	92.53	108
7.	5620	6885.95	1.81	93.67	100

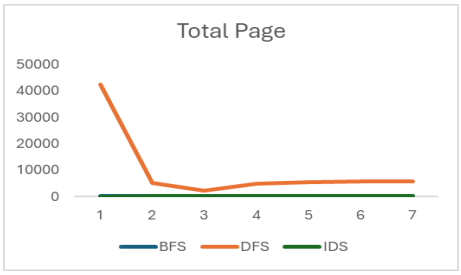
TABEL IV. Hasil Penelusuran Iterative Deepening Search

Entry Point	Total Page	Memori (MB)	CPU (%)	Efisiensi (Page / Durasi)	Kedalaman
1.	63	7062.26	2.25	1.05	2
2.	114	7056.94	2.5	1.90	2
3.	135	7046.31	2.17	2.25	5
4.	111	7047.54	2.03	1.85	2
5.	112	7021.65	2.27	1.87	2
6.	103	6981.55	2.37	1.72	2
7.	108	6960.89	2.48	1.80	2

D. Analisis Hasil Pengujian

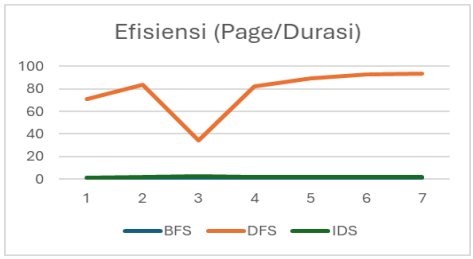
Berdasarkan jumlah halaman yang dikunjungi, kita dapat mengamati coverage dari ketiga metode penjelajahan. Traversal menggunakan BFS mengunjungi halaman paling sedikit. Penelusuran secara DFS menelusuri halaman paling banyak, disusul oleh IDS. BFS memiliki jumlah kunjungan paling lama karena menggunakan struktur data queue untuk menentukan simpul selanjutnya yang akan diekspansi. Jika ada

hasil ekspansi, elemen queue akan mengonkat hasil ekspansi tersebut ke dalam queue. Namun, pada DFS dan IDS, algoritma tidak menggunakan struktur data apa pun untuk menyimpan simpul selanjutnya yang akan diekspansi, karena kedua algoritma tersebut memanfaatkan rekursi.



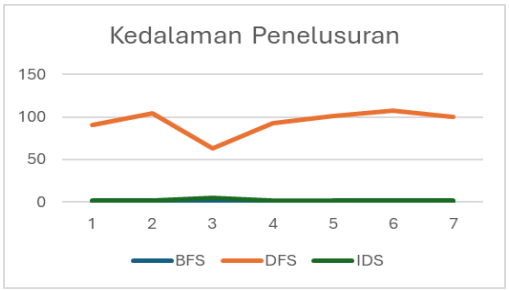
Gambar 5 Grafik Data Total Page

Berdasarkan efisiensi, traversal menggunakan DFS memiliki nilai efisiensi paling tinggi dibandingkan IDS dan BFS. BFS memiliki nilai efisiensi terendah, sementara IDS berada di antara keduanya. Berdasarkan kesimpulan coverage, hal ini disebabkan oleh iterasi menggunakan DFS yang mencakup lebih banyak halaman dibandingkan keduanya.



Gambar 6 Grafik Data Efisiensi

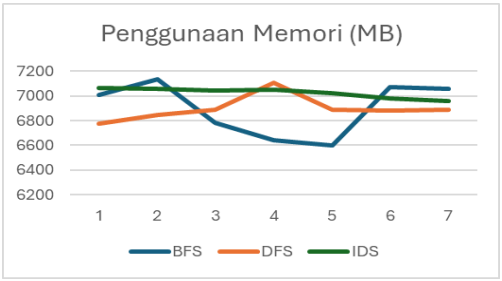
Berdasarkan kedalaman iterasi yang dilakukan oleh ketiga metode, DFS menelusuri kedalaman paling dalam dibandingkan IDS dan BFS. Hal ini disebabkan oleh sifat dasar algoritma DFS yang menelusuri suatu jalur sampai habis sebelum melakukan backtracking terhadap kedalaman sebelumnya. IDS memiliki konsep yang serupa dengan DFS, tetapi kedalamannya akan meningkat apabila seluruh simpul telah dilalui. BFS memiliki penelusuran kedalaman yang paling rendah karena sifat alami dari algoritma tersebut adalah menelusuri semua simpul tetangga terlebih dahulu sebelum melanjutkan traversal pada kedalaman lain.



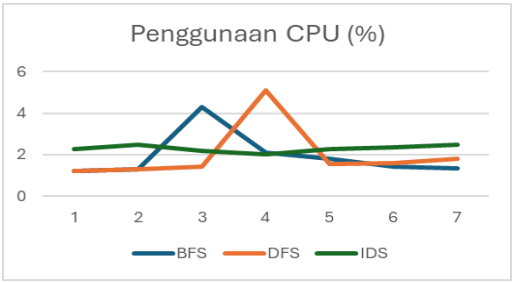
Gambar 7 Grafik Data Kedalaman Penelusuran

Berdasarkan penggunaan resource, perbedaan antara ketiga metode tidak terlalu signifikan karena banyak sampel tidak terlalu besar pada skala MB mengakibatkan penggunaan proses memiliki alokasi yang serupa untuk setiap proses. Variansi yang terjadi pada data penggunaan memori dan data penggunaan CPU disebabkan oleh urutan eksekusi program pada komputer serta program yang berjalan pada latar belakang komputer.

Namun, kita dapat memandang penggunaan resource dari banyaknya halaman yang dikunjungi. Sebuah web crawler akan menyimpan seluruh data dari suatu halaman untuk melakukan pengindeksan. Penelusuran menggunakan DFS menghasilkan penggunaan memori yang lebih besar karena jumlah halaman yang dikunjungi lebih banyak dibandingkan metode lainnya. Penggunaan memori selanjutnya disusul oleh IDS dan BFS.



Gambar 8 Grafik Data Penggunaan Memori



Gambar 9 Grafik Data Penggunaan CPU

IV. KESIMPULAN

Dalam era digital saat ini, web crawlers atau perayap web memainkan peran penting dalam pengindeksan dan pengumpulan data dari internet. Web crawlers adalah program otomatis yang menavigasi web secara sistematis untuk mengumpulkan informasi dari halaman web. Di antara berbagai algoritma traversal web, Breadth-First Search (BFS), Depth-First Search (DFS), dan Iterative Deepening Search (IDS) adalah yang paling dikenal dan diterapkan. Meskipun ketiganya sering digunakan, masing-masing memiliki karakteristik dan performa yang berbeda dalam hal efisiensi waktu, penggunaan sumber daya, dan kedalaman pencarian

Berdasarkan hasil pengujian yang telah dilakukan terhadap tiga metode penjelajahan yaitu Breadth-First Search (BFS), Depth-First Search (DFS), dan Iterative Deepening Search (IDS), beberapa kesimpulan dapat diambil.

Traversal menggunakan DFS mengunjungi halaman terbanyak, disusul oleh IDS dan BFS. Hal ini menunjukkan bahwa DFS memiliki cakupan penjelajahan yang paling luas dibandingkan kedua metode lainnya.

DFS menunjukkan nilai efisiensi tertinggi dalam hal jumlah halaman yang dijelajahi per satuan waktu, diikuti oleh IDS dan BFS. BFS memiliki efisiensi terendah karena waktu yang diperlukan lebih lama akibat penggunaan struktur data queue.

DFS menelusuri kedalaman struktur situs yang paling dalam dibandingkan IDS dan BFS. IDS memiliki kedalaman penelusuran yang serupa dengan DFS namun meningkat secara bertahap, sedangkan BFS memiliki penelusuran kedalaman paling rendah karena sifatnya yang menelusuri semua simpul tetangga terlebih dahulu.

Penelusuran menggunakan DFS menghasilkan penggunaan memori yang lebih besar karena jumlah halaman yang dikunjungi lebih banyak. IDS mengikuti di belakangnya, dan BFS memiliki penggunaan memori yang paling sedikit. Perbedaan penggunaan CPU di antara ketiga metode tidak terlalu signifikan.

Tidak ada jawaban tunggal yang tepat untuk menentukan algoritma terbaik bagi Web Crawler karena pilihan algoritma harus disesuaikan dengan kondisi dan kebutuhan spesifik. Jika penghematan memori adalah prioritas utama, IDS adalah pilihan yang baik. Untuk melakukan parsing secara menyeluruh pada suatu halaman, BFS lebih sesuai. Namun, jika tujuan utamanya adalah menjelajahi banyak halaman, maka DFS adalah opsi terbaik.

Pada akhirnya, pemilihan algoritma untuk web crawling harus mempertimbangkan berbagai faktor seperti efisiensi memori, kedalaman pencarian, dan cakupan halaman yang ingin dijelajahi. Dengan memahami kekuatan dan kelemahan dari masing-masing algoritma, pengembang dapat memilih metode yang paling sesuai untuk mencapai tujuan spesifik mereka dalam mengindeks dan mengumpulkan data dari web

V. Penutup

Syukur Alhamdulillah penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan makalah ini tepat pada waktunya. Tidak lupa, penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya kepada seluruh dosen pengajar mata kuliah Strategi Algoritma, terutama kepada Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc., yang telah dengan sabar dan penuh dedikasi membimbing kami di kelas K02.

Penulis juga ingin mengucapkan terima kasih yang mendalam kepada kedua orang tua penulis yang selalu memberikan dukungan, baik moral maupun material, selama penulis menempuh pendidikan. Tanpa dukungan, doa, dan kasih sayang mereka, penulis tidak akan berada di posisi saat ini.

Penulis sangat menghargai setiap bentuk bantuan dan dukungan yang telah diberikan, baik secara langsung maupun tidak langsung. Semoga makalah ini dapat memberikan manfaat dan kontribusi yang positif bagi perkembangan ilmu pengetahuan, khususnya dalam bidang Strategi Algoritma.

Akhir kata, penulis menyadari bahwa makalah ini masih jauh dari sempurna. Oleh karena itu, penulis sangat terbuka terhadap kritik dan saran yang membangun guna perbaikan di masa yang akan datang.

VI. Referensi

[1] R. Munir, "IF2211 Strategi Algoritma - Semester II Tahun 2023/2024," [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/stima23-24.htm. [Diakses 11 June 2024].

[2] J. M. Hsieh, S. D. Gribble dan H. M. Levy, "The Architecture and Implementation of an Extensible Web Crawler," *NSDI*, vol. 10, 2010.

[3] N. K. K. R. Rao dan P. S. Varma, "Architecture of a WebCrawler," *National Conference on Research Trends in Technology in Information Technology*, 2010.

[4] R. K. Rana dan N. Tyagi, "A Novel Architecture of Ontology-based Semantic Web Crawler," *International Journal of Computer Applications*, vol. 44, no. 18, 2012.

[5] GeekforGeeks, "Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS)," GeekforGeeks, 23 February 2023. [Online]. Available: https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/. [Diakses 10 June 2024].

[6] Google, "Overview of crawling and indexing topics," Google, 11 April 2024. [Online]. Available: https://developers.google.com/search/docs/crawling-indexing. [Diakses 11 June 2024].

[7] Microsoft, "Bing Webmaster Tools," Microsoft, [Online]. Available: https://www.bing.com/webmasters/help/help-center-661b2d18. [Diakses 12 June 2024].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Raden Rafly Hanggaraksa Budiarto
13522014