Kajian Struktur Data Pohon Pencarian Biner (BST) dan Kinerjanya dalam Berbagai Aplikasi Software

Meliza T.M. Silalahi – NIM 13506116
Jurusan Teknik Informatika Institut Teknologi Bandung
Jl.Ganesa 10 Bandung
email: if16116@if.itb.ac.id

Abstract
Ditinjau dari kinerja , berbagai aplikasi dan pengembangan system operasi sangat memerlukan struktur data yang sesuai untuk menyimpan data dan melakukan operasi-operasi dasar seperti penyisipan, pencarian , atau penghapusan seefisien mungkin karena sangat berdampak pada kinerja dari aplikasi maupun system operasi tersebut.
Stuktur data pohon pencarian biner merupakan struktur data yang baik dalam menjawab persoalan di atas. Memilih struktur data pohon pencarian biner yan tepat akan lebih berdampak baik. Pada makalah ini dikaji 6 struktur data implementasi pohon pencarian biner yang terdiri dari 3 pohon seimbang (pohon AVL, pohon merah-hitam, dan pohon splay) dan 3 pohon tidak seimbang (pohon treap, skip list dan radix.) Pilihan kedua yang tak kalah pentingnya adalah representasi simpul. Kedua hal ini mempengaruhi kinerja pohon pencarian biner.

Kata Kunci*:* pohon pencarian biner , stuktur data, kompleksitas waktu.

1. PENDAHULUAN

Pohon pencarian biner (*binary search tree*) merupakan salah satu struktur data penting yang dipakai dalam banyak persoalan operasi seperti ; operasi pencarian, penyisipan, dan penghapusan elemen. Pohon pencarian biner memiliki kinerja yang lebih baik dibandingkan struktur data lainnya dalam hal waktu pencarian.

Simpul pada pohon pencarian berupa kunci (key) pada data record yang nilainya unik, berbeda dengan simpul lain. Jika R adalah akar pohon, maka semua simpul pada subpohon kiri memiliki nilai yang lebih kecil dari Kunci (R) dan demikian juga semua simpul pada subpohon kanan memiliki nilai yang lebih besar dari Kunci (R).

2. POHON PENCARIAN BINER

2.1. Struktur Data

2.1.1 Stuktur Data Pohon AVL

AVL merupakan pohon pencarian biner seimbang (*self-balancing binary search tree*). Pencarian, penyisipan dan penghapusan memiliki kompleksitas waktu O (log n) untuk kasus rata-rata dan kasus terburuk. Jika factor seimbang (*balance factor*) -1, 0, 1 maka pohon masih AVL dan tidak perlu rotasi pohon (penyeimbangan lagi), namun jika factor seimbangnya -2 , 2 maka perlu rotasi pohon. Untuk meyeimbangkan kembali.

2.1.1 Stuktur Data Pohon Merah Hitam (*Red Black Tree*)

Karakteristik pohon merah hitam : [5]

1. Simpul berwarna hitam atau merah
2. Akar selalu berwarna hitam
3. Semua daun pada pohon berwarna hitam
4. Kedua anak dari simpul merah harus berwarna hitam
5. Setiap jalur yang dilalui dari akar sampai ke daun memiliki jumlah simpul hitam yang sama.

Jumlah simpul hitam dalam satu jalur disebut *black-height* pohon. Karakteristik di atas menjamin jalur yang dilalui dari akar ke daun tidak lebih dari dua kali . Jumlah memory yang dibutuhkan untuk menyimpan simpul merah hitam harus dijaga agar tetap minimum, terutama bila simpul ini telah dialokasi.

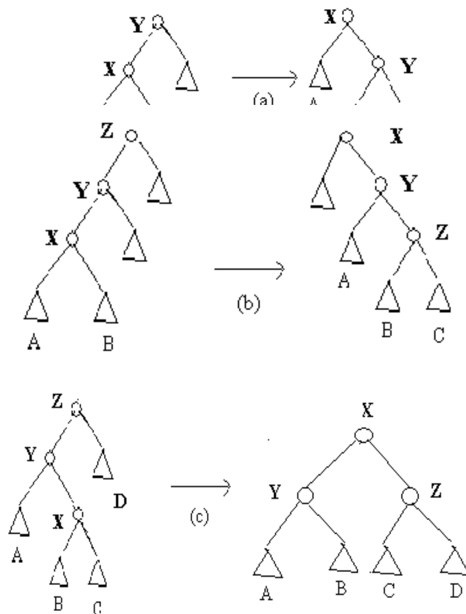
Pencarian, penyisipan dan penghapusan memiliki kompleksitas waktu O (log n).

2.1.1 Stuktur Data Pohon Splay (*Splay Tree*)

Pohon splay merupakan self-adjusting binary search tree yang strukturnya telah dimodifikasi (tidak unutk isinya). Operasi splay pada simpul x pada pohon pencarian biner terdiri dari langkah-langkah berikut, hingga x menjadi akar pohon . Berikut disebutkan langkah splay [3] :

- Kasus 1 : jika x adalah anak kiri dari akar y, maka maka rotasi kanan y menjadi anak kanan x, seperti pada gambar 1(a). Berlaku pada arah kebalikannya. Kasus ini disebut kasus zig.
- Kasus 2 : jika x adalah anak kiri y dan y adalah anak kiri dari z, maka rotasi kanan z, diikuti rotasi kanan y, seperti gambar 1(b). Berlaku juga untuk arah kebalikannya. Kasus ini disebut kasus zig-zig.
- Kasus 3 : jika x adalah anak kanan dari y dan y adalah anak kiri dari z, maka rotasi kiri y

diikuti rotas kanan z, seperti gambar 1(c). Kasus ini disebut kasus zig-zag.



Gambar 1: Langkah splay. Simpul yang diakses adalah x . (a) zig : rotasi tunggal. (b) Zig-zig : dua rotasi tunggal. (c) Zig-zag : rotasi ganda.

2.1.4 Stuktur Data Treap

Treap adalah struktur data dasar pada pohon pencarian random/acak. Kata treap berasala dari *tree* dan *heap* [3]. Lebih jelasnya, misalkan x menyatakan satu set item di mana setiap item memiliki kunci dan priority. Treap dari set x menyata kan kasus istimewa dari pohon pencarian biner yang mana set simpul disusun berurutan (tergantung kunci) seperti *heap fashion* (tergantung pada prioritas).
Waktu pencarian proposional terhadap kedalaman elemen pada pohon. Penyisipan elemen memerlukan proses 2-langkah. Langkah pertama menempatkan posisi daun (sesuai dengan nilai kunci) dan langkah kedua rotasi item ke atas (sesuai dengan priority dalam sturktur pohon). Begitu juga dengan penghapusan yang memerlukan proses 2-langkah.
Kompleksitas waktu untuk pencarian, penyisipan dan penghapusan adalah O (log n)[3].

2.1.5 Stuktur Data Skip List.

Skip list menyatakan list berkait yang terurut di mana setiap simpul berisi variable nomor dari link ke simpul lain dalam suatu struktur [3. Sebagai gambaran link n^{th} dari simpul point yang diberikan simpul-simpul di dalam suatu list melewati/skip bebrapa nomor simpul perantara.

Untuk beberapa aplikasi , skip list menyatakan gambaran yang lebih umum dibandingkan struktur pohon karena menggunakan algoritma yang lebih simple untuk diimplementasikan. Namun, ukuran simpul yang bervariasi merupakan salah satu kelemahan skip list.
Kompleksitas waktu rata-rata pada pencarian, penghapusan dan penyisipan adalah O (logn). Kemungkinan menemui kinerja yang buruk (kompleksitas waktu O(n))memang sangat kecil, meskipun demikian, kemungkinan itu tetap ada [
4].

2.1.6. Stuktur Data Pohon Radix.

Pada pohon pencarian radix , data disimpan sebagai objek daun sehingga simpul internal tidak memiliki nilai kunci . Simpul internal dari anak menyatakan simpul internal yang lain atau data yang sebenarnya.
Untuk pengurutan (sort), memiliki kompleksitas waktu O (nk), di mana n menyatakan jumlah item dan k menyatakan rata-rata dari panjang kunci. Kelemahan algoritma pengurutan pada pohon radix adalah tidak bisa langsung dieksekusi pada tempatnya. Hal ini menyebabkan, kebutuhan penambahan memory O (n). Pengurutan radix membutuhkan 1 pass/langkah untuk setiap symbol kunci, sehingga kurang effisien [4].

2.2. Representasi Simpul

Idealnya, operasi traversal untuk menemukan suksesor dan predesor secara urut harus mencapai O (1). Berikut kompleksitas waktu kasus normal, terurut dari yang tercepat .[3]

- **list berkait** O(1) traversal dengan sigle branchless pointer dereference.
- **Threads** O(1) traversal dengan mengikuti pointer tunggal upward atau downward
- **Right treads** O(1) operasi suksesor, sepeerti simpul thread, tapi menemukan simpul predessor memerlukan O(log n) , pencarian dari akar pohon di mana simpul tidak memiliki anak kiri.
- **Parent pointer** O(1) , traversal dengan menngikuti O(1) pointer upward atau downward dalam pohon.
- **Plain** O(1) traversal dengan mempertahankan tumpukan eksplisit (explicit stack). Jika pohon dimodofikasi maka kinerja menurun ke O(kg n).

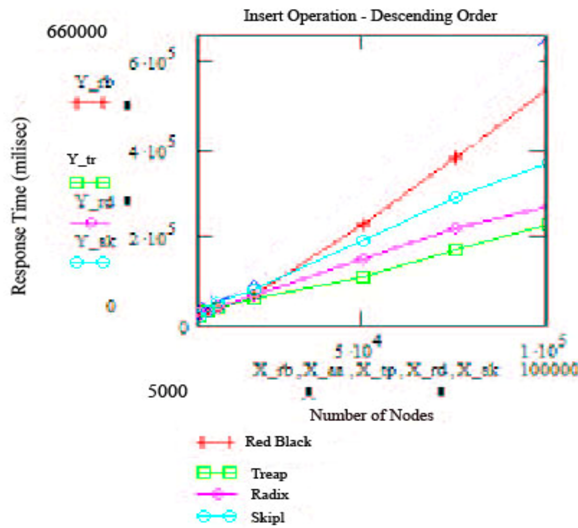
Operasi rotasi yang digunakan untuk mempertahankan keseimbangan pohon seimbang, harus menanggulangi banyak representasi simpul. Dengan parent pointer, setiap rotasi membutuhkan 3 ekstra pointer assignment, jumlah assignment yang dilakukan menjadi ganda. Pada pohon threaded , setiap rotasi membutuhkan 1 atau 2 ekstra assignment dan dua cara cabang. Pada pohon right-threaded , setipa rotasi

membutuhkan dua cara cabang dan 1 ekstra assignment.
Representasi simpul juga membutuhkan memory yang berbeda. Parent pointer menambah pointer tunggalk pada setiap simpul ; simpul list berkait menambahkan dua. Simpul threaded menambahkan 2 “tag” bit setiap simpul untuk membedakan treads dari pointer anak dan simpul right threaded menambahkan 1 tag bit.

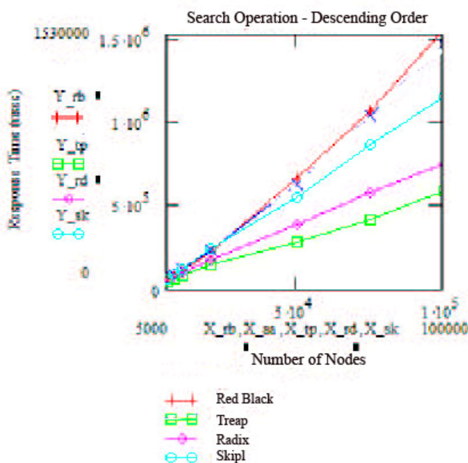
3. PEMBAHASAN KINERJA POHON PENCARIAN BINER

3.1. Operasi Penyisipan, Pencarian, dan Penghapusan.

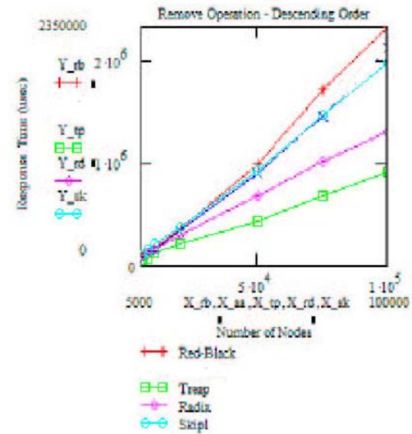
Berikut bebrapa sample variasi struktur data pohon pencarian biner dalam proses penyisipan.



Gambar 2. Operasi Penyisipan



Gambar 3. Operasi Pencarian



Gambar 4. Operasi Penghapusan.

Pada kasus di atas, simpul terurut menurun, stuktur data treap merupakan yang terbaik. Namun, analisa lebih lanjut melalui pengukuran data , fluktuasi penyisipan, pencarian dan penghapusan yang berbeda (menaik, menurun, maupun random) membuktikan bahwa struktur pohon merah hitam memiliki kekonsistenan dan lebih cemerlang dalam hal implementasi. [3]

3.3.1 Kinerja Pada Sytem Software

Perbandingan antara pohon yang tidak seimbang (treap, radix, split) dengan pohon seimbang (pohon merah hitam,pohon AVL, dan pohon splay) menghasilkan setiap struktur pohon lebih baik dalam situasi yang berbeda. Pohon tak seimbang paling baik saat input/ masukan data adalah random yang………liat kamus .., tapi jika perintah randomnya adalah normal dan perintah pengurutan/sort berjalan seperti yang diharapkan naka pohon merah hitam merupakan piilihan yang tepat. Di lain pihak, jika penyisipan sering terjadi dalam perintah pengurutan,pohon AVL baik ketika akses selanjutnya cenderung random, sedangkan pohon splay memiliki kinerja yang baik ketika akses selanjutnya adalah berurut/sequential atau berkelompok/clustered .

Berikut merupakan penjelasan dari eksperimen yang dilakukan untuk mengetes kinerja dari struktur data pohon pencarian biner [2]

3.3.1 Virtual Memory Areas

Setipa proses dalam Unix seperti kernel memiliki virtual memory areas (VMAs). Minimum scara statis pohon berkait memiliki satu VMA untuk setiap kode, data dan segmen tumpukan (stack segments). Secara

dinami, pohon berkait juga mmiliki kode dan data VMAs untuk setipa library yang dinamis. Pemroses dapat menciptakan sebuah VMAs yang berubah-ubah, bergantung pada btasa suatu system opersai atau arsitektur mesin dengan memetakan (*mapping*) disk files ke memory dengan mmap system call.
VMAs bervariasi dalam ukuran dari 4 kB hinggnan 1 GB, sehingga konvensional hash tables tidak dapat mejaga track secara efisien. HAsg tables juga tidak dapat mendukung secara efisien jangkauan antrian yang dibutuhkan untuk menentukan yang mana VMAs yang ada dan yang tumpang tindih di area permintaan oleh mmap atau munmap call.
Untuk mengatasi kedua hal di atas, digunakan sturktur data pohon pencarian biner (BST). Trebukti dari banyak kernel mnggunakan BSTs untuk menjaga track VMAs : Linux sebelum 2.4.10 menggunakan pohon AVL, OpenBSD adan versi Linux selanjutnya menggunakan pohon merah hitam, Free BSD menggunaka n pohon splay. Begitu juga sengan Winows NT [2].

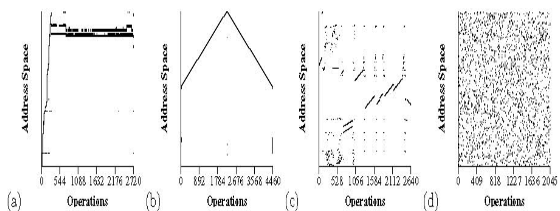
Eksperimen pertama mensimulasikan aktivitas VMA pada saat program dieksekusi. Moifikasi table VMA oleh mmap dan munmap calls disimulasikan dari modifikasi pohon.

- Mozilla 1.0 [2] . Total pamggilam mmap adalah 1,459 kali dan munmap 1,264 kali. Pada puncak nya ada 108 VMAs dengan rata-rata 98.
- VNware GSX Server 2.0.1 x86 monitor virtual mesin [2] setelah boot dan penggunaan Linux virtual mesin. Total panggilan mmap dan munmap masing-masing 2,233 kali. Pada puncaknya ada 2,233 VMAs dengan rta-rata 1,117.
- Squid web cache 2.4 STABLE4 running under User-Mode Linux 2.4.18.48, untuk simulasi virtual memory. Total panggilan mmap 1,278 kali dan munmap 1,403 kali Ada paling banyak 735 VMAs dengan rata rata 400.
- Test sintetik/buatan yang terdiri dari 1,024 mmap pangilan diikuti 1,204 munmap Dimulai saat random page-aligned location dan dilanjutkan pada random number o pages. Ini merupakan hal yang tidak realistik, namun merupakan contoh kasu terbaik dari pencarian pohon biner.

VMware GSX Server test set merupakan kasus terburuk untuk BST tidak seimbang karena sequences of anepage mmaps at sequential virtual address. Tapi di pihaklain, random test set merupakan kasus terbaik bagi BST tidak seimbang.

Mozilla , VMware dan Squid data sets merupakan “real world” data sets karena tergambar langsung dati program yang sebenarnya/nyata. Pohon splay terkemuka untuk ketiga real world test ini. Lebih baik 23 % ke 40% disebabkan oleh kmampuan pohon splay untuk tetap menggunakan simpul dekat puncak pohon dan kecenderungan VMware dan Squid test set dalam akses sequential yang menyebabkan waktu yang dibutuhkan menjadi linear. [2]
Seperti yang diduga BST lebih lama dibandingkan struktur data yang lain dalam hal real world tests. Pada pohon AVL secara kosisten lebih cepat dibanding pohon merah hitam , hingga mencapai 20%.
Berdasarkan eksperimen operasi gabungan[2], disimpulkan ;
1. Representasi simpul plain lambat karena penyisipan dan penghapusan yang dilakukan tidak memvalisai stack yang dibuthkan dalanm proses traversal . Dengan simpul plianm memakskan t_next untuk mencari lebih extra ; representasi simpul right threaded juga lama, disebabkan penggunaan t_prev.
2. Parent pointer lebih cepat dibanding representasi threaded karena membedakan urutan dari pointer anak yng membutuhkan langkah yang lebih ekstra.

Untuk random data set, semua implemntasi dikelompokkan dengan jangkauan yang relative kecil sekitar 2.3 kali perbedaan. Secara intuitif, implementasi tercepat haruslah yang memiliki ekstra kerja yang paling sedikit. Pada random data set , pohon pencarian biner biasa tidak ada rotasi, pohon merah-hitam 209 rotasi, pohon AVL 267, pohon splay 2678. Sehinggan urutan mulai dari tercepat hingga yang paling lama adalah pohon pencarian biner yang biasa, pohon merah hitam, pohon AVL dan pohon splay.
Dalam representasi simpul, dari yang tercepat adalah simpul list berkait, simpul denngan parent pointer, simpul threaded, simpul right-threaded dan terakhir simpul plain.



Gambar 4. (a) Mozilla, (b) VMware GSX Server 2.0.1, (c) squid UserMode Linux 2.4.18.48 , dan (d) data tes acak. [2]

3.3.2 Internet Peer Cache

RFC 791 [2] mmbutuhkan setiap IP dikirim dengan host internet yang ditandai dengan 16-bit identifikasi yang harus unik dari source tujuan dan protocol pada saat datagram aktif dalam internet system.
Kernel Linux menggunakan pohon AVL yang indexnya dari peer IP address. Implementasi ini

merupakan cara yang memudahkan dan membuat DoS menjadi lebih efisien. Contohnya hashing yang dapat membuat *collision attack* dapat diatasi dengan menggunakan teknik pohon seimbang.
Peer pohon AVL digunakan sebagai *second-level cache* . Paket data dikirim ke host tertentu , sebuah simpul dan IP address nya disisipkan ke pohon. Penunjuk langsung (*direct pointer*) ke simpul ini diletakkan pada route cache (first-level cache).
Berikut merupakan data hasil test pada kondisi normal dan attack.

Tabel 1. Waktu dalam detik scenario pada normal dan attack simulasi. * menandakan lebioh besar dari 60 detik.

3.3.2.1 Data tes normal

Dari hasil simulasi didapat urutan yang tercepat berdasarkan representasi simpul dimulai dari parent pointer, threads, plain, list berkait, right threads. Kecepatan parent pointer dan thread mempengaruhi kemampuan untuk menggunakan t_delete untuk penghapusan O(1). Di antara representasi simpul lainnya , plain merupakan yang tercepat karena tidak terhambat oleh manipulasi list berkait atau right thread yang tidak menguntungkan dalam eksperimen ini [2]. Namun ada pengecualian, untuk pohon splay yang walaupun menggunakan representasi simpul thread, tetap merupakan yang paling lama. Hal ini disebakan oleh hasil yang tidak sesuai pada saat implementasi pohon splay. Untuk mengatasi keanehan ini, pohon splay dengan representasi simpul thread ini dimodifikasi untuk mempertahankan tumpukan parent pointer dan bergantung pada algoritma untuk mendapatkan simpul parent.

Tes Set	Representasi	BST	AVL	RB	Splay
normal	plain	4.74	5.10	5.06	7.70
	parent	3.94	4.07	3.78	7.19
	threads	3.99	4.45	4.17	13.25
	right thread	5.52	5.71	5.64	8.29
	list berkait	4.93	5.40	5.25	8.41
attack	plain	*	3.76	4.32	4.14
	parent	*	2.65	2.92	3.21
	threads	*	2.97	3.27	7.77
	right thread	*	4.04	4.77	4.62
	list berkait	*	4.10	4.46	4.78

Tabel 1. Waktu dalam detik , scenario simulasi , * lebih besar dari 60 detik

Ketika eksperimen ini dijalankan pada system analisa didapat bahwa Pentium IV memiliki kerja ekstra terhadap cabang. Pomter parent dan list berkait membutuhkan lebih sedikit cabang dibandingkan yang lain. Itulah sebabnya mereka bisa lebih cepat.Untuk kasus normal dengan representasi simpul yang konsisten , maka urutan yang tercepat adalah pohon tak seimbang, pohon merah-hitam, pohon AVL dan pohon splay.

3.3.2.2 Data tes attack

Pohon AVL lebih baik dibanding pohon merah hitam karena pada pohon AVL yang seimbang sangat membantu dalam hal menjaga tinggi pohon menjadi minimum pada penyisipan selanjutnya dan menjaga langkah maksimum untuk semua operasi. Berdasarkan data yang diperoleh rata-rata internal panjang langkah semua operasi bervariasi kurang dari 1 % antara pohon AVL dengan pohon merah hitam.
Pohon splay sangat buruk pada data set attack, karena proses sekuensial akses menjadi waktu linear.

Pentium IV [2] , hasilnya berneda dengan sebelumnya di mana urutan yang tercepat adalah parent pointer, ,list berkait, threads, right threads dan plain.

3.4 Cross Reference Collator

Penggunaan *cross reference collator* ini berguna untuk pengembangan perangkat lunak . Dikatakan "*cross reference collator*" karena menyisipkan satu set pengenal yang dibaca ke pohon tunggal . Setiap pengenal memilikki pohon sendiri yang terdiri dari nama file di mana pengenal itu muncul. Setiap file memiliki satu set jumlah baris yang berkorespondensi dengan baris file pengenal itu muncul.

3.4.1 Data tes normal

Untuk kasus normal pohon splay merupakan yang tercepat diikuti pohon merah hitam, pohon AVL, dan pohon tak seimbang.[2]

3.4.1 Data tes terurut

Untuk kasus terurut, pohon splay merupan yang terbaik diikuti pohon AVL dan pohon merah hitam.

3.4.1 Data tes acak

Untuk kasus acak, justru pohon splay merupakan yang terburuk, sehingga lebih baik menggunakan pohon merah hitam.

3.5 Perbandingan

Berdasarkan hasil percobaan dibandingkan hingga diperoleh struktur data da representasi simpul yang sesuai untuk berbagai kasus. [2]

3.5.1 Pemilihan Stuktur Data

Pada kasus penyisipan pada urutan acak pohon tidak seimbang merupakan pilihan yang terbaik karena waktu yang dibutuhkan singkat dan ruangan yang kosong banyak. Sebaliknya jika penyisipan dilakukan

pada data yang terurut pohon tidak seimbang ini jauh lebih buruk dibandingkan dengan pohon merah hitam ataupun pohon AVL.
Pada kasus penyisipan data acak, namun dalam menjalankan program mengikutsertakan program pengurutan maka pohon merah hitam lebih baik dibandingkan pohon AVL karena pohon merah hitam lebih memerlukan kerja yang lebih sedikit dalam aturan penyeimbangan dalam menyeimbangkan data acak.Pohon splay sangat tidak direkomendasikan karena pengorbanan dalam hal *splaying* setiap akses. Untuk pohon splay merupakan pilihan terbaik saat program yang berikutnya merupakan data yang sekuensial (seperti pada percobaan VMA).

3.5.2 Pemilihan Representasi Simpul

Parent pointer merupakan yang terbaik diikuti representasi thread. Namun thread lebih baik digunakan saat ruang minimum karena penggunaan memory lebih sedikit. Representasi plain tanpa parent pointer atau thread lebih hemat waktu dalam meng-*update* penambahan bidang, namun tampaknya menghalangi efisiensi dalam implementasi t_delete.
Representasi simpul list berkait merupakan pilihan terbaik pada percobaan VMA yang menggunakan operasi traversal t_next dan t_prev, namun sangat susah untuk internet peer. Hal ini disebabkan pengorbanan memory yang tinggi karena menggunakan dua penunjuk/pointer dalam satu simpul terutama jika traversal jarang dilakukan.
Perpresentasi right thread tidak terlalu mendukung karena cara yang tidak efisien dalam hal implementasi t_delete.

4. KESIMPULAN

Berdasarkan percobaan dan perbandingan didapat dalam hal memilih struktur data yang akan digunakan tergantung pada kasu yang dihadapi. Pohon tidak seimbang paling biakm saat data acak, jika data acak itu diurutkan oleh program pengurutan yang sedang berjalan sebiknya pohon merah hitam yang digunakn. Jika penyisipan sering terjadi dalam data terurut, pohon AVL merupakan yang terbaik, dan jika data berikutnya merupakan data yang sekuensial pohon splay sangat mendukung.
Untuk representasi simpul, parent pointer lebih cepat , jika ruangan nya premium simpul threads sebaiknya digunakan . Untuk plain memiliki bidang yang sedikit untuk di *update,* dan lsit berkait merupakan pilihan bagus saat traversal namun memerlukan pengorbanan memory. Representasi right-thread sangat buruk di semua percobaan.

Memilih representasi simpul dan struktur data pohon pencarian biner yang akan digunakan sangat berpengaruh pada kinerja system perangkat lunak .

DAFTAR REFERENSI

[1] Munir, Rinaldi. 2006. *Diktat Kuliah IF2153 Matematika Diskrit.* Program studi Teknik Informatika, Institut Teknologi Bandung. Bandung.
[2] Ben Ptaff , “Performance Analysis of BSTs in System Software”
[3] The Eurean Journal for Informatics Professional Vol.V,No 5, October 2004. p 67-74
[4] http://en.wikipedia.org/wiki/BST
Tanggal akses 26 Desember 2006
pukul 17.00 WIB
[5] http://en.wikipedia.org/wiki/tree_html
Tanggal akses 28 Desember 2006
pukul 17.00 WIB

.