Pemanfaatan Pohon Biner dalam Pencarian Username pada Instagram

Steven Gianmarg Haposan Siahaan (13520145)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13520145@std.stei.itb.ac.id

***Abstrak*—Makalah ini menjelaskan mengenai salah satu metode pencarian suatu kata, angka, ataupun data dengan menggunakan *Binary Search Tree*. Metode ini terbilang lebih efektif dibandingkan dengan metode pencarian biasa karena memiliki waktu proses yang 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

Seiring dengan berkembangnya teknologi di dunia, semua sisi kehidupan sudah dimasuki oleh berbagai inovasi dalam bidang teknologi guna kenyamanan manusia sebagai user/pengguna.

Perkembangan paling nyata dapat terlihat pada dunia internet. Internet yang tadinya adalah barang langka khususnya bagi mereka yang tinggal di daerah pedalaman, sekarang menjadi sebuah kebutuhan dasar bahkan daerah pedalaman pun sudah semakin banyak yang mendapatkan akses internet.

Hal ini membuat pengguna di internet semakin banyak, terlebih khusus dalam sosial media. Pengguna sosial media setiap hari terus meningkat. Pada makalah ini akan secara spesifik membahas sosial media yaitu Instagram.

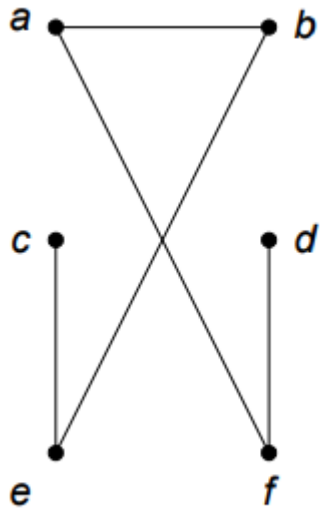
Tentunya diperlukan alokasi memori yang besar untuk menyimpan data – data pengguna Instagram yang banyak tersebut, serta waktu yang cukup lama untuk melakukan pencarian terhadap pengguna tertentu (misal : melakukan pengecekan terhadap *username* yang sudah terdaftar ketika akan membuat akun baru agar tidak terjadi akun yang kembar). Pada umumnya, setelah penambahan akun dilakukan, maka data tersebut akan tersimpan di *database* dengan urutan biasa yaitu data kedua ditambahkan di bawah data pertama, dan seterusnya. Dengan cara seperti ini, jika terdapat ratusan juta data, maka pencarian data tersebut akan memakan waktu yang relative lama.

Dengan memanfaatkan struktur data Pohon maka pencarian tersebut dapat dipercepat. Pohon yang digunakan di sini adalah pohon biner yang merupakan salah satu implementasi dari graf dengan komponen utamanya adalah akar, *sub*pohon kiri, dan *sub*pohon kanan. Setiap data yang ditambahkan akan dimasukkan ke dalam pohon biner terurut dengan membandingkan abjad yang kemudian akan dimanfaatkan dalam pencarian. Pada makalah ini 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

Selain itu, dapat terlihat juga pada Pohon tidak ada simpul yang tidak terhubung dengan lintasan manapun dan di antara 2 simpul hanya selalu terdapat 1 lintasan.

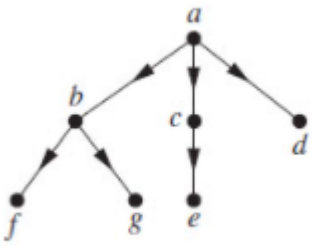
Adapun Sifat Pohon :

1. Misalkan G merupakan suatu graf dengan n bua h simpul dan tepat n – 1 buah sisi

2. Jika G tidak mempunyai sirkuit maka G merupa kan pohon.
3. Suatu pohon dengan n buah simpul mempunyai n – 1 buah sisi.
4. Setiap pasang simpul di dalam suatu pohon terh ubung dengan lintasan tunggal.
5. Misalkan G adalah graf sederhana dengan jumla h simpul n,jika G tidak mengandung sirkuit mak a penambahan satu sisi pada graf hanya akan me mbuat satu sirkuit.
6. Pohon adalah suatu graph yang banyak vertexnya sama dengan n (n>1)

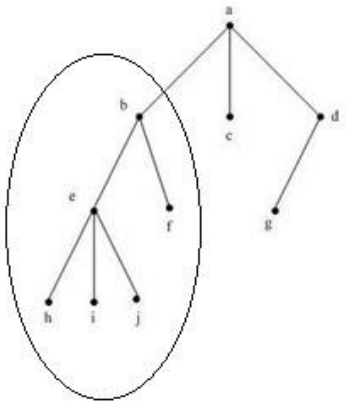
II.B. Pohon Berakar

Pohon berakar adalah graf pohon yang satu titik dari graf pohon tersebut dijadikan sebagai akar dan setiap sisi mengarah keluar dari akar tersebut. Atau bisa juga diartikan 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. Simpul dikatakan anaknya bila terdapat lintasan antara simpul tersebut dengan simpul di atasnya.
2. Lintasan
   Lintasan dari *a* ke *g* adalah *a*, *b*, *g*. Panjang lintasan tersebut adalah 2. Lintasan di sini adalah simpul yang ditempuh untuk mencapai simpul tujuan.
3. Saudara kandung (*sibling*)
   f adalah saudara kandung *g*, tetapi bukan saudara kandung *e* karena orang tuanya berbeda. 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 :

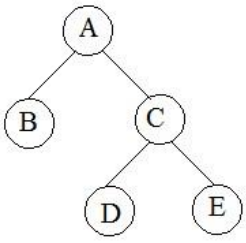


Gambar 2.3 Contoh subpohon

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 (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

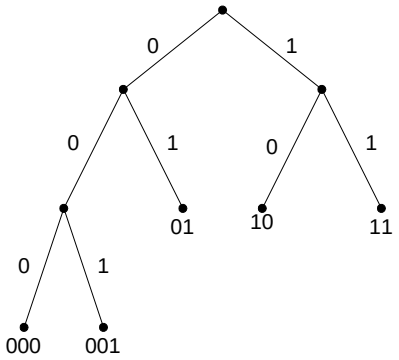
Pohon biner adalah sebuah pohon yang maksimal memiliki 2 anak untuk setiap simpulnya. Anak kiri dari sebuah simpul disebut upapohon kiri, dan anak kanan sebuah simpul disebut upapohon kanan. Berikut adalah contoh gambar pohon biner :



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

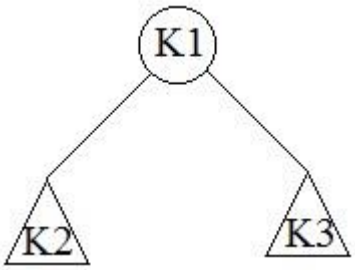
II.D. Pohon Pencarian Biner (*Binary Search Tree*)

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 mungkin 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

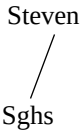
Agar Pohon pencarian biner dapat dimanfaatkan sebagaimana mestinya, maka penambahan data pohon pencarian biner tidak bisa dilakukan dengan sembarangan, melainkan harus mengikuti aturan bahwa data pada anak kiri harus lebih kecil dari data pada simpul, serta data pada anak kanan harus lebih besar dari 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	Steven
2	Sghs
3	Sghsgianmarg
4	25sadedadeda
5	Tom Holland
6	Xean
7	Sghssiahaan
8	Venom

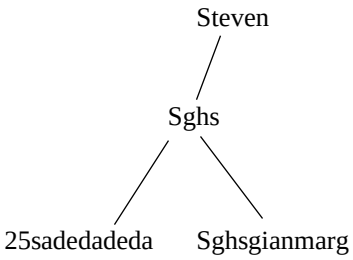
Tabel 3.1 Tabel *username* dengan asumsi tidak ada simbol dan angka lebih didahulukan daripada huruf

Langkah pembuatan pohon pencarian biner :

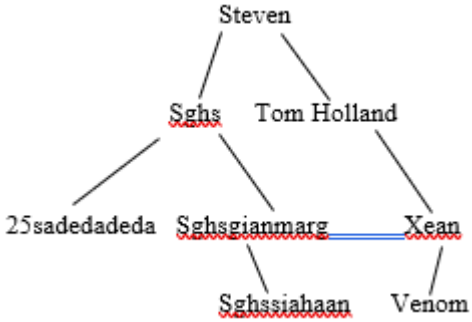
1. Data pertama digunakan sebagai simpul pertama. Yaitu Steven
2. Ambil data berikutnya, lalu periksa :
   - Jika data tersebut lebih awal dari simpul, maka letakkan data sebagai anak kiri.
   - Jika data tersebut lebih akhir dari simpul, maka letakkan data sebagai anak kanan.



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

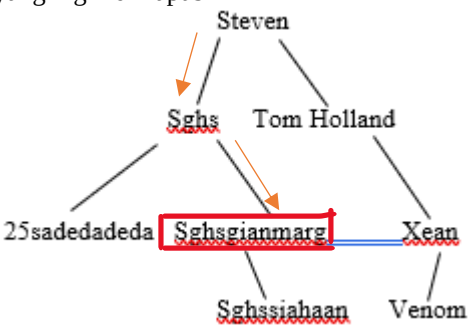
Dapat dilihat bahwa setelah pembentukan pohon pencarian biner, bila kita telusuru pohon secara *infix* maka kita akan dapatkan *string* nama yang terurut membesar. Penambahan data setelah itupun harus mengikuti langkah di atas agar tidak mengubah keterurutan elemen.

III.B. Penghapusan Data Pada Pohon Pencarian Biner

Penghapusan data memerlukan algoritma tertentu agar tidak mengubah keterurutan data pada pohon. Misal ingin dihapus *username* Sghsgianmarg

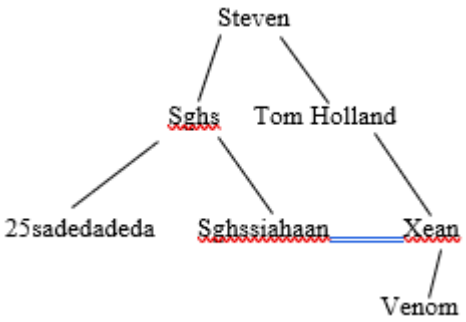
Langkah penghapusan data pada pohon biner :

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



Gambar 3.2 Proses Pencarian *username* Sghsgianmarg

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.

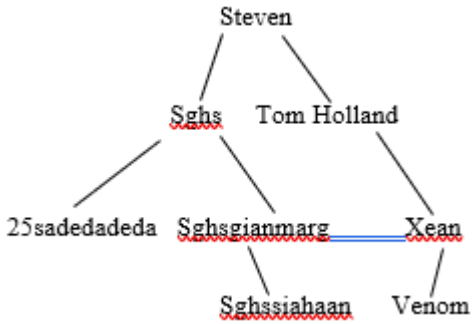


Gambar 3.2 Pohon Pencarian Biner hasil penghapusan *username* Sghsgianmarg

Setiap 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 :

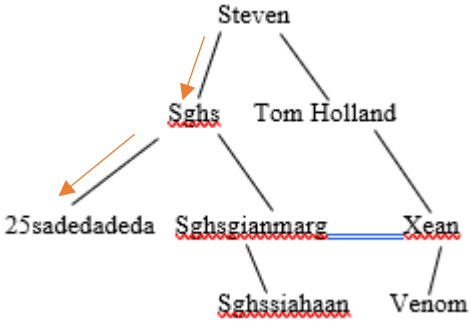


Gambar 4.1 Pohon Pencarian Biner berdasarkan table 3.1

Akan dilakukan proses pencarian terhadap beberapa *username* berikut :

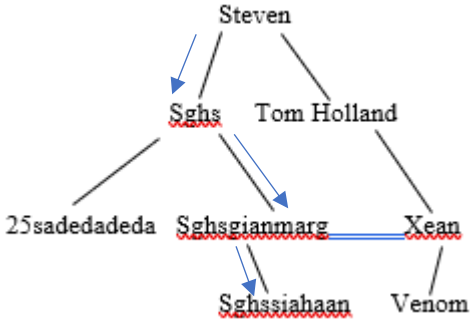
- 25sadedadeda
- Sghssiahaan
- Venom

1. Pencarian pertama : 25sadedadeda



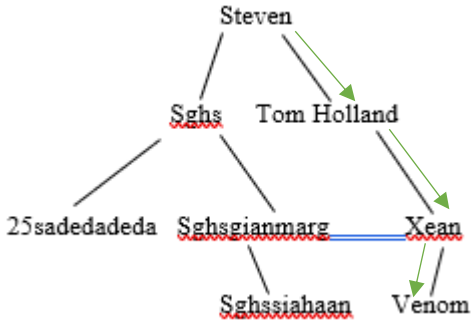
Gambar 4.2 Proses Pencarian Username 25sadedadeda

2. Pencarian kedua : Sghssiahaan



Gambar 4.3 Proses Pencarian Username Sghssiahaan

3. Pencarian ketiga : Venom



Gambar 4.4 Proses Pencarian Username Sghssiahaan

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

1. 25sadedadeda :
   - "25sadedadeda" < "Steven" ke kiri
   - "25sadedadeda" < "Sghs" ke kiri
   - "25sadedadeda" = "25sadedadeda" ke kanan

   Total : 3 kali perbandingan
2. Sghssiahaan :
   - "Sghssiahaan" < "Steven" ke kiri
   - "Sghssiahaan" > "Sghs" ke kanan
   - "Sghssiahaan" > "Sghsgianmarg" ke kanan
   - "Sghssiahaan" = "Sghssiahaan" berhenti

   Total : 4 kali perbandingan

3. Venom :
   - "Venom" > "Steven" ke kanan
   - "Venom" > "Tom Holland" ke kanan
   - "Venom" < "Xean" ke kiri
   - "Venom" = "Venom" berhenti
   - Total : 4 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* "Venom", maka akan terjadi proses perbandingan sebanyak 4 kali karena nama "Unreal" berada pada urutan ke 4 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 instagram maka pencarian data bisa dipercepat hingga ratusan, ribuan, bahkan jutaan kali lipat.

n	$T(n)={}^2\log n + 1$
256	9
2045	12
1152921504606846976	61
11805916207174113034 24	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 instagram yang memiliki user/pengguna sangat banyak.

VI. UCAPAN TERIMA KASIH

Pada bagian akhir makalah ini, saya berterima kasih kepada Tuhan YME karena atas berkat-Nya makalah ini dapat terselesaikan. Tidak lupa saya juga berterima kasih kepada tim dosen Matematika Diskrit, yakni bapak Rinaldi Munir, ibu harlili, serta dosen kelas saya ibu Nur Ulfa Maulidevi. Tentunya makalah ini dapat terselesaikan berkat bantuan tim dosen yang sudah memberikan pengajaran selama semester 3 ini berlangsung. Tidak lupa juga saya ucapkan terima kasih kepada tim asisten dosen yang sudah membantu selama proses pembelajaran mata kuliah Matematika Diskrit.

REFERENSI

[1] Munir, Rinaldi.*Matematika Diskrit rev. 5.*Informatika Bandung,
2014

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

[3] http://ibrahimmanorek.blogspot.com/2011/03/1.html, diakses 28 Desember 2013, pkl 08.00 WIB

[4] http://cslibrary.stanford.edu/110/BinaryTrees.html, diakses 28 Desember 2021, pkl 18.00. WIB

[5] http://bertzzie.com/knowledge/analisis-algoritma/2-KompleksitasAlgoritma.html, diakses 29 Desember 2021, pkl
11.00 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, 29 Desember 2021

Steven Siahaan/ 13520145