

**IMPLEMENTASI MODUL INDEXING PADA SEARCH ENGINE
TELUSURI DENGAN INTEGRASI INVERTED INDEX DAN
GENERALIZED SUFFIX TREE UNTUK MEREDUKSI WAKTU
PENCARIAN**

Skripsi

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



*Mencerdaskan dan
Memartabatkan Bangsa*

Oleh:
Mochammad Hanif Ramadhan
1313619025

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

2023

LEMBAR PERNYATAAN

Saya menyatakan dengan sesungguhnya bahwa skripsi dengan judul "**Implementasi Modul Indexing Pada Search Engine Telusuri Dengan Integrasi Inverted Index Dan Generalized Suffix Tree Untuk Mereduksi Waktu Pencarian**" 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 peneliti lain yang telah dipublikasikan yang disebutkan dalam teks Skripsi ini, telah dicantumkan dalam Daftar Pustaka sesuai dengan norma, kaidan 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, 21 Agustus 2023

Mochammad Hanif Ramadhan

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah SWT, karena dengan rahmat dan karunia-Nya, penulis dapat menyelesaikan proposal skripsi yang berjudul *Implementasi Modul Indexing Pada Search Engine Telusuri Dengan Integrasi Inverted Index Dan Generalized Suffix Tree Untuk Mereduksi Waktu Pencarian*.

Keberhasilan dalam penyusunan proposal skripsi ini tidak lepas dari bantuan berbagai pihak yang mana dengan tulus dan ikhlas memberikan masukan guna sempurnanya proposal skripsi ini. Oleh karena itu dalam kesempatan ini, dengan kerendahan hati penulis mengucapkan banyak terima kasih kepada:

1. Yth. Para petinggi di lingkungan FMIPA Universitas Negeri Jakarta.
2. Yth. Ibu Dr. Ria Arafiyah, M.Si selaku Koordinator Program Studi Ilmu Komputer.
3. Yth. Bapak Muhammad Eka Suryana, M.Kom selaku Dosen Pembimbing I yang telah membimbing, mengarahkan, serta memberikan saran dan koreksi terhadap proposal skripsi ini.
4. Yth. Bapak Med Irzal, M.Kom selaku Dosen Pembimbing II yang telah membimbing, mengarahkan, serta memberikan saran dan koreksi terhadap proposal skripsi ini.
5. Kedua orang tua dan kakak penulis yang telah mendukung dan memberikan semangat serta doa untuk penulis.
6. Wulan sebagai seseorang yang mau mendengarkan keluh kesah dan mendukung penulis.
7. Teman-teman Warung Tegang yang walau dengan segala keanehannya selalu memberikan semangat dan motivasi bagi penulis
8. Teman-teman SMAN 68 yang seringkali menemani penulis bertukar pikiran ketika beristirahat sejenak selama penulisan
9. Teman-teman Program Studi Ilmu Komputer 2019 yang telah memberikan dukungan dan memiliki andil dalam penulisan proposal skripsi ini.

10. Kedua kucing milik Pak Eka yang seringkali menghibur penulis selama melaksanakan bimbingan di Jasinga

Penulis menyadari bahwa penyusunan proposal skripsi ini masih jauh dari sempurna karena keterbatasan ilmu dan pengalaman yang dimiliki. Oleh karenanya, kritik dan saran yang bersifat membangun akan penulis terima dengan senang hati. Akhir kata, penulis berharap tugas akhir ini bisa bermanfaat bagi semua pihak khususnya penulis sendiri. Semoga Allah SWT senantiasa membela kebaikan semua pihak yang telah membantu penulis dalam menyelesaikan proposal skripsi ini.

Jakarta, 21 Agustus 2023

Mochammad Hanif Ramadhan

ABSTRAK

MOCHAMMAD HANIF RAMADHAN. Implementasi Modul Indexing Pada Search Engine Telusuri Dengan Integrasi Inverted Index Dan Generalized Suffix Tree Untuk Mereduksi Waktu Pencarian. Skripsi. Program Studi Ilmu Komputer. Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta. Agustus 2023. Di bawah bimbingan Muhammad Eka Suryana, M.Kom dan Med Irzal, M.Kom.

Mesin pencari atau *search engine* adalah program komputer yang digunakan untuk melakukan pencarian situs web. Pencarian dapat dilakukan dengan mengumpulkan informasi tentang halaman web terlebih dahulu. Karena ukuran data yang besar sebagai sumber informasi utama bagi mesin pencari, penggunaan *index* bisa dimanfaatkan untuk mereduksi waktu pencarian. Penelitian ini merupakan bagian dari rangkaian penelitian mesin pencari *Telusuri*, dan bertujuan untuk membuat implementasi *index* berdasarkan struktur *inverted index*, yaitu struktur *index* yang digunakan oleh mesin pencari *Google*, dan melakukan integrasi dengan struktur *Generalized Suffix Tree*. Tujuan utamanya adalah untuk mereduksi waktu pencarian suatu *keyword* dan memberikan peringkat terhadap dokumen berdasarkan kesesuaian antara informasi dalam dokumen dengan teks yang dimasukkan oleh pengguna. Hasil akhir dari implementasi modul menunjukkan reduksi waktu yang signifikan, berkisar antara 89 - 98%.

Kata kunci: *mesin pencari, indeks, basis data, teori informasi*

ABSTRACT

MOCHAMMAD HANIF RAMADHAN. Implementasi Modul Indexing Pada Search Engine Telusuri Dengan Integrasi Inverted Index Dan Generalized Suffix Tree Untuk Mereduksi Waktu Pencarian. Mini Thesis. Computer Science. Faculty of Mathematics and Natural Sciences, Universitas Negeri Jakarta. August 2023. Under guidance from Muhammad Eka Suryana, M.Kom and Med Irzal, M.Kom.

Search engine is a computer program that is used to search for a webpage. Before searching can be done, it is required to gather information about the webpage first. Due to the large collection of data needed for a search engine to be usable, indexes can be used to reduce time to retrieve information. This research is a part of a longer research for *Telusuri* search engine, and aims to implement an index structure based on inverted index form, which are used by *Google*, and integrates it with the *Generalized Suffix Tree*. The goal is to reduce the time taken to get informations from a given keyword and ranks the result based on the relevancy between the webpage and the input keyword. The index implementations are shown to be able to reduce the time taken significantly, ranging from 89 to 98%.

Kata kunci: *search engine, indexing, database, information retrieval, tree*

DAFTAR ISI

KATA PENGANTAR	iv
ABSTRAK	v
ABSTRACT	vi
DAFTAR ISI	vii
DAFTAR GAMBAR	ix
I PENDAHULUAN	1
1.1 Latar Belakang Masalah	1
1.2 Rumusan Masalah	4
1.3 Batasan Masalah	4
1.4 Tujuan Penelitian	4
1.5 Manfaat Penelitian	4
DAFTAR TABEL	1
II KAJIAN PUSTAKA	6
2.1 Proses <i>Indexing</i>	6
2.2 <i>Repository</i>	7
2.3 Daftar URL	7
2.4 <i>Document Index</i>	7
2.5 Daftar Kosakata	8
2.6 <i>docID</i> dan <i>wordID</i>	8
2.7 <i>Inverted Index</i>	8
2.7.1 Struktur Index	9
2.7.2 Skema Penyimpanan Index	11
2.7.3 Pemeringkatan Query	11
2.7.4 <i>Similarity Scoring</i>	12
2.8 Proses Pembuatan Index	12
2.8.1 Pengolahan Query	14
2.9 <i>Jaccard Distance</i>	15
2.10 Komunikasi antar proses	16
2.11 Modul <i>typing</i> pada Python	17
2.12 Perbandingan Metode <i>Indexing</i>	17
2.12.1 <i>Signature File Index</i>	17
2.12.2 <i>Bitmap Index</i>	18
2.12.3 <i>WordNet</i>	18

III METODOLOGI PENELITIAN	19
3.1 Tahapan Penelitian	19
3.2 Konstruksi Index	20
3.3 Daftar kata umum	22
3.4 Metode Pemeringkatan	23
3.4.1 <i>Word distance</i> (γ)	23
3.4.2 <i>Word similarity</i> (β)	24
3.5 Modifikasi Arsitektur <i>Telusuri</i>	25
3.6 Skema Pencarian	26
3.7 Alat dan Bahan Penelitian	30
3.8 Tahapan Pengembangan	30
3.8.1 Meningkatkan kemampuan modul <i>indexing</i> saat ini	30
3.8.2 Rancangan Eksperimen	31
IV HASIL DAN PEMBAHASAN	33
4.1 Implementasi	33
4.1.1 Pengolahan <i>Dataset</i>	34
4.1.2 Pembentukan <i>Inverted Index</i>	35
4.1.3 Pengolahan <i>User Query</i>	38
4.1.4 Proses Penggabungan <i>Hitlist</i> dan Pemeringkatan Hasil	41
4.1.5 Integrasi dengan <i>Generalized Suffix Tree (GST)</i>	45
4.1.6 Modifikasi kode <i>TFIDF</i>	49
4.1.7 Struktur Direktori Kode	54
4.2 Pengujian	54
4.2.1 Statistik Pengujian	54
4.2.2 Pengujian Relevansi	56
4.2.3 Pengujian Performa	64
4.3 Analisis Hasil	64
V KESIMPULAN DAN SARAN	66
5.1 Kesimpulan	66
5.2 Saran	66
DAFTAR PUSTAKA	68
LAMPIRAN	69
A Dokumentasi Pengujian Metode <i>Inverted Index</i>	69
B Dokumentasi Pengujian Metode <i>Inverted Index</i> dan Integrasi <i>GST</i>	71

DAFTAR GAMBAR

Gambar 2.1	Penggunaan alokasi memori dari <i>plain hit</i>	9
Gambar 2.2	Penggunaan alokasi memori dari <i>fancy hit</i>	10
Gambar 2.3	Rangkaian <i>hit</i> dengan struktur data <i>linked list</i>	10
Gambar 2.4	Penempatan jumlah <i>hit</i> dalam struktur index	11
Gambar 2.5	Contoh perbandingan kemiripan kata <i>beli</i> dan <i>dibeli</i> dengan <i>Jaccard index</i>	16
Gambar 3.1	Flowchart tahapan penelitian modul <i>indexing</i>	19
Gambar 3.2	Revisi arsitektur <i>Telusuri</i>	25
Gambar 3.3	Struktur objek <i>UserQuery</i>	26
Gambar 3.4	Flowchart pengolahan masukan query	27
Gambar 3.5	Ilustrasi hubungan struktur data	28
Gambar 3.6	Penulisan struktur data pada <i>Python</i>	28
Gambar 3.7	Flowchart pengolahan masukan query dengan modul <i>GST</i> . .	31
Gambar 4.1	<i>Flowchart</i> tahapan penelitian yang sudah jadi	33
Gambar 4.2	Statistik dataset hasil <i>crawling</i>	34
Gambar 4.3	Informasi pada tabel <i>page_paragraph</i>	35
Gambar 4.4	Pengambilan dataset dari database	35
Gambar 4.5	Pembuatan struktur index dari dataset	36
Gambar 4.6	Pengolahan paragraf menjadi <i>hitlist</i>	37
Gambar 4.7	Penyimpanan struktur index ke file persisten	38
Gambar 4.8	Fungsi pencarian utama	39
Gambar 4.9	Pengolahan teks input dari pengguna	40
Gambar 4.10	Fungsi untuk menyimpan posisi yang sesuai dengan teks input berdasarkan urutan	41
Gambar 4.11	Proses kalkulasi peringkat	42
Gambar 4.12	Grafik jumlah kata dalam dokumen	43
Gambar 4.13	Grafik jumlah kata dalam dokumen setelah menyingkirkan 20 entri tertinggi	43
Gambar 4.14	Penyaringan terhadap dokumen yang telah di- <i>blacklist</i> . . .	44
Gambar 4.15	Pengambilan dokumen dari database dan pemetaan informasi dokumen berdasarkan skor	44
Gambar 4.16	Perubahan pada iterasi ketika mengambil data dari database .	45
Gambar 4.17	Perubahan pada iterasi ketika membentuk struktur <i>tree</i> . . .	45
Gambar 4.18	Implementasi <i>class GST</i> untuk memudahkan integrasi . . .	46
Gambar 4.19	Penambahan inisiasi variabel untuk akomodasi penggunaan <i>GST</i>	47
Gambar 4.20	Perubahan implementasi perhitungan peringkat untuk integrasi <i>GST</i>	48
Gambar 4.21	Fungsi untuk mengambil dokumen berdasarkan hasil dari <i>GST</i>	49

Gambar 4.22 Operasi filter pada seluruh dataset dari <i>table_information</i> untuk menyisakan seluruh kemunculan kata dasar pada tiap dokumen	50
Gambar 4.23 Pengisian tabel <i>page_information_filtered</i> dengan data yang sudah di filter dengan daftar kata dasar	51
Gambar 4.24 Proses skoring kata dari data yang sudah di filter	52
Gambar 4.25 Pengisian tabel <i>tfidf_word</i> dari file persisten	53
Gambar 4.26 Statistik dataset setelah proses <i>filtering</i>	54

DAFTAR TABEL

Tabel 2.1	Contoh teks dari lirik lagu anak-anak	12
Tabel 2.2	<i>Inverted file index</i> dari lirik lagu pada tabel 2.1	13
Tabel 2.3	<i>Word-level index</i> dari lirik lagu pada tabel 2.1	14
Tabel 4.1	Durasi proses <i>reindexing</i>	54
Tabel 4.2	Durasi pembuatan struktur <i>GST</i>	55
Tabel 4.3	Durasi proses penyimpanan struktur data ke file persisten . . .	55
Tabel 4.4	Durasi proses muat ulang struktur data dari file persisten . . .	55
Tabel 4.5	Ukuran struktur data yang terbentuk	56
Tabel 4.6	<i>Query</i> yang digunakan untuk uji relevansi	56
Tabel 4.7	Hasil pencarian dengan kata kunci <i>bupati</i> dengan metode <i>inverted index</i>	56
Tabel 4.8	Hasil pencarian dengan kata kunci <i>bupati</i> dengan metode <i>inverted index</i> dan integrasi dengan <i>GST</i>	57
Tabel 4.9	Hasil pencarian dengan kata kunci <i>bupati</i> dengan metode <i>TF-IDF</i>	58
Tabel 4.10	Hasil pencarian dengan kata kunci <i>waskita</i> dengan metode <i>inverted index</i>	58
Tabel 4.11	Hasil pencarian dengan kata kunci <i>waskita</i> dengan metode <i>inverted index</i> dan integrasi dengan <i>GST</i>	59
Tabel 4.12	Hasil pencarian dengan kata kunci <i>waskita</i> dengan metode <i>TF-IDF</i>	59
Tabel 4.13	Hasil pencarian dengan kata kunci <i>nuklir</i> dengan metode <i>inverted index</i>	60
Tabel 4.14	Hasil pencarian dengan kata kunci <i>nuklir</i> dengan metode <i>inverted index</i> dan integrasi dengan <i>GST</i>	60
Tabel 4.15	Hasil pencarian dengan kata kunci <i>nuklir</i> dengan metode <i>TF-IDF</i>	61
Tabel 4.16	Hasil pencarian dengan kata kunci <i>tiket coldplay</i> dengan metode <i>inverted index</i>	61
Tabel 4.17	Hasil pencarian dengan kata kunci <i>tiket coldplay</i> dengan metode <i>inverted index</i> dan integrasi dengan <i>GST</i>	62
Tabel 4.18	Hasil pencarian dengan kata kunci <i>tiket coldplay</i> dengan metode <i>TF-IDF</i>	62
Tabel 4.19	Hasil pencarian dengan kata kunci <i>ganjar pranowo</i> dengan metode <i>inverted index</i>	63
Tabel 4.20	Hasil pencarian dengan kata kunci <i>ganjar pranowo</i> dengan metode <i>inverted index</i> dan integrasi dengan <i>GST</i>	63
Tabel 4.21	Hasil pencarian dengan kata kunci <i>ganjar pranowo</i> dengan metode <i>TF-IDF</i>	64
Tabel 4.22	Perbandingan durasi waktu pencarian	64

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

Mesin pencari atau *search engine* adalah program komputer yang digunakan untuk melakukan pencarian situs web. Pada awal kemunculannya, mesin pencari lebih diperuntukkan bagi kalangan akademisi untuk mencari jurnal atau dokumen akademik lainnya. Namun, seiring dengan meluasnya adopsi internet ke berbagai lapisan masyarakat, mesin pencari memiliki peran yang lebih besar dalam penggunaan internet. Mesin pencari berubah menjadi sarana utama dalam pencarian informasi secara umum.

Bila dilihat dari penggunaannya, mesin pencari bekerja dengan cara mengolah *query* yang diberikan oleh pengguna dan menampilkan hasil terkait informasi tertentu di internet yang paling sesuai dengan *query* tersebut. Tetapi mesin pencari tidak dapat langsung mencari halaman secara langsung melalui internet. Mesin pencari membutuhkan database yang berisi informasi pada halaman yang tersebar di seluruh jaringan internet.

Ketika internet masih memiliki lingkup pengguna yang kecil, metode pengumpulan data untuk database mesin pencari masih terbilang primitif. Beberapa orang membuat katalog dari berbagai situs dengan kategori tertentu secara manual dan memperbaruiinya dari waktu ke waktu (Seymour et al., 2011). Namun, makin tingginya adopsi internet membuat jumlah data pada internet meledak. Hal ini berakibat pada diperlukannya metode pengumpulan data yang lebih efisien dan metode penerimaan informasi yang lebih cepat.

Mesin pencari membutuhkan serangkaian proses yang perlu dilakukan sebelum dapat menerima *query* terkait informasi tertentu. *Google*, mesin pencari paling populer saat ini, mempunyai beberapa komponen yang menjalankan tugas secara spesifik. Arsitektur *Google* terdiri dari beberapa komponen utama seperti *crawler*, modul *indexer*, modul pemeringkatan *PageRank* dan modul pencari (Brin et al., 1998). Seluruh modul tersebut dibuat secara mandiri tanpa menggunakan aplikasi atau layanan pihak ketiga.

Meski dengan informasi tentang rancangan arsitektur dari publikasi artikel, implementasi ulang arsitektur *Google* secara menyeluruh cukup sulit dilakukan

karena banyak detail dari arsitektur yang tidak dicantumkan. Selain itu *Google* juga banyak membuat implementasi yang telah dioptimalkan untuk kebutuhan mesin pencari baik dari segi performa maupun penggunaan ruang penyimpanannya, tanpa mempublikasikan kode program atau algoritma yang digunakan.

Upaya implementasi ulang arsitektur *Google* telah dilakukan (Khatulistiwa 2023) dan menghasilkan integrasi mesin pencari secara utuh. Arsitektur tersebut merupakan pengembangan dari penelitian yang telah dilakukan sebelumnya yang menghasilkan *web crawler* (Qoriiba 2021). *Web crawler* bertugas mengumpulkan halaman web berdasarkan *entrypoint* tertentu. Halaman web tersebut kemudian di ekstrak dan dikumpulkan daftar kata yang termuat pada halaman web tersebut serta *outgoing link* yang merujuk kepada halaman web lain. Kedua data tersebut kemudian disimpan pada database.

Data yang tersimpan pada database selanjutnya di proses oleh modul *PageRank*. Modul *PageRank* menghitung peringkat suatu halaman web berdasarkan seberapa banyak halaman web lain yang merujuk kepada halaman web tersebut. Setelah kalkulasi peringkat halaman, data akan diolah oleh modul *TF-IDF*. Modul *TF-IDF* akan menghitung bobot kata pada dokumen berdasarkan frekuensi kemunculan kata tersebut dalam suatu dokumen tertentu dan jumlah dokumen yang mengandung kata tersebut. Hasil perhitungan skor dari modul *PageRank* dan *TF-IDF* kemudian akan digabungkan dengan menggunakan algoritma *similarity scoring* yang akan menghasilkan skor relevansi suatu halaman.

Arsitektur *Telusuri* saat ini masih memiliki berbagai kekurangan, dan implementasinya juga cukup berbeda dibandingkan dengan arsitektur milik *Google* karena keterbatasan detailnya. Salah satu modul yang belum memiliki implementasi yang sesuai adalah modul *indexing*. *Indexing* adalah suatu proses pemetaan *record* pada database dengan tujuan mempercepat proses pengambilan *record* dari database dan meningkatkan relevansi hasil pencarian. Proses *indexing* akan menghasilkan *index*, yaitu suatu representasi data yang merujuk pada lokasi data yang lebih lengkap. Konsep penggunaan *index* pada mesin pencari sama dengan index yang biasa ditemukan di bagian belakang buku.

Representasi data pada *index* memiliki tingkat akurasi lokasi data (*granularity*) yang dapat diatur, seperti suatu frasa dalam suatu paragraf, atau unit data yang lebih kecil seperti satu kata tertentu saja. Pemilihan tingkat *granularity* dapat memengaruhi performa dan akurasi dari proses pengambilan data. Sebagai contoh, penggunaan *synonym ring* yang dapat melakukan pengelompokan kata yang

bermakna sama seperti *WordNet* dapat memberikan hasil yang lebih baik dibandingkan dengan hanya menggunakan potongan kata biasa (Gonzalo et al., 1998).

Terdapat berbagai implementasi *index* yang disesuaikan dengan jenis data yang disimpan pada database. Mesin pencari membutuhkan implementasi *index* yang dioptimalkan untuk teks secara menyeluruh. Dalam artikel yang berjudul *An Efficient Indexing Technique for Full-Text Database Systems*, implementasi *index* yang difokuskan kepada penyimpanan teks setidaknya perlu melakukan tiga hal secara efisien. Yang pertama adalah mendukung pengambilan dokumen berdasarkan *query* berupa beberapa kata yang digabungkan oleh operator logika. Yang kedua adalah memiliki kemampuan untuk menambahkan *record* baru secara efisien. Yang ketiga adalah kemampuan untuk membuat pemeringkatan terhadap *record* yang ada apabila tidak ada data yang memenuhi *query* secara penuh (Zobel et al., 1992).

Dari persyaratan di atas, solusi yang umum digunakan untuk database penyimpanan teks adalah *inverted file index* atau biasa disebut *inverted index* saja. Wujud dari *inverted index* adalah sebuah struktur data yang memetakan kosakata dengan dokumen atau teks utama tempat kosakata tersebut berada(Hersh 2001). *Google* juga menggunakan struktur *inverted index* pada implementasi arsitekturnya (Brin et al., 1998).

Proses *indexing* yang akan di replika akan menggunakan detail yang terbatas yang dapat diakses pada *paper Google*. Untuk memroses *query*, *Google* melakukan beberapa langkah berikut. Pertama, *query* dipotong per kata. Setiap kata kemudian di konversi menjadi *wordID*, yang dapat mengidentifikasi suatu kata secara unik.

Dari kumpulan *wordID* tersebut, untuk setiap kata akan dicari kemunculannya pada permulaan daftar *index* yang berisi judul dan *outgoing links*. Pencarian dilakukan dengan proses *scanning* secara menyeluruh pada daftar *index* sampai ditemukan dokumen yang memenuhi seluruh kata pada *query*. Untuk setiap dokumen yang ditemukan, akan dihitung skor dokumen berdasarkan *query* yang ada.

Jika telah sampai pada akhir daftar dokumen dan posisi pencarian berada pada daftar judul dan *outgoing links*, maka posisi akan berpindah ke daftar kata pada seluruh dokumen, dan mulai melakukan pencarian dokumen lagi. Tetapi jika belum mencapai akhir daftar dokumen, maka posisi pencarian akan diulang dari posisi awal dan memulai pencarian lagi. Setelah seluruh daftar *index* dikunjungi, nantinya akan diakhiri dengan mengurutkan dokumen berdasarkan skor.

Penelitian tentang struktur data untuk keperluan *indexing* telah dilakukan dan

menghasilkan implementasi modul *indexing* dengan menggunakan struktur *generalized suffix tree (GST)* (Pratama 2023). Implementasi dari metode *inverted index* dapat menggantikan implementasi modul yang sudah ada secara menyeluruh atau diintegrasikan dengan modul yang sudah ada sebagai bentuk peningkatan kemampuan modul. Modul *indexing* nantinya juga dapat berperan sebagai metode *query ranking* yang berbeda, atau melakukan enkapsulasi nilai *query ranking* yang sudah ada.

1.2 Rumusan Masalah

Berdasarkan uraian pada latar belakang yang diutarakan di atas, maka perumusan masalah pada penelitian ini adalah "**bagaimana meningkatkan performa waktu pencarian dengan menggunakan index ?**".

1.3 Batasan Masalah

Batasan masalah pada penelitian ini adalah pembuatan salah satu komponen dalam arsitektur mesin pencari yaitu modul *indexing*. Sistem *indexing* yang akan dibuat mengacu pada konsep arsitektur *Google* (Brin et al., 1998). Selain itu, implementasi sistem *indexing* ini hanya akan mencakup proses konstruksi awal index berdasarkan data yang sudah dikumpulkan oleh *crawler*.

1.4 Tujuan Penelitian

1. Membuat implementasi *indexing* berdasarkan konsep *inverted index* untuk memenuhi kebutuhan mesin pencari
2. Melakukan integrasi antara struktur *inverted index* dengan *Generalized Suffix Tree* yang sudah dibuat pada penelitian sebelumnya

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 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 Proses *Indexing*

Indexing adalah proses pemetaan seluruh data yang pada *database* dengan tujuan untuk mempercepat proses pencarian (*lookup*) dari suatu informasi. Layaknya halaman *index* dalam sebuah buku, dengan mencari kata kunci yang sesuai / relavan dengan informasi yang ingin dicari, kita bisa mendapatkan halaman yang mengandung informasi tersebut. Hal tersebut adalah target yang sama yang ingin dicapai dalam implementasi modul *indexing* untuk mesin pencari.

Penggunaan index memberikan pengaruh signifikan dalam proses penerimaan data dari database karena dapat mengurangi waktu akses *storage*. Hal ini sejalan dengan pola penggunaan mesin pencari, di mana mayoritas operasi yang dilakukan adalah memberikan informasi berdasarkan *query* dari pengguna. Selain itu, representasi index tertentu juga dapat memengaruhi pengolahan *query*, penggunaan ruang pada media penyimpanan dan akurasi dari hasil *query* tersebut.

Proses dimulai dengan mengambil data mentah dari tempat penyimpanan halaman web yang telah dikumpulkan oleh *crawler* (*repository*). Untuk setiap dokumen yang sudah di-*crawl*, akan di ekstrak seluruh kata yang memiliki relevansi terhadap informasi tertentu. Seluruh kata dalam dokumen kemudian akan dipetakan sesuai dengan struktur index yang digunakan.

Database yang digunakan oleh mesin pencari adalah database yang dirancang untuk kebutuhan penyimpanan teks secara penuh. Untuk keperluan tersebut, setidaknya terdapat tiga syarat yang perlu dipenuhi ketika ingin menentukan representasi index (Zobel et al., 1992).

1. Memungkinkan untuk mendapatkan data berdasarkan suatu query

Dari suatu query pencarian, database harus dapat memberikan hasil yang dapat memenuhi query tersebut. Pada konteks penerimaan data yang berbentuk teks, konjungsi query sangat umum digunakan. Dalam kasus ini, index harus mampu memberikan informasi apakah suatu *record* mengandung kata tertentu, dan secara langsung memengaruhi penentuan granularitas index.

2. Menambahkan *record* baru secara efisien

Database yang hanya menyimpan teks umumnya digunakan untuk keperluan penyimpanan. Pada kasus mesin pencari, operasi yang sering dilakukan adalah penulisan data dari *crawler* ke database dan pembacaan data berdasarkan query. Operasi untuk mengubah atau menghapus data jarang diperlukan, walaupun operasi tersebut tetap perlu didukung.

3. Memberikan skor terhadap hasil dari query yang bersifat ‘informal’

Pada kasus mesin pencari, pengguna seringkali memberikan query yang bersifat ‘informal’, yaitu query yang tidak dapat dipenuhi seluruhnya oleh database. Dalam hal ini, database perlu memberikan hasil yang paling mendekati ekspektasi dari query tersebut dengan cara menyertakan nilai relevansi suatu data terhadap query.

2.2 Repository

Repository merupakan tempat penyimpanan seluruh kode HTML dari setiap halaman web yang telah di-*crawl*. Setiap dokumen memiliki data tambahan yang disematkan kepadanya, yaitu nomor *id* yang dibuatkan untuk dokumen tersebut (*docID*), panjang dari dokumen, dan *URL* dari dokumen tersebut. Seluruh data tersebut dikompres dengan library *zlib* untuk mengurangi penggunaan ruang pada *repository*.

2.3 Daftar URL

Ketika mengolah data dari *repository*, terdapat langkah tambahan yang dilakukan jika menemui *anchor text*. *Anchor text* adalah teks berupa URL yang merujuk kepada dokumen lain. Setiap *anchor text* yang ditemukan pada data dari *repository* akan disimpan pada sebuah daftar URL.

2.4 Document Index

Document index menyimpan beberapa informasi tambahan pada setiap dokumen. Setiap index akan memiliki nomor dokumen, status dokumen, alamat tempat dokumen tersimpan di *repository*, dan *checksum*. Apabila pada suatu dokumen telah dilakukan proses *crawling*, maka akan disematkan data tambahan yang berisi sepasang informasi berupa URL dan judul dari dokumen tersebut. Jika

belum melalui tahap *crawling*, maka data tambahan tersebut hanya merujuk kepada URL yang terdapat pada daftar URL.

2.5 Daftar Kosakata

Setiap hasil pengolahan data dari *repository* akan memberikan daftar kosakata yang bersifat unik. Daftar kosakata tersebut kemudian akan digabungkan kedalam daftar kosakata utama yang menampung seluruh kosakata yang sudah pernah di-*index* sebelumnya.

Implementasi daftar kosakata ini terbagi menjadi dua bagian, yaitu daftar kosakata itu sendiri dan daftar *pointer* yang merujuk kepada dokumen tempat kosakata itu berada. Selain itu, daftar kosakata dapat menyematkan informasi tambahan seperti jumlah dokumen tempat kosakata tersebut muncul.

Untuk mencegah akses ke ruang penyimpanan yang berlebihan, salah satu syarat dasar dari implementasi index adalah daftar seluruh kosakata dapat dimuat dalam memori. Hal ini bertujuan untuk memastikan bahwa dalam situasi pencarian hasil dengan query boolean, hanya membutuhkan setidaknya satu kali akses ke ruang penyimpanan per kata.

2.6 *docID* dan *wordID*

Untuk merepresentasikan dokumen dan kosakata secara unik, digunakan *id* yang bersifat sekuensial. Dokumen tidak lagi diidentifikasi berdasarkan *URL* halaman.

2.7 *Inverted Index*

Inverted index merupakan jenis index yang memetakan potongan isi dari suatu dokumen atau koleksi dokumen terhadap lokasi aslinya di dokumen atau koleksi dokumen tersebut. Karena bentuk strukturnya, inverted index cocok digunakan untuk kebutuhan penerimaan data berdasarkan query seperti mesin pencari.

2.7.1 Struktur Index

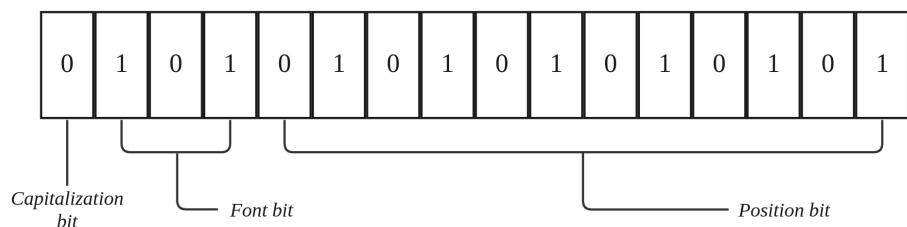
Representasi dari kemunculan suatu kata pada dokumen, atau biasa disebut sebagai *hit*, memiliki beberapa data tertentu terkait kata yang dikandung. Setiap *hit* merepresentasikan suatu kemunculan kata beserta data berikut:

- Lokasi kata dalam dokumen
- Jenis *font* yang digunakan
- Informasi terkait penggunaan huruf kapital
- Properti teks lainnya yang tidak dijelaskan kegunaannya

Hit terbagi menjadi tiga jenis berdasarkan lokasinya dalam struktur halaman web. Yang pertama adalah *fancy hit*, yaitu *hit* yang berada di dalam *URL*, judul halaman, atau *meta tag*. Yang kedua adalah *anchor hit*, yaitu *hit* yang khusus berada di dalam *anchor text* yang merupakan *URL* yang merujuk kepada dokumen atau halaman lain. Yang ketiga adalah *plain hit*, yaitu *hit* yang berada di luar lingkup dari *fancy hit*.

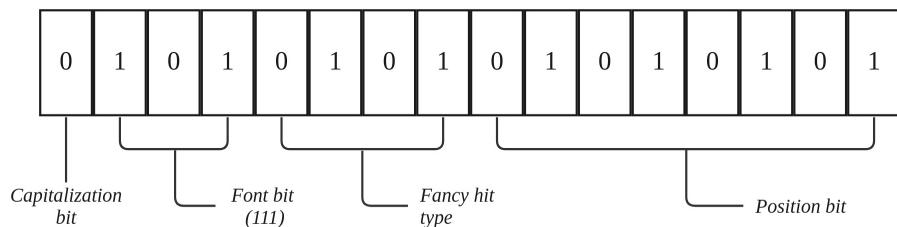
Alokasi memori yang dibutuhkan untuk menyimpan sebuah *hit* adalah 8 byte atau 16 bit. Penggunaan bit tersebut berbeda tergantung dari jenis *hit*, tetapi memiliki cara yang hampir sama dalam menggunakannya.

Bit pertama digunakan sebagai penanda untuk penggunaan huruf kapital. Selanjutnya, bit kedua hingga keempat digunakan sebagai penanda ukuran font yang digunakan. Ukuran font ini bersifat relatif terhadap seluruh kata dalam dokumen. Selain itu, nilai bit *111* tidak digunakan sebagai ukuran penanda ukuran font, namun digunakan sebagai penanda untuk *fancy hit*. Bit kelima hingga terakhir digunakan sebagai penanda posisi kata dalam dokumen. Karena keterbatasan bit, posisi yang memiliki nilai lebih dari 4095 akan dianggap bernilai 4096.



Gambar 2.1: Penggunaan alokasi memori dari *plain hit*

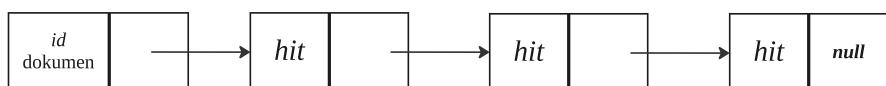
Pada penggunaan *fancy hit*, alokasi bit untuk penanda posisi dibagi menjadi dua. Bit ke-9 hingga ke-12 digunakan sebagai penanda jenis dari *fancy hit*, sementara bit ke-13 hingga terakhir tetap digunakan sebagai penanda posisi.



Gambar 2.2: Penggunaan alokasi memori dari *fancy hit*

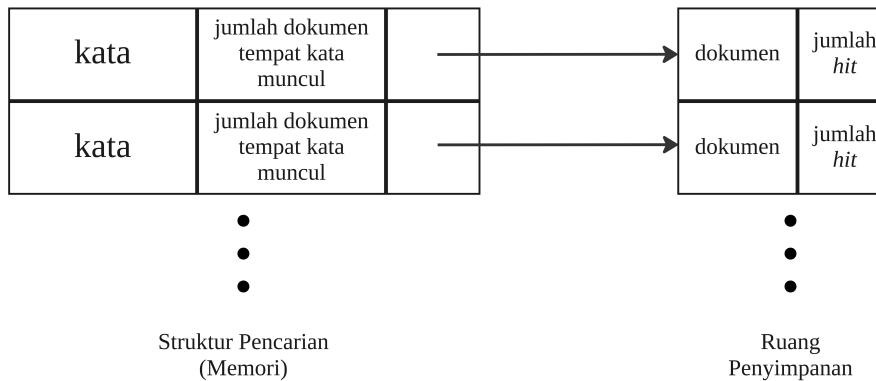
Dari penggunaan *fancy hit*, penggunaan *anchor hit* membagi sisa delapan bit terakhir menjadi dua, yaitu empat bit awal sebagai penanda posisi dan empat bit terakhir sebagai nilai *hash* dari dokumen di mana kata tersebut berada.

Hit dirangkai dengan menggunakan struktur *linked list* agar penambahan hit lebih mudah dilakukan ketika proses indexing ulang. Urutan dari *hit* dimulai dari kemunculan pertama kata tersebut dalam dokumen tertentu. *Hit* pertama dalam dokumen kemudian disambungkan ke *docID* di depannya dengan menggunakan pointer.



Gambar 2.3: Rangkaian *hit* dengan struktur data *linked list*

Iterasi terhadap seluruh *hit* dapat dihindari dengan menyimpan total *hit* untuk suatu kata dalam suatu dokumen di depan rangkaian *hit*. Hal ini akan membuat proses penambahan *hit* menjadi lebih efisien dan memudahkan kalkulasi tingkat kepentingan suatu kata. Jumlah *hit* memiliki jumlah bit yang berbeda tergantung dari lokasi daftar index. Untuk *inverted index* digunakan 5 bit, dan untuk *forward index* digunakan 8 bit.



Gambar 2.4: Penempatan jumlah *hit* dalam struktur index

2.7.2 Skema Penyimpanan Index

Proses indexing akan menghasilkan *forward index*, yaitu index yang telah terurut berdasarkan *docID*. Untuk membuat *inverted index*, data pada *forward index* akan diurutkan berdasarkan *wordID*.

Kemudian, dibuat dua salinan dari daftar index. Salinan pertama berisi daftar *hit* yang termasuk dalam kategori *fancy hit* atau *anchor hit*. Salinan kedua adalah daftar *hit* sisanya. Hal ini bertujuan untuk mengurangi kemungkinan akses dari index, karena pada sebagian besar kasus query dapat dipenuhi dengan hanya mengakses data pada *fancy hit* dan *anchor hit*.

Pada contoh kasus mesin pencari, jumlah data di seluruh internet yang terus bertambah membuat potensi penggunaan media penyimpanan menjadi hampir tidak terbatas. Hal tersebut dapat menyebabkan tidak cukupnya kapasitas memori untuk menampung seluruh index. Untuk mengatasinya, skema penyimpanan index perlu di partisi menjadi beberapa bagian.

Selain itu, pada tiap salinan index, index juga akan di sortir berdasarkan *id* dari dokumen. Tujuannya adalah untuk memudahkan proses *merging* ketika dihadapkan pada situasi di mana query memiliki banyak kata.

2.7.3 Pemeringkatan Query

Untuk memenuhi kebutuhan tambahan seperti pemeringkatan query, struktur pencarian perlu menampung data tambahan seperti frekuensi kemunculan kata, posisi kata dalam dokumen, dan lain-lain. Apabila data tersebut tidak dapat dimuat seluruhnya pada memori, maka struktur pencarian dapat dipecah dan disimpan

sebagian pada ruang penyimpanan. Sebagai contoh, kata yang bersifat umum dapat disimpan pada memori untuk mengurangi kemungkinan akses ke ruang penyimpanan.

2.7.4 Similarity Scoring

Arsitektur *Telusuri* saat ini menggunakan algoritma *similarity scoring* untuk mendapatkan nilai relevansi suatu halaman dengan rumus berikut

$$W = \alpha \cdot PR + \beta \cdot QR \quad (2.1)$$

di mana

PR = Nilai *PageRank* dari dokumen

α = Bobot nilai *PageRank*

QR = Skor pemeringkatan *query*

β = Bobot nilai peringkat *query*

Untuk metode pemeringkatan *query* sendiri, terdapat beberapa metode yang dapat digunakan secara bersamaan seperti *TF-IDF*, *Skipgram* dan lain-lain. Hasil dari peringkat tersebut digabungkan dengan menggunakan rumus berikut

$$QR = \beta_1 \cdot QR_1 + \cdots + \beta_N \cdot QR_N \quad (2.2)$$

2.8 Proses Pembuatan Index

Sebagai contoh, pada tabel 2.1 digunakan sampel teks berupa lirik lagu anak-anak. Pada kasus ini, diasumsikan bahwa setiap baris merupakan isi dari dokumen terpisah.

Tabel 2.1: Contoh teks dari lirik lagu anak-anak

Dokumen	Teks
1	Pease porridge hot, pease porridge cold,
2	Pease porridge in the pot,
3	Nine days old.
4	Some like it hot, some like it cold,
	Dilanjutkan pada halaman berikutnya

Tabel 2.1: Contoh teks dari lirik lagu anak-anak

Dokumen	Teks
5	Some like it in the pot,
6	Nine days old.

Seluruh kata dalam seluruh koleksi dokumen kemudian akan diambil jumlah kemunculannya beserta nomor dokumen tempat kata tersebut ditemukan, dan dikumpulkan berdasarkan keunikan kata-nya.

Tabel 2.2: *Inverted file index* dari lirik lagu pada tabel 2.1

Nomor	Kata	Index
1	cold	$\langle 2; 1, 4 \rangle$
2	days	$\langle 2; 3, 6 \rangle$
3	hot	$\langle 2; 1, 4 \rangle$
4	in	$\langle 2; 2, 5 \rangle$
5	it	$\langle 2; 4, 5 \rangle$
6	like	$\langle 2; 4, 5 \rangle$
7	nine	$\langle 2; 3, 6 \rangle$
8	old	$\langle 2; 3, 6 \rangle$
9	pease	$\langle 2; 1, 2 \rangle$
10	porridge	$\langle 2; 1, 2 \rangle$
11	pot	$\langle 2; 2, 5 \rangle$
12	some	$\langle 2; 4, 5 \rangle$
13	the	$\langle 2; 2, 5 \rangle$

Hasil index pada tabel 2.2 merupakan bentuk *inverted file index*, yang memetakan kata ke dokumen tempat kemunculan kata tersebut. Untuk membuat implementasi ulang modul index milik Google, maka diperlukan index dengan granularitas yang lebih halus. Modul index milik Google menggunakan *word-level index*, di mana index juga menyimpan posisi kata dalam dokumen. *Word-level index*

dapat mengatasi situasi di mana jumlah kemunculan suatu kata dalam dokumen lebih dari satu kali.

Dengan mengabaikan detail selain posisi kata dalam dokumen, index pada tabel 2.2 dapat dikembangkan menjadi seperti berikut

Tabel 2.3: *Word-level index* dari lirik lagu pada tabel 2.1

Nomor	Kata	Index
1	cold	$\langle 2; (1; 6), (4; 8) \rangle$
2	days	$\langle 2; (3; 2), (6; 2) \rangle$
3	hot	$\langle 2; (1; 3), (4; 4) \rangle$
4	in	$\langle 2; (2; 3), (5; 4) \rangle$
5	it	$\langle 2; (4; 3, 7), (5; 3) \rangle$
6	like	$\langle 2; (4; 2, 6), (5; 2) \rangle$
7	nine	$\langle 2; (3; 1), (6; 1) \rangle$
8	old	$\langle 2; (3; 3), (6; 3) \rangle$
9	pease	$\langle 2; (1; 1, 4), (2; 1) \rangle$
10	porridge	$\langle 2; (1; 2, 5), (2; 2) \rangle$
11	pot	$\langle 2; (2; 5), (5; 6) \rangle$
12	some	$\langle 2; (4; 1, 5), (5; 1) \rangle$
13	the	$\langle 2; (2; 4), (5; 5) \rangle$

2.8.1 Pengolahan Query

Untuk query yang hanya mengandung satu kata, hasil bisa langsung didapatkan dengan melakukan pencarian kosakata yang memenuhi query pada daftar kosakata. Setelah ditemukan dokumen yang membuat kata yang memenuhi query tersebut diambil dengan petunjuk dari data yang disematkan pada daftar kosakata.

Query yang melibatkan banyak kata akan memerlukan proses penggabungan hasil dari kosakata yang sesuai. Sebagai contoh mudahnya, query yang melibatkan banyak kata dapat digabungkan dengan menggunakan operator logika. Operator *OR* akan memberikan gabungan dari hasil. Sementara operator *AND* akan memberikan irisan dari hasil. Jika diberikan query sebagai berikut

$$Q_1 \text{ AND } Q_2 \text{ AND } \dots \text{ AND } Q_N$$

maka dilakukan pencarian untuk setiap dokumen yang mengandung kata yang memenuhi seluruh rangkaian Q_1 hingga Q_n . Dari seluruh dokumen yang sesuai, maka dicari irisan dari hasil tersebut (dokumen yang memiliki kedua query tersebut).

Sebagai contoh, dari index pada tabel 2.2, apabila diberikan query

hot AND like

maka akan didapatkan lokasi dari index dengan kosakata *hot* dan *like*, yaitu $\langle 1, 4 \rangle$ dan $\langle 4, 5 \rangle$. Dari data lokasi tersebut, mereka kemudian digabungkan (diambil irisannya), sehingga didapatkan dokumen 4 yang memenuhi kedua query tersebut.

Untuk query yang mengandung banyak kata yang tidak digabungkan dengan operator boolean, penggunaan *inverted file* seperti contoh di atas akan menimbulkan banyaknya *false match* yang mengurangi kualitas hasil dari query. *False match* sendiri dapat di atas dengan melakukan pemindaian untuk memastikan hasil dari query. Tetapi pemindaian akan mengurangi performa keseluruhan dari proses pengambilan data.

Penggunaan *word-level index* dapat menangani hal tersebut dengan menggunakan informasi yang lebih lengkap yang tersemat pada tiap *hit*. Karena terdapat informasi terkait posisi kata dalam dokumen, jarak antara kata dapat dibandingkan sesuai dengan query.

2.9 Jaccard Distance

Jaccard index adalah suatu metode statistik yang digunakan untuk mengukur kemiripan pada suatu kumpulan sampel. *Jaccard index* mengukur kemiripan berdasarkan suatu kumpulan data yang terbatas, dan dapat dijabarkan sebagai nilai dari perpotongan dibagi dengan *union* dari data (Jaccard 1912).

Diberikan dua buah himpunan A dan B , maka nilai dari *Jaccard index* didapatkan melalui rumus

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (2.3)$$

dengan kondisi $0 \leq J(A, B) \leq 1$. Apabila $|A \cap B|$ adalah himpunan kosong, maka nilai dari $J(A, B)$ adalah 0.

Jaccard distance (d_J) adalah salah satu penerapan dari metode statistik dengan *Jaccard index*. *Jaccard distance* digunakan untuk mengukur nilai ketidakmiripan di

antara dua sampel data. Nilai d_J didapatkan dengan cara mengurangi skor 1 dengan nilai yang didapatkan pada perhitungan *Jaccard index*.

$$d_J(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \quad (2.4)$$

Berikut adalah contoh penggunaan *Jaccard distance* untuk membandingkan dua buah kata. Himpunan yang ada terdiri dari seluruh kemungkinan pasangan *bigram* huruf yang dapat dibentuk.

	di	ib	be	el	li
beli	0	0	1	1	1
dibeli	1	1	1	1	1

Gambar 2.5: Contoh perbandingan kemiripan kata *beli* dan *dibeli* dengan *Jaccard index*

2.10 Komunikasi antar proses

Pada implementasi sistem operasi modern, dalam situasi di mana banyak proses berjalan secara bersamaan, maka setiap proses akan terisolir dari proses lainnya. Sistem operasi akan membuat sebuah ruang virtual khusus untuk tiap proses, menjadikan setiap proses seakan-akan dapat mengakses seluruh sumber daya dari perangkat. Sistem operasi lalu mengelola pembagian sumber daya asli yang tersedia untuk seluruh proses. Karena isolasi tersebut, proses pada situasi normal tidak memiliki akses terhadap data pada memori yang dialokasikan oleh proses lain. Perlu dilakukan penulisan data pada media penyimpanan persisten agar proses dapat saling bertukar informasi.

Karena waktu akses media penyimpanan persisten membutuhkan waktu yang sangat lama, maka sistem operasi menyediakan beberapa metode komunikasi antar proses yang dapat digunakan untuk menghindari hal tersebut. Sistem operasi Linux memberikan beberapa pilihan metode, diantaranya yaitu *signal*, *pipe*, *shared memory* dan *socket*.

Socket adalah metode komunikasi antar proses yang sering digunakan saat ini. *Socket* sendiri umumnya terbagi menjadi dua jenis, yaitu *network socket* yang biasa digunakan oleh seluruh perangkat untuk berkomunikasi melalui jaringan internet, dan *UNIX domain socket* yang merupakan metode komunikasi antar proses standar pada sistem operasi yang bersifat *POSIX compliant*.

2.11 Modul *typing* pada Python

Untuk deklarasi struktur data pada modul *indexing*, penulis menggunakan bahasa pemrograman *Python*. *Python* dipilih karena seluruh kode program yang ada pada arsitektur *Telusuri* menggunakan bahasa yang sama.

Selain itu penulis menggunakan modul *typing* yang mulai diperkenalkan pada *Python* versi 3.5 untuk memberikan keterangan yang lebih jelas terkait tipe data apa yang digunakan (*type annotation*). Dengan memberikan keterangan tipe data yang digunakan secara eksplisit, kode juga menjadi lebih mudah ditulis dan dibaca.

Fitur *type annotation* yang disediakan oleh modul *typing* diperkenalkan dengan *PEP-484* pada tahun 2014 dengan tujuan untuk memberikan standar penulisan sintaks untuk tipe data (Rossum et al., 2015).

2.12 Perbandingan Metode *Indexing*

2.12.1 *Signature File Index*

Signature file adalah suatu metode probabilistik yang digunakan sebagai alternatif untuk mengindeks teks. Pada implementasi *signature file*, setiap dokumen memiliki suatu *descriptor*, yaitu rangkaian *bit* yang merepresentasikan konten dari dokumen. *Descriptor* dibuat dengan cara menggunakan beberapa *hash function* untuk mendapatkan suatu rangkaian bit unik yang merepresentasikan suatu kata dalam dokumen.

Karena sifatnya yang probabilistik, *descriptor* tidak dapat bersifat unik sepenuhnya. Terdapat kemungkinan konflik antara *descriptor* dari tiap kata. Oleh karena itu, setiap hasil *hashing* dari suatu kata yang sesuai dengan *descriptor* tertentu perlu diartikan sebagai suatu kemungkinan, bukan suatu kepastian.

Untuk memastikannya, perlu dilakukan *scanning* pada dokumen yang diduga mengandung kata tersebut. Kemungkinan terjadinya konflik dapat dikurangi dengan

menggunakan lebih banyak bit, tetapi proses *scanning* tetap perlu dilakukan untuk memastikan ada atau tidaknya suatu kata.

Dibandingkan dengan inverted file, signature file memiliki dua kekurangan yang memiliki dampak langsung untuk kebutuhan mesin pencari. Yang pertama adalah performa yang lebih lambat dalam hal pengambilan data dari database. Hal ini dikarenakan perlunya melakukan *scanning* untuk memastikan suatu kata ada di dalam suatu dokumen. Yang kedua adalah perlunya proses yang rumit untuk mengolah query yang mengandung disjungsi atau negasi.

Signature file memiliki satu keunggulan dibandingkan inverted file, yaitu tidak membutuhkan penyimpanan daftar kosakata pada memori. Oleh karena itu, di masa lalu signature file lebih umum digunakan karena alasan keterbatasan ruang penyimpanan. Tetapi seiring berjalannya waktu, inverted file mengadopsi teknik kompresi yang membuat keunggulan signature file ini menjadi tidak berlaku lagi.

2.12.2 *Bitmap Index*

Bitmap adalah. Bitmap menggunakan *bitvector* sebagai representasi index. Setiap kata memiliki *bitvector* yang menunjukkan keberadaan kata tersebut dalam berbagai dokumen. Panjang dari *bitvector* tergantung dari jumlah dokumen secara keseluruhan. Konsep bitmap sangat mudah dan cepat untuk digunakan. Tetapi, bentuk struktur *bitvector* membuat kebutuhan ruang penyimpanan menjadi sangat besar.

Karena kekurangannya dalam hal ukuran struktur data, bitmap tidak dapat digunakan pada berbagai kebutuhan dunia nyata, terutama mesin pencari.

2.12.3 *WordNet*

WordNet adalah suatu *synonym ring* yang mampu mengelompokkan kata berdasarkan makna yang ekuivalen secara semantik (sama secara makna) (Gonzalo et al., 1998). *WordNet* dapat digunakan pada proses indexing dengan harapan hasil dari pengelompokan kata yang ditemukan dalam dokumen dapat menjadi lebih baik. Hal ini dapat memberikan peningkatan performa dalam proses pengambilan data.

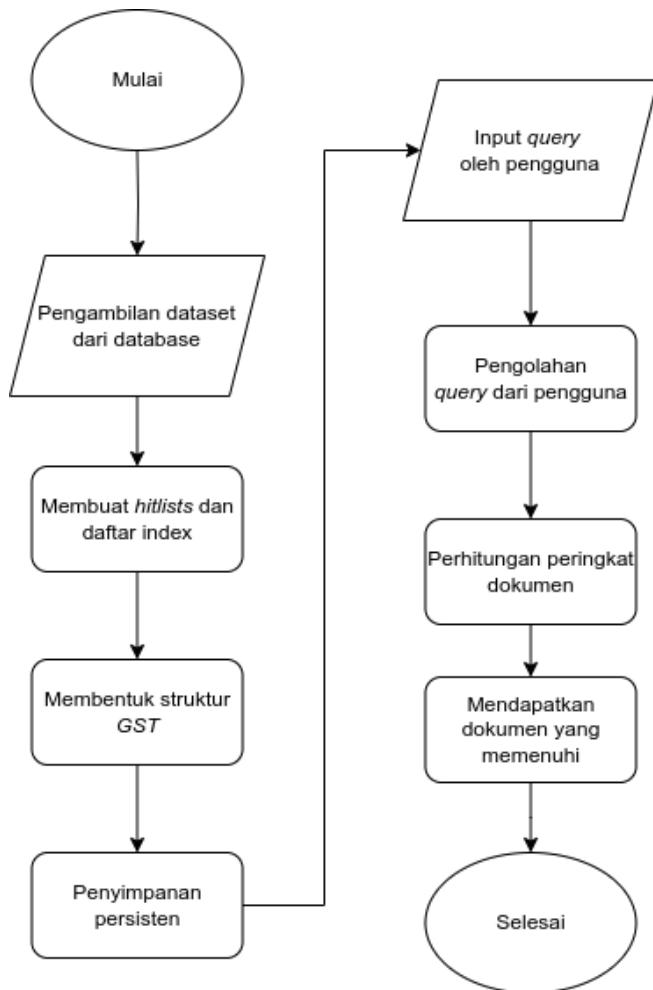
Implementasi index menggunakan *WordNet* dapat digunakan sebagai pembanding untuk metode *inverted file* yang telah dijelaskan sebelumnya.

BAB III

METODOLOGI PENELITIAN

3.1 Tahapan Penelitian

Gambar *flowchart* berikut mengilustrasikan proses pembuatan *inverted index* dari dataset yang ada pada database hingga ke proses pencarian oleh pengguna.



Gambar 3.1: Flowchart tahapan penelitian modul *indexing*

Data utama yang diperlukan oleh modul *indexing* dari database adalah teks yang berisi bagian informasi inti dari halaman tersebut. Umumnya, sebuah halaman web memiliki banyak teks di luar informasi inti yang tidak diperlukan. Oleh karena itu, ditetapkan bahwa modul *indexing* hanya akan mengambil bagian paragraf dari

halaman web. Untuk mengidentifikasi sebuah paragraf, cukup dengan mencari *tag* HTML yang sesuai (*<p>*).

Walaupun sudah menspesifikasikan *tag* tertentu yang akan digunakan, pada praktiknya sering ditemukan penggunaan *tag* paragraf di luar informasi inti halaman web. Selain itu, seringkali ditemukan paragraf yang memiliki beberapa simbol yang tidak diperlukan seperti karakter *newline*. Untuk memastikan bahwa modul *indexing* hanya akan membaca paragraf inti saja, diperlukan *filter* khusus untuk menghindari masuknya informasi yang tidak diperlukan. Parameter dari penentuan informasi ini cukup beragam, dan akan diatur sedemikian rupa pada masa pengujian.

Pada kode program mesin pencari yang telah ada saat ini, paragraf yang disimpan masih memiliki banyak informasi yang tidak diperlukan. Untuk memudahkan implementasi modul *indexing*, maka proses *filtering* akan dilakukan pada kode mesin pencari di bagian ekstraksi informasi dari halaman web. Akan dibuat sebuah table baru yang berisi paragraf inti saja.

3.2 Konstruksi Index

Sebelum proses *indexing* dilakukan, diperlukan sebuah *hash table* pada memori. *Hash table* digunakan dengan tujuan untuk menyimpan kata secara unik dalam dokumen. Jika nantinya ditemukan lebih dari satu kali kemunculan kata dalam dokumen yang sama, *hash table* dapat melakukan penambahan kepada *value* dari kata tersebut, dan membentuk suatu rangkaian seperti *linked list*.

Proses *indexing* dimulai dengan mengambil seluruh daftar paragraf yang tersimpan pada database yang diurutkan berdasarkan *docID*. Kemudian, setiap paragraf akan dipecah berdasarkan karakter spasi menjadi sebuah array berisi kata. Setiap kata lalu disimpan dalam sebuah hit bersama dengan informasi tambahan yang ada pada paragraf.

Algorithm 1 Algoritma pembuatan index (Brin et al., 1998)

```

 $hitData \leftarrow \text{HITDATA}(false)$                                  $\triangleright$  Inisiasi hash table
 $docMap \leftarrow []$ 

 $docs[..] \leftarrow \text{GETDOCS}()$ 
for  $doc \in docs[..]$  do
     $paragraphs[..] \leftarrow \text{GETPARAGRAPHS}(doc)$ 
    for  $paragraph \in paragraphs[..]$  do
         $words[..] \leftarrow \text{SPLITWORDS}(paragraph)$ 

        for  $word \in words[..]$  do
             $\text{STOREHIT}(hitData, word)$ 
        end for

         $\text{DOCMAP}(doc) \leftarrow hitData$ 
    end for
end for

```

Dari proses ekstraksi kosakata beserta data tambahannya, akan didapatkan *forward index* berupa daftar *hit* dari dokumen. Daftar *hit* kemudian didistribusikan ke tempat penyimpanan.

Untuk melakukan konversi *forward index* menjadi *inverted index*, dilakukan proses *sorting* pada *forward index*. *Sorting* dilakukan berdasarkan *id* dari kosakata, kemudian dilanjutkan dengan *id* dari dokumen, dan diakhiri dengan posisi kata pada dokumen. Hasil dari proses *sorting* disimpan kembali ke tempat penyimpanan dengan kondisi sudah menjadi *inverted index*.

Algorithm 2 Operasi *sorting* pada *hitlist* (Brin et al., 1998)

```

function SORTHITLIST( $H$ )
     $sorted \leftarrow false$ 
     $i \leftarrow 0$ 

    do
        if  $i = \text{LEN}(H) - 1$  then
            if  $sorted = true$  then
                 $break$ 
            else
                 $i \leftarrow 0$ 
            end if
        end if

         $tempHit \leftarrow H[i]$ 

        if  $H[i].text > H[i + 1].text$  OR
             $H[i].docID > H[i + 1].docID$  OR
             $H[i].oset > H[i + 1].oset$  then
                 $H[i] \leftarrow H[i + 1]$ 
                 $H[i + 1] \leftarrow tempHit$ 
            end if

         $i \leftarrow i + 1$ 
        while  $sorted = false$ 

        return
    end function

```

3.3 Daftar kata umum

Untuk meningkatkan kualitas hasil pencarian, maka skema pencarian perlu mengabaikan kata-kata yang bersifat umum. Dari hal tersebut, perlu di dapatkan daftar kata umum dari daftar kosakata yang telah dibuat oleh modul *indexing*. Metode yang penulis gunakan untuk mendapatkan kata umum adalah dengan

menggunakan *sorting* terhadap seluruh daftar kosakata berdasarkan tingkat kemunculannya secara global. Kemudian, dengan menggunakan batasan tertentu, akan diambil sebagian persen dari kata terbanyak yang ada.

Sebagai langkah awal, penulis berencana untuk mengambil sebanyak 1% kata teratas sebagai kata umum. Nilai tersebut kemudian akan diatur ulang berdasarkan hasil *user testing*.

3.4 Metode Pemeringkatan

Dari hasil rangkaian *hit* yang didapatkan, perlu dilakukan beberapa operasi untuk mendapatkan urutan hasil dokumen yang paling sesuai. Hasil peringkat yang didapatkan dilakukan berdasarkan dua faktor berikut

- *Word distance* (γ), yaitu jarak kata yang memenuhi query
- *Word similarity* (β), yaitu kemunculan hasil yang memenuhi beberapa bagian dari query

3.4.1 *Word distance* (γ)

Perhitungan *word distance* digunakan untuk melakukan validasi terhadap urutan kata yang muncul pada dokumen. Urutan dari kata berpengaruh terhadap maksud dari query. Selain itu, jarak yang terlalu jauh antara kata yang ditemukan dapat mengurangi relevansi informasi.

Untuk membandingkan jarak kata, seluruh kata pada query perlu diberikan urutan yang sesuai dengan menggunakan angka terlebih dahulu. Setelah *hitlist* didapatkan, posisi dari kata yang ditemukan akan dikurangi dengan posisi asli pada query. Nilai dari dokumen bisa didapatkan dengan rumus

$$\gamma = \left(\sum_{n=1}^N |L_n - L'_n| \right) - (L_1 - L'_1) \times N \quad (3.1)$$

di mana

N = Jumlah kata pada query

L_n = Posisi awal kata ke- n pada query

L'_n = Posisi kata ke- n pada query di dokumen

Ketika urutan kata pada dokumen memiliki nilai $\gamma = 0$, maka dokumen memenuhi kondisi *exact match*. Pada kondisi tersebut, dokumen akan mendapatkan peringkat berdasarkan jumlah kemunculan *exact match*. Sementara jika nilai $\gamma \neq 0$, maka dokumen akan mendapatkan peringkat berdasarkan rumus berikut

$$D = \frac{n}{N} + \left(\frac{n}{N} \times \frac{1}{G} \times K \right) \quad (3.2)$$

di mana

n = Jumlah *match* pada dokumen

N = Jumlah kata pada query

G = Faktor kemunculan terhadap nilai γ

K = Jumlah kemunculan *partial match*

Apabila dalam suatu dokumen, terdapat berbagai kondisi *match* dengan nilai γ yang berbeda-beda, maka akan dipilih kondisi *match* dengan nilai γ paling rendah untuk merepresentasikan nilai dokumen.

3.4.2 Word similarity (β)

Word similarity menilai suatu kata berdasarkan kemiripannya dari kata pada query. Untuk membandingkan apakah suatu kata memiliki makna atau maksud yang sama dengan kata lainnya, akan digunakan *Jaccard distance*. Perbandingan akan dilakukan berdasarkan hasil kombinasi pada seluruh karakter yang ada pada kosakata untuk membuat pasangan dua huruf.

Karena *Jaccard distance* mengukur nilai ketidakmiripan dari dua model, maka tinggi nilai β maka nilai dokumen akan makin rendah. Dari detail tersebut, didapatkan rumus

$$\beta = 1 - \frac{m}{M} \quad (3.3)$$

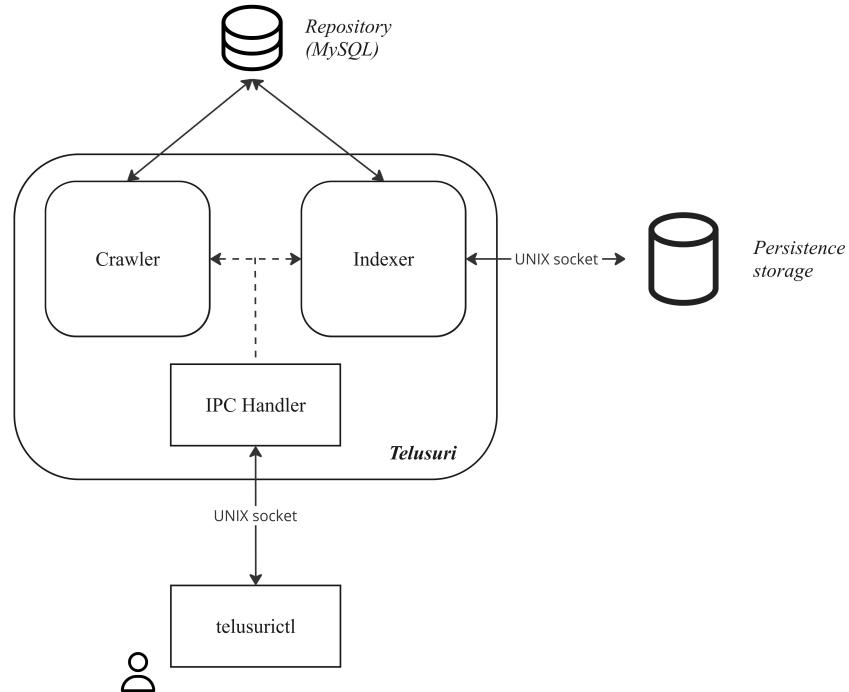
di mana

m = Jumlah pasangan dua huruf berurutan yang ada pada kata

M = Jumlah kemungkinan seluruh pasangan dua huruf berurutan yang mungkin dibentuk

Karena proses perhitungannya yang cukup kompleks, metode pemeringkatan ini hanya digunakan ketika nilai dari *word distance* bernilai kurang dari satu.

3.5 Modifikasi Arsitektur *Telusuri*



Gambar 3.2: Revisi arsitektur *Telusuri*

Dari arsitektur *Telusuri* yang sudah ada saat ini, terdapat beberapa perubahan yang perlu dilakukan untuk mengintegrasikan modul *indexing* ini.

- *Telusuri* dibagi menjadi dua mode *runtime*, yaitu *crawling* dan *indexing*.
- Kontrol terhadap berjalannya *Telusuri* akan dikendalikan oleh aplikasi *command-line*, yaitu *telusurictl*. *telusurictl* akan berkomunikasi dengan *Telusuri* melalui sebuah metode komunikasi antar proses. Untuk metode komunikasi akan digunakan, ditentukan untuk menggunakan *UNIX domain socket* dengan alasan didukung secara langsung oleh sistem operasi yang digunakan (Linux) dan bersifat *language-agnostic* sehingga mudah untuk dikembangkan kedepannya.
- Terdapat modul database primitif yang berperan sebagai *persistent storage* untuk *hitlists* yang telah dibuat oleh modul *indexing*. Modul database ini berjalan sebagai proses terpisah, tetapi tetap dalam satu mesin yang sama. Modul akan memberikan *hitlists* sesuai dengan rentang yang diminta oleh

modul *indexing*. Untuk komunikasi antar modul, akan digunakan *UNIX domain socket*.

3.6 Skema Pencarian

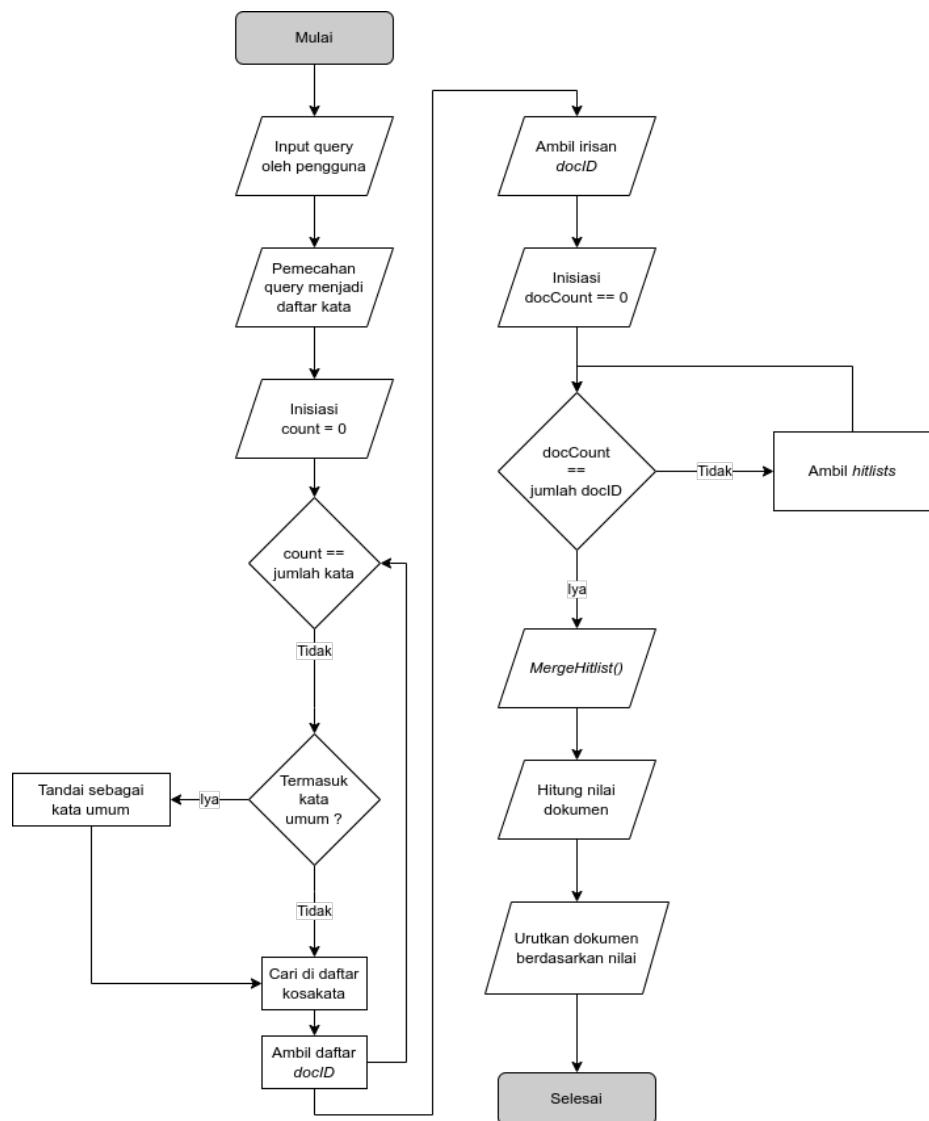
Untuk mendapatkan informasi pencarian berdasarkan input teks dari pengguna, diperlukan beberapa informasi tertentu yang bisa didapatkan dengan cara mengolah informasi yang ada pada teks tersebut. Objek *UserQuery* digunakan untuk mengakomodasi kebutuhan itu.

```
class UserQuery:
    __slots__ = ("pairs", "expectedPos", "documentRank", "globalModifier",
                "rootHitlists", "mergedHitlists")

    def __init__(self) -> None:
        self.pairs: Dict[str, Tuple[WordInfo, HitLists]] = {}
        self.documentRank: Dict[int, float] = defaultdict(lambda: 1.0)
        self.globalModifier: float = 1.0
        self.mergedHitlists: HitLists = []
        self.expectedPos: List[int] = []
        self.rootHitlists: HitLists = []
```

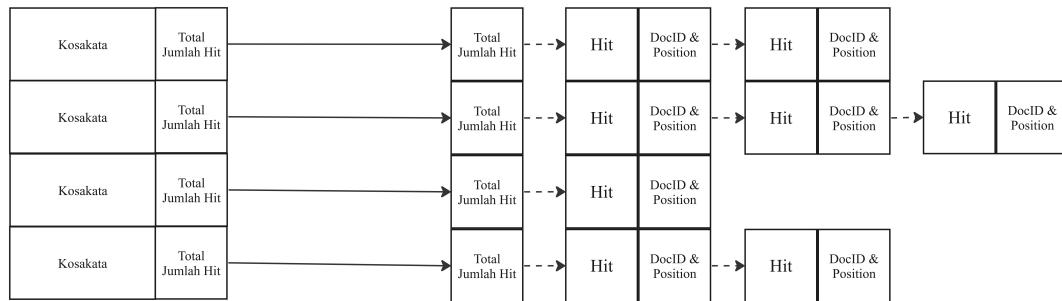
Gambar 3.3: Struktur objek *UserQuery*

Berdasarkan input query yang diberikan oleh pengguna, query dipecah menjadi potongan kata dan ditetapkan beberapa informasi tambahan bedasarkan karakteristik dari masing-masing kata. Informasi tersebut adalah posisi kata dalam teks input, apakah kata tersebut termasuk kata umum dan apakah kata tersebut termasuk kapital. Ketiga informasi ini akan disimpan sebagai *WordInfo* yang ada pada objek *UserQuery*.



Gambar 3.4: Flowchart pengolahan masukan query

Untuk setiap kosakata pada daftar kosakata yang sesuai dengan kosakata pada query, akan di dapatkan rangkaian data berupa *id* dari dokumen tempat kosakata tersebut, total jumlah *hit* pada dokumen serta seluruh daftar *hit* yang berhubungan dengan kosakata pada dokumen.

**Gambar 3.5:** Ilustrasi hubungan struktur data

```

from typing import Dict, Optional
from typing_extensions import Self

class Hit:
    def __init__(self, docID: int, pos: int, next=None) -> None:
        self.docID: int = docID
        self.pos: int = pos
        self.next: Optional[Self] = next

class HitLists:
    def __init__(self, hit: Hit) -> None:
        self.head = hit
        self.count = 1

class HitData:
    table: Dict[str, HitLists]

    def __init__(self, wordID: str, h: HitLists):
        self.table = {wordID: h}

```

Gambar 3.6: Penulisan struktur data pada *Python*

Untuk setiap daftar *hit* yang didapatkan, dilakukan *merging* sehingga tersisa dokumen yang memenuhi query.

Algorithm 3 Operasi *merging* pada seluruh *hitlists* (Brin et al., 1998)

```

function MERGEHITLIST( $p_1, p_2$ )
     $M \leftarrow [..]$ 
    while  $p_1 \neq \text{nil}$  AND  $p_2 \neq \text{nil}$  do
        if  $p_1.docID = p_2.docID$  then                                 $\triangleright$  Cek kesamaan docID
            if  $-(p_1.oset - p_2.oset) = 1$  then                       $\triangleright$  Cek selisih offset
                APPEND( $M, p_1$ )
            end if
        end if
        end if
         $p_1 \leftarrow \text{NEXT}(p_1)$ 
         $p_2 \leftarrow \text{NEXT}(p_2)$ 
    end while
    return  $M$ 
end function

```

```

 $H \leftarrow [..]$ 
 $W \leftarrow \text{PARSEQUERY}(query)$                                  $\triangleright$  Pecah query menjadi daftar kata
for  $w \in W$  do
     $h \leftarrow \text{GETHITLIST}(w)$ 
    if  $h = \text{nil}$  then
        continue
    end if
    APPEND( $H, h$ )
end for

```

```

 $i \leftarrow 0$ 
while  $i \neq \text{LEN}(H)$  do
     $H[0] = \text{MERGEHITLIST}(H[0], H[i])$ 
     $i \leftarrow i + 1$ 
end while

```

result $\leftarrow \text{GETRESULT}(H[0])$

3.7 Alat dan Bahan Penelitian

Pada penelitian ini, terdapat beberapa alat yang digunakan sebagai penunjang dalam pembuatan modul *indexing* dengan rincian sebagai berikut:

- Laptop dengan konfigurasi Intel Core i7-8650U, 32GB RAM)
- Sistem operasi *Linux*
- *Neovim* sebagai *code editor*
- Database *MySQL* versi 8.0.31
- *Podman* untuk menjalankan database *MySQL*
- *Python 3.8*

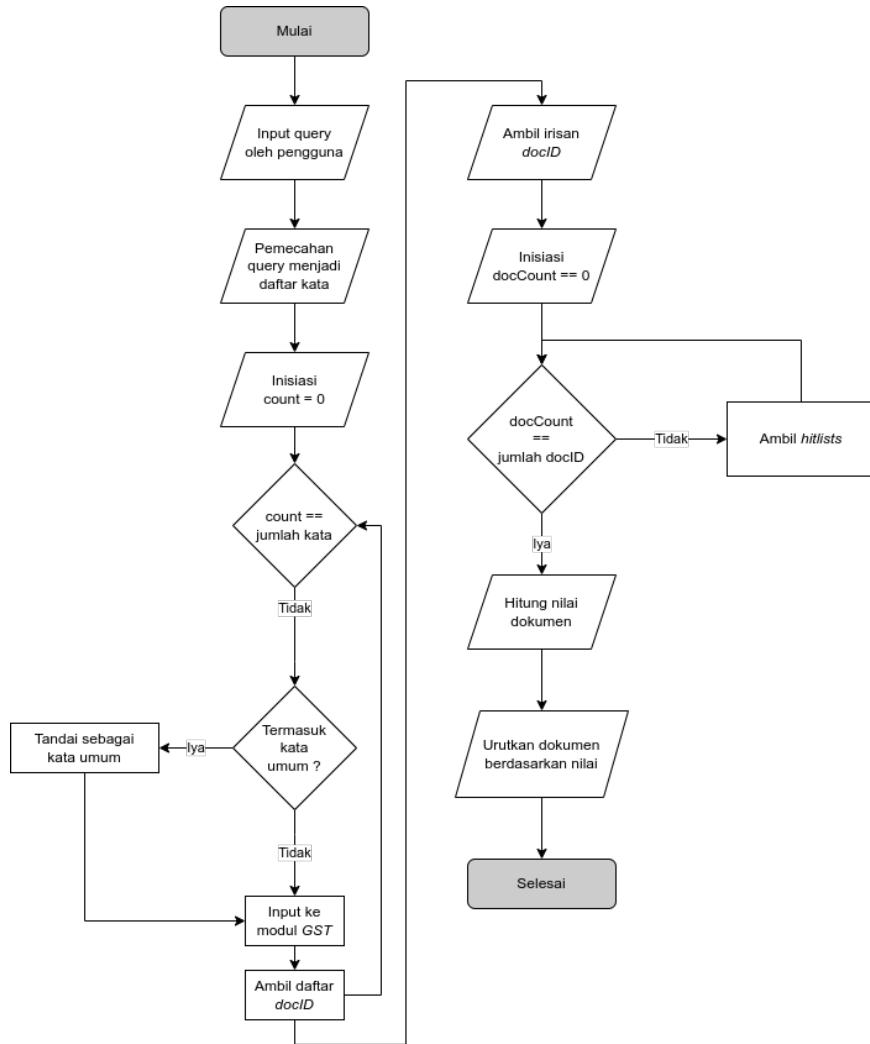
3.8 Tahapan Pengembangan

3.8.1 Meningkatkan kemampuan modul *indexing* saat ini

Pencarian akan dilakukan secara sekuensial terhadap daftar index yang telah dihasilkan oleh modul *indexing*. Sebagai pembanding, modul dari penelitian ini akan digabungkan dengan modul *indexing* yang sudah ada saat ini (Pratama 2023).

Hasil penelitian tersebut menghasilkan modul *indexing* dengan bentuk *Generalized Suffix Tree (GST)*. Penggunaan modul *GST* bertujuan untuk menghindari proses iterasi daftar kosakata yang bisa memakan waktu yang cukup lama. Modul *GST* dapat langsung memberikan hasil berupa dokumen tempat kata tersebut berada. Dengan demikian, modul index pada penelitian ini hanya perlu menunjukkan lokasi kata pada dokumen melalui informasi yang ada pada *hit*. Selain itu, penggunaan *GST* juga dapat melakukan koreksi terhadap adanya *typo* pada query yang diberikan oleh pengguna.

Untuk mendukung integrasi dengan modul *GST*, diperlukan pemetaan dari dokumen ke *hitlists* yang sesuai. Oleh karena itu, akan dilakukan modifikasi pada 1 untuk menambahkan pemetaan dokumen ke *hitlists* yang terbentuk.



Gambar 3.7: Flowchart pengolahan masukan query dengan modul *GST*

Integrasi dilakukan dengan modul *GST* yang telah dibuat pada penelitian sebelumnya, dengan beberapa perbaikan dan penambahan fungsi pembantu untuk mempermudah integrasi.

3.8.2 Rancangan Eksperimen

1. Skenario pertama (Pencarian dengan iterasi daftar kosakata)
 - Pencatatan waktu akan mulai mencatat
 - *Tester* memberikan sebuah query ke dalam mesin pencari
 - Sistem memecah kata berdasarkan spasi

- Sistem melakukan iterasi pada daftar kosakata untuk mencari kata yang sesuai dengan query
- Sistem melakukan pencarian pada kata dengan mencari pada daftar kosakata
- Sistem menggabungkan seluruh hasil pencarian per kata
- Sistem melakukan pemeringkatan hasil
- Sistem mengembalikan daftar dokumen hasil pencarian yang diurutkan berdasarkan hasil pemeringkatan
- Pencatatan waktu berhenti mencatat
- *Tester* memberikan penilaian terhadap relevansi dari hasil pencarian
- Hasil perhitungan waktu akan dibandingkan dengan skenario lain

2. Skenario kedua (Pencarian dengan integrasi modul GST)

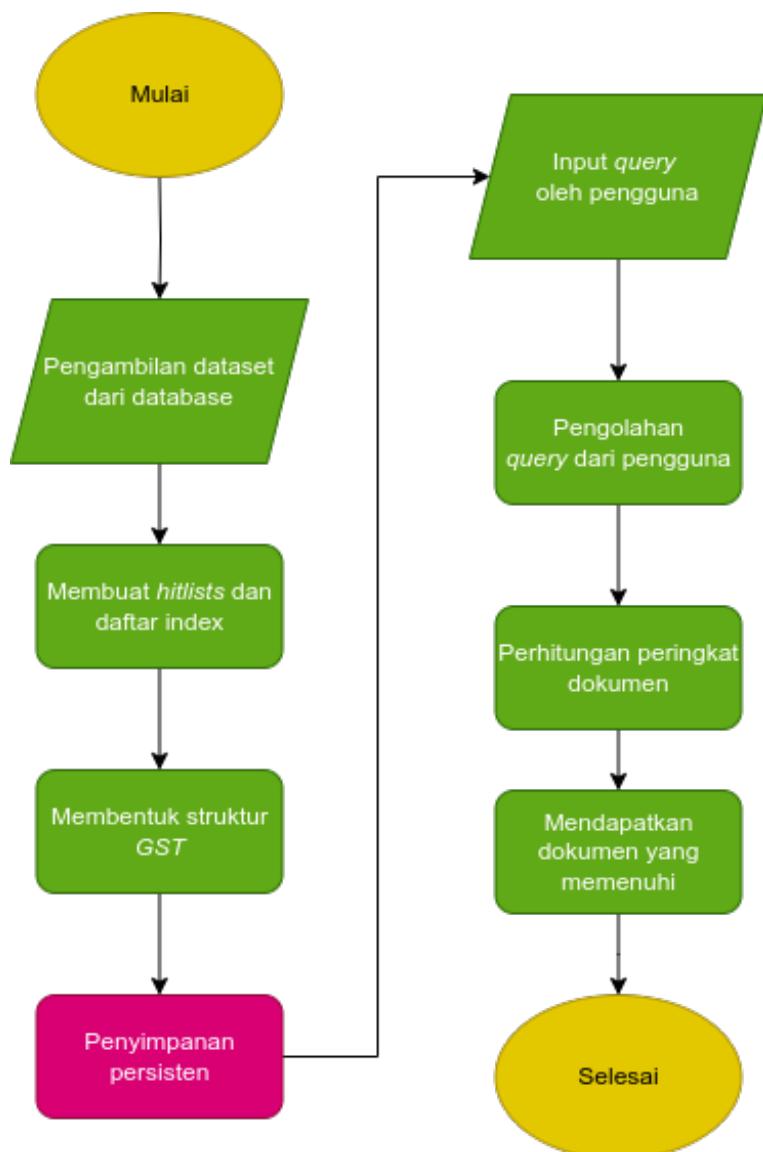
- Pencatatan waktu akan mulai mencatat
- *Tester* memberikan sebuah query ke dalam mesin pencari
- Sistem memecah kata berdasarkan spasi
- Sistem melakukan input kata ke modul *GST* dan mendapatkan output berupa hasil pencarian
- Sistem menggabungkan seluruh hasil pencarian per kata
- Sistem melakukan pemeringkatan hasil
- Sistem mengembalikan daftar dokumen hasil pencarian yang diurutkan berdasarkan hasil pemeringkatan
- Pencatatan waktu berhenti mencatat
- *Tester* memberikan penilaian terhadap relevansi dari hasil pencarian
- Hasil perhitungan waktu akan dibandingkan dengan skenario lain

BAB IV

HASIL DAN PEMBAHASAN

4.1 Implementasi

Berikut adalah keterangan *flowchart* tentang apa saja yang telah berhasil diimplementasikan. Warna hijau menandakan implementasi telah berhasil dibuat, sementara warna merah menandakan implementasi belum sempurna.



Gambar 4.1: *Flowchart* tahapan penelitian yang sudah jadi

Untuk penyimpanan persisten saat ini berjalan, tetapi tidak secara *concurrent* karena keterbatasan kemampuan penulis.

4.1.1 Pengolahan Dataset

Sebelum menjalankan modul *indexer*, dataset perlu dikumpulkan terlebih dahulu. Proses pengumpulan dataset dilakukan dengan menggunakan modul *crawler* dari hasil penelitian sebelumnya. Titik pengumpulan dataset adalah situs *Forum News Network (fnn.co.id)*, dengan durasi *crawling* selama 48 jam. Modul *crawler* dijalankan pada hari Jum'at, 23 Juni 2023 dan selesai pada hari Minggu, 25 Juni 2023.

Pada database, terdapat dua tabel yang akan digunakan dalam modul *indexer*. Pertama adalah tabel *page_paragraph* yang berisi seluruh teks dalam *tag* paragraf pada halaman web. Data dari tabel ini akan digunakan untuk membentuk struktur index yang akan digunakan dalam proses pencarian data. Yang kedua adalah tabel *page_information* yang berisi informasi tentang judul dan *URL* dari halaman web. Data dari tabel ini akan digunakan untuk mendapatkan detail dari halaman setelah proses kalkulasi skor tiap dokumen.

Table Name	Engine	Auto Increment	Data Length
crawling	InnoDB	1	16K
page_forms	InnoDB	84,604	24M
page_images	InnoDB	1,226,298	134M
page_information	InnoDB	42,655	658M
page_linking	InnoDB	5,489,314	456M
page_list	InnoDB	2,315,955	333M
page_paragraph	InnoDB	1,010,083	505M
page_scripts	InnoDB	343,974	214M
page_styles	InnoDB	81,926	149M
page_tables	InnoDB	82	144K
pagerank	InnoDB	0	16K
pagerank_changes	InnoDB	0	16K
tfidf	InnoDB	0	16K
tfidf_word	InnoDB	0	16K

Gambar 4.2: Statistik dataset hasil *crawling*

	123 id_list	123 page_id	abc paragraph
62		62	6 ↗ Jakarta, FNN – Empat mantan ketua Himpunan Mahasiswa
63		63	6 ↗ Bandung, FNN – Idham Maulana, alumni Teknik Geologi
64		64	6 ↗ Jakarta, FNN – Menteri Pendidikan, Kebudayaan, Riset, dan
65		65	6 ↗ Jakarta, FNN - Akademisi Kaukus Indonesia untuk Keber
66		66	6 ↗ Bandung, FNN - Ketua Umum Ikatan Alumni Universitas
67		67	6 ↗ Denpasar, FNN - Analis Sosial Politik UNJ Ubedilah Badr
68		68	6 ↗ Denpasar, FNN – Musyawarah Nasional (Munas) Asosiasi
69		69	6 ↗ Denpasar, FNN – Musyawarah Nasional (Munas) Asosiasi
70		70	6 ↗ Bali, FNN - Asosiasi Program Studi Pendidikan Sosiologi
71		71	6 ↗ Forum News Network (FNN.co.id) adalah Portal Berita L
72		72	7 ↗ AKP menjadi kuat seperti saat ini setelah melalui perjalan
73		73	7 ↗ Adu kuat pengeras suara telah menjadi budaya baru kor
74		74	7 ↗ Raja Mohammed VI menggunakan kaum sufi untuk men
75		75	7 ↗ Di bawah AKP dan Erdogan Islam bangkit kembali di Tur
76		76	7 ↗ Jakarta, FNN – Dewan Pimpinan Pusat Asosiasi Serikat
77		77	7 ↗ Film India yang baru-baru ini dirilis, The Kerala Story, m
78		78	7 ↗ Jakarta, FNN - Sejarah baru kembali terungkap. Untuk p
79		79	7 ↗ Oleh Muhammad Chirzin - Guru Besar UIN Sunan Kalijaga
80		80	7 ↗ Oleh Pierre Suteki - Guru Besar Hukum Undip PA. Penga

Gambar 4.3: Informasi pada tabel *page_paragraph*

4.1.2 Pembentukan *Inverted Index*

Fungsi *getRepoDump* mengambil dataset dari database dengan menggunakan objek *Database* yang sudah diinisiasi sebelumnya. Kemudian didapatkan sebuah *list* berisi pasangan *id* dari dokumen dan paragraf yang ditemukan pada dokumen tersebut.

```
def getRepositoryDump(self) -> List[Dict[str, Union[int, str]]]:
    print("Getting data from database...")
    start = time.perf_counter()
    conn: pymysql.Connection[
        pymysql.cursors.Cursor] = self.db.connect() #type: ignore

    try:
        with conn.cursor(cursor=pymysql.cursors.DictCursor) as cursor:
            cursor.execute("SELECT page_id, paragraph FROM page_paragraph")
            result = cursor.fetchall()
    finally:
        conn.close()

    end = time.perf_counter()
    print(f"Time elapsed getting data from database: {end - start:.4f}s")
    return result #type: ignore
```

Gambar 4.4: Pengambilan dataset dari database

Setelah dataset didapatkan, fungsi *generateIndex* akan membuat struktur

index berdasarkan data tersebut. Untuk setiap pasangan pada dataset, paragraf akan dikumpulkan berdasarkan *id* dari dokumen. Setelah pemetaan antara dokumen dan paragraf selesai dibuat, maka untuk setiap dokumen daftar paragraf akan diolah oleh fungsi *generateHitlists*.

```
def generateIndex(self, data: List[Dict[str, Union[int, str]]]):
    start = time.perf_counter()
    print("Generating hitlists...")
    docMap: Dict[int, List[str]] = {}
    curDoc = int(data[0]["page_id"])
    docMap[curDoc] = []
    wordDocCount: Dict[int, int] = {}

    # Gather all paragraph as a single list value per word
    for pair in data:
        if pair["page_id"] != curDoc:
            curDoc = int(pair["page_id"])
            docMap[curDoc] = []

        docMap[curDoc].append(str(pair["paragraph"]))

    for docID, paragraphs in docMap.items():
        wordDocCount[docID] = self.generateHitlists(docID, paragraphs)

    # Generate document blacklist
    self.__generateDocumentBlacklist(wordDocCount)

    # Generate list of common words
    self.__generateCommonLists()

    end = time.perf_counter()
    print(f"Time elapsed creating indexes: {end - start:.4f}s")
```

Gambar 4.5: Pembuatan struktur index dari dataset

Pada fungsi *generateHitlists*, setiap paragraf akan dibersihkan terlebih dahulu sehingga hanya tersisa karakter alfanumerik yang dipisahkan oleh spasi. Paragraf yang sudah dibersihkan kemudian akan dibagi berdasarkan spasi menjadi sebuah daftar kata. Kemudian, untuk setiap kata akan disimpan informasi tentang *id* dari dokumen saat ini, posisi kata pada dokumen, dan apakah kata tersebut termasuk kapital (singkatan dari sebuah istilah).

```

def generateHitlists(self, docID: int, paragraphs: List[str]) -> int:
    sdocID = bitarray(bin(docID)[2:].zfill(19))
    wordCount = 1
    totalCount = 0
    for paragraph in paragraphs:
        paragraph = sub(r'\W+', ' ', paragraph)
        for char in FILTERED_CHAR:
            paragraph = paragraph.replace(char, ' ')
        words = str(paragraph).split()

        # Strip empty or single-character word
        for word in words:
            # Skip word longer than 30 characters
            # Honestly, if there is an abnormal condition where there are
            # word longer than 30 chars, then there's a problem with filtering
            if len(word) > 30:
                continue

            # Word limit
            if wordCount > 4094:
                wordCount = 4095

            if word == "" or len(word) == 1:
                continue

            isCapital = bool(match(CAPITAL_PATTERN, word))
            if not isCapital:
                word = word.lower()

            # Merge docID, word offset and capital information
            hit = ba2int(sdocID + bitarray(bin(wordCount)[2:]).zfill(12)) +
                  bitarray(bin(int(isCapital))[2:]).zfill(1))
            self.pairs[word].append(hit)

            wordCount += 1
            totalCount += 1

    return totalCount

```

Gambar 4.6: Pengolahan paragraf menjadi *hitlist*

Fungsi *storeIndex* digunakan untuk menyimpan struktur index pada sebuah file persisten dengan nama *telusuri_store.pkl*. File ini nantinya dapat langsung digunakan kembali ketika modul tidak menggunakan konfigurasi *reindex*.

```

def storeIndex(self):
    print("Storing indexes...")
    barrelSize = int(len(self.pairs) / 64)
    print(f"Barrel size: {barrelSize}")
    maxedOut = True
    firstWord = ""
    tempBarrel = Barrel()
    for k, v in sorted(self.pairs.items(), key=lambda x: x[0]):
        if maxedOut:
            firstWord = k
            maxedOut = False
            tempBarrel = Barrel()

        tempBarrel.pairs[k] = v

        if len(tempBarrel.pairs) == barrelSize:
            self.persistence[firstWord] = tempBarrel
            maxedOut = True

    print("Done storing indexes")
    self.persistence.sync()

```

Gambar 4.7: Penyimpanan struktur index ke file persisten

4.1.3 Pengolahan *User Query*

Setelah menerima input *query* dari pengguna, teks akan diolah pada fungsi *getInputPairs*. Fungsi tersebut akan memecah teks input dari pengguna berdasarkan spasi, menjadi sebuah daftar kata yang berurutan. Untuk setiap kata pada teks input, akan diolah informasi relatif terhadap teks input. Informasi tersebut adalah posisi kata pada teks input, apakah kata tersebut masuk dalam kategori kata umum dan apakah kata tersebut adalah kata kapital. Seluruh informasi disimpan dalam sebuah objek *UserQuery*, yang nantinya akan menyimpan seluruh informasi yang telah diolah berdasarkan input pengguna.

```
def search(self, input: str) -> Dict[int, Tuple[int, float, str, str]]:
    query = UserQuery()
    infoPairs = self.__parseInput(input)
    res: Dict[int, Tuple[int, float, str, str]] = {}

    try:
        self.__getInputPairs(query, infoPairs)

        query.generateExpectedPos()
        query.processQuery()
        query.calculateRanking()

        # Filter docID result based on document blacklists
        self.filterQuery(query)

        res = self.__getDocuments(query)
        prettyPrint(res)
    except Exception as e:
        print(f"Error on intermediate process: {e}")

    return res
```

Gambar 4.8: Fungsi pencarian utama

```

def __getInputPairs(self, query: UserQuery, infoPairs: Dict[str,
                                                               WordInfo]):
    for word in infoPairs.keys():
        # Skip storing hitlists if it's a common word
        if not infoPairs[word][1]:
            # Check if word is exist in lexicon
            try:
                query.pairs[word] = (infoPairs[word], self.pairs[word])
            except KeyError:
                # If capital, check if lowercase version exist
                if infoPairs[word][2]:
                    lowerVer = word.lower()
                    if lowerVer in self.pairs.keys():
                        query.pairs[lowerVer] = (infoPairs[word],
                                                 self.pairs[lowerVer])
                    continue

        # Find semantically nearest word with Jaccard index
        bestMatch = self.rankSimilarity(word)
        if len(bestMatch) == 0:
            query.pairs[word] = (infoPairs[word], [])
        else:
            # Use word with highest similarity
            for m in bestMatch:
                if m[0] in self.commonwords:
                    continue

                query.pairs[m[0]] = ((infoPairs[word][0], False,
                                      bool(
                                          match(
                                              CAPITAL_PATTERN,
                                              m[0]))),
                                      self.pairs[m[0]])
                break

    else:
        query.pairs[word] = (infoPairs[word], [])

```

Gambar 4.9: Pengolahan teks input dari pengguna

Dalam proses pengolahan teks input, untuk setiap kata yang bersifat umum maka *hitlist* tidak akan diambil untuk digabungkan karena alasan efisiensi. Selain itu, untuk kata yang bersifat kapital, apabila kata tersebut tidak ada pada daftar kosakata, maka akan dicoba untuk didapatkan varian non-kapitalnya terlebih dahulu. Seluruh kata yang sudah didapatkan *hitlist*-nya, posisi relatifnya akan dimasukkan ke dalam variabel *expectedPos* untuk dibandingkan ketika proses pemeringkatan. Proses ini

dilakukan pada fungsi *generateExpectedPos*.

```
def generateExpectedPos(self):
    data = list(sorted(self.pairs.values(), key=lambda x: x[0][0]))
    for i in data:
        # If common words, skip
        if not i[0][1]:
            self.expectedPos.append(0)
```

Gambar 4.10: Fungsi untuk menyimpan posisi yang sesuai dengan teks input berdasarkan urutan

Setelah *hitlist* didapatkan pada masing-masing kata, seluruh data akan digabungkan menjadi satu *hitlist* lengkap, dan kemudian diurutkan dari nilai terkecil.

4.1.4 Proses Penggabungan *Hitlist* dan Pemeringkatan Hasil

Pada fungsi *calculateRanking*, akan dilakukan iterasi pada seluruh *hitlist* untuk menghitung nilai per dokumen. Untuk setiap iterasi, setiap posisi akan disimpan dalam variabel *curIter*, dan akan dicek apakah isi dari variabel tersebut telah sesuai dengan *expectedPos*. Apabila iterasi telah pindah ke dokumen lain sebelum seluruh kata terkumpul, maka dokumen sebelumnya akan masuk dalam golongan *substring match*. Selain jenis *match*, jumlah kemunculan *match* tersebut juga akan disimpan, dan dihitung ketika terjadi perpindahan dokumen pada iterasi.

```

for info in self.mergedHitlists:
    docID = getDocID(info)
    pos = getPosition(info)

    if info in self.rootHitlists:
        # If len are the same, chances are not a partial match
        if len(curIter) == len(self.expectedPos):
            # Normalize curIter
            diff = curIter[0] - self.expectedPos[0]
            for i, x in enumerate(curIter):
                curIter[i] = x - diff

            # Compare to expectedPos
            if curIter == self.expectedPos:
                exactCount += 1
        # If different len, then its a partial match
        else:
            subScore = len(curIter) / len(self.expectedPos)
            try:
                subMatch[subScore] += 1
            except KeyError:
                subMatch[subScore] = 1

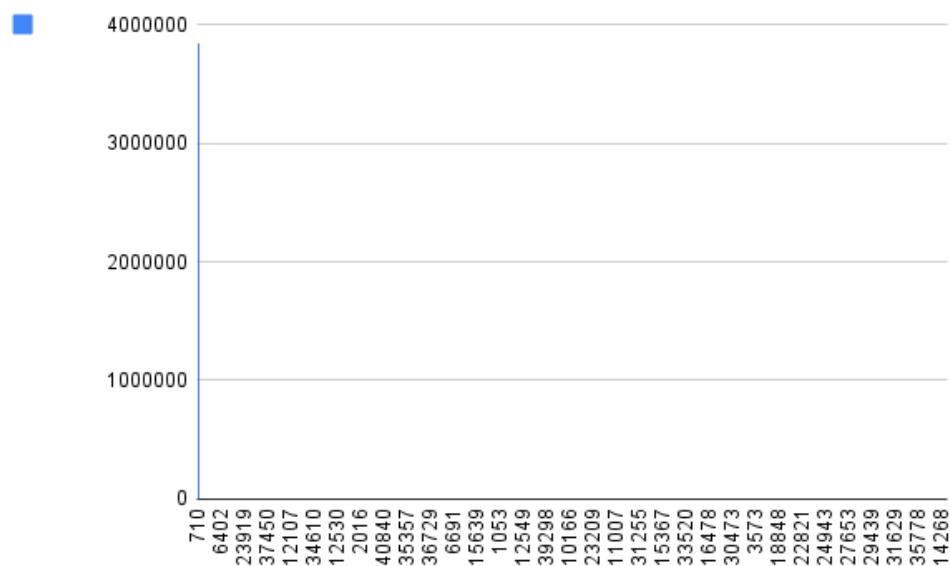
            curIter.clear()
            curIter.append(pos)
    elif curDoc != docID:
        # Exact match exist
        if exactCount > 0:
            self.documentRank[
                curDoc] = exactCount * self.globalModifier
    elif len(subMatch) > 0:
        # For submatch, get the highest submatch occurrence and
        # calculate the result with the occurrence
        for v in sorted(subMatch.keys(), reverse=True):
            self.documentRank[curDoc] = (
                v + (v / 3 * subMatch[v])) * self.globalModifier
        break

    # Renew docID, reset context
    subMatch.clear()
    curIter.clear()
    curDoc = docID
    curIter.append(pos)
else:
    curIter.append(pos)

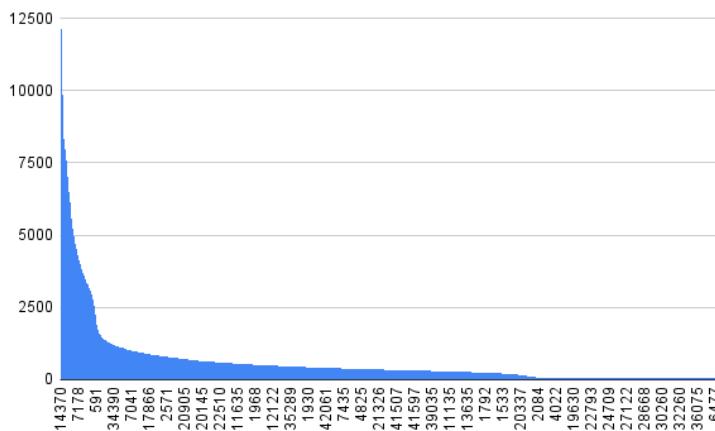
```

Gambar 4.11: Proses kalkulasi peringkat

Setelah proses pemeringkatan selesai, dokumen akan disaring terlebih dahulu berdasarkan variabel *documentBlacklist* melalui fungsi *filterQuery*. Hasil yang telah disaring akan digunakan untuk mendapatkan data halaman web dari database. Melalui fungsi *getDocuments*, akan diambil seluruh data yang mengandung nilai dari seluruh *id* dokumen. Kemudian, data akan dicetak berdasarkan peringkat dokumen tertinggi.



Gambar 4.12: Grafik jumlah kata dalam dokumen



Gambar 4.13: Grafik jumlah kata dalam dokumen setelah menyingkirkan 20 entri tertinggi

```

def filterQuery(self, query: UserQuery):
    filteredDoc: List[int] = []
    for k in query.documentRank.keys():
        if k in self.doc_blacklist:
            filteredDoc.append(k)

    for doc in filteredDoc:
        del query.documentRank[doc]

```

Gambar 4.14: Penyaringan terhadap dokumen yang telah di-*blacklist*

```

# Get documents from database
def __getDocuments(self, query: UserQuery):
    if len(query.documentRank) == 0:
        raise IndexError(
            "Unable to get documents because document ranking mapping is empty"
        )
    docs: List[int] = []
    for data in sorted(query.documentRank.items(), key=lambda x: x[1]):
        docs.append(data[0])

    conn: pymysql.Connection[
        pymysql.cursors.Cursor] = self.db.connect() #type: ignore
    queryStr = f"SELECT id_page, title, url FROM page_information WHERE id_page IN {tuple(docs)}"
    print(queryStr)

    try:
        with conn.cursor(cursor=pymysql.cursors.DictCursor) as cursor:
            cursor.execute(queryStr)
            result = cursor.fetchall()
    finally:
        conn.close()

    resDict: Dict[int, Tuple[int, float, str, str]] = {}
    for r in result:
        resDict[r["id_page"]] = (r["id_page"],
                                 query.documentRank[r["id_page"]],
                                 r["title"], r["url"])

    d: Dict[int, Tuple[int, float, str, str]] = {}
    for data in sorted(query.documentRank.items(),
                       key=lambda x: x[1],
                       reverse=True):
        d[data[0]] = resDict[data[0]]

    return d

```

Gambar 4.15: Pengambilan dokumen dari database dan pemetaan informasi dokumen berdasarkan skor

4.1.5 Integrasi dengan *Generalized Suffix Tree (GST)*

Implementasi *GST* yang dipilih adalah implementasi milik Zaidan yang telah dibuat sebelumnya. Tetapi karena kondisi kode yang tidak bisa langsung berjalan, diperlukan beberapa perbaikan pada kode. Bagian kode yang perlu diperbaiki adalah penggunaan iterasi yang lebih *proper* dan beberapa penambahan operasi kondisional untuk menangani kesalahan pada data.

Perubahan pertama adalah pada kondisi yang digunakan untuk iterasi ketika mengambil informasi dari database. Ditambahkan operasi kondisional untuk mengecek apakah kolom *title* tidak bernilai *NULL* dan memiliki panjang lebih dari 0.

```

25     for data in result:
26         # menjadikan title lower case untuk dimasukkan ke tree
27         if data["title"] is not None and len(data["title"]) > 0:
28             data["title"] = data["title"].lower()
29             # cleaning title dari simbol untuk input tree
30             data["title"] = re.sub('[^A-Za-z0-9 ]+', " ", data["title"])
31             # print(data)
32             data["title"] = data["title"].lower()
33             # cleaning title dari simbol untuk input tree
34             data["title"] = re.sub('[^A-Za-z0-9 ]+', " ", data["title"])
35
36     return result

```

Gambar 4.16: Perubahan pada iterasi ketika mengambil data dari database

Perubahan kedua adalah pada kondisi yang digunakan untuk interasi ketika membuat struktur *tree*. Ditambahkan operasi kondisional untuk memastikan bahwa nilai dari *title* tidak kosong.

```

154     for title in data:
155         # pemisahan setiap kata pada title untuk diproses
156         if title["title"] is not None and len(title["title"]) > 0:
157             for word in title["title"].split():
158                 wordIndex = title["id_page"]
159                 # penambahan terminal node untuk pembentukan GST
160                 word += "$"
161                 # proses memasukkan tiap sufiks dari kata pada title
162                 for i in range(len(word)):
163                     suf = word[i:]
164                     addChild(suf=suf,
165                             parent="root",
166                             tree=root,
167                             index=wordIndex)
168
169             for word in title["title"].split():
170                 wordIndex = title["id_page"]
171                 # penambahan terminal node untuk pembentukan GST
172                 word += "$"
173                 # proses memasukkan tiap sufiks dari kata pada title
174                 for i in range(len(word)):
175                     suf = word[i:]
176                     addChild(suf=suf, parent="root", tree=root, index=wordIndex)
177
178     return root

```

Gambar 4.17: Perubahan pada iterasi ketika membentuk struktur *tree*

Setelah implementasi diperbaiki dan dikonfirmasi dapat berjalan melalui pengujian terpisah, dilakukan *refactoring* pada kode untuk memudahkan integrasi. *Class GST* dibuat, dan ditambahkan beberapa fungsi untuk memudahkan integrasi. Modul *indexer* hanya perlu melakukan impor modul untuk dapat menggunakannya.

```
class GST:

    __slots__ = ("db", "tree")

    def __init__(self, db: Database) -> None:
        self.db = db
        self.tree = Node("root")

    def generateTree(self):
        start = time.perf_counter()
        db = self.getTitle()
        self.tree = self.makeTree(db)
        end = time.perf_counter()
        print(f"Time elapsed generating GST: {end - start:.04f}s")

    def findTree(self, input: str) -> List[Tuple[int, int]]:
        (_, res) = self.searchTree(input)
        ranked = self.rankResult(res)
        result: List[Tuple[int, int]] = []
        for d in ranked:
            try:
                result.append((d["index"], d["count"]))
            except KeyError:
                result = [(d["index"], d["count"])]
        return result
```

Gambar 4.18: Implementasi *class GST* untuk memudahkan integrasi

Selain itu, agar lebih mudah dalam proses pengujian, implementasi diberikan sebuah *guard* kondisional agar dapat dinyalakan / dimatikan dengan menggunakan *environment variable*.

Untuk mengakomodasi implementasi *GST*, terdapat beberapa penambahan kode. Yang pertama adalah pemetaan antara dokumen dengan *hitlists*. Untuk itu dibuat variabel *documentPairs*. Variabel ini akan menambahkan *hit* dalam fungsi *generateIndex*, tetapi akan ditambahkan per dokumen.

```

if useGST == "true":
    self.useGST = True
    self.gst = GST(self.db)
    self.documentPairs: Dict[int, HitLists] = defaultdict(
        list) # Array / standard list

    if barrelMode == "local" and status == "reindex":
        # Remove file if reindexing
        if os.path.exists(PERSISTENT_DOCPAIRS_FILE):
            os.remove(PERSISTENT_DOCPAIRS_FILE)
        if os.path.exists(GST_FILE):
            os.remove(GST_FILE)
        self.documentPersistence = open(PERSISTENT_DOCPAIRS_FILE, 'ab')
        self.treePersistence = open(GST_FILE, 'ab')
    elif status == "search":
        self.documentPersistence = open(PERSISTENT_DOCPAIRS_FILE, 'rb')
        self.treePersistence = open(GST_FILE, 'rb')
    else:
        self.useGST = False

```

Gambar 4.19: Penambahan inisiasi variabel untuk akomodasi penggunaan *GST*

Selanjutnya, pada proses pengolahan *query*, dibuat sebuah fungsi baru yaitu *calculateRankingGST*. Karena bentuk input yang berupa daftar *hitlists* berdasarkan pemetaan dokumen, maka proses perhitungan peringkat agak berubah. Proses perhitungan menghindari iterasi secara menyeluruh dari *hitlist* dengan cara mengambil irisan dari *hit* yang sesuai dengan kata dan dokumen. Dari hasil irisan yang sesuai, barulah perhitungan peringkat dimulai.

```

for doc in self.docHitlists.keys():
    exactCount = 0
    pos: List[int] = []
    subMatch: Dict[float, int] = {}
    for i in temp:
        t = list(set(self.docHitlists[doc]).intersection(i))
        if len(t) > 0:
            pos.extend(t)

    if len(pos) == 0:
        continue

    # Extra sort to make sure
    pos.sort()

    if len(pos) >= len(self.expectedPos):
        marked = list(set(pos).intersection(self.rootHitlists))

        curIter: List[int] = []
        maxLen = len(pos)-1
        for idx, p in enumerate(pos):
            if p in marked or idx == maxLen:
                if idx == maxLen:
                    curIter.append(p)

                if len(curIter) > 0:
                    # Calculate
                    for i, item in enumerate(curIter):
                        curIter[i] = getPosition(item)

                    # Normalize
                    diff = curIter[0] - self.expectedPos[0]
                    for j, _ in enumerate(curIter):
                        curIter[j] -= diff

                    if curIter == self.expectedPos:
                        exactCount += 1
                    # If different len, then its a partial match
                    else:
                        subScore = len(pos) / len(self.expectedPos)
                        try:
                            subMatch[subScore] += 1
                        except KeyError:
                            subMatch[subScore] = 1

            # Sum up calculation
            if idx == maxLen:
                if exactCount > 0:
                    self.documentRank[doc] = exactCount * self.globalModifier
                else:
                    # For submatch, get the highest submatch occurrence
                    # and calculate the result with the occurrence
                    maxSubScore = max(subMatch.keys())
                    self.documentRank[doc] = (
                        maxSubScore +
                        (maxSubScore / PARTIAL_MATCH_OCCUR_FACTOR *

```

Gambar 4.20: Perubahan implementasi perhitungan peringkat untuk integrasi *GST*

Yang terakhir adalah penambahan fungsi untuk mengambil *hitlist* dari pemetaan dokumen. Fungsi *getDocumentPairs* dimulai dengan melakukan input kata ke modul *GST*. Untuk setiap kata, modul *GST* akan mengembalikan daftar dokumen yang memiliki kata tersebut beserta jumlahnya. Dokumen kemudian akan diurutkan berdasarkan jumlah kata, dan untuk setiap dokumen akan diambil *hitlist* yang memenuhi.

```

def __getDocumentPairs(self, query: UserQuery):
    docPairs: Dict[str, List[Tuple[int, int]]] = defaultdict(
        list) # For (document, count)
    docList: Dict[str, List[int]] = defaultdict(list) # For document only
    intersectPairs: Dict[float, List[int]] = defaultdict(list)
    # TODO Confirm if common word is already filtered
    for word in query.wordPairs.keys():
        # Assumptions are there will always be a result
        # since the word is found in lexicon
        docPairs[word] = self.gst.findTree(word)

    # Sort by highest count
    for val in docPairs.values():
        val.sort(key=lambda x: x[1])

    # Extract doc lists
    for k, v in docPairs.items():
        docList[k] = [d[0] for d in v]

    # Get intersection
    if len(docList) > 1:
        for iter in range(1, len(docList.keys())):
            for pair in combinations(docList.keys(), iter):
                for p in pair:
                    intersectPairs[iter / len(docList.keys())] = list(
                        set(intersectPairs[iter / len(
                            docList.keys())]).intersection(docList[p]))

    # Get all docID-mapped hitlists
    storeDoc: List[int] = []
    for d in docList.values():
        storeDoc.extend(d)
    for doc in set(storeDoc):
        # Filter document blacklist
        if doc not in self.documentBlacklist:
            query.docHitlists[doc] = self.documentPairs[doc]

```

Gambar 4.21: Fungsi untuk mengambil dokumen berdasarkan hasil dari *GST*

4.1.6 Modifikasi kode *TFIDF*

Metode *TFIDF* digunakan sebagai patokan untuk memverifikasi kesesuaian kata pada hasil pencarian. Kode yang berdasarkan implementasi pemeringkatan dokumen pada penelitian sebelumnya. Kode tersebut tidak bisa dijalankan secara langsung karena penggunaan *dataset* yang besar membuat proses skoring kata dalam dokumen terlalu lama.

Modifikasi yang dilakukan adalah dengan membagi operasi menjadi beberapa bagian, dan menyimpan hasil dari setiap operasi kedalam file persisten untuk menghindari melakukan proses ulang data.

Operasi pertama yang dilakukan adalah melakukan penyaringan terhadap dataset yang ada. Agar metode *TF-IDF* dapat bekerja dengan baik, maka dataset perlu dibersihkan dari kata sambung, atau disisakan kata dasar saja. Metode yang dipilih adalah penyaringan untuk menyisakan kata dasar saja. Untuk mengetahui

apakah sebuah kata termasuk kata dasar, diperlukan sebuah daftar yang berisi kata dasar untuk digunakan sebagai patokan. Daftar kata yang digunakan diambil dari proyek *Sastrawi* (Librian 2023).

```

db_connection = Database().connect()

# Ambil semua data halaman yang sudah di crawl ke dalam pandas dataframe
query = "SELECT * FROM `page_information`"
df = pd.read_sql(query, db_connection)
text_content = df[
    "content_text"] # Konten teks dari halaman yang sudah dicrawl

# Buat model menggunakan TfIdfVectorizer
vectorizer = TfIdfVectorizer(
    lowercase=True, # Untuk konversi ke lower case
    use_idf=True, # Untuk memakai idf
    norm="l2", # Normalisasi
    smooth_idf=True, # Untuk mencegah divide-by-zero errors
)

f = open('filterWithCount.pkl', 'wb')
alphaFile = open('alphaMap.pkl', 'rb')
alphaDict = pickle.load(alphaFile)
filterWithCount: Dict[int, str] = {}
alphaList = alphaDict.keys()

for idx, content in enumerate(text_content):
    print(f"Processing document {idx+1}/{len(text_content)}")
    tempList: List[str] = []
    if content is not None and len(content) > 0:
        lowered = content.lower()
        normalized = normalize('NFKC', lowered)
        cleaned = re.sub('\W+', ' ', normalized)
        split = cleaned.split()

        for word in split:
            if len(word) > 0:
                if word[0] in alphaList and word in alphaDict[word[0]]:
                    tempList.append(word)

    print(tempList)
    filterWithCount[idx] = " ".join(tempList)
print("Processing document...DONE")

```

Gambar 4.22: Operasi filter pada seluruh dataset dari *table_information* untuk menyisakan seluruh kemunculan kata dasar pada tiap dokumen

Pada percobaan awal, proses filter secara langsung masih memakan waktu yang cukup signifikan karena daftar kata dasar yang cukup besar (29932 kata). Oleh karena itu, dibuat pemetaan antara huruf pertama dari kata dasar ke seluruh kata yang memiliki huruf pertama yang sama.

Setelah terbentuk *dataset* yang telah di filter, seluruh data tersebut dimasukkan ke dalam database. Untuk menghindari hal yang tidak diinginkan, dibuat salinan tabel *page_information* ke tabel baru *page_information_filtered* yang nantinya akan menjadi sumber data pengganti. Kolom *content_text* kemudian diperbarui dengan data yang sudah di filter.

```
query = "UPDATE page_information_filtered SET content_text = %s WHERE id_page = %s"

def update(db: Database, docID: int, content_text: str):
    print(f"Updating document {docID}")
    conn: pymysql.Connection[
        pymysql.cursors.Cursor] = db.connect() #type: ignore

    try:
        with conn.cursor(cursor=pymysql.cursors.DictCursor) as cursor:
            cursor.execute(
                f"UPDATE page_information_filtered SET content_text = '{content_text}' WHERE id_page = {docID}"
            )
    finally:
        conn.close()

if __name__ == "__main__":
    dotenv.load_dotenv()
    uFile = open('filterWithCount.pkl', 'rb')

    uniqueWordMap = pickle.load(uFile)

    db = Database()

    for k, v in uniqueWordMap.items():
        update(db, k, re.sub('\W+', ' ', v))
```

Gambar 4.23: Pengisian tabel *page_information_filtered* dengan data yang sudah di filter dengan daftar kata dasar

Operasi selanjutnya adalah pembuatan peringkat kata per dokumen. Pada implementasi sebelumnya, setiap kata yang telah mendapatkan skor akan segera di masukkan nilainya ke database. Karena proses ini diidentifikasi sebagai proses yang paling memperlambat pembuatan skoring kata, maka operasi database dipisah dan dilakukan setelah seluruh skor kata per dokumen terbentuk.

```

query = "SELECT * FROM `page_information_filtered`"
df = pd.read_sql(query, db_connection)
text_content = df[
    "content_text"] # Konten teks dari halaman yang sudah dicrawl

# Buat model menggunakan TfIdfVectorizer
vectorizer = TfIdfVectorizer(
    lowercase=True, # Untuk konversi ke lower case
    use_idf=True, # Untuk memakai idf
    norm="l2", # Normalisasi
    smooth_idf=True, # Untuk mencegah divide-by-zero errors
)

h = open('finalDataTFIDF.pkl', 'wb')

tfidf_matrix = vectorizer.fit_transform(text_content)
words = vectorizer.get_feature_names_out()
idf_vector = vectorizer.idf_

df_tfidf = pd.DataFrame.sparse.from_spmatrix(tfidf_matrix, columns=words)

data_tfidf = []
for i in range(len(df_tfidf)):
    print(f"Processing TF-IDF for document {i+1}/{len(df_tfidf)}")
    page_id = df["id_page"].loc[i]
    for j in range(len(words)):
        word = words[j]
        tf_idf = df_tfidf[word].loc[i]
        if tf_idf == 0.0:
            continue
        idf = idf_vector[j]
        tf = tf_idf / idf

        # Simpan setiap bobot/score pada kata ke table "tfidf_word"
        # save_one_tfidf_word(db_connection, word, page_id, tf_idf)
        data_tfidf.append({
            "kata": word,
            "page_id": page_id,
            "tf": tf,
            "idf": idf,
            "tfidf": tf_idf,
        })

pickle.dump(data_tfidf, h)
h.close()

```

Gambar 4.24: Proses skoring kata dari data yang sudah di filter

```

query = (
    "INSERT INTO tfidf_word (word, page_id, tfidf_score) VALUES (%s, %s, %s)")

if __name__ == "__main__":
    dotenv.load_dotenv()
    tfidfFile = open('finalDataTFIDF.pkl', 'rb')

    print("Loading TF-IDF file...")
    tfDict = pickle.load(tfidfFile)
    print("Loading TF-IDF file...DONE")

    tfidfFile.close()

    data = []

    print("Generating tuple list...")
    count = 0
    for d in tfDict:
        count += 1
        print(f'Data {count}/{len(tfDict)}')
        data.append((d["kata"], d["page_id"], d["tfidf"]))
    print("Generating tuple list...DONE")

    db = Database()

    conn = pymysql.connect(host="localhost",
                           port=3308,
                           user="root",
                           password="toor",
                           db="crawler")

    with conn:
        cursor = conn.cursor()
        cursor.executemany(query, data)
        conn.commit()
        conn.close()

```

Gambar 4.25: Pengisian tabel *tfidf_word* dari file persisten

Table Name	Engine	Auto Increment	Data Length
crawling	InnoDB	1	16K
page_forms	InnoDB	84,604	24M
page_images	InnoDB	1,226,298	134M
page_information	InnoDB	42,655	686M
page_information_filtered	InnoDB	42,654	714M
page_linking	InnoDB	5,489,314	469M
page_list	InnoDB	2,315,955	333M
page_paragraph	InnoDB	1,010,083	505M
page_scripts	InnoDB	343,974	214M
page_styles	InnoDB	81,926	149M
page_tables	InnoDB	82	144K
pagerank	InnoDB	1,974	112K
pagerank_changes	InnoDB	0	16K
tfidf	InnoDB	13,193	1.5M
tfidf_word	InnoDB	10,255,054	411M

Gambar 4.26: Statistik dataset setelah proses *filtering*

4.1.7 Struktur Direktori Kode

Struktur penulisan kode yang digunakan mengikuti struktur yang telah ada pada penelitian sebelumnya. Pada direktori *src*, dibuat folder baru dengan nama *indexing* sebagai penanda modul baru. Kemudian di dalamnya terdapat file *inverted_index.py* sebagai tempat implementasi utama.

Untuk menjalankan modul *indexer*, pada direktori utama dibuat file *run_index.py* yang memanggil objek dari modul *inverted_index*.

4.2 Pengujian

4.2.1 Statistik Pengujian

Pengujian dimulai dengan merekam waktu yang terpakai dalam proses pembuatan struktur index berdasarkan data dari database.

Tabel 4.1: Durasi proses *reindexing*

Mode	Percobaan Ke-1	Percobaan Ke-2	Percobaan Ke-3
Pengambilan dataset	7.6138s	7.5740s	7.5910s
Inverted Index	204.2576s	209.5963s	204.3641s
Inverted Index + GST	526.0686s	527.3835s	535.1130s

Karena terdapat peningkatan signifikan dalam pembuatan index dengan integrasi *GST*, maka direkam waktu pembuatan untuk struktur *GST* sendiri.

Tabel 4.2: Durasi pembuatan struktur *GST*

Percobaan	Durasi
1	302.1005s
2	308.6661s
3	319.0878s

Selanjutnya adalah merekam waktu yang digunakan untuk menyimpan struktur yang telah dibuat ke dalam file persisten dengan modul *pickle*.

Tabel 4.3: Durasi proses penyimpanan struktur data ke file persisten

Mode	Percobaan Ke-1	Percobaan Ke-2	Percobaan Ke-3
Inverted Index	6.3289s	6.1939s	6.2439s
Inverted Index + GST	6.2494s	6.2009s	6.6043s

Setelah itu, direkam proses muat ulang dari file persisten ke struktur data asli pada memori sebelum digunakan untuk pencarian.

Tabel 4.4: Durasi proses muat ulang struktur data dari file persisten

Struktur	Percobaan Ke-1	Percobaan Ke-2	Percobaan Ke-3
Pemetaan Kata – <i>Hitlists</i>	2.2921s	2.3181s	2.2682s
Pemetaan Dokumen – <i>Hitlists</i>	1.9319s	1.9767s	1.9284s
GST	2.1220s	2.1438s	2.1274s

Berikut adalah beberapa statistik penggunaan memori dan media penyimpanan yang dikumpulkan selama pengujian berlangsung.

Tabel 4.5: Ukuran struktur data yang terbentuk

Struktur	Ukuran di memori	Ukuran file persisten
Pemetaan Kata – <i>Hitlists</i>	2035.57 MB	264 MB
Pemetaan Dokumen – <i>Hitlists</i>	2026.64 MB	263 MB
GST	93.75 MB	7.3 MB

4.2.2 Pengujian Relevansi

Berikut adalah kata kunci yang digunakan untuk uji relevansi hasil pencarian. Kata kunci dipilih karena dapat digunakan dengan metode *TF-IDF* sebagai patokan. Implementasi *TF-IDF* sendiri sepertinya masih memiliki masalah di mana terdapat sekumpulan kata yang tidak menghasilkan output apa pun dengan menggunakan *dataset* yang digunakan. Sempat dilakukan pengujian dengan menggunakan *dataset* baru yang lebih kecil, dan berhasil. Tetapi tetap gagal untuk *dataset* utama.

Tabel 4.6: *Query* yang digunakan untuk uji relevansi

<i>Query</i> satu kata	<i>Query</i> dua kata
bupati	ganjar pranowo
waskita	tiket coldplay
nuklir	

Tabel 4.7: Hasil pencarian dengan kata kunci *bupati* dengan metode *inverted index*

Peringkat	Hasil
1	Mochamad Toha http://fnn.co.id/author/Mochamad Toha
2	daerah http://fnn.co.id/category/daerah?page=5
3	kesehatan http://fnn.co.id/category/kesehatan?page=13
Dilanjutkan pada halaman berikutnya	

Tabel 4.7: Hasil pencarian dengan kata kunci *bupati* dengan metode *inverted index*

Peringkat	Hasil
4	nasional http://fnn.co.id/category/nasional?page=11
5	daerah http://fnn.co.id/category/daerah?page=11

Tabel 4.8: Hasil pencarian dengan kata kunci *bupati* dengan metode *inverted index* dan integrasi dengan *GST*

Peringkat	Hasil
1	Selain Bupati Saiful Ilah, Ada Raja Koruptor yang Sedang diburu KPK! http://fnn.co.id/post/selain-bupati-saiful-ilah-ada-raja-koruptor-yang-sedang-diburu-kpk
2	Ruslan Tawari: Pejabat Bupati Seram Barat Jangan Bikin Resah Rakyat http://fnn.co.id/post/rustlam-tawari-pejabat-bupati-seram-barat-jangan-bikin-resah-rakyat-#back-to-top
3	Ruslan Tawari: Pejabat Bupati Seram Barat Jangan Bikin Resah Rakyat http://fnn.co.id/post/rustlam-tawari-pejabat-bupati-seram-barat-jangan-bikin-resah-rakyat-#1
4	Ruslan Tawari: Pejabat Bupati Seram Barat Jangan Bikin Resah Rakyat http://fnn.co.id/post/rustlam-tawari-pejabat-bupati-seram-barat-jangan-bikin-resah-rakyat-#1
5	Ruslan Tawari: Pejabat Bupati Seram Barat Jangan Bikin Resah Rakyat http://fnn.co.id/post/rustlam-tawari-pejabat-bupati-seram-barat-jangan-bikin-resah-rakyat-#1

Tabel 4.9: Hasil pencarian dengan kata kunci *bupati* dengan metode *TF-IDF*

Peringkat	Hasil
1	http://fnn.co.id/post/jahiliyah-yang-belum-berevolusi
2	http://fnn.co.id/post/wakil-ketua-dpr-santri-harus-jadi-penopang-kekuatan-ekonomi-baru
3	http://fnn.co.id/post/pemprov-sulsel-resmikan-ruas-jalan-perbatasan-sidrap-soppeng
4	http://fnn.co.id/category/daerah?page=5#back-to-top
5	http://fnn.co.id/category/daerah?page=18#3

Tabel 4.10: Hasil pencarian dengan kata kunci *waskita* dengan metode *inverted index*

Peringkat	Hasil
1	Dimas Huda http://fnn.co.id/author/Dimas Huda
2	infrastruktur http://fnn.co.id/category/infrastruktur
3	hukum http://fnn.co.id/category/hukum?page=4
4	Mochamad Toha http://fnn.co.id/author/Mochamad Toha
5	editorial http://fnn.co.id/category/editorial

Tabel 4.11: Hasil pencarian dengan kata kunci *waskita* dengan metode *inverted index* dan integrasi dengan *GST*

Peringkat	Hasil
1	Waskita Karya dan PT API Tandatangani Divestasi Tol Cibitung-Cilincing http://fnn.co.id/post/waskita-karya-dan-pt-api-tandatangani-divestasi-tol-cibitung-cilincing
2	Sim Salabim Abracadabra Waskita Karya http://fnn.co.id/post/simbsalabim-waskita-karya#back-to-top
3	Sim Salabim Abracadabra Waskita Karya http://fnn.co.id/post/simbsalabim-waskita-karya#1
4	Sim Salabim Abracadabra Waskita Karya http://fnn.co.id/post/simbsalabim-waskita-karya#2
5	Sim Salabim Abracadabra Waskita Karya http://fnn.co.id/post/simbsalabim-waskita-karya#3

Tabel 4.12: Hasil pencarian dengan kata kunci *waskita* dengan metode *TF-IDF*

Peringkat	Hasil
1	http://fnn.co.id/post/ptm-100-persen-di-surabaya-belum-bisa-dilaksanakan-secara-penuh#carousel-example-generic
2	http://fnn.co.id/post/waskita-dan-pt-smi-teken-pjbb-divestasi-tol-cimanggis-cibitung#back-to-top
3	http://fnn.co.id/post/waskita-dan-pt-smi-teken-pjbb-divestasi-tol-cimanggis-cibitung#1
4	http://fnn.co.id/post/waskita-dan-pt-smi-teken-pjbb-divestasi-tol-cimanggis-cibitung#2
5	http://fnn.co.id/post/waskita-dan-pt-smi-teken-pjbb-divestasi-tol-cimanggis-cibitung#3

Tabel 4.13: Hasil pencarian dengan kata kunci *nuklir* dengan metode *inverted index*

Peringkat	Hasil
1	Mochamad Toha http://fnn.co.id/author/Mochamad Toha
2	energi http://fnn.co.id/category/energi?page=2
3	energi http://fnn.co.id/category/energi?page=7
4	Dimas Huda http://fnn.co.id/author/Dimas Huda
5	nasional http://fnn.co.id/category/nasional?page=5

Tabel 4.14: Hasil pencarian dengan kata kunci *nuklir* dengan metode *inverted index* dan integrasi dengan *GST*

Peringkat	Hasil
1	Malaysia Prihatin Perlucutan Senjata Nuklir Melambat http://fnn.co.id/post/malaysia-prihatin-perlucutan-senjata-nuklir-melambat
2	Penumpukan Nuklir China Mengkhawatirkan Amerika Serikat http://fnn.co.id/post/penumpukan-nuklir-china-mengkhawatirkan-amerik-serikat
3	Indonesia Minta Perjanjian Nonproliferasi Nuklir Ditegakkan http://fnn.co.id/post/indonesia-minta-perjanjian-nonproliferasi-nuklir-ditegakkan
4	PLTN Ukraina Terbakar, Media Rusia Sebut Radiasi Nuklir Aman http://fnn.co.id/post/pltn-ukraina-terbakar-media-rusia-sebut-radiasi-nuklir-aman#back-to-top
5	PLTN Ukraina Terbakar, Media Rusia Sebut Radiasi Nuklir Aman http://fnn.co.id/post/pltn-ukraina-terbakar-media-rusia-sebut-radiasi-nuklir-aman#1

Tabel 4.15: Hasil pencarian dengan kata kunci *nuklir* dengan metode *TF-IDF*

Peringkat	Hasil
1	http://fnn.co.id/post/bsi-perluas-ekosistem-ekonomi-digital-lewat-dewan-masjid-indonesia
2	http://fnn.co.id/post/dpp-psi-tegaskan-viani-limardi-bukan-lagi-kader-partai
3	http://fnn.co.id/post/kementrian-investasi-permudah-kemitraan-umkm-usaha-besar
4	http://fnn.co.id/post/wapres-bersurat-ke-menkeu-dan-bpjph-guna-percepat-kodifikasi-halal
5	http://fnn.co.id/post/anies-baswedan-senang-bisa-bantu-tugas-kpk

Tabel 4.16: Hasil pencarian dengan kata kunci *tiket coldplay* dengan metode *inverted index*

Peringkat	Hasil
1	Mochammad Toha http://fnn.co.id/author/Mochammad Toha
2	politik http://fnn.co.id/category/politik?page=14
3	hukum http://fnn.co.id/category/hukum?page=10
4	hukum http://fnn.co.id/category/hukum?page=9
5	ekonomi http://fnn.co.id/category/ekonomi?page=6

Tabel 4.17: Hasil pencarian dengan kata kunci *tiket coldplay* dengan metode *inverted index* dan integrasi dengan *GST*

Peringkat	Hasil
1	Coldplay "LGBT" Kok Kagum Tokoh Syiah? http://fnn.co.id/post/coldplay-lgbt-kok-kagum-tokoh-syiah
2	Coldplay "LGBT" Kok Kagum Tokoh Syiah? http://fnn.co.id/post/coldplay-lgbt-kok-kagum-tokoh-syiah#back-to-top
3	Coldplay "LGBT" Kok Kagum Tokoh Syiah? http://fnn.co.id/post/coldplay-lgbt-kok-kagum-tokoh-syiah#1
4	Coldplay "LGBT" Kok Kagum Tokoh Syiah? http://fnn.co.id/post/coldplay-lgbt-kok-kagum-tokoh-syiah#2
5	Coldplay "LGBT" Kok Kagum Tokoh Syiah? http://fnn.co.id/post/coldplay-lgbt-kok-kagum-tokoh-syiah#3

Tabel 4.18: Hasil pencarian dengan kata kunci *tiket coldplay* dengan metode *TF-IDF*

Peringkat	Hasil
1	http://fnn.co.id/post/era-jokowi-sudah-padam#carousel-example-generic
2	http://fnn.co.id/post/bareskrim-mendalami-dugaan-penipuan-penjualan-tiket-online-coldplay#back-to-top
3	http://fnn.co.id/post/bareskrim-mendalami-dugaan-penipuan-penjualan-tiket-online-coldplay#1
4	http://fnn.co.id/post/bareskrim-mendalami-dugaan-penipuan-penjualan-tiket-online-coldplay#2
5	http://fnn.co.id/post/bareskrim-mendalami-dugaan-penipuan-penjualan-tiket-online-coldplay#3

Tabel 4.19: Hasil pencarian dengan kata kunci *ganjar pranowo* dengan metode *inverted index*

Peringkat	Hasil
1	Mochammad Toha http://fnn.co.id/author/Mochammad_Toha
2	Ganjar Calon Presiden yang Paling Lemah http://fnn.co.id/post/ganjar-calon-presiden-yang-paling-lemah
3	politik http://fnn.co.id/category/politik?page=3
4	politik http://fnn.co.id/category/politik?page=10
5	politik http://fnn.co.id/category/politik?page=15

Tabel 4.20: Hasil pencarian dengan kata kunci *ganjar pranowo* dengan metode *inverted index* dan integrasi dengan *GST*

Peringkat	Hasil
1	Ganjar Semakin "Bandel", dia Tahu Dilema Megawati http://fnn.co.id/post/ganjar-semakin-bandel-dia-tahu-dilema-megawati
2	Tiga persoalan Serius Pasca Pencapresan Ganjar http://fnn.co.id/post/tiga-persoalan-serius-pasca-pencapresan-ganjar
3	Megawati Akhirnya Akan Mendukung Ganjar Pranowo http://fnn.co.id/post/megawati-akhirnya-akan-mendukung-ganjar-pranowo
4	Megawati Akhirnya Akan Mendukung Ganjar Pranowo http://fnn.co.id/post/megawati-akhirnya-akan-mendukung-ganjar-pranowo#back-to-top
5	Megawati Akhirnya Akan Mendukung Ganjar Pranowo http://fnn.co.id/post/megawati-akhirnya-akan-mendukung-ganjar-pranowo#1

Tabel 4.21: Hasil pencarian dengan kata kunci *ganjar pranowo* dengan metode *TF-IDF*

Peringkat	Hasil
1	http://fnn.co.id/post/yaqut-harus-kukut
2	http://fnn.co.id/post/dimulai-dari-palesrina-dunia-diambang-perang-agama
3	http://fnn.co.id/post/dpr-ri-usulkan-pemilu-serentak-6-maret-2024
4	http://fnn.co.id/post/presiden-main-sandiwara
5	http://fnn.co.id/post/ketimbang-melakukan-amandemen-mpr-diminta-fokus-sosialisasi-empat-pilar

4.2.3 Pengujian Performa

Berikut adalah hasil pengujian perbandingan performa antara penggunaan struktur index dengan integrasi *GST*.

Tabel 4.22: Perbandingan durasi waktu pencarian

Percobaan	Inverted Index	Inverted Index + GST	Perubahan
<i>nuklir</i>	0.12565734s	0.00722047s	+94.3%
<i>waskita</i>	0.07239101s	0.00734384s	+89.8%
<i>bupati</i>	1.11867232s	0.03204097s	+97.1%
<i>ganjar pranowo</i>	6.17200237s	0.08626969s	+98.6%
<i>tiket coldplay</i>	0.25978417s	0.00856094s	+96.7%

Apabila dilihat dari perbandingan waktu yang dihabiskan, maka waktu yang digunakan untuk mengolah *query* yang diberikan oleh pengguna diproses lebih cepat dengan menggunakan integrasi *GST*. Oleh karena itu, berdasarkan skenario pengujian yang direncanakan, dapat disimpulkan bahwa hasil rancangan sesuai atau berhasil.

4.3 Analisis Hasil

Berdasarkan data yang telah dikumpulkan, analisis hasil pengujian adalah sebagai berikut:

1. Pencarian berhasil dilakukan dalam waktu kurang dari satu detik untuk semua kasus pengujian.
2. Integrasi dengan struktur *GST* memberikan peningkatan performa waktu pencarian karena menghindari melakukan iterasi pada seluruh *hitlist* dari sebuah kata. Karena struktur *GST* dapat langsung memberikan lokasi dokumen, maka hanya diperlukan perpotongan yang sesuai.
3. Karena struktur *GST* membutuhkan pemetaan antara dokumen dengan kata yang ada di dalamnya, maka diperlukan *hitlist* yang sama persis, dengan perbedaan hanya pada dengan apa data tersebut dipetakan. Hal ini membuat penggunaan memori dan penyimpanan persisten naik dua kali lipat.
4. Proses pembuatan struktur *GST* menghabiskan mayoritas waktu pada proses indeks ulang, bahkan lebih lama dari proses pembuatan struktur indeks itu sendiri.
5. Peningkatan performa dari integrasi *GST* dapat berasal dari dua hal, yaitu karena dokumen yang mengandung kata tersebut bisa langsung ditemukan dan daftar *hit* yang perlu dihitung lebih pendek karena hanya menghitung kata per dokumen ketimbang seluruh kata yang terekam di database.

Hasil pencarian dengan index menunjukkan relevansi yang kurang baik jika dibandingkan dengan hasil metode *TF-IDF*. Walaupun hasil pencarian hampir tidak menunjukkan halaman dengan judul yang sesuai, tetapi beberapa menghasilkan halaman web yang memiliki kategori yang sesuai.

Ketika dilakukan perbandingan lebih lanjut, nilai dari *exact match* tertimpa oleh banyaknya jumlah kata yang sesuai dengan input. Penggunaan filter *outlier* tidak memberikan hasil yang diharapkan.

Tetapi jika dibandingkan dengan hasil pada integrasi *GST*, hasil yang diberikan lebih akurat. Jika dilihat dari sumber datanya, terdapat dua perbedaan yang cukup signifikan. Struktur *inverted index* menggunakan informasi yang berasal hanya dari paragraf yang ada pada suatu halaman. Sementara struktur *GST* menggunakan informasi yang berasal dari judul halaman.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil dari implementasi dan pengujian terhadap arsitektur *inverted index*, maka diperoleh kesimpulan sebagai berikut:

1. Data pada kolom *paragraph* dengan jumlah 1.010.083 baris dari tabel *page_paragraph* berhasil digunakan untuk membuat struktur *inverted index*.
2. Modul *indexer* berbasis *inverted index* yang digunakan untuk melakukan pemeringkatan pada *query* yang diberikan oleh pengguna telah selesai dibuat.
3. Penyimpanan persisten untuk struktur *inverted index* diimplementasikan sebagai penyimpanan sederhana dalam satu proses yang sama dengan modul *indexer*.
4. Integrasi dengan struktur *GST* yang telah dibuat pada penelitian sebelumnya berhasil diintegrasikan dengan struktur *inverted index*.
5. Integrasi dengan struktur *GST* dapat memberikan reduksi waktu pencarian yang signifikan jika dibandingkan hanya dengan penggunaan *inverted index* saja, dengan rasio perbedaan berkisar antara 85 - 98%.
6. Relevansi pencarian tidak terlalu baik jika hanya menggunakan *inverted index*. Hasil menjadi lebih relevan ketika menggunakan integrasi *GST*.
7. Informasi yang ada pada *tag* paragraf (*<p>*) tidak cukup untuk menjadi dasar dalam pemeringkatan hasil pencarian.

5.2 Saran

Adapun saran untuk penelitian selanjutnya adalah:

1. Melanjutkan penelitian tentang informasi lain yang dapat di ekstrak dari suatu halaman web untuk meningkatkan hasil pencarian.

2. Melakukan perombakan terhadap struktur *hitlists* agar bisa direferensikan melalui alamat memori untuk menghilangkan redundansi pada struktur dengan integrasi *GST*.
3. Melanjutkan penelitian tentang penggunaan kompresi pada struktur *hit* untuk mereduksi penggunaan memori.
4. Menuliskan implementasi *indexer* dengan menggunakan bahasa lain yang lebih cepat dan mampu mengeksplorasi sistem *multi-thread* dengan lebih baik seperti *Rust* atau *C++*.
5. Melanjutkan penelitian tentang distribusi penyimpanan dan penggunaan *hitlists* pada memori.

DAFTAR PUSTAKA

- Brin, S. and L. Page (1998). “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. In: *Computer Networks* 30, pp. 107–117.
- Gonzalo, J. et al., (1998). “Indexing with WordNet synsets can improve Text Retrieval”. In.
- Hersh, W. (2001). *Managing Gigabytes Compressing and Indexing Documents and Images (Second Edition)*. Vol. 4. 1, pp. 109–113.
- Jaccard, P. (1912). *The Distribution of the Flora in the Alpine Zone*.
- Khatulistiwa, L. (2023). “Perancangan Arsitektur Search Engine dengan Mengintegrasikan Web Crawler, Algoritma Page Ranking, dan Document Ranking”. In.
- Librian, A. (2023). *High quality stemmer library for Indonesian Language (Bahasa)*. URL: <https://github.com/sastrawi/sastrawi>.
- Pratama, Z. (2023). “Perancangan Modul Pengindeks Pada Search Engine Berupa Generalized Suffix Tree Untuk Keperluan Pemeringkatan Dokumen”. In.
- Qoriiba Muhammad, F. (2021). “Perancangan Crawler Sebagai Pendukung pada Search Engine”. In.
- Rossum, G. van, J. Lehtosalo, and. Langa (2015). *PEP 484 – Type Hints*. URL: <https://peps.python.org/pep-0484/>.
- Seymour, T., D. Frantsvog, S. Kumar, et al., (2011). “History of search engines”. In: *International Journal of Management & Information Systems (IJMIS)* 15.4, pp. 47–58.
- Zobel, J., A. Moffat, and R. Sacks-Davis (1992). “An Efficient Indexing Technique for Full-Text Database Systems”. In: *Proceedings of the 18th VLDB Conference*, pp. 352–362.

LAMPIRAN A

Dokumentasi Pengujian Metode *Inverted Index*

```
DocID: 88 | Score: 334.0
Title: Mochamad Toha
URL: http://fnn.co.id/author/Mochamad Toha
=====
DocID: 342 | Score: 23.0
Title: daerah
URL: http://fnn.co.id/category/daerah?page=5
=====
DocID: 174 | Score: 16.0
Title: kesehatan
URL: http://fnn.co.id/category/kesehatan?page=13
=====
DocID: 302 | Score: 14.0
Title: nasional
URL: http://fnn.co.id/category/nasional?page=11
=====
DocID: 348 | Score: 13.0
Title: daerah
URL: http://fnn.co.id/category/daerah?page=11
=====
DocID: 352 | Score: 6.0
Title: daerah
URL: http://fnn.co.id/category/daerah?page=15
=====
```

LAMPIRAN B

Dokumentasi Pengujian Metode *Inverted Index* dan Integrasi GST

```
DocID: 13164 | Score: 23.0
Title: Selain Bupati Saiful Ilah, Ada Raja Koruptor yang Sedang Diburu KPK!
URL: http://fnn.co.id/post/selain-bupati-saiful-ilah-ada-raja-koruptor-yang-sedang-diburu-kpk
=====

DocID: 38438 | Score: 15.0
Title: Ruslan Tawari: Pejabat Bupati Seram Barat Jangan Bikin Resah Rakyat
URL: http://fnn.co.id/post/ruslan-tawari-pejabat-bupati-seram-barat-jangan-bikin-resah-rakyat-#back-to-top
=====

DocID: 38439 | Score: 15.0
Title: Ruslan Tawari: Pejabat Bupati Seram Barat Jangan Bikin Resah Rakyat
URL: http://fnn.co.id/post/ruslan-tawari-pejabat-bupati-seram-barat-jangan-bikin-resah-rakyat-#
=====

DocID: 38440 | Score: 15.0
Title: Ruslan Tawari: Pejabat Bupati Seram Barat Jangan Bikin Resah Rakyat
URL: http://fnn.co.id/post/ruslan-tawari-pejabat-bupati-seram-barat-jangan-bikin-resah-rakyat-#2
=====

DocID: 38441 | Score: 15.0
Title: Ruslan Tawari: Pejabat Bupati Seram Barat Jangan Bikin Resah Rakyat
URL: http://fnn.co.id/post/ruslan-tawari-pejabat-bupati-seram-barat-jangan-bikin-resah-rakyat-#3
=====

DocID: 38442 | Score: 15.0
Title: Ruslan Tawari: Pejabat Bupati Seram Barat Jangan Bikin Resah Rakyat
URL: http://fnn.co.id/post/ruslan-tawari-pejabat-bupati-seram-barat-jangan-bikin-resah-rakyat-#carousel-example-generic
=====

DocID: 6339 | Score: 15.0
Title: Ruslan Tawari: Pejabat Bupati Seram Barat Jangan Bikin Resah Rakyat
URL: http://fnn.co.id/post/ruslan-tawari-pejabat-bupati-seram-barat-jangan-bikin-resah-rakyat-
=====
```

DAFTAR RIWAYAT HIDUP



MOCHAMMAD HANIF RAMADHAN. Lahir di Jakarta, 25 Desember 1999. Anak Kedua dari pasangan Bapak Akhadiyat Karsono dan Ibu Herlarti Kresnoyorini. Saat ini penulis tinggal di Jl. Tembok Nomor 12 RT 002/RW 003, Kelurahan Kayu Putih, Kecamatan Pulogadung, Kota Jakarta Timur, Provinsi DKI Jakarta.

No. Ponsel : 081219853881

Email : hanifrmn@protonmail.com

Riwayat Pendidikan : Penulis mengikuti pendidikan sekolah dasar di SD Islam Al-Azhar 13 Rawamangun pada tahun 2006 - 2012. Setelah itu, penulis melanjutkan pendidikan di SMP Islam Al-Azhar 12 Rawamangun pada tahun 2012 - 2015. Kemudian penulis melanjutkan pendidikan di SMAN 68 Jakarta pada tahun 2015 - 2018. Setelah lulus penulis sempat menjalani jenjang perkuliahan di Universitas Gunadarma pada tahun 2018, sebelum akhirnya melanjutkan perkuliahan di Universitas Negeri Jakarta pada tahun 2019.

Riwayat Organisasi : Selama di bangku perkuliahan, penulis terlibat dalam organisasi Badan Eksekutif Mahasiswa Program Studi Ilmu Komputer sebagai staff Departemen Akademik periode 2019-2020 dan Badan Eksekutif Mahasiswa Program Studi Ilmu Komputer sebagai kepala Departemen Akademik periode 2020-2021. Penulis juga berpartisipasi dalam organisasi di lingkungan Program Studi Ilmu Komputer yaitu DEFAULT, di mana penulis membantu dalam upaya membangun ulang organisasi tersebut kembali.

METADATA

Judul : Implementasi Modul Indexing Pada Search Engine Telusuri Dengan Integrasi Inverted Index Dan Generalized Suffix Tree Untuk Mereduksi Waktu Pencarian

Nama : Mochammad Hanif Ramadhan

NIM : 1313619025

Pembimbing I : Muhammad Eka Suryana, M.Kom

Pembimbing II : Med Irzal, M.Kom

Keyword : 1. Mesin Pencari
2. Indeks
3. Basis Data
4. Teori Informasi