Perancangan Modul Pengindeks pada Search Engine berupa Generalized Suffix Tree untuk Keperluan Pemeringkatan Dokumen

Zaidan Pratama[1], Muhammad Eka Suryana[2], Med Irzal[3]

Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Negeri Jakarta, Jakarta Timur, Indonesia
Email: [1)]zaidanpratamaa@gmail.com, [2), 3)]

Abstract

To create a search engine, many components are needed to support the architecture of the search engine. In a previous study, Muhammad Fathan Qoriiba successfully developed the crawler component in his research entitled "Designing a Crawler as Support for a Search Engine." One component of the search engine that has not been implemented is the indexing module, or indexer. The purpose of this study is to create an indexing module using the Generalized Suffix Tree data structure for the purpose of document ranking and to continue the series of search engine research. The process of forming the Generalized Suffix Tree and all programs were developed using the Python language. The final result of this research is an indexing module in the form of a console program that is used to perform searches with a Mean Average Precision value of 0.658 from 3 testers, and a Generalized Suffix Tree data structure with a maximum depth of 11 levels and a total of 34,573 leaves stored in the same directory.

Keywords: *search engine, document retrieval, indexing module, generalized suffix tree, document ranking*

Abstrak

Untuk membuat *search engine* diperlukan banyak komponen yang mendukung kebutuhan arsitektur *search engine*. Pada penelitian sebelumnya, komponen *crawler* berhasil dibuat oleh Muhammad Fathan Qoriiba dalam penelitiannya yang berjudul "Perancangan *Crawler* Sebagai Pendukung Pada *Search Engine*". Salah satu komponen dari *search engine* yang belum diimplementasikan adalah modul pengindeks atau *indexer*. Penelitian ini bertujuan untuk membuat modul pengindeks menggunakan struktur data *Generalized Suffix Tree* untuk keperluan pemeringkatan dokumen sekaligus melanjutkan rangkaian penelitian *search engine*. Proses pembentukan *Generalized Suffix Tree* dan seluruh program dibuat menggunakan bahasa *Python*. Hasil akhir dari penelitian ini adalah modul pengindeks berupa *program console* yang digunakan untuk melakukan pencarian dengan nilai *Mean Average Precision* dari 3 *tester* sebesar 0.658 dan struktur data *Generalized Suffix Tree* dengan kedalaman maksimal 11 level dan total jumlah daun sebanyak 34.573 daun yang tersimpan dalam direktori yang sama.

Kata-kata kunci: *search engine, document retrieval, modul pengindeks, generalized suffix tree, pemeringkatan dokumen*

PENDAHULUAN

Web Search Engine adalah program perangkat lunak yang melakukan pencarian di internet yang memiliki banyak *website* dan bekerja berdasarkan kata kunci atau *query words* yang kita tentukan.

Search engine mencari kata kunci yang kita tentukan di dalam *database* atau basis data informasi yang dimiliki. Alan Emtage beserta dua orang temannya, yaitu Bill Heelan dan J. Peter Deutsch memperkenalkan *search engine* pertama kali pada tahun 1990 dengan nama *Archie*. *Archie* mengunduh daftar direktori semua file yang terletak di situs anonim publik FTP (*File Transfer Protocol*) lalu membuat *database* nama file yang dapat dicari. Namun, *Archie* tidak mengindeks isi situs-situs tersebut karena jumlah datanya sangat terbatas sehingga dapat dengan mudah dicari secara manual. (Seymour dkk., 2011).

TABEL 1. Persentase *global market share search engine* Februari 2022 (StatCounter, 2022)

Search Engine	Market Share (%)
Google	92.01
Bing	2.96
Yahoo!	1.51
Baidu	1.17
Yandex	1.06
DuckDuckGo	0.68

Dapat dilihat dari data di atas, *Google* masih menjadi *search engine* favorit bagi para pengguna internet untuk saat ini. *Google* sendiri ditemukan dan diperkenalkan pada tahun 1998 oleh Larry Page dan Sergey Brin. Salah satu keunggulan *Google* adalah pengaplikasian *PageRank*. Manfaat terbesar dari *PageRank* adalah kehebatannya dalam mengatasi *underspecified queries* atau kueri yang kurang spesifik. Sebagai contoh, pada mesin pencari konvensional jika kita mencari "Universitas Stanford" maka hasil pencariannya dapat mengembalikan sejumlah halaman web yang mencantumkan Stanford di dalamnya. Jika menggunakan *PageRank*, maka halaman web Universitas Stanford akan menjadi halaman pertama yang dikembalikan. (Brin dkk., 1998).

Google sebagai search engine terdiri dari beberapa komponen utama. Komponen-komponen tersebut antara lain *crawler, repository, indexer, searcher, page rank,* dan komponen lainnya. (Brin dkk., 1998). Pada penelitian sebelumnya yang dilakukan oleh Muhammad Fathan Qoriiba yang berjudul "Perancangan Crawler Sebagai Pendukung Pada Search Engine", rangkaian komponen *crawler* berhasil dibuat. Untuk membuat *search engine*, mulanya dibutuhkan sebuah *crawler*. *Crawler* merupakan sebuah program untuk mengambil informasi yang ada di halaman web. Dibutuhkan perancangan *crawler* yang baik untuk mendapatkan hasil yang baik. Algoritma yang digunakan mengikuti algoritma awal perkembangan *Google*, yaitu *breadth first search crawling* dan *modified similarity based crawling*. *Crawler* yang dibuat dapat dijalankan untuk situs web statis yang dibuat menggunakan *html* versi 5 atau 4. Hasil *crawling* disimpan di dalam *database MySQL*. (Qoriiba, 2021).

Rangkaian komponen *crawler* yang dibuat menghasilkan dataset berisi 8 tabel yang berukuran 176 MiB atau sekitar 185 MB. Dataset ini disusun menggunakan *tools crawling* yang dikembangkan. Situs yang digunakan untuk proses crawling adalah situs *indosport.com* dan *curiouscuisiniere.com.* Dataset ini akan digunakan untuk kelanjutan penelitian *search engine*. (Qoriiba, 2021). Salah satu komponen lain yang merupakan kelengkapan dari arsitektur search engine adalah pengindeks atau *indexer*.

Algoritma yang terkait dengan *indexing* dan *document retrieval* sendiri telah diteliti oleh cukup banyak orang sebelumnya. Salah satu pelopornya yaitu S. Muthukrishnan pada tahun 2002 dengan memanfaatkan penggunaan *Generalized Suffix Tree*. Algoritma Muthukrishnan menekankan pada pengambilan semua dokumen yang memiliki jumlah kemunculan pola *input query* jika melebihi ambang batas tertentu. Ambang batas tersebut ditentukan oleh pencari informasi atau dengan kata lain oleh *user*. (Hon dkk., 2010).

Pada penelitian yang berjudul "Privacy-preserving string search on encrypted genomic data using a generalized suffix tree", penggunaan *Generalized Suffix Tree* untuk pengindeksan memberikan peningkatan yang signifikan terhadap waktu eksekusi *query*. Struktur indeks dari *generalized suffix tree* ini digunakan untuk membuat sebuah protokol yang aman dan efisien dalam melakukan pencarian *substring* dan set-maksimal pada data genom. Hasil eksperimen menunjukkan bahwa

metode yang diajukan pada penelitian ini dapat mengkomputasi pencarian *substring* dan set-maksimal yang aman pada sebuah dataset *single-nucleotide polymorphism* (SNPs) dengan 2184 *records* (setiap record berisi 10000 SNPs) dalam 2.3 dan 2 detik. (Mahdi dkk., 2021).

Wing-Kai Hon dan beberapa temannya berhasil mengembangkan algoritma baru dengan pemanfaatan dari *Generalized Suffix Tree* yang lebih efisien dari segi ruang dan waktu. Algoritma ini digunakan untuk mendapatkan indeks yang efisien dalam masalah pencarian dokumen. Pendekatan ini didasarkan pada penggunaan baru dari pohon sufiks atau *suffix tree* yang dinamakan *induced generalized suffix tree (IGST*). (Hon dkk., 2010).

Di dalam penelitian ini, penulis akan membuat modul pengindeks atau *indexer* sebagai salah satu komponen arsitektur *search engine*. Penulis akan membuat *indexer* menggunakan algoritma yang sudah dikembangkan oleh Wing-Kai Hon dan teman-temannya. Dataset yang akan digunakan sebagai kumpulan dokumen adalah dataset hasil *crawling* yang sudah dibuat oleh Muhammad Fathan Qoriiba sebelumnya. Penelitian ini sekaligus akan melanjutkan rangkaian penelitian sebelumnya terkait *search engine.*

KAJIAN PUSTAKA

A. Search Engine

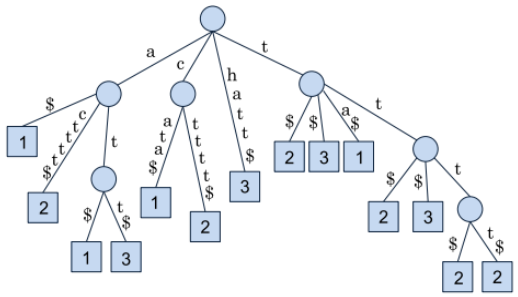
Search engine merupakan contoh dari *Information Retrieval System* berskala besar. Sejarah *search engine* bisa dibilang dimulai dari tahun 1990 ketika Alan Emtage dan kawan-kawannya membuat *Archie*. Setelah *Archie*, banyak *search engine* lain yang mulai diperkenalkan. *Gopher* dan *Veronica and Jughead* diperkenalkan pada tahun 1991. Pada tahun 1993, *W3Catalog and Wanderer*, *Aliweb*, dan *Jump Station* diperkenalkan ke publik. Menyusul di tahun 1994 ada *WebCrawler* dan pada tahun 1995 *MetaCrawler*, *AltaVista*, serta *Excite* diperkenalkan. (Seymour dkk., 2011).

B. Document Retrieval

Dalam basis data yang berisi teks atau *string*, kita memiliki kumpulan beberapa dokumen yang berisi teks atau *string* alih-alih hanya satu dokumen *string* saja. Dalam kasus ini, masalah dasarnya adalah mengambil semua dokumen di mana pola *query* P berada. Masalah ini disebut juga masalah pengambilan dokumen atau *document retrieval problem*. (Hon dkk., 2010). Masalah ini sudah pernah diteliti sebelumnya oleh S. Muthukrishnan.

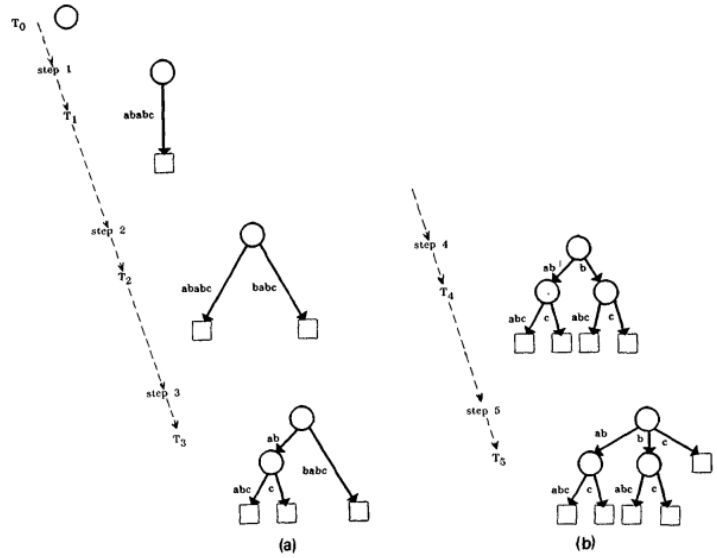
C. Generalized Suffix Tree

Diketahui $\Delta = \{T1, T2, ..., Tm\}$ menunjukkan satu set dokumen. Setiap dokumen adalah *string* karakter dengan karakter yang diambil dari $\sum$ alfabet umum yang ukurannya $|\sum|$ bisa tak terbatas. Kita asumsikan untuk setiap i, karakter terakhir dari dokumen Ti ditandai dengan $i , yang mana bersifat *unique* untuk semua karakter pada semua dokumen. *Generalized Suffix Tree (GST)* untuk Δ adalah sebuah *compact trie* yang menyimpan semua sufiks dari dokumen Ti . Lebih tepatnya, setiap sufiks dari setiap dokumen sesuai dengan daun yang berbeda di GST. Setiap tepi diberi label oleh urutan karakter, sehingga untuk setiap daun yang mewakili beberapa sufiks *s*, rangkaian label tepi di sepanjang jalur akar ke daun adalah *s*. Selain itu, untuk setiap *internal node u*, tepi yang mengarah ke anak-anaknya semuanya berbeda dengan karakter pertama pada label tepi yang sesuai, sehingga anak-anak *u* diurutkan menurut urutan abjad dari karakter pertama tersebut.



Gambar 1. Contoh *Generalized Suffix Tree* dengan T1 = cata, T2 = actttt, dan T3 = hatt. (Hon dkk., 2010).

Proses pembentukan *suffix tree ababc* digambarkan pada gambar di bawah ini:



Gambar 2. Pembentukan *Suffix Tree ababc* (McCreight, 1976)

Dari ilustrasi di atas dapat disimpulkan langkah-langkah pembentukan pohon sufiks adalah sebagai berikut:

1. Buat *tree* awal yaitu root
2. Tambahkan *terminal node* agar setiap sufiks dari kata *input* terwakilkan dalam *tree*
3. Tambahkan setiap sufiks ke *tree* jika pada *tree* belum ada sufiks tersebut
4. Jika prefiks dari sufiks yang akan ditambahkan sudah ada maka penambahan sufiks ke *tree* menyesuaikan dengan menjadikan sufiks sebagai anak dari prefiks tersebut

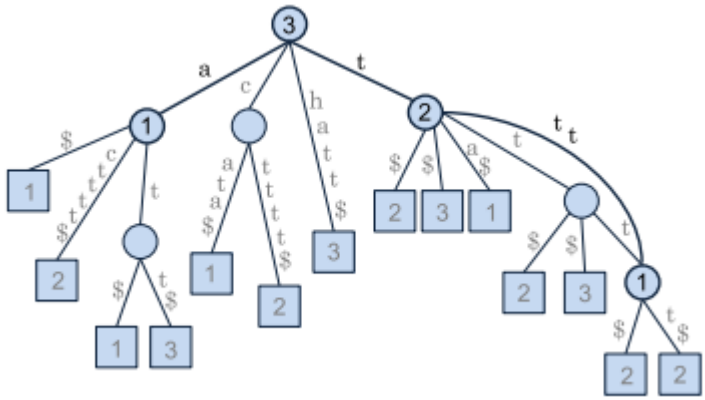
D. Induced Generalized Suffix Tree

Misalkan ada GST untuk Δ dan sebuah bilangan bulat f dengan $1 \leq f \leq D$. Anggap setiap daun pada GST diberi label original dari sufiks yang sesuai. Untuk setiap *internal node v* dari GST, kita katakan *v* adalah f-*frequent* jika di dalam *subtree* yang berakar pada *v*, setidaknya ada *f* daun yang memiliki label yang sama. Subpohon terinduksi dari GST atau *induced subtree* yang dibentuk dengan mempertahankan semua *node* frekuensi-f disebut *pre-IGST-f*. Subpohon terinduksi yang terbentuk oleh *node* yang disorot ini adalah *pre-IGST-f*. (Hon dkk., 2010).

Misalkan v adalah sebuah *node* di dalam *pre-IGST-f* dan *v* adalah sebuah *internal node* di dalam GST tersebut. Istilah *count(f, v)* digunakan untuk menyatakan jumlah dokumen berbeda yang memiliki label *f* dalam subpohon yang berakar pada *v* di GST. Istilah *count(v)* digunakan ketika konteksnya jelas. Nilai *count(v)* harus berada di antara 1 dan *m*, di mana *m* adalah jumlah dokumen dalam Δ. (Hon dkk., 2010).

Misalkan ada sebuah *pre-IGST-f*. Untuk setiap *internal node v*, kita katakan *v* redundan apabila *v* adalah *node* derajat-1 atau *degree-1 node* dan *count(v) = count(child(v))* di mana *child v* menyatakan

unique child dari *v*. Pohon induksi atau *induced tree* yang dibentuk dengan menyusutkan atau mereduksi semua node redundan pada *pre-IGST-f* disebut *IGST-f*. (Hon dkk., 2010).

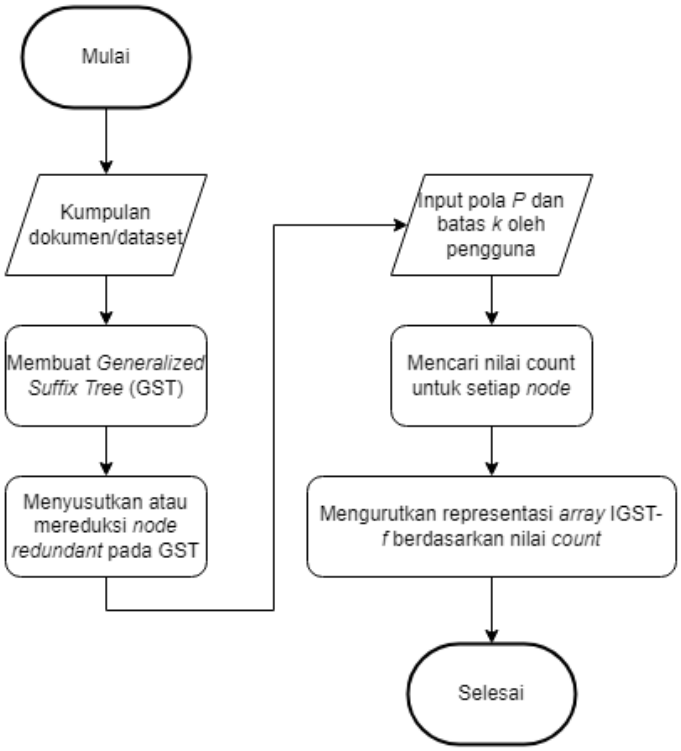


Gambar 3. IGST-2 (Hon dkk., 2010)

METODOLOGI PENELITIAN

A. Tahapan Penelitian

Diagram alir atau *flowchart* di bawah ini merupakan tahapan demi tahapan yang akan dilaksanakan pada penelitian ini. *Flowchart* disusun berdasarkan algoritma *Efficient Index For Retrieving Top-k Most Frequent Document.*



Gambar 1. *Flowchart* Tahapan Penelitian Modul Pengindeks

Diagram alir dari algoritma pengindeksan dimulai dari masukan berupa kumpulan dokumen yang akan dibuat menjadi GST. Langkah selanjutnya yaitu membuat GST dari kumpulan dokumen tersebut. GST menjadi struktur data awal dari kumpulan dokumen. Dari GST tersebut kemudian dilakukan penyusutan atau reduksi untuk *node* yang redundan sehingga akan terbentuk pohon yang terinduksi untuk frekuensi f yang bernama *Induced Generalized Suffix Tree-f* atau *IGST-f*. *IGST-f* sendiri seperti yang dijelaskan, menjadi kunci atau komponen utama dari pengindeksan. Langkah

selanjutnya adalah program menerima masukan berupa pola kata dan batas *k* untuk dicari pada kumpulan dokumen. Dari sini, kita akan mencari nilai *count* dari setiap *node*. Langkah terakhir yaitu mengurutkan representasi *array IGST-f* yang sudah memiliki nilai *count* dan mengembalikan hasil *top-k* dokumen yang memiliki pola *P*.

B. Dataset

Pada penelitian ini, dataset yang akan digunakan sebagai kumpulan dokumen adalah tabel *page information* khususnya kolom *title*. Dataset ini disusun oleh Muhammad Fathan Qoriiba dalam penelitiannya yang berjudul PERANCANGAN CRAWLER SEBAGAI PENDUKUNG PADA SEARCH ENGINE. Dataset ini disusun menggunakan *tools crawling* yang telah dikembangkan. Situs yang digunakan untuk proses *crawling* adalah situs *indosport.com* dan *curiouscuisiniere.com.* Total data yang tersimpan di dalam dataset tersebut berukuran 176 MiB atau sekitar 185 MB. Tabel *page information* berisi 1060 halaman web dengan atribut:

1. *Baseurl* : Pranala dari halaman web
2. *Html5* : Menyatakan halaman tersebut tersusun dari HTML5 atau bukan
3. *Title* : Judul halaman web
4. *Description* : Deskripsi halaman web
5. *Keywords* : Kata kunci dari halaman web
6. *Content text* : Isi dari halaman web
7. *Hot url* : Menyatakan halaman web tersebut termasuk hot url atau tidak
8. *Model crawl* : Menyatakan cara crawling yang digunakan pada halaman web

C. Tahapan Pengembangan

Pada penelitian ini, penulis akan membuat modul pengindeks atau indexer yang menerima input *string* berupa kata kunci atau pola kata dan *integer* input berupa batasan berapa dokumen yang dikembalikan sebagai hasil. *Indexer* akan mengembalikan beberapa dokumen sesuai dengan batasan yang diberikan dan dengan relevansi paling tinggi yang mengandung kata kunci tersebut. Algoritma yang akan digunakan adalah algoritma *Efficient index for retrieving top-k most frequent documents.*

Penelitian ini merupakan penelitian pendukung dari keseluruhan penelitian *search engine.* Penelitian induk dari rangkaian penelitian *search engine* antara lain PERANCANGAN CRAWLER SEBAGAI PENDUKUNG PADA SEARCH ENGINE oleh Muhammad Fathan Qoriiba dan PERANCANGAN ARSITEKTUR SEARCH ENGINE DENGAN MENGINTEGRASIKAN WEB CRAWLER, ALGORITMA PAGE RANKING, DAN DOCUMENT RANKING oleh Lazuardy Khatulistiwa.

Pembuatan modul pengindeks pada penelitian ini akan dilakukan bertahap sesuai dengan setiap proses pada *flowchart* sebelumnya. Waktu dari pembuatan setiap langkah atau proses diestimasikan 1-2 minggu dan paling maksimal 3 minggu tergantung kompleksitas dari proses tersebut. Jika penelitian ini selesai lebih dulu dibandingkan penelitian induknya yaitu PERANCANGAN ARSITEKTUR SEARCH ENGINE DENGAN MENGINTEGRASIKAN WEB CRAWLER, ALGORITMA PAGE RANKING, DAN DOCUMENT RANKING yang dibuat oleh Lazuardy Khatulistiwa, maka penelitian ini akan berjalan secara otonom dan hasil akhirnya adalah modul pengindeks. Namun, jika penelitian induknya selesai lebih dulu dibandingkan penelitian ini, maka struktur direktori serta *design class* dan fungsi dari penelitian induk akan diikuti skemanya atau dengan kata lain modul pengindeks akan disesuaikan dengan arsitektur *search engine* besar.

D. Pengujian

Untuk pengujian dari hasil penelitian ini, akan ada dua poin penting pengujian, yaitu:

1. Relevansi pencarian yaitu kesesuaian hasil pencarian yang dihasilkan oleh program dengan pola kata kunci yang diberikan pada saat awal memasukkan input. Relevansi pencarian akan diujikan dengan *Mean Average Precision*. Indikator modul pengindeks berjalan dengan baik adalah dokumen yang dikembalikan relevan dengan kata yang dicari oleh pengguna.

2. Waktu pengindeksan atau *indexing time* yaitu kecepatan pengindeksan dari awal proses input masuk hingga mengembalikan hasil. Waktu pengindeksan akan diujikan dengan *performance testing* yang berfokus pada *speed*. Indikator modul pengindeks berjalan dengan baik adalah waktu pengembalian dokumen dalam waktu singkat yaitu hitungan detik atau kurang.

Mean Average Precision (MAP) adalah nilai rata-rata aritmatika dari nilai *average precision* pada sistem temu kembali informasi untuk sekumpulan n topik pencarian. Metrik evaluasi *Mean Average Precision* telah lama digunakan sebagai standar evaluasi sistem pengambilan informasi yang diakui secara umum pada Konferensi Pemulihan Teks NIST (TREC). Selama 25 tahun terakhir, banyak penelitian yang dipublikasikan dalam bidang pengambilan informasi mengandalkan perbedaan yang diamati pada MAP untuk menarik kesimpulan tentang efektivitas teknik atau sistem yang dipelajari. (Beitzel dkk., 2009)

$$MAP = \frac{1}{n}\sum APn \quad (1)$$

AP adalah *average precision* dari untuk topik tertentu dari n topik. (Beitzel dkk., 2009). Nilai *precision* untuk konteks pencarian informasi didapatkan dari hasil pembagian antara total dokumen yang relevan dengan total dokumen yang dihasilkan. (Ting, 2010)

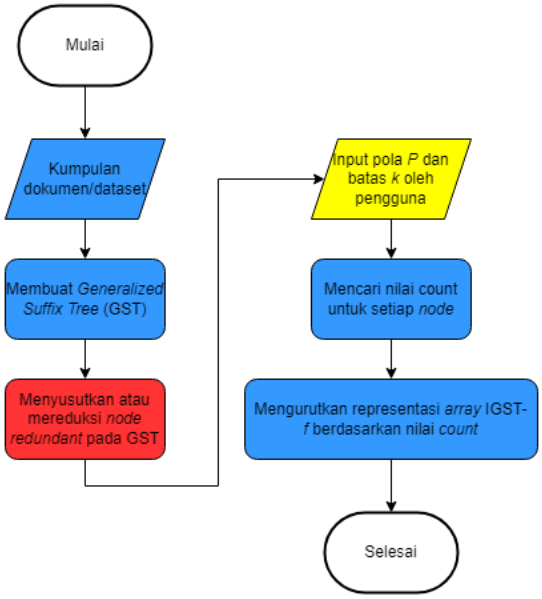
Dalam penelitian ini, 3 *tester* dengan pola input yang sama akan berpartisipasi dalam pengujian *mean average precision* untuk menilai apakah dokumen yang dikembalikan relevan dengan pencarian yang diinginkan. Dokumen yang akan dijadikan pengujian adalah 5 dokumen teratas pada hasil pencarian. Penggunaan *mean average precision* menghasilkan skala hasil 0 sampai 1.

Performance testing merupakan pengujian yang melakukan verifikasi pada kebutuhan non-fungsional yaitu performa perangkat lunak salah satunya kecepatan atau *speed*. Untuk waktu pengindeksan, skenarionya adalah sebagai berikut:

1. *Developer* memasukkan 5 input pola kata kunci dengan panjang yang berbeda-beda dan input limit *k* sebesar 3, 5, dan 7 sebagai batasan berapa dokumen yang dikembalikan.
2. Program mencari dokumen yang memiliki pola input.
3. Program mengembalikan top-*k* dokumen yang memiliki pola input.
4. *Developer* mengukur dan membandingkan waktu yang dihabiskan dari mulai awal proses hingga pengembalian dokumen dengan input yang berbeda-beda.

HASIL DAN PEMBAHASAN

Pada bagian ini akan dibahas mengenai hasil penelitian dan implementasi dari rancangan yang sebelumnya telah dibuat. Rancangan yang telah dibuat diimplementasikan dengan proses coding menggunakan bahasa pemrograman *Python*. Berikut adalah hasil implementasi dari rancangan yang telah dibuat:



Gambar 1. Komponen *Flowchart* yang Diimplementasikan

Pada penelitian ini komponen *flowchart* yang ditandai dengan warna biru berhasil diimplementasikan. Komponen *flowchart* yang berwarna merah yaitu mereduksi *node* redundan belum berhasil diimplementasikan karena keterbatasan kemampuan dan waktu. Untuk bagian input yang ditandai dengan warna kuning terdapat modifikasi di mana input yang sudah diimplementasikan hanya memerlukan pola kata yang ingin dicari dan tidak memerlukan batas *k* agar lebih mudah untuk digunakan dan mengikuti alur natural ketika melakukan pencarian.

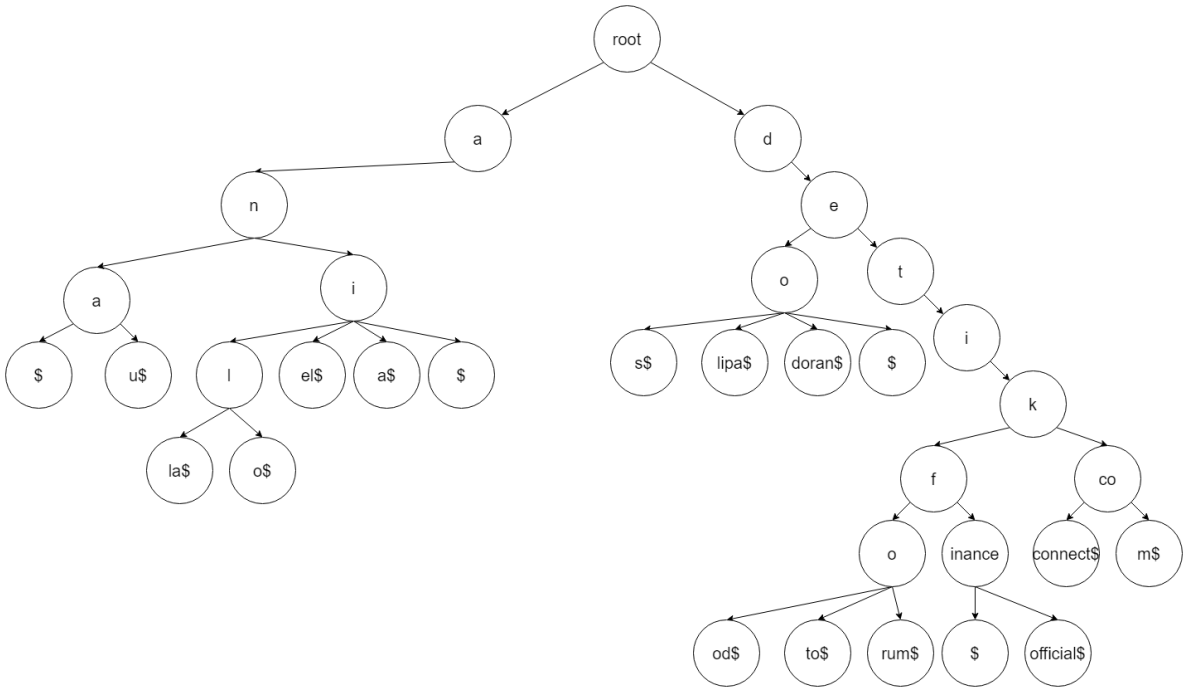
A. Dataset

Dataset yang digunakan sebagai *input* untuk membentuk *tree* adalah tabel *page information* dan kolom *title* yang berjumlah 10.686 *rows* atau data. Dataset ini didapatkan dari hasil *crawling* pada penelitian induk yaitu PERANCANGAN ARSITEKTUR SEARCH ENGINE DENGAN MENGINTEGRASIKAN WEB CRAWLER, ALGORITMA PAGE RANKING, DAN DOCUMENT RANKING oleh Lazuardy Khatulistiwa. Dataset ini juga dapat ditemukan pada penelitian induk sebagai sumber data yang digunakan untuk perhitungan *page rank, TF-IDF, dan similarity score*. Penggunaan dataset ini adalah salah satu bentuk pengintegrasian dengan penelitian induk.

Pada tabel *page information* terdapat kolom *id page* yaitu id dari halaman, *crawl id* yang merupakan *id* dari *crawl*, *url* yaitu pranala dari halaman web, *title* yaitu judul halaman web, *description* yaitu deskripsi dari halaman web, *keywords* yaitu kata kunci dari halaman web, *hot url* yaitu *hot url* dari halaman web, *content text* yaitu konten dari halaman web, *size bytes* yaitu ukuran dari halaman web, *model crawl* yaitu model *crawling* yang digunakan, *duration crawl* yaitu durasi dari *crawling*, dan created at yaitu kapan data tersebut dibuat. Kolom *title* adalah data yang akan digunakan sebagai bahan pembentukan *Generalized Suffix Tree*.

B. Pembentukan *Generalized Suffix Tree*

Data *title* yang sudah dibersihkan selanjutnya akan dimasukkan ke dalam *tree* atau dengan kata lain *Generalized Suffix Tree* akan dibentuk. Hasil dari penelitian ini yaitu *Generalized Suffix Tree* menjadi struktur data terakhir yang digunakan sebagai modul pengindeks untuk keperluan pemeringkatan dokumen.



Gambar 2. Ilustrasi Potongan *Tree* yang Terbentuk

Ilustrasi ini merupakan sebagian kecil dari pohon yang terbentuk. Seluruh sufiks dari data *title* dimasukkan ke dalam *tree*. Sebagai contoh pada ilustrasi di atas, sufiks *detikcom*, *detikconnect*, *detikfood*, *detikfinance* dan sufiks lainnya terdapat pada *tree*. Dari 10.686 data *title* didapatkan jumlah maksimal kedalaman *tree* yang terbentuk yaitu 11 level dan jumlah maksimal daun pada *tree* sebanyak 34.573 daun.

C. Mereduksi *Node* Redundan pada *Tree*

Penjelasan mengenai reduksi node redundan dijelaskan pada bagian *Induced Generalized Suffix Tree*. Pada penelitian ini tahapan ini belum bisa diimplementasikan karena keterbatasan kemampuan dan waktu. Tahapan ini berperan dalam efisiensi segi ruang atau *space*. Seluruh tahapan *flowchart* pada penelitian ini masih bisa berjalan meskipun tahapan ini tidak terlaksana. Tahapan ini yang disebut dengan mengubah *Generalized Suffix Tree* menjadi *Induced Generalized Suffix Tree*. Karena tahapan ini tidak terlaksana maka *Generalized Suffix Tree* menjadi struktur data terakhir yang diimplementasikan pada penelitian ini.

D. *Input* Pola *P*

Terdapat sedikit perubahan pada input yang digunakan untuk melakukan pencarian. Pada *flowchart* terdapat input pola *P* dan batas *k* yaitu seberapa banyak dokumen yang ingin dikembalikan. Untuk membuat proses pencarian lebih simpel dan mengikuti alur natural pencarian maka input batas *k* dihilangkan dan dokumen yang dikembalikan adalah seluruh dokumen terkait dengan pencarian.

```
hasil pencarian:  {'query': 'resesi', 'result': Node('/root/r/e/s/e/s/i$', index=[352, 805, 807, 813, 829, 834, 2061, 2072, 3816, 4147, 4402, 4415
, 4645, 4894, 4895, 4898, 4900, 4901, 4903, 4904, 4905, 4906, 4907, 4908, 4926, 9878, 10185, 10591, 10592, 10593, 10607, 10608, 10609, 10645, 1064
6, 10647])}
++++++++++++++++++++
hasil pencarian:  {'query': '2023', 'result': Node('/root/2/0/2/3$', index=[85, 301, 316, 325, 335, 527, 566, 567, 637, 649, 660, 895, 970, 1016,
1212, 1375, 1421, 1952, 1984, 2253, 2630, 2635, 3003, 3059, 3083, 3084, 3085, 3882, 3898, 3907, 3923, 3926, 4042, 4054, 4055, 4137, 4384, 4400, 44
93, 4497, 4498, 4510, 4590, 4591, 4592, 4593, 4679, 4680, 4757, 4903, 4928, 4944, 4958, 4959, 4990, 5085, 5630, 5841, 5937, 5938, 5953, 6488, 6517
, 6575, 6613, 6628, 6631, 6638, 6640, 6645, 6662, 6664, 6665, 6671, 6673, 6674, 6683, 6707, 6710, 6883, 6886, 7414, 7415, 7416, 7417, 7418, 7419,
7420, 7421, 7422, 7423, 7424, 7425, 7785, 7794, 7799, 7812, 7946, 8234, 8312, 8347, 8382, 8475, 8481, 8482, 8550, 8619, 8679])}
```

Gambar 3. Contoh Hasil Pencarian dengan *Query* Resesi 2023

Untuk hasil pencarian berupa *array* berisi *dictionary*. *Key* dari *dictionary* hasil adalah *query*, *result*, dan *index*. Pada gambar di atas, elemen pertama dari *array* adalah *dictionary* hasil dari kata resesi dengan *result* berupa *node* dari *tree* dan *index* yang berisi *id* dokumen yang pada judulnya terdapat kata resesi. Elemen kedua dari *array* adalah *dictionary* hasil dari kata 2023 dengan *result* berupa *node* dari *tree* dan *index* yang berisi *id* dokumen yang pada judulnya terdapat kata 2023.

E. Mencari Nilai *Count*

Pada tahapan ini nilai *count* berfungsi sebagai parameter peringkat pada tiap *node* atau hasil pencarian. Setelah berhasil mendapatkan hasil pencarian dari kalimat yang dimasukkan, hasil pencarian perlu dilakukan pemeringkatan terkait relevansi hasil pencarian. Dengan menghitung nilai *count* atau jumlah kemunculan kata pada judul dan mengakumulasikan jumlah dari tiap kata tersebut maka dokumen dengan relevansi tertinggi akan dihasilkan.

```
masukkan kata yang ingin dicari: resesi 2023
{'index': 4903, 'count': 2, 'query': 'resesi 2023'}
{'index': 352, 'count': 1, 'query': 'resesi'}
{'index': 805, 'count': 1, 'query': 'resesi'}
{'index': 807, 'count': 1, 'query': 'resesi'}
{'index': 813, 'count': 1, 'query': 'resesi'}
{'index': 829, 'count': 1, 'query': 'resesi'}
{'index': 834, 'count': 1, 'query': 'resesi'}
{'index': 2061, 'count': 1, 'query': 'resesi'}
{'index': 2072, 'count': 1, 'query': 'resesi'}
{'index': 3816, 'count': 1, 'query': 'resesi'}
{'index': 4147, 'count': 1, 'query': 'resesi'}
{'index': 4402, 'count': 1, 'query': 'resesi'}
{'index': 4415, 'count': 1, 'query': 'resesi'}
{'index': 4645, 'count': 1, 'query': 'resesi'}
{'index': 4894, 'count': 1, 'query': 'resesi'}
{'index': 4895, 'count': 1, 'query': 'resesi'}
{'index': 4898, 'count': 1, 'query': 'resesi'}
{'index': 4900, 'count': 1, 'query': 'resesi'}
{'index': 4901, 'count': 1, 'query': 'resesi'}
{'index': 4904, 'count': 1, 'query': 'resesi'}
{'index': 4905, 'count': 1, 'query': 'resesi'}
{'index': 4906, 'count': 1, 'query': 'resesi'}
{'index': 4907, 'count': 1, 'query': 'resesi'}
{'index': 4908, 'count': 1, 'query': 'resesi'}
{'index': 4926, 'count': 1, 'query': 'resesi'}
{'index': 9878, 'count': 1, 'query': 'resesi'}
{'index': 10185, 'count': 1, 'query': 'resesi'}
{'index': 10591, 'count': 1, 'query': 'resesi'}
{'index': 10592, 'count': 1, 'query': 'resesi'}
{'index': 10593, 'count': 1, 'query': 'resesi'}
{'index': 10607, 'count': 1, 'query': 'resesi'}
{'index': 10608, 'count': 1, 'query': 'resesi'}
```

Gambar 4. Contoh *Array* Hasil Pemeringkatan *Index* Berdasarkan Nilai *Count*

Array berisi *dictionary* telah diurutkan berdasarkan nilai *count* dengan *key index*, *count*, dan *query*. Pada *query* resesi 2023, *index* atau *id* dokumen teratas dengan *count* 2 mengandung *query* resesi 2023. Selanjutnya, *index* dengan *count* lebih kecil ditampilkan sesuai urutan. Untuk nilai *count* yang sama atau senilai maka relevansi dokumen dinilai sama atau setara karena terdapat pola *query* pada judul sesuai nilai *count*.

F. Merepresentasikan Array Hasil Berdasarkan Nilai Count

Arriay hasil pemeringkatan yang sebelumnya sudah diurutkan berdasarkan nilai couint ditampilkan dengan detail untuk *query*, nilai *count*, judul halaman web, dan *URL* dari hasil pencarian. 5 data hasil pencarian teratas ditampilkan ke *console*.

G. Hasil Pengujian

Setelah semua komponen sudah dijalankan, hal yang selanjutnya dilakukan adalah melakukan pencarian dengan beberapa kalimat dan panjang yang berbeda. Pencarian dilakukan dengan 3 skema input berbeda yaitu dengan panjang 1 kata, 2 kata, dan 2 *input* dengan kata yang salah ketik sehingga total ada 8 pencarian yang dilaksanakan. Berikut merupakan seluruh *input* dan waktu pengembalian pencariannya:

TABEL 1. Tabel *input* dan waktu pengembalian pencarian

***Input* Kata**	**Waktu Pengembalian**
bjorka	0,0004 detik
messi	0,0005 detik
masak	0,0015 detik
resesi 2023	0,0046 detik
bitcoin naik	0,0023 detik
world cup	0,0005 detik
teknolpgi	0,0012 detik
reseso	0,0001 detik

Sesuai dengan skenario pengujian sebelumnya, modul pengindeks akan diuji relevansinya oleh 3 orang tester. Hasil pengujian relevansi menggunakan metode *Mean Average Precision* dari modul pengindeks adalah sebagai berikut:

TABEL 2. Hasil Pengujian Relevansi 3 *Tester* Menggunakan Metode *Mean Average Precision*

***Input* Kata**	**AP Tester 1**	**AP Tester 2**	**AP Tester 3**
bjorka	1	1	1
messi	1	1	1
masak	0.4	1	1
resesi 2023	0.2	0.2	0.2
bitcoin naik	0.2	1	1
world cup	0	1	1
teknolpgi	0.8	1	1
reseso	0	0	0
MAP	**0.45**	**0.775**	**0.8**

Berdasarkan hasil pengujian *Mean Average Precision* mengenai relevansi *query*, MAP *tester* 1 untuk seluruh *query* adalah 0.45. MAP *tester* 2 untuk seluruh *query* adalah 0.775. MAP *tester* 3 untuk seluruh *query* adalah 0.8. Untuk menghitung total skor MAP dari ketiga *tester* maka seluruh nilai MAP akan dijumlah dan dibagi dengan 3 sesuai dengan jumlah *tester*. Total skor MAP = (0.45 + 0.775 + 0.8)/3 = 0.658 Total skala penilaian MAP adalah dari rentang 0 sampai 1 sehingga untuk nilai 0.658 dapat disimpulkan bahwa relevansi modul pengindeks tergolong cukup relevan namun perlu ditingkatkan lagi.

H. Analisis Hasil

Berikut analisis hasil berdasarkan pengujian yang telah dilakukan:

1. Pencarian berhasil dilakukan dalam hitungan detik untuk seluruh input pengujian.
2. Nilai total *Mean Average Precision* dari 3 tester sebesar 0.658 menandakan dokumen yang ditampilkan cukup relevan dengan yang diinginkan oleh *tester* namun perlu ditingkatkan lagi.

Pencarian dengan 1 kata menampilkan dokumen dengan relevansi yang baik yaitu dokumen dengan judul yang mengandung kata tersebut dan nilai *Average Precision* dari ketiga *tester* mendekati maksimal. Pencarian dengan 2 kata pada input resesi 2023 menghasilkan daftar dokumen dengan relevansi kurang memuaskan karena nilai *Average Precision* dari ketiga *tester* tidak ada yang maksimal. Pencarian dengan 2 kata pada input bitcoin naik hanya mengembalikan daftar dokumen dengan judul yang mengandung kata bitcoin karena pada dataset tidak ada dokumen dengan judul yang mencakup kedua kata bersamaan. Pencarian dengan 2 kata pada input world cup tergolong baik karena nilai *Average Precision* dari 2 *tester* maksimal sementara untuk 1 *tester* nilainya 0.

Pencarian dengan kata yang salah ketik pada input *teknolpgi* menghasilkan daftar dokumen dengan relevansi sangat baik di mana daftar dokumen yang ditampilkan adalah dokumen yang mengandung kata *teknologi* dan nilai *Average Precision* dari ketiga *tester* mendekati maksimal. Dengan kata lain, dokumen yang dikembalikan sesuai dengan yang dimaksud. Pencarian dengan kata yang salah ketik pada input *reseso* hanya mengembalikan dokumen yang mencakup kata *reses* di dalamnya karena pencarian kata pada *tree* dimulai dari depan dan hanya bisa mengembalikan *longest common substring* atau substring terpanjang yang sesuai dengan *node* di *tree* sehingga nilai *Average Precision* dari ketiga *tester* adalah 0.

KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil penelitian yang sudah dilakukan dapat ditarik kesimpulan sebagai berikut:

1. Struktur direktori serta dataset dari penelitian induk telah diikuti untuk membuat modul pengindeks berupa *Generalized Suffix Tree*.
2. Data title berjumlah 10.686 *rows* dari tabel *page information* berhasil digunakan untuk membuat *Generalized Suffix Tree* dengan kedalaman maksimal 11 level serta jumlah daun sebanyak 34.573 daun sebagai struktur data indeks yang dapat dimanfaatkan untuk pencarian kata atau *query*.
3. Modul pengindeks berupa *Generalized Suffix Tree* untuk keperluan pemeringkatan dokumen pada *search engine* berhasil dibuat dengan menggunakan *naive algorithm* dengan notasi waktu $O(n^2)$.
4. Pencarian dan pemeringkatan dokumen hasil pencarian berjalan dengan relevansi cukup baik dengan nilai total *Mean Average Precision* sebesar 0.658 dari ketiga *tester* di mana daftar dokumen dengan judul yang mencakup kata *query* ditampilkan dan waktu rata-rata pencarian berhasil didapatkan dalam hitungan detik.
5. Pencarian dengan *query* salah ketik bekerja cukup baik pada kata *teknolpgi* karena substring yang paling panjang yang bisa ditemui dan dikembalikan adalah *teknol* sehingga seluruh dokumen dengan kata teknologi bisa dikembalikan.
6. Pencarian dengan *query* salah ketik pada kata *reseso* tidak menghasilkan daftar dokumen yang diinginkan karena *substring* yang paling panjang yang bisa ditemui dan dikembalikan adalah *reses* sehingga hanya dokumen dengan kata *reses* yang ditampilkan.

B. Saran

Adapun saran untuk penelitian selanjutnya adalah:

1. Melanjutkan penelitian dengan struktur data *Induced Generalized Suffix Tree* agar dihasilkan struktur data yang lebih efisien dari segi ruang.
2. Melanjutkan penelitian dengan struktur pohon kustomisasi buatan sendiri atau menggunakan graph library yang bernama *i*.
3. Melanjutkan penelitian dengan input data ke *tree* berupa *tag* atau *resume* dari suatu halaman web/dokumen agar relevansi meningkat.
4. Mengembangkan fungsi pencarian agar dapat menghasilkan relevansi lebih baik terutama untuk pencarian dengan *query* salah ketik.

DAFTAR PUSTAKA

Al Aziz, M. M., Thulasiraman, P., dan Mohammed, N. (2022). Parallel and private generalized suffix tree construction and query on genomic data. BMC genomic data, 23(1):1–16.

Beitzel, S. M., Jensen, E. C., dan Frieder, O. (2009). MAP, pages 1691–1692. Springer US, Boston, MA.

Brin, S., Motwani, R., Page, L., dan Winograd, T. (1998). What can you do with a web in your pocket? IEEE Data Eng. Bull., 21(2):37–47.

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

Hon, W.-K., Patil, M., Shah, R., dan Wu, S.-B. (2010). Efficient index for retrieving top-k most frequent documents. Journal of Discrete Algorithms, 8(4):402–417.

McCreight, E. M. (1976). A space-economical suffix tree construction algorithm. Journal of the ACM (JACM), 23(2):262–272.

Muthukrishnan, S. (2002). Efficient algorithms for document retrieval problems. In SODA, volume 2, pages 657–666. Citeseer.

Qoriiba, M. F. (2021). Perancangan crawler sebagai pendukung pada search engine.

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

StatCounter (2022). Search engine market share worldwide.

Ting, K. M. (2010). Precision and Recall, pages 781–781. Springer US, Boston, MA.