Penerapan Pohon Untuk Algoritma Pencarian Kata Pada *Inverted File* Dalam Sistem Temu Balik Informasi

Ibnu Hikam – NIM: 13505038

Program Studi Informatika, Institut Teknologi Bandung
Jl.Ganesha 10, Bandung 40135,
email: if15038@students.if.itb.ac.id

Abstract – *Makalah ini membahas penerapan pohon yang merupakan graf tidak berarah terhubungkan dan tidak mengadung sirkuit untuk mengoptimasi algoritma pencarian pada inverted file dalam sistem temu balik informasi.*

Konsep ataupun teori mengenai pohon digunakan sebagai konsep optimalisasi pencarian kata dan pengambilan nilai kemunculan kata tersebut dari inverted file yang merupakan daftar kata-kata hasil pengekstrakan dokumen beserta nilai kemunculannya pada sistem temu balik informasi. Optimalisasi menggunakan konsep pohon ini merupakan alternatif dari pencarian kata secara sekuensial yang membutuhkan komputasi yang berat atau brute force menjadi lebih efektif dan efisien.

Pencarian kata dan nilai kemunculan itu sendiri sangat penting dalam sistem temu balik informasi yang menemukan kembali dokumen-dokumen yang relevan dengan kebutuhan informasi pengguna atau paling tidak menemukan kembali dokumen-dokumen yang mengandung kata yang diberikan pengguna atau biasa disebut query. Dan nilai kemunculan kata tersebut akan digunakan sebagai masukan komputasi untuk mendapatkan nilai similaritas atau tingkat kesamaan antara dokumen dan query sehingga dokumen-dokumen yang tinggi nilai similaritasnya akan dikembalikan oleh sistem temu balik informasi sebagai dokumen-dokumen yang dianggap relevan terhadap kebutuhan informasi pengguna.

Di dalam makalah ini akan dipelajari bagaimana konsep pohon diterapkan atau diimplementasiakn dalam proses pencarian kata dan nilai kemunculan kata tersebut pada inverted file dalam sistem temu balik informasi.

Kata Kunci*: pohon, inverted file, sistem temu balik informasi, nilai kemunculan, bruto force, query, dokumen, similaritas.*

1. PENDAHULUAN

Pada saat ini informasi sangatlah mudah didapat salah satunya adalah dari internet kita dapat mendapatkan informasi yang sangat luas. Dengan semakin bertambahnya informasi, pendayagunaan sistem temu balik informasi menjadi penting adar dapat menghemat waktu dan kerja untuk mendapatkan informasi yang terkandung di dalam dokumen-dokumen tersebut. Misalnya pencarian dokumen-dokumen yang relevan terhadap kebutuhan informasi pengguna.

Pada prinsipnya, penyimpanan informasi dan proses pencarian kembali informasi tersebut sifatnya sederhana, selama ada kumpulan dokumen yang disimpan dan pengguna yang memberikan pertanyaan ataupun kebutuhan [1]. Maka sistem temu balik informasi dapat mengembalikan kumpulan dokumen yang dianggap relevan dengan menghitung *similarity* atau tingkat kesamaan antara dokumen dengan *query* yang diberikan.

Hasil pencarian q*uery* atau kombinasi kata yang diberikan pengguna dikembalikan oleh sistem temu balik informasi ketika kata-kata tersebut ditemukan pada kumpulan dokumen. Sehingga jumlah kemunculan dari *query* pada tiap dokumen dan posisi rinci kata tersebut juga diperlukan. Oleh karena itu, lalu digunakanlah algoritma pencarian kata secara sequensial dan *pattern matching* karena sederhana dan mudah diimplementasikan. Namun yang menjadi tren kunci kemudian adalah bagaimana menstrukturkan dokumen teks dan teknik-teknik pengompresan dokumen.

Pilihan yang paling sederhana untuk algoritma pencarian adalah melakukan *scanning* pada teks secara sekuensial. Namun kemudian pada implementasinya algoritma ini hanya cocok jika dokumen koleksi berukuran kecil saja, tidak cocok digunakan untuk koleksi dokumen yang berukuran besar atau banyak.

Sehingga kemudian dipertimbangkanlah untuk membangun struktur data pada koleksi dokumen yang disebut indeks untuk mempercepat proses pencarian. Perubahan ini sangatlah memuaskan dan mampu

meningkatkan performansi pencarian sehingga lebih cepat untuk koleksi dokumen yang besar dan banyak jumlahnya.

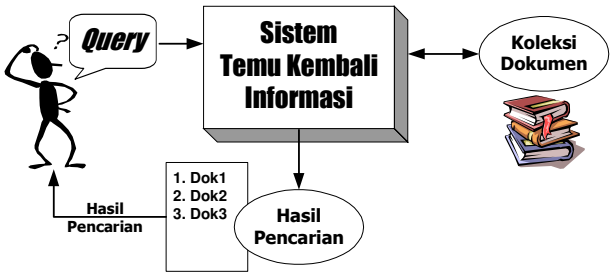
Sehingga konsep inilah yang melatar belakangi kesuksesan pembuatan mesin pencari web sekarang ini. Dan tercatat bahwa sampai sekarang teknik ini sukses diimplementasi pada dokumen koleksi yang cukup besar atau sekitar berukuran 200Mb. Dan implementasi dari teknik pengindeksan ini berupa *inverted file* yang terdiri dari daftar kata-kata yang telah diekstraksi, nilai kemunculan ataupun posisi rincinya. Dan struktur pohon sangatlah cocok jika diimplementasikan untuk membantu algoritma pencarian kata pada *inverted file* sehingga diharapkan lebih efektif dan efisien.

2. DASAR TEORI

Beberapa dasar teori yang berkaitan dalam penerapan pohon untuk algoritma pencarian kata pada inverted file dalam sistem temu balik informasi ini antara lain:

2.1. Sistem temu balik informasi

Sistem temu balik informasi (*information retrieval system)* adalah sistem yang menemukan kembali (*retrieve)* informasi-informasi yang relevan terhadap kebutuhan pengguna dari suatu kumpulan informasi secara otomatis [2]. Hal ini dapat dengan jelas dipahami dengan mengamati gambar 1.



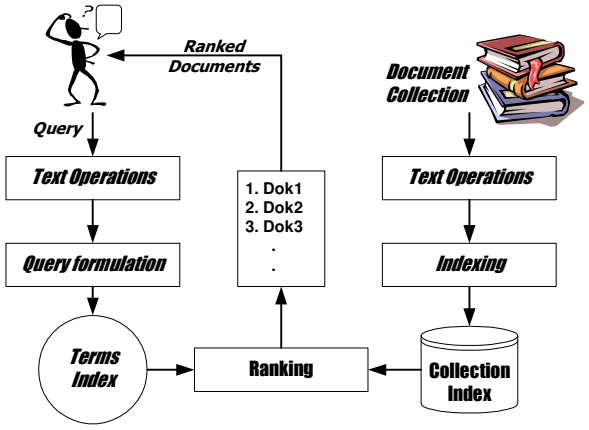
Gambar 1: Ilustrasi sistem temu balik informasi

Salah satu aplikasi dari sistem temu balik informasi adalah search engine atau mesin pencarian yang terdapat pada jaringan internet. Pengguna dapat mencari halaman-halaman web yang dibutuhkannya melalui *search engine.* Contoh lain penerapan dari sistem temu balik informasi adalah sistem informasi perpustakaan, data/*text mining, knowledge-acquisition,* dan lain sebagainya.

Salah satu aplikasi dari sistem temu balik informasi adalah search engine atau mesin pencarian yang terdapat pada jaringan internet. Pengguna dapat mencari halaman-halaman web yang dibutuhkannya melalui *search engine.* Contoh lain penerapan dari sistem temu balik informasi adalah sistem informasi perpustakaan, data/*text mining, knowledge-acquisition,* dan lain sebagainya.

Sistem temu kembali informasi terutama berhubungan dengan pencarian informasi yang isinya tidak memiliki struktur. Demikian pula ekspresi kebutuhan pengguna yang disebut *query*, juga tidak memiliki struktur. Hal ini yang membedakan sistem temu kembali informasi dengan sistem basis data. Dokumen adalah contoh informasi yang tidak terstruktur. Isi dari suatu dokumen sangat tergantung pada pembuat dokumen tersebut.

Sebagai suatu sistem, sistem temu kembali informasi memiliki beberapa bagian yang membangun sistem secara keseluruhan. Gambaran bagian-bagian yang terdapat pada suatu sistem temu kembali informasi digambarkan pada gambar 2.



Gambar 2: Bagian-bagian Sistem Temu Kembali Informasi

Pada makalah ini yang patut dikaji dari sistem temu balik informasi ini adalah pada bagian *indexing*. *Indexing* atau pengindeksan, membangun basis data indeks dari koleksi dokumen. Dilakukan terlebih dahulu sebelum pencarian dokumen dilakukan. Pengindeksan ini dapat diimplementasikan dalam berbagai cara yaitu misalnya dengan menggunakan tabel-tabel dan kolom-kolom basis data dengan menggunakan fasilitas yang sudah ada seperti mysql, oracle, dll. Namun cara implementasi ini sangatlah membebani program pada saat dijalankan karena mengkonsumsi proses yang yang sangat besar karena menggunakan *library* yang sudah disediakan sehingga kemudian cara seperti ini tidak banyak digunakan karena dinilai tidak efektif dan efisien dalam penggunaanya. Karena sistem basisdata yang demikian itu terlalu besar lingkupnya untuk dipakai pada inverted file ini yang hanya menghabiskan beberapa tabel dan kolom saja.

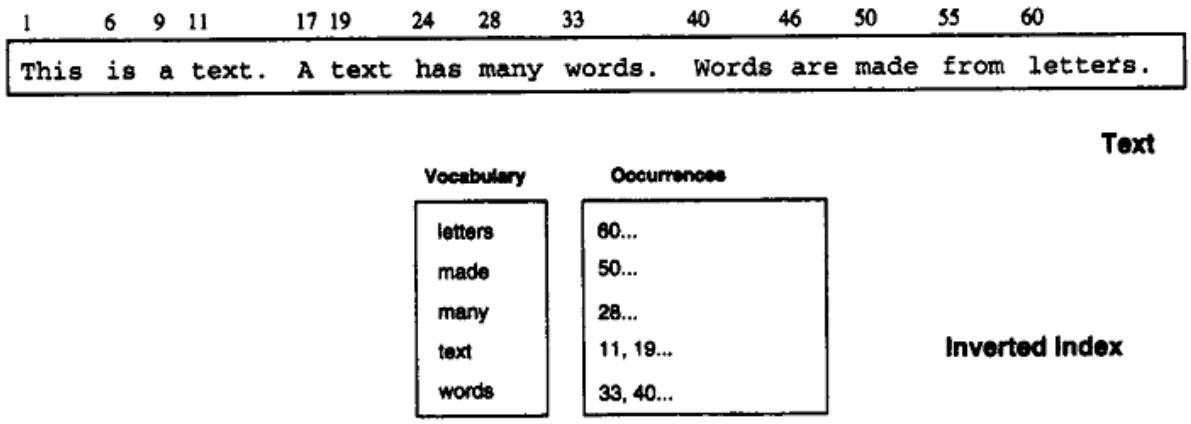
Lalu kemudian dipertimbangkan teknik implementasi pengindeksan tersebut dalam suatu *inverted file* yang menggunakan *file text* yang tersrtuktur yang diharapkan lebih efisien karena tidak memakan proses yang besar seperti basisdata yang sudah disebutkan diatas.

2.2. Inverted File

Inverted file atau inverted index adalah mekanisme untuk pengindeksan kata dari koleksi teks yang digunakan untuk mempercepat proses pencarian [3].

Struktur *inverted file* terdiri dari dua elemen, yaitu: kata (*vocabulary*) dan kemunculan (*occurences).* Kata-kata tersebut adalah himpunan dari kata-kata yang ada pada teks, atau merupakan ekstraksi dari kumpulan teks yang ada.

Dan tiap kata terdapat juga informasi mengenai semua posisi kemunculannya secara rinci. Jumlah dari posisi-posisi inilah yang dimaksudkan dengan nilai kemunculan atau *occurences*. Posisi dapat merefer kepada posisi kata ataupun karakter. Hal ini dapat dilihat dengan jelas dengan memperhatikan gambar 3.

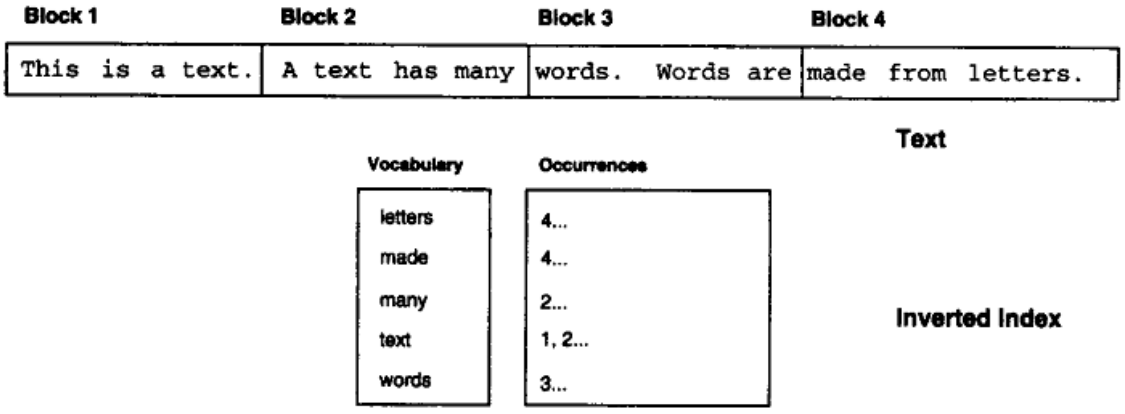


Gambar 3: Contoh 1 teks dan *inverted file*-nya.

Pada gambar 3 kata-kata dikonversikan menjadi karakter huruf kecil (*lower-case).* Kolom vocabulary adalah kata-kata yang telah diekstraksi dari dari koleksi teks, sedangkan occurences adalah posisi kemunculan pada teks. Struktur inverted file seperti ini masihlah sangat sederhana sehingga berikutnya muncul beberapa masalah.

Nilai kemunculan dari kata-kata memerlukan ruangan (*space)* yang tidak sedikit, karena tiap kata muncul pada teks sekali pada struktur *occurences,* sehingga ada ruangan ekstra atau dilambangkan dengan *O(n*). Walaupun tidak semua kata diindekskan karena ada kata-kata *stopword* yang dibuang, overhead yang muncul akibat penambahan indeks ini sampai mencapai 30% sampai dengan 40% dari ukuran besar koleksi teks.

Sehingga kemudian untuk mengurangi kebutuhan ruangan yang besar tersebut, maka digunakanlah teknik yang disebut *block addressing.* Teknik pengindeksan ini sama seperti teknik klasik sebelumnya yang disebut *full inverted indices* [3], karena tetap sama elemennya yaitu *vocabulary* dan *occurences.* Namun perbedaan yang ada dan membuat teknik ini lebih unggul adalah pada pengalamatanya yang tidak satu-satu pada tiap kataseperti yang dilakukan oleh teknik yang klasik, namun pengalamatannya berdasarkan blok-blok tertentu yang sudah didefiniskan. Hal ini akan lebih jelas dengan memperhatikan gambar 4.



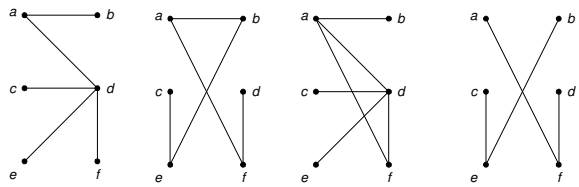
Gambar 4: Contoh 2 teks dan *inverted file*-nya.

Dengan teknik ini kebutuhan ruangan untuk membuat tambahan pengindeksan akan lebih berkurang, karena dapat dipastikan bahwa jumlah blok akan lebih sedikit dibandingkan dengan jumlah keseluruhan kata. Secara eksperimental hanya diperlukan 5% dari koleksi teks untuk membuat tambahan pengindeksan dengan teknik ini [3]. Sungguh efisien dalam penggunaan ruangan atau *space demand.*

Namun *trade off* yang terjadi adalah pada tiap kali me-*retriev* kata maka yang akan di tunjuk adalah alamat blok kata tersebut. Sehingga harus dilakukan iterasi berikutnya pada blok tersebut untuk menemukan kata yang dimaksud. Tapi *trade off* ini tidak perlu dikhawatirkan karena tidak begitu banyak berpengaruh terhadap sistem karena hanya merupakan komputasi perbandinga sederhana jika dibandingkan efek positif yang sangat baik karena mampu mengefisienkan ruangan yang dibutuhkan untuk pengindeksan.

2.3. Pohon

Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit [4]. Sekilas mengenai pohon dapat secara jelas dipahami dengan memperhatikan gambar 5.



a. b. c. d.

Gambar 5: a dan b adalah pohon, sedangkan c dan d bukanlah pohon

Karena definisi pohon tersebut diacu dari teori graf, maka sebuah pohon dapat mempunyai sebuah simpul tanpa sebuah sisipun. Dengan kata lain, jika G = (V,E) merupakan sebuah pohon, maka V tidak boleh berupa himpunan kosong, tetapi E boleh merupakan himpunan kosong.

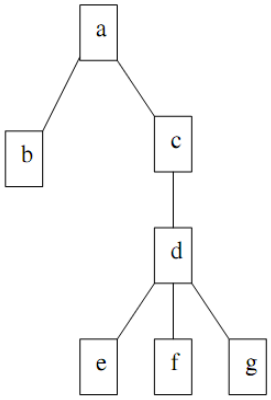
Berdasarkan definisi tersebut, ada dua sifat penting pada pohon yaitu terhubung dan tidak mengandung sirkuit. Terhubung artinya pada setiap pasang simpul pada pohon terdapat lintasan yang menghubungkan. Tidak mengandung sirkuit berarti tidak terdapat lebih dari satu lintasan yang menghubungkan setiap pasang simpul pada pohon.

Selain itu, dapat terlihat bahwa di dalam pohon, jumlah sisinya adalah jumlah simpul dikurangi satu. Terlihat pula bahwa pohon hanya memerlukan dua buah warna untuk mewarnai simpul-simpul di dalam pohon sedemikian rupa sehingga tidak ada dua buah simpul bertetangga yang mempunyai warna sama. Dengan kata lain, ditinjau dari teori pewarnaan graf, maka pohon mempunyai bilangan kromatik sama dengan 2.

2.4. Pohon Berakar

Pohon yang sebuah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah dinamakan pohon berakar [4].

Akar mempunyai derajat-masuk sama dengan nol dan simpul-simpul lainnya berderajat-masuk sama dengan satu. Simpul yang mempunyai derajat-keluar sama dengan nol disebut daun atau simpul terminal. Simpul yang mempunyai derajat-keluar tidak sama dengan nol disebut simpul dalam atau simpul cabang. Setiap simpul di dalam pohon dapat dicapai dari akar dengan sebuah lintasan tunggal (unik). Untuk lebih jelasnya perhatikan gambar 6.



Gambar 6. Contoh pohon berakar

Gambar 6 di atas (gambar pohon berakar dengan simpul a sebagai akar) akan digunakan untuk menjelaskan terminologi pohn berakar.

Anak (*child*) dan Orangtua (*parent*). Misalkan d adalah sebuah simpul di dalam pohon berakar. Simpul e dikatakan anak simpul d jika ada sisi dari simpul d ke simpul e. Dalam hal ini, d merupakan orangrua dari e. Pada gambar 1, simpul b, e, f, dan g tidak mempunyai anak.

Saudara Kandung (*sibling*). Simpul yang mempunyai orangtua yang sama dikatakan merupakan saudara kandung satu sama lain. Pada gambar 1, simpul b dan c merupakan saudara kandung satu sama lain.

Lintasan (path). Lintasan adalah runtunan simpul-simpul dari simpul awal sampai simpul tujuan. Panjang lintasan adalah jumlah sisi yang

dilalui dalam suatu lintasan.

3. PENERAPAN TEORI POHON PADA ALGORITMA PENCARIAN

Algoritma pencarian pada *inverted file* terdiri dari tiga langakah umum, yaitu:

1. *Vocabulary search*, kata-kata yang ada pada *query* dicari pada *vocabulary*, dan frase pun dipecah menjadi dua buah kata yang berbeda.

2. *Retrieaval of occurrences,* daftar kemunculan seluruh kata yang ditemukan di *retriev.*

3. *Manipulation of occurrences,* nilai kemunculan diproses sebagai kombinasi perhitungan *similarity* antara dokumen dengan *query.*

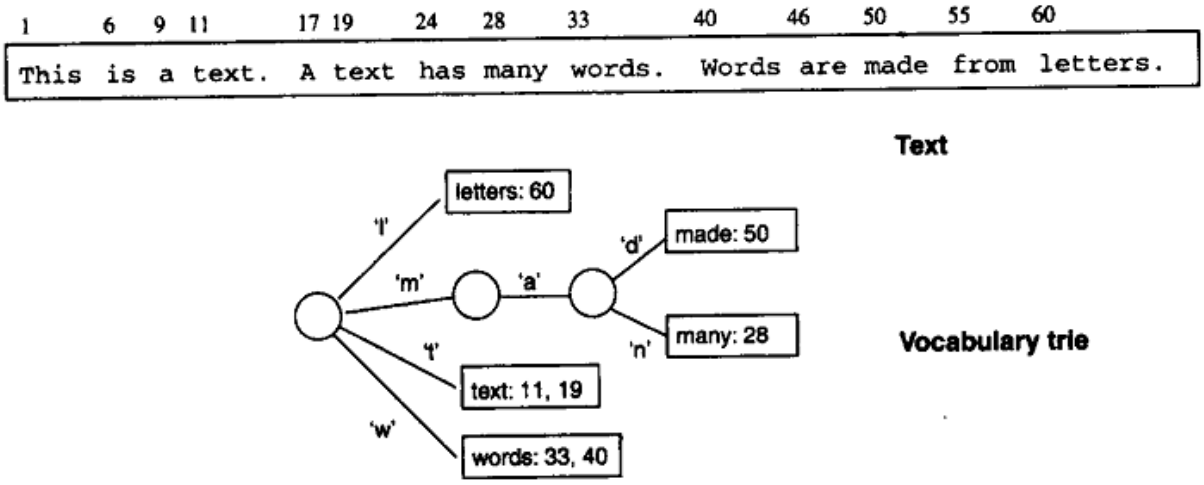
Oleh karena itu, pencarian pada *inverted file* selalu dimulai pada *vocabulary.* Maka membagi *file* ini menjadi beberapa *file* yang terpisah merupakan ide yang baik, sehingga sebesar apapun koleksi teks tersebut maka akan tetap tertampung atau dapat ditangani oleh memori utama.

Suatu kata tunggal dapat dicari menggunakan beberapa macam struktur data yang sesuai yang dapat mempercepat pencarian, seperti *hashing,* pohon, ataupun pohon *binary.* Dua yang pertama dari struktur data tersebut memiliki biaya sebesar *O(n),* sedangkan untuk pohon *binary* memiliki biaya sebesar *O(log n).*

3.1. Penerapan Struktur Data Pohon

Berikut akan dikaji penerapan pohon untuk pencarian kata pada inverted file.

Membangun dan mengelola *inverted file* adalah membutuhkan biaya proses yang relatif kecil. *Inverted file* dengan struktur data pohon untuk pencarian kata pada prinsipnya dapat mencari kata yang memiliki *n* karakter dengan biaya waktu *O(n).* Setiap kata pada *vocabulary* disimpan dalam struktur pohon dan setiap kata memiliki nilai *occurrences* masing-masing. Semua kata sudah tersusun rapi dan siap di retrieve. Jika suatu kata tidak diketemukan pada *vocabulary* maka keterangan ini dapat dimasukkan kedalam pohon sebagai kata yang *occurrences*-nya nol. Hal ini bisa dipahami secara jelas dengan mengamati gambar 7.



Gambar 7: Contoh teks dan *inverted file*-nya dengan struktur data pohon

Dengan skema ini *file* disebut juga '*posting file*' [3]. Pada file tersebut vocabulary tersusun secara terurut menurut abjad. Dan untuk tiap kata,terdapat juga pointer untuk tiap *list* nya. Hal ini memungkinkan *vocabulary* untuk disimpan pada memory pada tiap pencarian di tiap kasus pencarian. Kemudian, nilai kemunculan dapat dengan segera diketahui dengan hanya sedikit biaya tambahan saja.

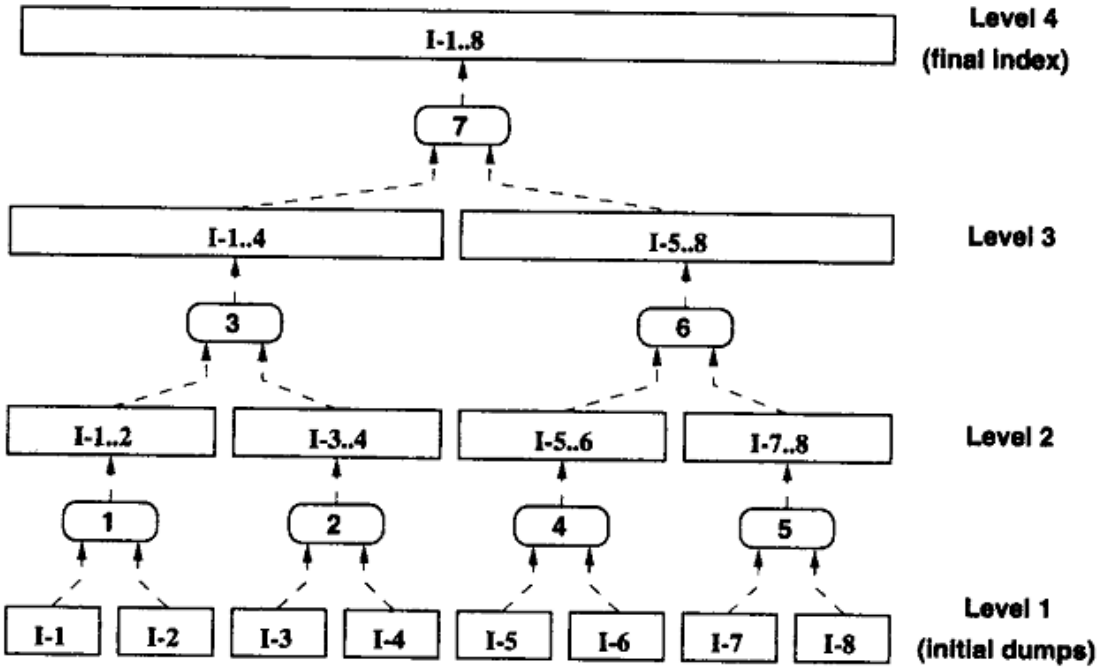
Namun mekanisme diatas tidak digunakan dalam praktek koleksi teks yang berukuran besar karena indeks tidak cukup semuanya di masukkan kedalam memori utama. Hal ini bisa sajadisiasati dengan cara melakukan *paging.* Namun teknik *paging* akan menurunkan performansi algoritma. Algoritma ini benar-benar memakan *resource* memori untuk me-*load* semua indeks pointer tiap list tiap kata, hal ini menyebabjan memori utama mengalami *exhausted.*

.Untuk selanjutnya akan dikaji alternatif cara untuk mengatasi hal ini dengan menggunakan struktur data pohon *binary*.

3.1. Penerapan Struktur Data Pohon *Binary*

Pohon *binary* adalah pohon yuang tiap simpulnya berderajat sama yaitu dua. struktur data ini akan dimanfaatkan untuk mengurangi penggunaan memori utama yang sangat berlebihan pada struktur data sebelumnya.

Sebelumnya indeks dibagi-bagi menjadi beberapa bagian, misalkan terdapat *Ii* pada *disk*. Maka indeks-indeks parsial ini akan digabungkan secara hierikal mengikuti struktur data pohon *binary.* Indeks I1 dan I2 digabungkan menjadi indeks I1..2, Indeks I3 dan I4 digabungkan menjadi indeks I3..4 dan begitu seterusnya. Jika semua telah digabungkan dengan cara tersebut indeks I1..2 akan digabungkan dengan I3..4 begitu pula yang lainnya mengikuti sampai tergabung semluruhnya. hal ini diilustrasikan pada gambar 8.



Gambar 8: Contoh teks dan *inverted file*-nya dengan struktur data pohon

Menggabungkan dua buah indeks berarti termasuk juga menggabungkan *vocabulary* yang sudah terurut. Dengan struktur ini maka indeks yang ada akan lebih sedikit jumlahnya yang logikanya sama dengan membadingkan posisi tiap kata menjadi beberapa blok. Karena indeks parsial akan lebih kecil jumlahnya daripada jumlah keseluruhan indeks kata. Sehingga dengan adanya operasi penggabungan ini makan untuk pencarian indeks yang memerlukan penggabungan maka biayanya adalah konkatinasi dari biaya komputasi teknik sebelumnya yaitu menjadi *O(n1+n2).* Biaya tambahan ini masih bisa ditolerir karena ada *trade off* yang sangat signifikan yaitu mengurangi konsumsi memori utama sehingga tidak terjadi *exhaust.*

3. HASIL DAN PEMBAHASAN

Hasil uji coba yang dilakukan [3], untuk full inverted index yang memiliki koleksi teks berukuran 250Mb maka didapatkan waktu pencarian untuk kata sederhana adalah 0.08 detik, sedangkan untuk frase adalah membutuhkan waktu sebesar 0.25 sampai dengan 0.35 detik.

Sedangkan hasil uji coba pada *inverted file indices* [3], didapatkan hasil yang menurun dari teknik sebelumnya yaitu 4-8Mb/menit untuk koleksi diatas 1 Gb. Penurunan ini disebabkan oleh faktor text yang berkembang dan 20%-30% digunakan untuk mengagabungkan indeks-indeks parsial.

Namun penurunan kecepatan tersebut dapat dimaklumi dan masih bisa ditolerir karena ada *trade off* yang besar yaitu penggunaan memori utama yang lebih efisien sehingga mampu menangani jumalah koleksi teks yang sangat besar sampai dengan 1Gb.

4. KESIMPULAN

Melalui makalah ini dapat kita simpulkan bahwa penggunaan konsep pohon untuk *inverted file* dalam sistem temu balik informasi dapat meningkatkan performansi salah satu bagian penting sistem temu balik informasi yaitu pengindeksan dan pencarian.

Pencarian dengan menggunakan konsep pohon ini mampu menjadi alternatif dari algoritma pencarian sekuensial sederhana dan *pattern matching* yang bersifat *bruto force* dan memerlukan proses komputasi yang berat.

DAFTAR REFERENSI

[1] Rijsbergen, van. "Information Retrieval". Butterworth. 1979.

[2] Mandala, Rila. "Sistem Temu-Kembali informasi dengan menggunakan Model Probabilistik", *Jurnal Informatika,* vol. 1, no. 2, 2002. hal 1-2

[3] Baeza-Yates, Ricardo & Ribeiro Neto, Berthier. "Modern Information Retrieval. ACM Press/Addison Wesley. 1999. hal 192-198.

[4] Munir, Rinaldi. (2006). "Diktat Kuliah IF2153 Edisi Keempat". Program Studi Informatika ITB. Bandung 2006. hal IX-1