Pemanfaatan Pohon Biner dalam Pencarian Nama Pengguna pada Situs Jejaring Sosial

Stephen (13512025)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13512025@std.stei.itb.ac.id

***Abstrak*—Makalah ini menjelaskan mengenai salah satu metode pencarian suatu kata, angka, ataupun data dengan menggunakan Pohon Pencarian Biner (*Binary Search Tree*). Metode ini cukup efektif dibandingkan dengan metode pencarian biasa karena waktu pemrosesan jauh lebih cepat. Algoritma yang digunakan juga cukup mudah yaitu pencarian dilakukan dengan membandingkan kata yang dicari dengan kata pada akar, jika huruf pertama kata yang dicari lebih awal dibanding akar, maka akan dilanjutkan pencarian ke kiri, sebaliknya bila lebih akhir maka akan dilanjutkan pencarian ke kanan. Penambahan dan Penghapusan data juga dapat dilakukan pada Pohon Biner.**

***Kata Kunci*—biner, nama, pencarian, pohon.**

I. PENDAHULUAN

Bidang informatika kini sangat berkembang di dunia ini, banyak hal yang sudah diciptakan demi kenyamanan hidup manusia sendiri dan salah satunya adalah internet. Internet memiliki jutaan situs yang bisa kita akses setiap harinya, situs – situs seperti *facebook*, *twitter*, *youtube* serasa tidak asing lagi di telinga kita karena hampir semua orang mengetahui situs terkenal tersebut, dan situs – situs tersebut juga telah memiliki puluhan bahkan ratusan juta orang pengguna.

Pengguna – pengguna situs tersebut yang sangat banyak jumlahnya, tentu saja memiliki kesulitan dalam mengelolanya, baik karena memerlukan alokasi memori yang besar untuk menyimpan data – data pengguna tersebut, serta waktu yang cukup lama untuk melakukan pencarian terhadap pengguna tertentu (misal : pencarian seorang pengguna yang melakukan penipuan di *facebook* dan akan dilakukan pemblokiran terhadap akun tersebut ,atau saat melakukan pengecekan terhadap *username* yang sudah terdaftar ketika akan membuat akun baru pada suatu situs agar tidak terjadi akun yang kembar).

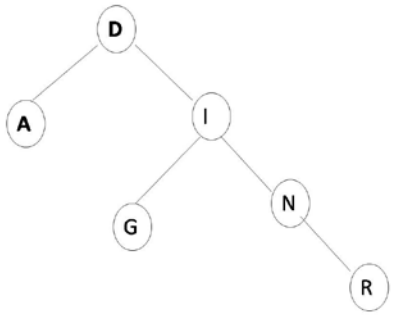
Pada umumnya, setelah penambahan akun dilakukan, maka data tersebut akan disimpat pada *database* dengan urutan biasa yaitu data kedua ditambahkan di bawah data pertama, data ketiga ditambahkan di bawah data kedua, dan seterusnya. Dengan cara seperti ini, jika terdapat ratusan juta data, maka pencarian data tersebut akan memakan waktu yang cukup lama.

Pencarian – pencarian tersebut dapat dipercepat dengan memanfaatkan salah satu struktur data pada bidang informatika yaitu pohon. Pohon yang digunakan di sini adalah pohon biner yang merupakan salah satu implementasi dari graf dengan komponen utamanya adalah akar, *sub*pohon / upapohon kiri, dan *sub*pohon / upapohon kanan. Setiap data yang ditambahkan akan dimasukkan ke dalam pohon biner terurut dengan membandingkan abjad yang kemudian akan dimanfaatkan dalam pencarian. Berikut akan dijelaskan dan diperlihatkan keefektifan pohon pencarian biner (*Binary Search Tree*) dalam pencarian suatu data.

II. DASAR TEORI

II.A. Definisi Pohon

Pohon merupakan graf tak-berarah terhubung yang tidak mengandung sirkuit, berikut merupakan contoh pohon :



Gambar 2.1 Pohon

Pohon juga memiliki sifat bahwa tidak ada simpul yang tidak terhubung dengan lintasan manapun dan di antara 2 simpul hanya selalu terdapat 1 lintasan. Misalkan $G = (V, E)$ adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pernyataan di bawah ini adalah ekivalen:

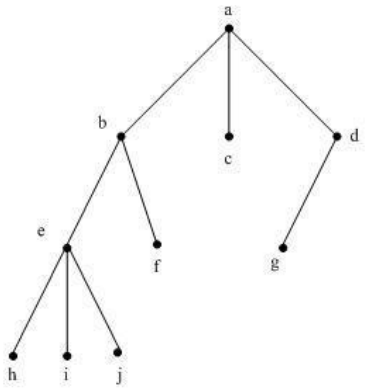
1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki $m = n - 1$ buah sisi.
4. G tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.

6. *G* terhubung dan semua sisinya adalah jembatan.

Definisi di atas merupakan definisi lain dari pohon.

II.B. Pohon Berakar

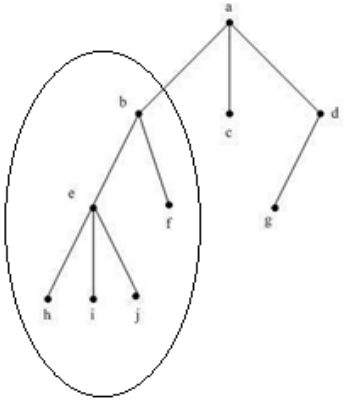
Pohon yang satu buah simpulnya diperlakukan sebagai akar dan sisi – sisinya diberi arah sehingga menjadi graf berarah dinamakan **pohon berakar**. Sebagai perjanjian, tanda panah pada sisi dapat dibuang dan berikut contoh pohon berakar :



Gambar 2.2 Pohon Berakar

Berikut akan dijelaskan mengenai terminologi pada pohon berakar dengan pohon pada gambar 2.2 sebagai subjek :

1. Anak (*child* atau *children*) dan orangtua (*parent*)
   b, *c*, dan *d* adalah anak dari simpul *a*, dengan *a* adalah orang tua dari anak – anak itu. Orang tua terletak di atas dan sebuah simpul dikatakan anaknya bila terdapat lintasan antara simpul tersebut dengan simpul di atasnya.
2. Lintasan
   Lintasan dari *a* ke *f* adalah *a*, *b*, *f*. Panjang lintasan tersebut adalah 2. Lintasan di sini adalah simpul yang ditempuh untuk mencapai simpul tujuan.
3. Saudara kandung (*sibling*)
   f adalah saudara kandung *e*, tetapi bukan saudara kandung *g* karena orang tuanya berbeda. Sebuah simpul dikatakan saudara kandungnya jika memiliki orang tua yang sama.
4. Upapohon (*subpohon*)
   Sebuah subpohon adalah anak dari sebuah simpul yang juga merupakah sebuah pohon, seperti tertera pada gambar di bawah ini :

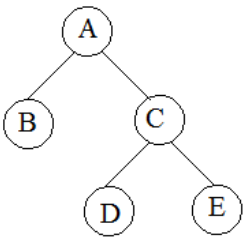


Gambar 2.3 Subpohon

5. Derajat (*degree*)
   Derajat sebuah simpul adalah jumlah upapohon (atau jumlah anak) pada simpul tersebut. Derajat *a* adalah 3, derajat *b* adalah 2, derajat *c* adalah 0. Jadi, derajat yang dimaksudkan di sini adalah derajat – keluar. Derajat maksimum dari semua simpul merupakan derajat pohon itu, sehingga pohon pada contoh memiliki derajat 3.
6. Daun (*leaf*)
   Simpul yang berderajat nol (atau tidak mempunyai anak) disebut daun. Simpul *h, i, j, f, c, g* adalah daun.
7. Simpul dalam (*internal nodes*)
   Simpul yang mempunyai anak disebut simpul dalam. Simpul *b, d, e* adalah simpul dalam.
8. Aras (*level*) atau tingkat
   Aras adalah tingkatan dari sebuah simpul – simpul dalam pohon, simpul *a* memiliki aras 0, simpul *b, c, d* memiliki aras 1, simpul *e, f, g* memiliki aras 2, dan seterusnya.
9. Tinggi (*height*) atau kedalaman (*depth*)
   Aras maksimum dari suatu pohon disebut tinggi atau kedalaman. Tinggi pohon pada contoh adalah 3.

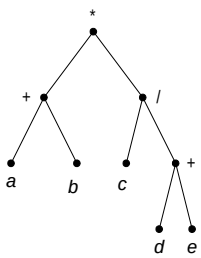
II.C. Pohon Biner

Pohon biner adalah sebuah pohon yang setiap simpulnya maksimal hanya memiliki 2 anak, karena itulah diberi nama biner. Anak kiri dari sebuah simpul disebut upapohon kiri, dan anak kanan sebuah simpul disebut upapohon kanan. Berikut adalah contoh gambar pohon biner :

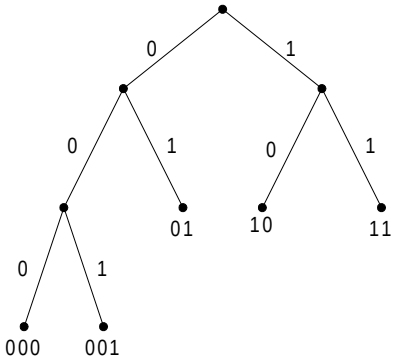


Gambar 2.4 Pohon biner

Pohon biner dikatakan Pohon biner lengkap jika setiap simpulnya memiliki 2 anak. Terdapat sangat banyak penggunaan dari pohon biner seperti pohon ekspresi yaitu pohon biner yang menggunakan daun sebagai *operand* dan simpul dalam sebagai operator, pohon keputusan untuk pengambilan keputusan berdasarkan kondisi – kondisi yang mungkin terjadi, kode prefiks, Pohon huffman untuk sistem kompresi, dan lain – lain. Berikut adalah contoh gambar aplikasi pohon biner :



Gambar 2.5 Pohon ekspresi dari *(a+b)*(c/(d+e)*



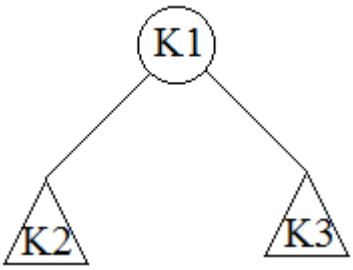
Gambar 2.6 Pohon biner dari kode prefiks {000, 001, 01, 10, 11}

II.D. Pohon Pencarian Biner (*Binary Search Tree*)

Pohon pencarian biner merupakan salah satu pemanfaatan pohon biner yang sangat efisien jika digunakan (akan dibahas kemudian) karena pencarian data dapat dilakukan dengan cepat dan efisien. Data pada pohon ini bersifat unik (tidak ada yang sama). Penambahan serta penghapusan elemen memiliki algoritma tersendiri, dan tidak memungkinkan akan menambahkan data yang sudah ada ke dalam pohon dan tentu saja tidak memungkinkan penghapusan data yang belum ada di pohon.

Terdapat ketentuan dalam peletakan data dalam pohon agar pencarian dapat dilakukan dengan lebih mudah, yaitu jika K1 adalah simpul, dan pohon tersebut unik maka berlaku :

a. Jika K1 memiliki subpohon kiri , maka nilainya harus lebih kecil dari K1.
b. Jika K1 memiliki subpohon kanan , maka nilainya harus lebih besar dari K1.



Gambar 2.7 BST dengan K2<K1, dan K3>K1

III. PEMBENTUKAN POHON PENCARIAN BINER

III.A. Penambahan Data Pada Pohon Pencarian Biner

Agar Pohon pencarian biner dapat dimanfaatkan sebagaimana mestinya, maka penambahan data pohon pencarian biner tidak bisa dilakukan dengan sembarangan, melainkan harus mengikuti aturan bahwa data pada anak kiri harus lebih kecil dari data pada simpul, serta data pada anak kanan harus lebih besar dari data pada simpul, penambahan data tersebut menggunakan algoritma tertentu sehingga nantinya pohon biner tersebut menjadi pohon biner yang terurut. Misal kita punya data *username* seperti yang tertera pada tabel di bawah ini :

No.	Nama
1	Stephen
2	Anthony
3	Anthony123
4	24doom
5	Tony Stark
6	xfactor
7	Junior
8	Unreal

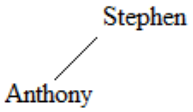
Tabel 3.1 Tabel *username* dengan asumsi tidak ada simbol dan angka lebih didahulukan daripada huruf

Langkah pembuatan pohon pencarian biner :

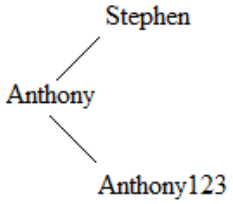
1. Data pertama digunakan sebagai simpul pertama.

   Stephen

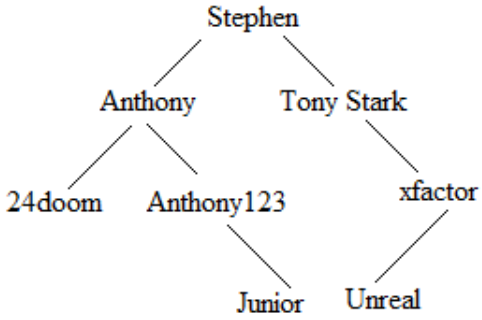
2. Ambil data berikutnya, lalu periksa :
   - Jika data tersebut lebih awal dari simpul, maka letakkan data sebagai anak kiri.
   - Jika data tersebut lebih akhir dari simpul, maka letakkan data sebagai anak kanan.



3. Ambil data berikutnya, lalu telusuri dari awal, dan mengulangi langkah 2, kemudian :
   - Jika setelah penelusuran menemukan adanya data lain (simpul tersebut tidak kosong), maka kembali telusuri dengan langkah 2.
   - Jika setelah penelusuran tempat tersebut kosong, tambahkan data sesuai aturan pada langkah 2.



4. Mengulangi langkah 3 sampai akhir dari data. Pada contoh, akan terbentuk pohon seperti di bawah ini :



Gambar 3.1 Pohon Pencarian Biner berdasarkan tabel 3.1

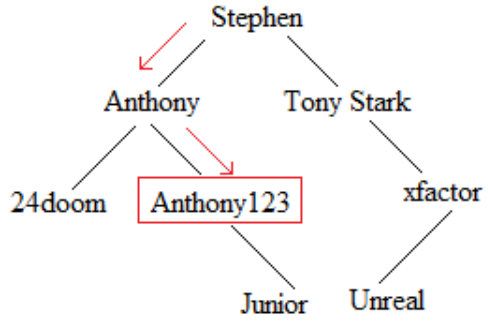
Dapat dilihat bahwa setelah pembentukan pohon pencarian biner, bila kita telusuru pohon secara *infix* maka kita akan dapatkan *string* nama yang terurut membesar yaitu : 24doom, Anthony, Anthony123, Junior, Stephen, Tonystark, xfactor, Unreal. Penambahan data setelah itupun harus mengikuti langkah di atas agar tidak mengubah keterurutan elemen.

III.B. Penghapusan Data Pada Pohon Pencarian Biner

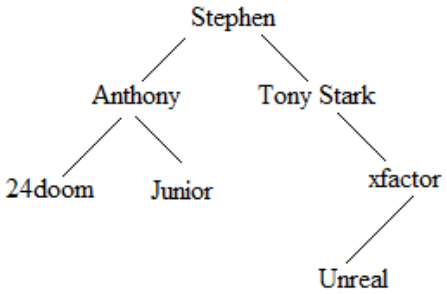
Sama seperti penambahan data pada pohon pencarian biner, penghapusan data juga perlu algoritma tertentu agar penghapusan tidak mengubah keterurutan data pada pohon. Misal ingin dihapus *username* Anthony123.

Langkah penghapusan data pada pohon biner :

1. Melakukan proses pencarian terhadap data yang ingin dihapus :
   - Jika nama yang dicari lebih awal dari simpul, maka lanjutkan proses pencarian ke anak kiri
   - Jika nama yang dicari lebih akhir dari simpul, maka lanjutkan proses pencarian ke anak kanan
2. Melanjutkan proses pencarian sampai menemukan data yang ingin di hapus



3. Setelah menemukan data :
   - Jika data ditemukan pada daun, cukup hapus daun tersebut
   - Jika data ditemukan pada simpul, hapus simpul tersebut, kemudian simpul tersebut akan digantikan upapohon kiri atau kanan, lakukan kembali perbandingan agar tidak mengubah keterurutan elemen.

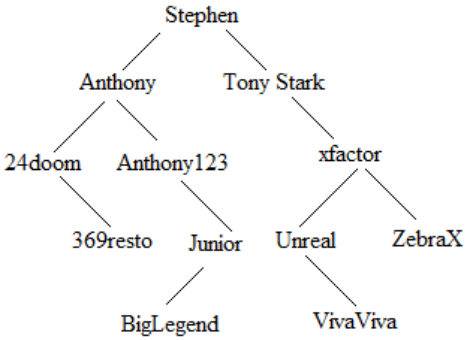


Gambar 3.2 Pohon Pencarian Biner hasil penghapusan *username* Anthony123

Setiap dilakukannya penghapusan elemen, tidak boleh ada pengubahan struktur yang mengakibatkan perubahan keterurutan elemen, karena hal tersebut akan menyebabkan pohon tidak berfungsi semestinya.

IV. ANALISIS PENCARIAN DATA MENGGUNAKAN POHON PENCARIAN BINER

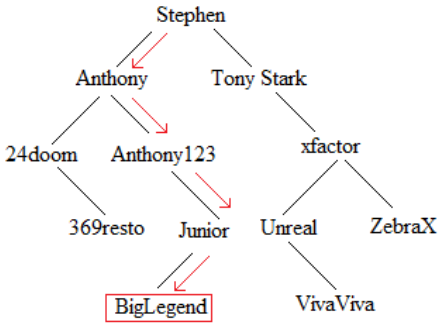
Misalkan percobaan akan dilakukan pada pohon pencarian biner *username* berikut :



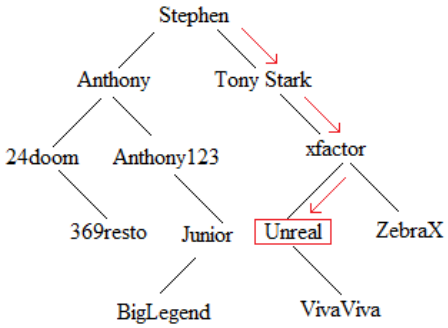
Akan dilakukan proses pencarian terhadap beberapa *username* berikut :

- BigLegend
- Unreal
- 24doom

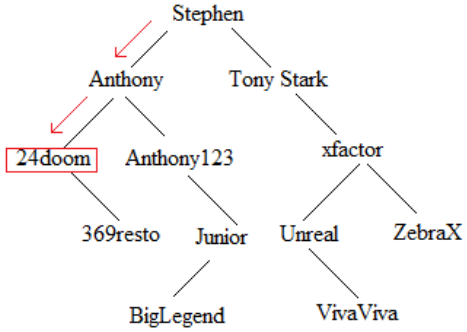
1. Pencarian pertama : BigLegend



2. Pencarian kedua : Unreal



3. Pencarian ketiga : 24doom



Dapat dilihat bahwa pada setiap kali pencarian, terdapat beberapa kali proses perbandingan yang dilakukan yaitu :

1. BigLegend :
   - "BigLegend" < "Stephen" → ke kiri
   - "BigLegend" > "Anthony" → ke kanan
   - "BigLegend" > "Anthony123" → ke kanan
   - "BigLegend" < "Junior" → ke kiri
   - "BigLegend" = "BigLegend" → berhenti

   Total : 5 kali perbandingan

2. Unreal :
   - "Unreal" > "Stephen" → ke kanan
   - "Unreal" > "Tony Stark" → ke kanan
   - "Unreal" < "xfactor" → ke kiri
   - "Unreal" = "Unreal" → berhenti

   Total : 4 kali perbandingan

3. 24doom :
   - "24doom" < "Stephen" → ke kiri
   - "24doom" < "Anthony" → ke kiri
   - "24doom" = "24doom" → berhenti

   Total : 3 kali perbandingan

Dapat dilihat bahwa proses pencarian selalu terbagi 2 (jika lebih kecil maka ke kiri dan sebaliknya jika lebih besar maka ke kanan), yang menyebabkan bahwa tidak semua data dibandingkan dan menyebabkan pencarian berjalan lebih cepat. Jika data tersebut tidak dibentuk dalam struktur pohon pencarian biner, melainkan hanya struktur tabel biasa, maka jumlah proses perbandingan bisa mencapai beberapa kali lipat.

Contoh bisa dilihat dari Tabel 3.1 dimana bila akan dicari *username* "Unreal", maka akan terjadi proses perbandingan sebanyak 8 kali karena nama "Unreal" berada pada urutan ke 8 pada tabel, dan setiap data akan dibandingkan sehingga waktu pencarian akan mencapai 2 kali lipat dibandingkan dengan pencarian pada *BST*.

Jika dilihat dari sisi kompleksitas algoritma, maka algoritma pencarian pada *Binary Search Tree* memiliki kasus terburuk sebesar :

$$T_{\max}(n) = {}^2\log n + 1$$

yang berarti pencarian akan dilakukan dari simpul pertama sampai ke daun, dan algoritma ini juga memiliki notasi *Big-Oh* :

$$T(n) = O(\log n)$$

yang berarti algoritma ini memiliki kompleksitas waktu asimptotik *log n* yang termasuk salah satu algorimta yang cukup baik di antara algoritma yang lain, dapat dilihat urutan kompleksitas waktu asimptotik berikut dari yang terbaik hingga yang terburuk :

$O(1) > O(\log n) > O(n) > O(n \log n) > O(n^2) > O(n^3) > O(2^n)$

Pada data yang cukup besar seperti data pengguna pada situs *facebook*, *twitter*, maka pencarian data bisa dipercepat hingga ratusan, ribuan, bahkan jutaan kali lipat.

n	$T(n)={}^2\log n + 1$
128	8
1024	11
1125899906842624	51
1180591620717411303424	71

Tabel 4.1 Perbandingan pertumbuhan *T(n)* dengan *n*

V. KESIMPULAN

Dengan menggunakan pohon pencarian biner (*Binary Search Tree*) maka proses pencarian akan jauh lebih mudah untuk dilakukan karena perbandingan tidak dilakukan terus menerus melainkan hanya melakukan perbandingan seperlunya yang mengakibatkan proses pencarian akan memakan waktu yang jauh lebih pendek dan sangat efektif untuk situs yang memiliki jumlah pengguna terdaftar di atas rata – rata seperti *facebook*, *twitter*, dan *youtube*.

VI. UCAPAN TERIMA KASIH

Pada kesempatan ini saya ingin mengucapkan terima kasih kepada Tuhan yang Maha Esa karena atas berkat-Nyalah makalah ini dapat diselesaikan. Saya juga ucapkan banyak terima kasih kepada Bapak Rinaldi Munir selaku dosen Matematika Diskrit karena berkat pendidikan bimbingannya saya mendapat ilmu yang bisa terapkan dalam penyelesaian makalah ini. Terima kasih juga kepada semua pihak yang membantu dalam penyelesaian makalah ini.

REFERENSI

[1] Munir, Rinaldi.*Matematika Diskrit rev. 5*.Informatika Bandung, 2012

[2] Kenneth H. Rosen, *Discrete Mathematics and Application to Computer Science 7th Edition*, Mc Graw-Hill, 2007. Hal. 757-760

[3] http://cslibrary.stanford.edu/110/BinaryTrees.html, diakses 13 Desember 2013, pkl 19.20 WIB

[4] http://ibrahimmanorek.blogspot.com/2011/03/1.html, diakses 13 Desember 2013, pkl 19.54 WIB

[5] http://10507276.blog.unikom.ac.id/tag/inilah-jumlah-atau-data-pengguna-facebook-indonesia/, diakses 14 Desember 2013, pkl 11.33 WIB

[6] http://bertzzie.com/knowledge/analisis-algoritma/2-KompleksitasAlgoritma.html, diakses 14 Desember 2013, pkl 14.11 WIB

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 15 Desember 2013

Stephen / 13512025