Implementasi *Fulltext Indexing* pada Dokumen Elektronik dengan Algoritma *B-Tree*

Diken Pradana Putra[1], Eko Darwiyanto[2], Alfian Akbar Gozali[3]
Program Studi Teknik Informatika Telkom University, Bandung
[1]diken.pp@gmail.com [2]ekodarwiyanto@telkomuniversity.ac.id [3]alfian@tass.telkomuniversity.ac.id

Abstrak
Dokumen merupakan sumber informasi yang mengandung fakta penting dari suatu kejadian atau keadaan tertentu dan dokumen tersebut menjadi suatu informasi penting bagi suatu instansi. Penggunaan dokumen elektronik sudah menggeser penggunaan dokumen konvensional yang memakai kertas sebagai bentuk fisiknya. Pengelolaan dokumen elektronik dapat dilakukan dengan menyimpannya pada media penyimpanan offline (media magnetik dan media optik) maupun online (database online dan cloud storage) yang mana keduanya memiliki fungsi indexing sebagai metode pengelolaannya. Salah satu metode indexing untuk meng-index teks biasa agar mengurangi kapasitas pemakaian storage dan meningkatkan kinerja searching adalah Fulltext Indexing. Dalam Fulltext Indexing indeks disimpan dalam struktur Balance Search Tree (B-Tree), dimana struktur penyimpanan database ini memudahkan Indexing dan Searching dokumen.Hasil penelitian Tugas Akhir ini adalah pengimplementasian Fulltext Indexing dan struktur B-Tree membuat sistem pengelolaan dokumen elektronik menjadi lebih cepat 0,3 kali dibandingkan tanpa pengimplementasian kedua metode tersebut dengan perbandingan jumlah kata ter-index dengan jumlah kata dari jumlah dokumen yang ditentukan adalah 1:8,6.

Kata Kunci : Dokumen Elektronik, *Fulltext Indexing, B-Tree*

Abstract
Document is an information source that contains important facts of a certain event or situation and the document became an important information for an institution. The use of electronic documents has shifted the use of conventional physical documents that put on paper. Electronic document management can be done by saving them in offline storage media (magnetic media and optical media) or online (online database, and cloud storage) which both have the function of indexing as a method of management. One of the indexing methods to index plain text in order to reduce the use of storage capacity and improve the performance of searching is Fulltext Indexing. Index in Fulltext Indexing is stored in the structure of the Balance of Search Tree (B-Tree), which is the database storage structure that makes it easier for searching and indexing documents. The output generated from research in this final project is by implementing Fulltext Indexing and B-Tree structure makes electronic document management system 0,3 faster than without the implementation of these methods with the comparison of the indexed word with the total words from the amount of the determined documents is 1:8,6.

Keyword : *electronic document, Fulltext Indexing, B-Tree*

1. Pendahuluan

Dokumen adalah sumber informasi yang mengandung fakta penting dan terekam pada media tertentu seperti dokumen atau surat penting dan memiliki fungsi tertentu bagi suatu instansi.. Penggunaan pengolahan manual dokumen konvensional membutuhkan ruang penyimpanan dokumen dengan kapasitas yang besar sedangkan dokumen yang keluar masuk selalu bertambah. Semakin banyak dokumen maka akan semakin sulit pengelolaannya karena terbatasnya kapasitas ruang dokumen dan kemampuan individu dalam melakukan pencarian dokumen yang menyebabkan dokumen menjadi tidak teratur, sulit untuk dicari, dan membutuhkan waktu lama untuk menemukan suatu dokumen. Untuk alasan ini, banyak organisasi mencoba untuk meningkatkan penanganan dokumen dengan menggunakan penerapan aplikasi teknologi informasi [16]

Fungsi dasar pengolahan adalah mengubah dokumen dan menyimpan dokumen dalam suatu tempat yang aman dan dengan suatu cara yang memungkinkan penemuan dokumen tertentu dengan cepat. Sehingga sistem pengolahan dokumen dalam bentuk dokumen elektronik akan meningkatkan efisien dan standar manajemen dokumen dibandingkan pengolahan secara manual [18]. Dokumen elektronik meliputi dokumen baik teks, grafik, atau speadsheets yang dihasilkan oleh software dan disimpan pada media magnetik (disk) atau media optik (CD, DVD) serta surat elektronik (e-mail) atau dokumen yang dikirim melalui Electronic Data Interchange [13]. Oleh karena itu, dokumen elektronik dipilih menjadi material yang digunakan dalam tugas akhir ini.

Pengelolaan dokumen elektronik dapat dilakukan dengan menyimpannya pada media penyimpanan offline (media magnetik dan media optik) maupun online (database online dan cloud storage) yang mana keduanya memiliki fungsi indexing sebagai metode pengelolaannya [1]. Terdapat banyak metode indexing yang biasa dipakai, seperti

Inverted Index, Suffix Tree, atau Ngram Index. Salah metode indexing yang bisa meng-index teks biasa untuk mengurangi kapasitas pemakaian storage dan meningkatkan kinerja searching adalah Fulltext Indexing [9]. Fulltext Indexing merupakan metode pengelolaan pengindeksan dokumen elektronik dengan proses pengindeksan yang dilakukan dengan membaca halaman yang di-scan kemudian mengindeks setiap kata dan meletakkannya pada lokasinya, yang akan mengurangi biaya proses pengindeksan dan meningkatkan kemampuan pencarian dokumen berdasarkan kata atau frase [6]. Fulltext indexing adalah teknik pencatatan atau penyimpanan dokumen di dalam tabel database dan dokumen disimpan dalam struktur Balance Search Tree (B-Tree), dimana struktur penyimpanan database ini memudahkan penelusuran dokumen [17]. Selain struktur B-Tree terdapat pula struktur B^+-Tree, k-d-tree, k-d-B-tree, atau hB-tree, dimana dibandingkan struktur yang lain B-Tree adalah struktur pengindeksan data yang cepat dan mengatur indeks ke dalam satu set multi-level node, di mana setiap node berisi data ter-index yang memungkinkannya untuk menyimpan banyak informasi di dalamnya serta memiliki tinggi leaf yang sama untuk setiap node yang ada.[11]. Oleh karena itu, metode Fulltext Indexing dengan menggunakan algoritma struktur data B-Tree dalam pengelolaan database yang dipilih dalam implementasi Tugas Akhir ini.

2. Tinjauan Pustaka

2.1 Dokumen Elektronik

Dokumen elektronik adalah segala bentuk informasi yang membutuhkan komputer atau alat elektronik lain untuk menampilkan, mengartikan, dan memproses dokumen tersebut [16].

Dokumen elektronik muncul sebagai hasil perkembangan teknologi *electronic display* sehingga dokumen dapat dilihat pada layar komputer daripada harus mencetak dokumen, dimana hal tersebut lebih menghemat penggunaan kertas dan kapasitas ruang penyimpanan. Dokumen dalam era teknologi digital semuanya disimpan sebagai *string bit*, sehingga bentuk fisik biasa dari dokumen (kertas atau *microfilm*) tidak lagi berperan penuh membantu dalam pengelolaan dokumen karena keunikan dari suatu dokumen dalam bentuk fisik menjadi lebih berkurang [10].

2.2 Indexing

Proses Indexing adalah proses pembuatan struktur data utuk mempercepat proses retrieval yang dibuat dari bagian suatu data. Tujuan dari penyimpanan index adalah untuk mengoptimalkan kecepatan dan kinerja dari sistem dalam melakukan pencarian data [1]. Saat tidak menggunakan index pencarian akan dilakukan dengan memindai setiap data yang ada dan akan membutuhkan waktu yang cukup lama dan resource yang banyak. Terdapat beberapa metode indexing yang sering digunakan pada database, berikut ini adalah contohnya [14] :

1. Suffix Tree : indeks yang hanya menyimpan akhiran darisetiap kata pada data
2. Inverted Index : indeks dari semua kemunculan bagian dari data
3. Ngram Index : indeks yang menyimpan panjang suatu sequence dari suatu data untuk keperluan text retrieval atau text mining
4. Fulltext Index : indeks dari setiap kemunculan kata dari suatu data

2.2.1 *Fulltext Indexing*

Fulltext Indexing adalah metode *information retrieval* di mana user dapat mencari informasi menggunakan kata-kata dari dokumen asli sebagai *search key* dengan menjadikan kata sebagai *index* [17]. Saat dokumen baru masuk ke dalam database, secara otomatis memicu transaksi *insertion* kata-kata bersama-sama dengan informasi di dalamnya ke dalam *index* [6]. Ide utama dari *Fulltext Indexing* adalah untuk mengambil dokumen berdasarkan kemiripan konten dokumen [5]. *Fulltext indexing* dirancang untuk meng-*index* teks polos untuk mengurangi kapasitas penyimpanan tetapnya, serta untuk meningkatkan kinerja pencarian [9].

Berikut ini adalah beberapa metode yang biasa digunakan untuk menerapkan *Fullttext indexing* [17] :

a. *fulltext scanning*, yaitu dengan melakukan *sequential scan* melalui *database*
b. *inversion of terms*. Idenya adalah mendaftar kata-kata yang muncul dokumen.
c. *signature files*. Idenya adalah untuk menghitung *hash function* dari setiap kata dalam dokumen dan bersama dengan *signature* dari setiap kata.

2.2 *Balanced Search Tree (B-Tree)*

B-Tree merupakan suatu struktur data yang digunakan untuk melakukan *indexing* dalam *database* [11]. *B-Tree* memiliki teknik pencarian 'Multiway Search Tree' yg dapat meminimalkan jumlah akses ke disk. *B-Tree* memiliki kelebihan yaitu dapat menampung banyak data pada setiap node-nya serta struktur data pohon yang memiliki tinggi setiap daunnya yang sama [8]. Pada perkembangannya, *B-Tree* diterapkan dalam berbagai kasus pengelolaan basis data dan bentuk *B-Tree* dikembangkan menjadi berbagai variasi. *B-Tree* digunakan pada pengelolaan data dari database (basis data), yaitu dengan memberikan indeks pada data secara teratur dan terurut. Dimana pemberian indeks pada data membuat proses menambah, menghapus, atau memperbarui

(*update*) suatu data menjadi mudah dan cepat. [7] [11].

B-Tree diciptakan oleh Rudolf Bayer dan Ed McCreight pada tahun 1972 [4] [15]. *B-Tree* dibuat untuk memungkinkan suatu *tree* memiliki node yang menyimpan banyak data, dengan jumlah *sub-tree* yang banyak pula. Karena hal tersebutlah *B-Tree* tepat digunakan untuk mengelola data pada *disk*. *B-Tree* dapat digunakan pada situasi dimana sebagian atau semua bagian dari pohon harus disimpan dalam *secondary storage* karena *B-Tree* bisa meminimalisasi jumlah akses ke *disk*. [2] [3]

B-Tree adalah generalisasi dari *Binary Search Tree* dimana lebih dari dua *path* menyatu dalam satu *node*. *Sebuah B-Tree* dengan *order m* memiliki karakteristik sebagai berikut [4] [11] :

a. Setiap *leaf node* (simpul daun) memiliki kedalaman yang sama.
b. Sebuah *node* (*leaf* atau *index*) x memiliki nilai x.num sebagai jumlah objek yang disimpan dalam x. *Key* dan nilai dari objek ke-i (1 <= i <= x.num) direpresentasikan sebagai x.key (i) dan x.value (i)
c. Sebuah *index node* x , menyimpan objek x:num dan menyimpan x:num + 1 pointer anak. Dimana setiap pointer anak adalah *pageid* dari *node* anak yang sesuai. Pointer anak ke-i dilambangkan sebagai x.child (i) dengan rentang *key* (x.key (i – 1), x.key (i)). Ini berarti bahwa dalam *sub-tree* ke-i, *key* harus lebih besar dari x.key (i – 1) dan lebih kecil dari x.key(i). Misalnya, suatu *sub-tree* yang direferensikan oleh x.child (1), maka *key* lebih kecil dari x.key(1) dan di *sub-tree* yang direferensikan oleh x.child (2), *key* ada di antara x.key (1) dan x.key (2), dan seterusnya.
d. Setiap *node* kecuali *root* minimal harus setengah penuh. Artinya, misalkan sebuah *index node* dapat menyimpan hingga 2m anak pointer, maka setiap *index node* kecuali *root* harus memiliki minimal m pointer anak.
e. Jika *root* adalah *index node*, maka *root* harus memiliki setidaknya dua anak
f. Semua *node* internal kecuali *root* memiliki paling banyak *m* anak yang tidak kosong, dan paling sedikit *m / 2* anak yang tidak kosong

B-Tree memiliki operasi *construct, insertion, deletion,* dan *search* [7] [8] [11]. Berikut ini adalah operasi yang ada pada *B-Tree* [2] [4] [11] :

*Search, u*ntuk melakukan pencarian data menggunakan indeks, maka terlebih dahulu mencari nilai pada node yang lebih besar dari nilai yang diinginkan. Setelah ditemukan maka penunjuk (pointer) akan menunjukkan nilai sebelumnya, lalu masuk ke anak yang ditunjuk oleh nilai tersebut, dan mengulang langkah tadi sampai nilai ditemukan. Apabila pada suatu node tidak ditemukan nilai yang lebih besar dari nilai yang dicari maka pointer akan mengikuti node yang ditunjuk oleh nilai terbesar dari node tersebut.

*Insert, tu*ntuk melakukan perintah *insert*, sebelumnya harus dilakukan perintah *search* terlebih dahulu untuk menetukan dimana data akan diletakkan. Bila *node* belum penuh maka perintah *insert* dapat langsung dilakukan, akan tetapi jika *node* telah penuh maka *node* harus dipecah terlebih dahulu untuk membuat *node* baru dan memasukkan anak baru dari *node* tersebut. Dan bila ternyata *node* induk dari *node* yang akan dipecah penuh, maka *node* induk harus dipecah terlebih dahulu. Perintah terus diulang sampai *node* tidak penuh.

*Delete, d*alam melakukan perintah *delete*, terdapat beberapa hal yang harus diperhatikan. Jika penghapusan dapat mengurangi nilai *key* dari suatu *node* dan nilainya menjadi di bawah *minimal key*, maka harus dilakukan penyesuaian seperti pengurangan tinggi pohon.

*Create / Construct, u*ntuk perintah *create* akan membuat *node* yang kosong. Pembuatan dilakukan dengan mengalokasikan sebuah akar baru dengan nilai *key* kosong dan menjadikannya daun.

2.4 Data Evaluation

Untuk mengukur efektivitas dan efisiensi dari sebuah sistem Information Retrieval terdapat beberapa cara yang dapat digunakan. Pada penelitian tugas akhir ini metode yang digunakan adalah Precision dan Recall dimana kedua metode ini adalah standar yang digunakan dalam sistem Information Retrieval untuk mengukur seberapa baik sistem tersebut berjalan [1] [14].

2.4.1 Precision

Precision digunakan untuk mengukur seberapa relevan suatu dokumen yang terambil dibandingkan dengan seluruh dokumen yang terambil, dimana terdapat pula dokumen yang tidak relevan terhadap suatu query pencarian. Berikut ini adalah rumus perhitungan Precision :

$$Precision = \frac{\sum dokumen\ relevan\ yang\ terambil}{\sum seluruh\ dokumen\ yang\ terambil}$$

2.4.2 Recall

Recall digunakan untuk mengukur kelengkapan suatu dokumen relevan yang terambil dari database dokumen terhadap suatu query pencarian. Recall diukur karena tidak semua dokumen yang relevan terhadap query dapat terambil. Berikut ini adalah rumus perhitungan Recall :

$$Recall = \frac{\sum dokumen\ relevan\ yang\ terambil}{\sum seluruh\ dokumen\ relevan}$$

Berikut ini adalah Tabel Evaluasi pada proses retrieval suatu dokumen :

Tabel 1 Tabel Evaluasi

dokumen			
relevan		tidak relevan	
terambil (true positive)	tidak terambil (false negative)	terambil (false positif)	tidak terambil (true negative)

Nilai Precision dan Recall juga dapat dihitung dengan :

$$Precision = \frac{tp}{(tp+fp)}$$

$$Recall = \frac{tp}{(tp+fn)}$$

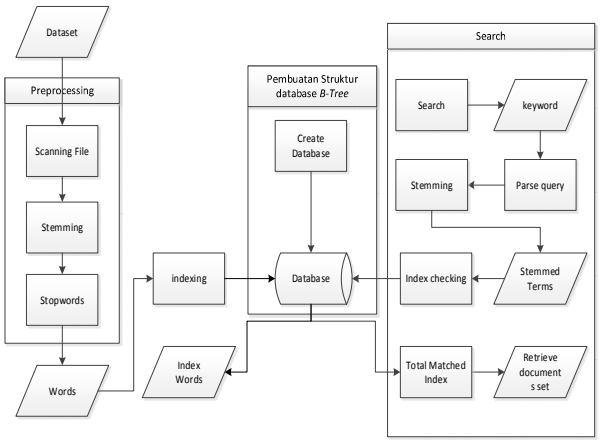
2.5 Execution Time

Execution time adalah waktu yang dibutuhkan oleh sistem untuk melakukan eksekusi atau menjalankan tugas, berbeda dengan fase lain dari siklus hidup suatu sistem seperti *compile time* (waktu kompilasi), *link time* (waktu link), *load time* (waktu buka). *Execution Time* dihitung dari semua waktu yang dikonsumsi oleh run-time layanan atas nama sistem tersebut pada CPU [1] [14]. Berikut ini adalah *Execution Time* yang digunakan dalam Tugas Akhir ini :

a) Waktu pembuatan indeks (*indexing*)
   Waktu yang dibutuhkan sistem untuk melakukan proses pembuatan indeks, mulai dari proses preprocessing sampai indeks di database terbentuk
b) Waktu pencarian (*searching*)
   Waktu yang dibutuhkan sistem untuk melakukan proses pencarian, mulai dari proses pencarian pada database indeks sampai memunculkan hasil document set dari keyword masukan pada searching

3. Perancangan dan Implementasi

3.1 Alur Proses sistem



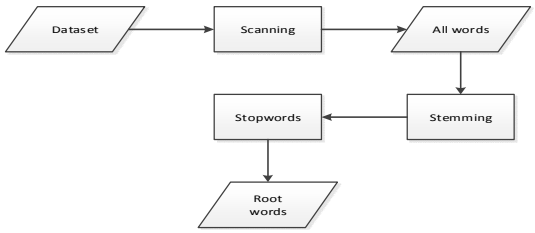
Gambar 1 Alur Proses Sistem

3.1.1 Dataset

Dataset berupa artikel dari Republika Online (www.republika.co.id) tanggal 1 Mei 2014 dan 2 Mei 2014 sebanyak 300 artikel yang telah diubah dan disimpan ke format dokumen Notepad Plain Text (*.txt).

3.1.2 Preprocessing

Tahapan preprocessing akan digambarkan seperti di bawah ini :



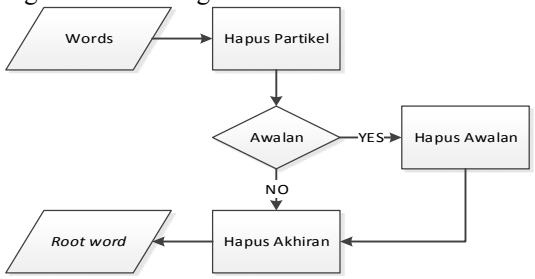
Gambar 2 Alur Preprocessing

3.1.2.1 Scanning

Scanning merupakan tahap *preprocessing* yang dimulai dari mengambil setiap kata yang ada pada *dataset*, dalam hal ini adalah dokumen dengan format *Notepad Plain Text* (*.*txt*) yang berisi kumpulan kata dari artikel Republika Online untuk nantinya akan dilanjutkan dengan proses *stemming* dan *stopwords.*

3.1.2.2 Stemming

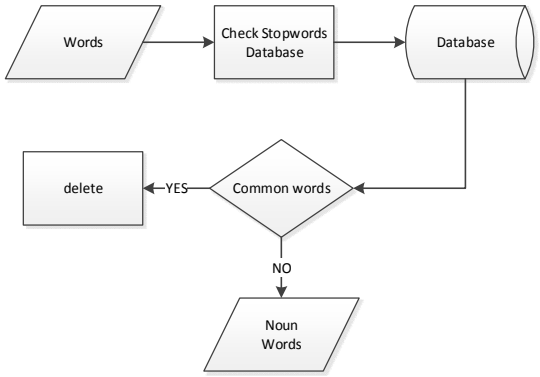
Stemming merupakan proses mengubah suatu kata bentukan menjadi kata dasar (root word) dengan menghapus awalan atau akhiran dari suatu kata bentukan. Tahap proses Stemming dapat digambarkan sebagai berikut :



Gambar 3 Alur Proses Stemming

3.1.2.3 Stopwords

Stopwords merupakan proses pada *preprocessing* dimana dilakukan penghapusan kata-kata selain kata kerja dan kata benda yang sering ditampilkan dalam dokumen. Proses penghapusan dilakukan dengan mengecek *database* dari kata *stopwords*, apakah termasuk kata kerja dan kata benda atau bukan kedua jenis kata tersebut. Tahap proses *Stopwords* dapat digambarkan sebagai berikut :



Gambar 4 Alur Proses Stopwords

3.1.3 Pembuatan Struktur Database B-Tree

Pada tahapan ini dibuat struktur *database B-Tree* yang digunakan untuk menyimpan *index* dari kata hasil proses *preprocessing*. Pembuatan dilakukan dengan mengalokasikan sebuah akar baru dengan nilai *key* kosong dan menjadikannya daun. Berikut ini adalah notasi algoritma untuk membuat *database B-Tree* :

```
x <- Allocate-Node()
leaf[x] <- TRUE
n[x] <- 0
Disk-Write(x)
root[T] <- x
```

Gambar 5 Notasi Algoritma Create B-Tree

Setelah *database B-Tree* sudah dibuat, maka dilakukan *Insertion* dari kata hasil proses *preprocessing* yang nanti menjadi *index* untuk proses pencarian.

3.1.4 Seearch menggunakan Metode Fulltext Indexing

Setelah *database B-Tree* sudah dibuat, maka dilakukan *Insertion* dari kata hasil proses *preprocessing* yang nanti menjadi *index* untuk proses pencarian. Proses pertama dari *search* adalah memasukkan *keyword* yang kemudian dijadikan sebagai *query* yang akan digunakan untuk pencarian di *database*. Kemudian dilakukan *stemming* terhadap *keyword* yang dimasukkan sehingga didapatkan *stemmed term* untuk di cek apakah *index* dari kata tersebut ada atau tidak pada *database*. Jumlah *index* pada *database* yang sesuai (*total matched index*) dengan masukan *keyword* akan digunakan untuk memunculkan *document set* berdasarkan *total matched index* terbanyak dan dengan total frekuensi kemunculan *index* dalam dokumen elektronik.

3.2 Implementasi

3.2.1 Pengumpulan Data

Dalam Tugas Akhir ini d*ataset* yang digunakan sebagai *input* dalam sistem ini adalah :

1) Artikel sampling secara acak dari Republika Online (www.republika.co.id) tanggal 1 Mei 2014 dan 2 Mei 2014
2) *Dataset* berjumlah 300 artikel
3) Artikel telah diubah dan disimpan ke format dokumen *Notepad Plain Text* (**.txt*) dengan isi dokumen menggunakan Bahasa Indonesia.

3.2.2 Implementasi Preprocessing

Pada bagian ini akan diberikan satu contoh yang memaparkan bagaimana proses *preprocessing* secara sederhana.

1) dokumen elektronik (dengan format file *.txt), dengan nama dokumen : "budi dan bapak budi.txt" dan isi dokumen adalah : Budi dan bapaknya pergi ke pasar membeli sepeda baru.
2) Scanning *words* (kata-kata) pada dokumen menggunakan fungsi php pada web
3) Didapatkan list kata hasil *scanning* sebagai berikut :

1. budi	4. pergi	7. membeli
2. dan	5. ke	8. sepeda
3. bapaknya	6. pasar	9. baru

4) List kata setelah proses *stopwords* adalah sebagai berikut :

1.budi	4.pasar
2.bapaknya	5.membeli
3.pergi	6.sepeda

5) List kata setelah proses *stemming* adalah sebagai berikut :

1.budi	4.pasar
2.bapak	5.beli
3.pergi	6.sepeda

6) Kata akan disimpan di *database* pada tabel *Word*

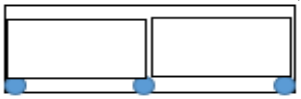
Tabel 2 Tabel word pada Database

word	*word_id*	*word_ count*	*doc_i d*
budi	1	1	0
bapak	2	1	0
pergi	3	1	0
pasar	4	1	0
beli	5	1	0
sepeda	6	1	0

3.2.3 Implementasi Struktur Database B-Tree

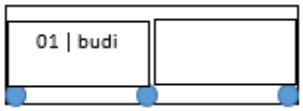
Berikut ini adalah contoh langkah-langkah implementasi struktur *database B-Tree* dengan *order* = 3 dan *Tree* diurutkan berdasarkan urutan masuknya :

1. Membuat *root B-Tree* baru (*Create B-Tree*)



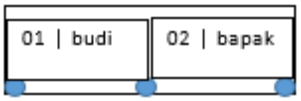
Gambar 6 Create New B-Tree Root

2. Melakukan insertion kata hasil preprocessing ke database
3. *Insert key for* "budi"



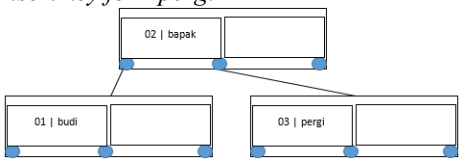
Gambar 7 Insertion word 1

4. *Insert key for"*bapak"



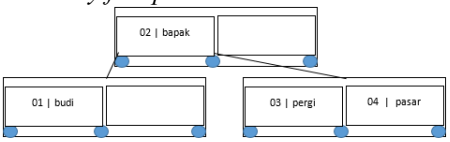
Gambar 8 Insertion word 2

5. *Insert key for "pergi"*



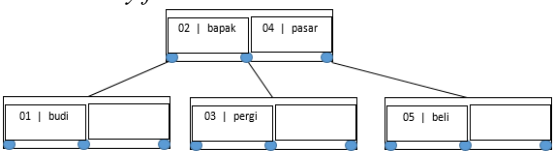
Gambar 9 Insertion word 3

6. *Insert key for"pasar"*



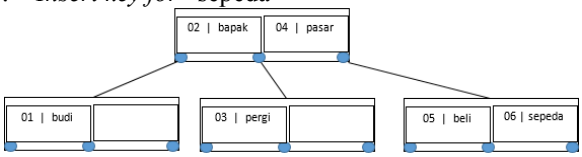
Gambar 10 Insertion word 4

7. *Insert key for"beli"*



Gambar 11 Insertion word 5

8. *Insert key for* "sepeda"



Gambar 12 Insertion word 6

4 Pengujian dan Analisis

4.1 Skenario Pengujian

1. Pengujian terhadap performansi kecepatan proses *indexing*
2. Pengujian terhadap performansi keakuratan keluaran dan kecepatan sistem pada proses *searching* dari suatu masukan *keyword* pencarian
3. Perbandingan performansi pembandingan efisiensi hasil proses *indexing* antara sistem yang menggunakan dan sistem tanpa menggunakan metode *Fulltext Indexing* maupun Algoritma struktur data *B-Tree*

4.2 Skenario1

Pada skenario 1 ini dilakukan pengujian terhadap performansi kecepatan proses *indexing* dari sistem. Jumlah dokumen yang digunakan pada skenario 1 ini adalah 50 dokumen, 100 dokumen, 150 dokumen, 200 dokumen, 250 dokumen, dan 300 dokumen.

4.2.1 Hasil Pengujian dan Analisis Skenario 1

Berikut ini adalah hasil pengujian dari skenario 1 yang dijelaskan pada tabel berikut ini :

Tabel 3 Tabel Hasil Skenario 2

jumlah dokumen	jumlah words	jumlah words ter-index	waktu pembuatan index (detik)	waktu pembuatan index (menit)
50	12205	2094	369.5136	6.1586
100	23033	3432	605.6211	10.0937
150	34376	4230	746.4386	12.4406
200	46634	5073	895.1969	14.9199
250	59456	5777	1019.4269	16.9904
300	71265	6355	1121.4226	18.6904
Rata-rata	41162	4493.5000	792.9366	13.2156

Dari tabel tersebut terlihat bahwa dari setiap jumlah dokumen yang di-*index* dengan jumlah *word* yang tersebut dalam tabel menghasilkan jumlah *word* ter-*index* yang rata-rata perbandingannya adalah **1 : 8,6** dimana **1** untuk jumlah *words* yang ter-*index* dan **8,6** untuk jumlah total *words* dari suatu kumpulan dokumen.

4.3 Skenario 2

Pada skenario 2 ini dilakukan pengujian terhadap performansi keakuratan keluaran dan kecepatan sistem pada proses *searching* dari suatu masukan *keyword* pencarian. Jumlah dokumen yang digunakan berjumlah 300 dokumen yang telah di-*index* dengan total *word* sebanyak 71265 dan jumlah masukan *keyword* pencarian sebanyak 12 buah.

4.3.1 Hasil Pengujian dan Analisis Skenario 2

Berikut ini adalah hasil pengujian dari skenario 2 yang dijelaskan pada tabel berikut ini :

Tabel 4 Tabel Hasil Skenario 2

jumlah keyword	*Precision*	*Recall*	waktu pencarian (detik)
1	1	1	0.0100
1	1	1	0.0097
1	1	1	0.0098
2	0.75	1	0.0186
2	0.6667	1	0.0145
2	0.5714	1	0.0150
3	1	1	0.0142
3	0.25	1	0.0153
3	0.2222	1	0.0141
4	0.1428	1	0.0140
4	0	0	0.0092
4	0.08333	1	0.0157

Tabel 4 menunjukkan nilai *Precision, Recall* dan, *Execution Time* terhadap *query* atau *keyword* pencarian. Nilai tersebut diperoleh dari pengujian dengan *dataset* dokumen yang memiliki kerelevanan acak dengan *keyword*. Dari Tabel 4 nilai Precision paling baik adalah 1 dengan waktu pencarian 0,0097 detik saat jumlah keyword pencarian 1. Saat jumlah keyword 2 hasil Precision paling baik adalah 0,75 dengan waktu pencarian 0,0186 detik. Saat jumlah keyword 3 hasil Precision paling baik adalah 0,25 dengan waktu pencarian 0,0153 detik sedangkan untuk jumlah keyword 4 hasil Precision paling baik adalah 0,1428 dengan waktu pencarian 0,0140 detik. Nilai *Precision* dan *Recall* yang kecil disebabkan oleh sedikitnya dokumen relevan yang ada karena *dataset* sebanyak 300 dokumen di-*sampling* secara acak sehingga memiliki konten relevansi yang acak. Dimana kondisi saat jumlah *keyword* lebih dari 2 rata-rata hanya memiliki 1 sampai 2 dokumen yang relevan.

Skenario 3

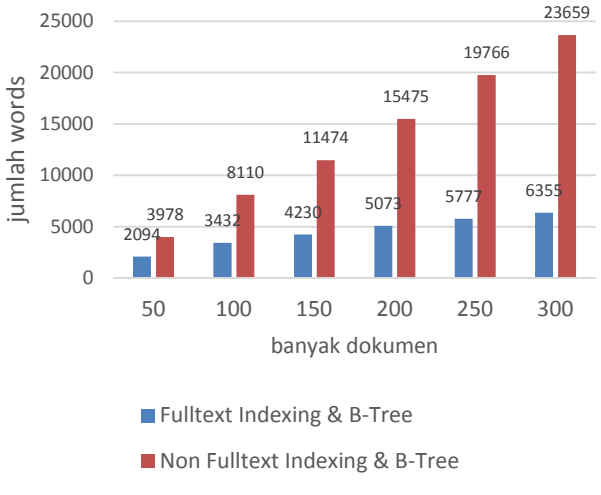
Pada skenario 3 ini dilakukan pembandingan efisiensi hasil proses *indexing* antara sistem yang menggunakan dan sistem tanpa metode *Fulltext Indexing* maupun Algoritma struktur data *B-Tree* dengan jumlah dokumen yang digunakan untuk proses *indexing* adalah 50 dokumen, 100 dokumen, 150 dokumen, 200 dokumen, 250 dokumen, dan 300 dokumen. Terdapat dua Execution Time pada Skenario 3 ini, yaitu waktu yang dibutuhkan sistem untuk melakukan proses indexing dan waktu saat melakukan proses searching.

4.3.2 Hasil Pengujian dan AnalisisSkenario 3

Berikut ini adalah hasil pengujian dari skenario 3 yang dijelaskan pada tabel berikut ini :

Tabel 5Perbandingan Sistem dengan dan tanpa *Fulltext Indexing* dan Struktur *B-Tree*

jumlah dokumen	Fulltext Indexing & B-Tree	Non Fulltext Indexing & B-Tree	Fulltext Indexing & B-Tree	Non Fulltext Indexing & B-Tree
	words		waktu pembuatan index (detik)	
50	2094	3978	369.5136	701.9700
100	3432	8110	605.6211	1431.115
150	4230	11474	746.4386	2024.737
200	5073	15475	895.1969	2730.765
250	5777	19766	1019.426	3379.598
300	6355	23659	1121.4226	4095.1793
Rata-rata	4493.500	13743.6667	792.9366	2393.894

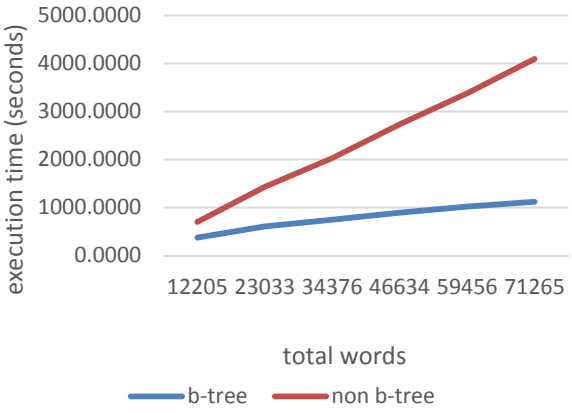


Dari Tabel perbandingan sistem di atas dapat dibentuk grafik :

Gambar 13 Perbandingan Execution Time B-Tree dan Non B-Tree

Gambar 14 Perbandingan Execution Time B-Tree dan Non B-Tree

Gambar 13 merupakan grafik yang menunjukkan jumlah *words* yang ter-*index* pada sistem saat menggunakan dan tidak menggunakan *Fulltext Indexing* maupun struktur *B-Tree*. Dari grafik di atas terlihat jelas bahwa banyak *words* yang ter-*index* saat menggunakan kedua metode tersebut jumlahnya relatif sedikit dengan penambahan yang stabil sedangkan pada sistem yang tidak menggunakan *Fulltext Indexing* maupun struktur *B-Tree* jumlahnya banyak dengan penambahan

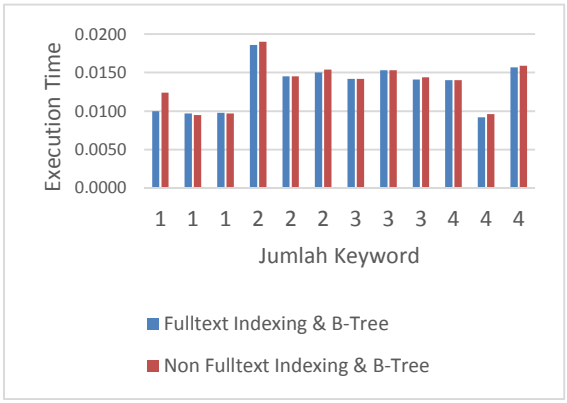


yang sebanding dengan total *words* dari berbagai jumlah dokumen yang telah ditentukan. Rata-rata nilai perbandingan jumlah *words* dari total dokumen yang ditentukan dengan jumlah *words* yang ter-*index* antara kedua sistem adalah 8,6 dan 3. Terlihat jelas bahwa untuk sistem tanpa menggunakan *Fulltext Indexing* maupun struktur *B-Tree* perbandingannya mencapai 2,9 yang berarti untuk rata-rata 9 kata yang akan ter-*index* adalah 3 kata. Dimana kondisi semakin banyak *words* yang masuk *index* akan semakin lama pula proses *indexing*-nya seperti yang ditampilkan pada Gambar 14 yang merupakan grafik perbandingan

lama waktu *Execution Time* dimana untuk sistem *B-Tree* kecepatan *execution time* untuk jumlah word yang sama dengan yang diproses sistem non *B-Tree* hasilnya adalah lebih cepat. Dan sistem yang menggunakan *Fulltext Indexing* dan struktur *B-Tree* hasilnya lebih efisien dalam melakukan *indexing* yaitu untuk setiap rata-rata 9 kata berbanding dengan 1 kata ter-*index* sehingga waktu yang dibutuhkan sistem untuk melakukan *indexing* menjadi lebih cepat

Tabel 6 Perbandingan Precision, Recall, dan Execution Time Searching

jumlah keyword	Precision		Recall		waktu pencarian (detik)	
	Fulltext Indexing & B-Tree	Non Fulltext Indexing & B-Tree	Fulltext Indexing & B-Tree	Non Fulltext Indexing & B-Tree	Fulltext Indexing & B-Tree	Non Fulltext Indexing & B-Tree
1	1.0000	1.0000	1.0000	1.0000	0.0100	0.0124
1	1.0000	1.0000	1.0000	1.0000	0.0097	0.0095
1	1.0000	1.0000	1.0000	1.0000	0.0098	0.0097
2	0.7500	0.7000	1.0000	1.0000	0.0186	0.0190
2	0.6667	0.6667	1.0000	1.0000	0.0145	0.0145
2	0.5714	0.4684	1.0000	1.0000	0.0150	0.0154
3	1.0000	0.8750	1.0000	1.0000	0.0142	0.0142
3	0.2500	0.2500	1.0000	1.0000	0.0153	0.0153
3	0.2222	0.2222	1.0000	1.0000	0.0141	0.0144
4	0.1428	0.1376	1.0000	1.0000	0.0140	0.0140
4	0.0000	0.0000	0.0000	0.0000	0.0092	0.0096
4	0.0833	0.0567	1.0000	1.0000	0.0157	0.0159
Rata	0.5572	0.5330	0.9167	0.9167	0.0133	0.0137



Gambar 15 Perbandingan Execution Time Searching

Gambar 15 menunjukkan perbandingan lama Execution Time antara sistem yang menggunakan dan tanpa Fulltext Indexing maupun B-Tree dimana untuk setiap jumlah keyword yang dimasukkan, lama Execution Time kedua sistem berada di bawah 0,02 detik. Untuk sistem yang menggunakan Fulltext Indexing dan B-Tree nilai Execution Time tercepatnya adalah 0,0092 detik dan yang terlama adalah 0,0186 detik sedangkan untuk sistem yang tidak menggunakan Fulltext Indexing maupun B-Tree nilai Execution Time tercepatnya adalah 0,0095 detik dan yang terlama adalah 0,0190 detik. Dari Gambar 15 dapat disimpulkan bahwa untuk sistem yang menggunakan Fulltext Indexing dan B-Tree memiliki rata-rata nilai Execution Time lebih cepat dibandingkan dengan sistem yang tidak menggunakan Fulltext Indexing maupun B-Tree.

5. Penutup

5.1 Kesimpulan

Berdasarkan hasil pengujian yang diperoleh dan analisis yang telah dilakukan dapat diberikan kesimpulan sebagai berikut :

1. Lama waktu proses *Indexing* berbanding lurus dengan jumlah *words* dari jumlah dokumen elektronik yang ditentukan.
2. Kecepatan proses *Indexing* dan *Searching* sistem relatif stabil dengan penambahan *execution time* yang juga stabil
3. Nilai *Precision* dan *Recall* yang relatif kecil dipengaruhi oleh sedikitnya dokumen relevan yang ada karena 300 dokumen *dataset* di-*sampling* secara acak sehingga memiliki rasio relevansi konten yang sedikit dengan kondisi saat jumlah *keyword* lebih dari 2 rata-rata hanya memiliki 1 sampai 2 dokumen yang relevan.
4. Penggunaan *Fulltext Indexing* dan struktur data *B-Tree* pada proses pengelolaan *database* dokumen elektronik mempercepat sistem saat melakukan *indexing* dan *searching* dimana kedua metode tersebut mempercepat sistem 0,3 kali dari sistem sebelumnya dengan perbandingan jumlah *word* yang ter-*index* dengan jumlah total *word* dari dokumen yang ditentukan adalah 1:8,6.

5.2 Saran

Adapun saran yang dapat diberikan mengenai penelitian dan pengembangan lebih lanjut adalah sebagai berikut :

1. Untuk penelitian lebih lanjut, metode penelitian dapat menggunakan metode *indexing* maupun struktur *database* lain seperti *Inverted Index,PAMs,* atau *SAMs* dengan struktur B^{+}-*Tree, k-d-tree, k-d-B-tree,* atau *hB-tree.*
2. Untuk penelitian lebih lanjut, pemilihan *dataset* yang digunakan di kategorikan terlebih dahulu dan dalam jumlah yang lebih banyak. *Dataset* dapat menggunakan format *file* dokumen elektronik lain seperti *PDF, DOC, DOCX, RTF*, atau *GDOC*
3. Untuk penelitian selanjutnya menggunakan algoritma *stemming* untuk Bahasa Indonesia yang lebih baik sehingga kesalahan pembentukan kata dasar menjadi lebih sedikit

4. Untuk penelitian selanjutnya membuat sistem yang dapat secara otomatis meng-*update* dan meng-*insert Index* baru pada saat suatu dokumen elektronik baru akan diunggah atau dimasukkan ke dalam *database.*

DAFTAR PUSTAKA

1] A. Goker, D. John. 2009. *Information Retrieval : Searching in 21st Century*, Chippenham: Wiley.

2] A. Mushofan. 2013. B-Tree dan Penerapan di Basis Data, Sekolah Teknik Elektro dan Informatika, ITB, Bandung.

3] D. Comer. 1979. *Ubiquitous B-Tree*, ACM Computing Surveys (CSUR), pp. 121-137, June 1979.

4] D. Zhang. 2014. *B-Trees*. CRC Press.

5] E. Chaves. 2006. *A Universal Full Text Index with Access Control and Annotation Driven Information Retrieval*, Computing, 2006. CIC '06. 15th International Conference on, pp. 135-140, November 2006.

6] E. Soisalon-Soininen, P. Widmayer. 1999. *Concurrency and Recovery in Full-Text Indexing*, String Processing and Information Retrieval Symposium, pp. 192-198.

7] G. Graefe. 2010. *A survey of B-tree locking techniques*, Transactions on Database Systems (TODS), January 2010.

8] G. Graefe. 2011. *Modern B-Tree Techniques*, Foundations and Trends in Databases, pp. 203-402.

9] J. Lian. 2009. *An Efficient Approach for Building Compressed Fulltext Index for Structured Data*, Computer Sciences and Convergence Information Technology. ICCIT '09. Fourth International Conference on, pp. 59-63, November 2009.

10] M. Buckland. 1998. *What is "digital document"?*, University of California, California.

11] P. Koruga, M. Bača. 2010. *Analysis of B-tree data structure and its usage in computer forensics*, Faculty of Organization and Informatics, Univercity of Zagreb, Zagreb.

12] Parliament of Canada. 2008. *Personal Information Protection and Electronic Documents Act*, Amandement Juli 2014. Canada.

13] Pemerintah Republik Indonesia. 2008. Undang-undang Informasi dan Transaksi Elektronik. Indonesia Patent 11.

14] R. Baeza-Yates, B. Ribeiro-Neto. 1999. *Modern Information Retrieval*, New York: Association for Computing (ACM) and Addison Wesley Longman.

15] R. Bayer, E. McCreight. 1972. *Organization and Maintenance of Large Ordered Indexes*, Acta Informatica, pp. 173-189.

16] R. S. J. Meier. 1996. *Towards a better understanding of electronic document management*, System Sciences, Proceedings of the Twenty-Ninth Hawaii International Conference , (Volume:5), pp. 53-61, January 1996.

17] T. Ylonen. 1992. *An Algorithm for Full Text Indexing*. Helsinki.

18] Y. Wang, B.-y. Sun, F. Cheng. 2011. *Electronic-Document-Based Management Process Model for Image Archives in Universities*, International Conference of Information Technology, Computer Engineering and Management Sciences..