Aplikasi Trie dalam Pembuatan Saran Pencarian Sederhana pada Mesin Pencari

Johanes Boas Badia 13517009
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
boas.marbun@yahoo.com, 13517009@std.stei.itb.ac.id

Abstrak – Pohon(Tree) merupakan struktur data yang sering dipakai dalam dunia komputer. Struktur data tree mempunyai banyak jenis, salah satunya adalah Trie. Trie merpakan jenis pohon pencarian yang biasa digunakan untuk menyimpan data yang dinamis, biasanya dalam bentuk string. Aplikasi Trie banyak digunakan pada pembuatan saran pencarian pada mesin pencari.

Kata kunci : Mesin Pencari , Pohon Pencarian, Struktur data, Trie.

I. PENDAHULUAN

Pohon(Tree) merupakan struktur data yang merepresentasikan hubungan antara satu simpul dengan yang lainnya. Penggunaan tree sudah sangat biasa dipakai dalam dunia komputer, karena keefektifan mengakses data untuk kebutuhan pencarian, dll. Ada banyak struktur data pohon yang biasa dipakai, salah satunya yang paling terkanal adalah pohon biner. Namun pada makalah ini akan dibahas jenis pohon yang lebih jarang dibahas, yaitu Trie.

Trie merupakan jenis pohon yang memiliki lebih dari satu anak, berbeda dengna binary tree yang hanya memiliki 2 anak. Dengan struktur seperti ini, Trie akan mempunyai waktu akses data yang sangat cepat, dan akan sangat efektif penggunaannya dalam saran pencarian mesin pencari. Pada makalah ini akan dibahas implementasi trie pada pembuatan saran pencarian sederhana pada mesin pencarian.

II. POHON

Sebelum masuk ke pembahasan Trie, ada baiknya terlebih dahulu mempunyai sedikit pemahaman tentang Pohon.

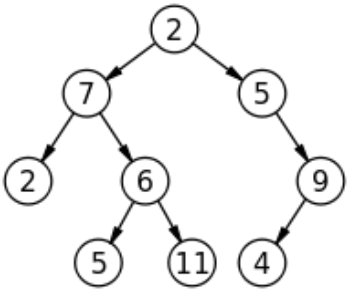
Graf terhubung yang tidak mengandung sirkuit disebut pohon. Properti sebuah pohon dinyatakan pada sebuah teorema dibawah ini :

Misalkan G = (V, E) adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pemyataan di bawah ini adalah ekivalen:

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung daun memiliki m = n - 1 buah sisi.
4. G tidak mengandullg sirkuit dan memiliki m = n- 1 buah sisi.
5. G tidak mengandung sirkuit dan penambaban satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan (jembatan adalah sisi yang bila dibapus menyebabkan graf terpecah menjadi dua komponen) [1].

III. POHON BERAKAR

Pohon yang sebuah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah menjauh dari akar dinamakan pohon berakar (*rooted tree*) [1].



Gambar 1.
Pohon Berakar
(sumber : https://en.wikipedia.org/wiki/Tree graph_theory#/media/File:Tree_graph.svg diakses pada 8 Desember 2018)

Terminologi untuk gambar 1:

1. Simpul (*node*) : semua lingkaran merupakan simpul dari gambar 1.

2. Akar (*root*) : simpul paling atas dari sebuah pohon (pada gambar 1, 2 yang berada di puncak merupakan akar).

3. Anak (*child*) : simpul yang terhubung langsung kepada simpul lain, saat pergerakan menjauhi akar. Contoh: 11 meruapakan anak dari 6, namum 9 bukan karena tidak terhubung langsung dengan 6.

4. Orang Tua (*parents*) : Kebalikan anak. Contoh : 6 adalah orang tua 11 dan 5, tapi bukan orang tua 2.

5. Daun (*leaf*) : node yang tidak memiliki anak lagi. Contoh: 2, 5(yang terletak di bagian bawah tree), 11, dan 4.

6. Tingkat (*level*) : panjang jalan dari akar sampai ke simpul tertentu. Tingkat dari akar = 1. Contoh tingkat(11) = 4

7. Kedalaman (*depth*) : tingkat terpanjang. Contoh : depth dari gambar 1 adalah 4.

8. Derajat (*degree*) : Banyaknya anak dari sebuah simpul. Contoh : derajat(9) = 1, derajat(6) = 2, derajat(4) = 0.

9. Keturunan (descendant) : simpul yang dapat dicapai dengan mencari jejak anak dari anak. Contoh : 7, 6, dan 5 merupakan keturunan dari 2(akar).

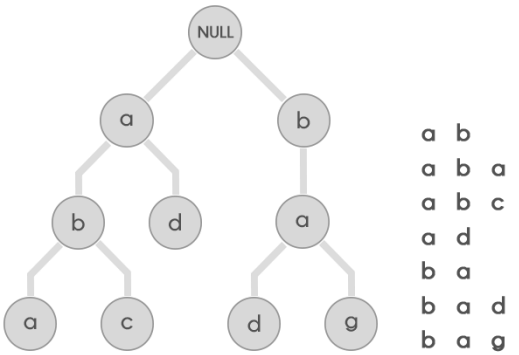
10. Leluhur (ancestor) : Kebalikan dari keturunan. Contoh : 2(akar) merupakan leluhur dari 5, 6, 7.

11. Saudara Kandung (siblings) : simpul yang memiliki orang tua yang sama. Contoh : 5 (daun) dan 11 merupakan saudara kandung, tetapi 6 dan 9 bukan.

III. TRIE

Trie merupakan salah satu contoh pohon berakar. Namun berbeda dengan pohon biner, yang setiap simpulnya maksimal memiliki 2 anak. Trie memiliki jumlah akar yang dinamis, sesuai kebutuhan. Namun pada umumnya, pada trie, satu simpul memiliki 26 anak, sejumlah dengan jumlah alphabeth pada umumnya, karena trie biasa digunakan untuk algoritma pencarian pada sebuah string.

Kata "trie" sendiri diambil dari kata reTRIEval, dan biasanya diucapkan seperti mengucapkan kata *try* agar tidak tertukar dengan kata *tree.* Dalam dunia sains komputer, Trie merupakan istilah yang relatif baru. Trie pertama kali digunakan pada tahun 1959, oleh seorang Prancis yang bernama René de la Briandais, yang berkata bahwa dengan Trie maka kompromi yang baik antara space dan *running time*[2].

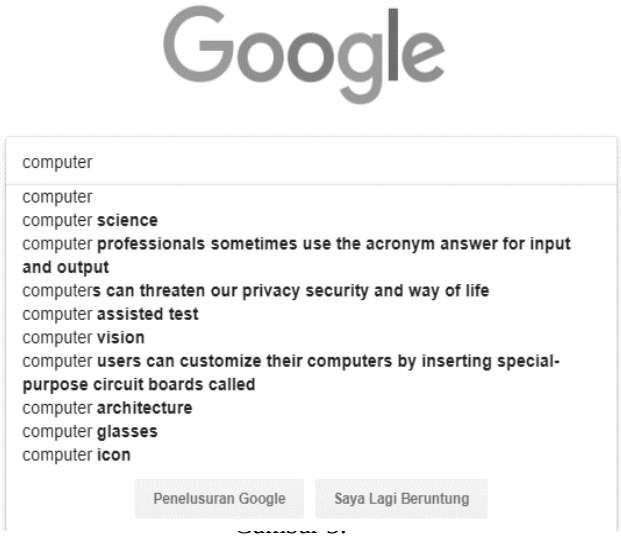


Gambar 2.
Trie Sederhana.
Sumber : (https://he-s3.s3.amazonaws.com/media/uploads/fb14630.png, diakses pada 8 Desember 2018 pukul 14: 50).

Dari contoh gambar diatas, dapat disimpulkan bahwa satu simpul pada trie hanya memiliki 2 aspek, yaitu isi simpul itu sendiri dan anak-anak dari simpul tersebut. Seiring bertambahnya trie, maka akan semakin banyak simpul anak yang terhubung dengan simpul orang tua. Walaupun terkesan memakan memori, penggunaan metode ini sangat efektif jika digunakan untuk pencarian.

IV. IMPLEMENTASI TRIE DALAM PEMBUATAN SARAN PENCARIAN SEDERHANA PADA MESIN PENCARI

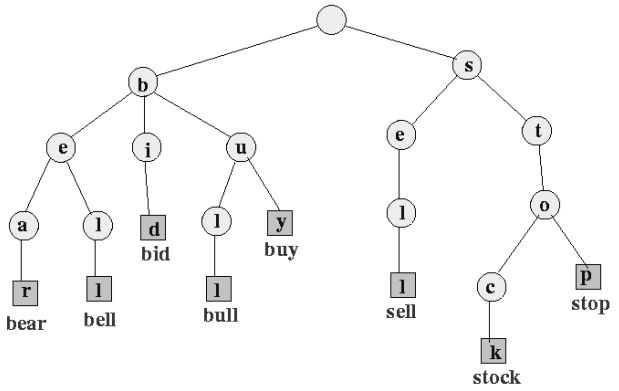
Pada mesin pencari pada umumnya, seperti *Google, Yahoo, dll*, akan ada fitur Saran Pencarian, atau biasa disebut *autocomplete*. *Autocomplete* adalah suatu fitur pada *search engine* yang memungkinkan *search engine* untuk mengeluarkan saran kata-kata yang akan pengguna ketik.



Gambar 3
Mesin Pencari *Google.*
(Sumber: google.co.id, diakses pada 8 Desember 2018 pada pukul 15:09)

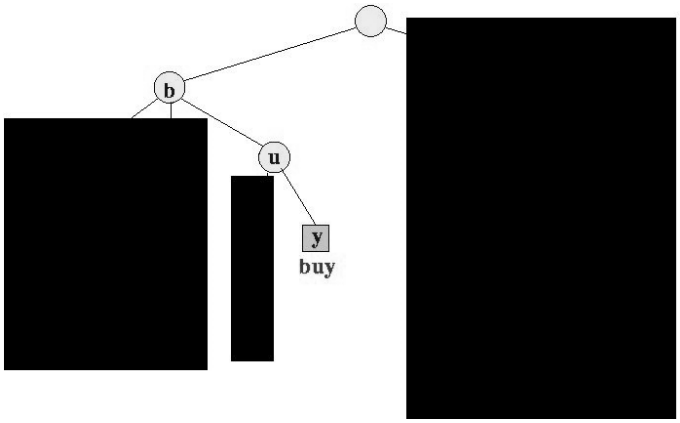
Dari gambar 3 diatas, dapat kita lihat bahwa ketika pengguna mesin pencarian mengetikkan kata "*computer"*, maka muncul saran-saran pencarian dari *Google*, dalam kasus ini "*computer science", "computer professionals sometimes use the acronym answer for input",* dll. Namun disini kita tidak akan membahas *autocomplete* yang serumit ini, karena alogirmta pada *search engine "Google"* sudah memakai algoritma yang sulit dan digabungkan dengan *data science*, sehingga bisa menghasilkan saran pencarian berdasarkan data-data yang telah dikumpulkan sebelumnya.Namun, pada makalah ini akan dibahas mengenai penggunaan sederhana Trie dalam algoritma saran pencarian.

Sebelum masuk ke algoritma pencarian, kita akan dibahas terlebih dahulu sistem struktur data tree dalam menambahkan dan mencari kata secara konsep.



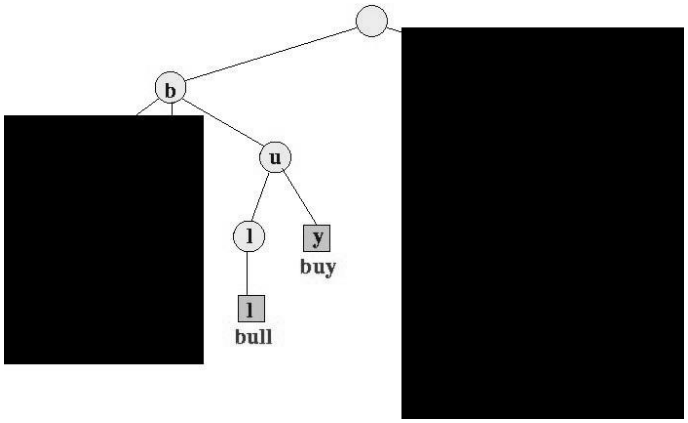
Gambar 4
Contoh Implementasi Trie
(sumber : https://cdn-images-1.medium.com/max/1600/1*e3549k5A9oCLn-vZTxsFEA.gif, diakses pada 8 Desember 2018 pada pukul 15:20)

Dari gambar diatas, ada 8 kata yang dapat dibentuk dari trie, yaitu "*bear, bell, bid, bull, buy, sell, stock, stop"*. Ini adalah contoh trie yang sudah dibentuk dan sudah membentuk berbagai kata. Sekarang anggap kita punya trie kosong, dan ingin membentuk kata "*buy".*



Gambar 5.
Contoh Implementasi Trie pada kata *buy*
(sumber : https://cdn-images-1.medium.com/max/1600/1*uhGbxMDw4CJKYpQIYhSmTw.jpeg, diakses pada 8 Desember pukul 15:27)

Ketika ingin menambahkan kata buy ke dalam sebuah Trie, maka kita harus mengalokasi memori untuk setiap huruf di dalam kata buy, dan menjadikannya sebuah simpul baru. Pada gambar 5, dari simpul kosong(akar), dialokasikan sebuah simpul baru yang berisi huruf 'b', dan kita hubungkan dengan akar. Setiap simpul baru harus dihubungkan dengan orang tuanya sebelum melanjutkan proses pembuatan kata. Lalu setelah huruf 'b' ada huruf 'u', dan kita harus mengalokasikan lagi satu simpul baru, dan menghubungkannya pada huruf 'b'. Begitu pula hal yang sama dilakukan pada huruf y. Lalu pada huruf y, dilakukan pewarnaan yang berbeda pada Trie untuk menyatakan bahwa y adalah akhir kata. Dengan langkah ini, kata *"buy"*sudah masuk kedalam Trie yang dibuat.



Gambar 6.
Contoh Implementasi Trie pada kata *bull*
(sumber : https://cdn-images-1.medium.com/max/1600/1*UKn6xTbYT-FuQ8kh269Keg.jpeg, diakses pada 12 Desmber 2018 pada pukul 15:36).

Sekarang ada kasus baru, yaitu kata *"bull"* ingin ditambahkan pada Trie. Karena pada Trie sudah ada simpul yang dibentuk sebelumnya, maka akan dilakukan pencarian dulu terlebih dahulu, huruf per huruf, apakah huruf yang ingin dimasukkan ke Trie sudah ada pada Trie atau belum. Pada gambar 6, ingin dimasukkan kata "*bull".* Kata ini berawalan huruf 'b', dan karena sudah ada huruf 'b' dalam Trie, maka tidak akan dilakukan alokasi memori, melainkan menjadikan simpul 'b' yang sudah ada menjadi simpul yang sama untuk kata *"bull".* Selanjutnya huruf 'u', dan karena sudah ada huruf u (setelah huruf b) pada Trie, maka tidak dilakukan lagi alokasi terhadap huruf u.

Lalu berlanjut ke huruf selanjutnya, yaitu huruf l. Karena huruf 'u' tidak memiliki anak 'l', melainkan hanya memiliki anak 'w', maka akan dialokasikan memori dan menambahkan simpul 'l' sebagai anak dari simpul 'u'. Terakhir, ditambahkan simpul 'l' ke simpul 'l' sebelumnya dan menandai simpul ini sebagai huruf terakhir dari sebuah kata.

Langkah-langkah ini dilakukan setiap akan dilakukan penambahan kata baru. Pengeckan per karakter, kemudian jika belum ada anak dari suatu simpul, maka akan ditambahkan simpul baru.

Selanjutnya ada algoritma pencarian, yang berfungsi sebagai cara untuk mencari kata yang disarankan. Pada implementasi yang akan dibahas nanti, adalah bentuk sederhana dari *autocomplete* pada mesin pencari pada umumnya. Perbedaan terletak pada output, dimana output pada implementasi bukanlah merupakan kata-kata, melainkan berapa kata yang bisa dibentuk dari string yang sudah dimasukkan oleh pengguna. Sebagai contoh, user memasukkan string "be", maka akan di output angka 2, yaitu jumlah kata yang dapat dibentuk dari huruf "be", yaitu "bear" dan "bell".

Selanjutnya akan dipaparkan kode untuk mengeluarkan output yang tadi sudah dijelaskan. Kode ditulis dalam bahas python.

```
from typing import Tuple


class TrieNode(object):

    def __init__(self, char: str):
        self.char = char
        self.children = []
        self.word_finished = False
        self.counter = 1


def add(root, word: str):
    node = root
    for char in word:
        found_in_child = False
        for child in node.children:
            if child.char == char:
                child.counter += 1
                node = child
                found_in_child = True
                break
        if not found_in_child:
            new_node = TrieNode(char)
            node.children.append(new_node)
            node = new_node
    node.word_finished = True


def find_prefix(root, prefix: str) -> Tuple[bool, int]:
    node = root
    if not root.children:
        return False, 0
    for char in prefix:
        char_not_found = True
        for child in node.children:
            if child.char == char:
                char_not_found = False
                node = child
                break
        if char_not_found:
            return False, 0
    return True, node.counter


if __name__ == "__main__":
    root = TrieNode('*')
    add(root, "hackathon")
    add(root, 'hack')

    print(find_prefix(root, 'hac'))
    print(find_prefix(root, 'hack'))
    print(find_prefix(root, 'hackathon'))
    print(find_prefix(root, 'ha'))
    print(find_prefix(root, 'hammer'))
```

Source code untuk program saran pencarian[3].

Cara kerja program adalah sebagai berikut:

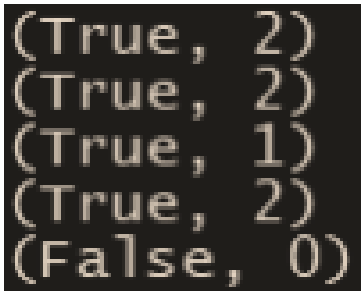
1. Fungsi add.

Fungsi add berfungsi untuk menambahkan sebuah string kata yang diinginkan ke dalam trie. Cara kerja mirip dengan apa yang telah dijelaskan sebelumnya. Fungsi add menerima 2 parameter, yaitu root dari TrieNode dan string yang ingin ditambahkan pada Trie.
Lalu program akan mengecek setiap karakter pada string. Pertama, boolean found_in_child diinisialisasi false. Lalu dilakukan traversal terhadap anak-anak dari sebuah simpul. Jika ada anak yang simpulnya sama seperti simpul karakter pada string, maka child.counter bertambah satu (bagian ini akan digunakan untuk menampilkan jumlah string *autocomplete* saat bagian fungsi find_prefix), dan akan berlanjut ke karakter selanjutnya. Lalu jika tidak, akan dialokasi node baru dan menghubungkan node baru ke node orangtua. Setelah itu node.word_finished ditandai True, yang menandakan bahwa suatu simpul adalah akhir dari suatu kata.

2. Fungsi findprefix

Fungsi ini menerima 2 parameter, yaitu root dari sebuah Trie dan juga prefix atau string masukkan dari pengguna. Lalu keluaran dari fungsi ini adalah sebuah tuple beru boolean dan integer. Boolean berisi apakah prefix ini ada atau tidak pada Trie, sementara int merupakan jumlah kata yang dapat dibentuk dari prefix yang telah dimasukkan oleh pengguna. Program akan melakukan pencarian secara traversal. Akan dilakukan pemeriksaan dari karakter pertama sampai karakter terakhir prefix. Apabila anak simpul sama dengan karakter dari prefix, maka anak simpul akan dijadikan simpul (node = child), dan karakter selanjutnya dari prefix akan dicari kembali.

Lalu masuk ke main program. Pada main program sudah di add kata "hackathon" dan "hack". Maka pada Trie kita sudah memiliki 2 kata tersebut. Lalu, ada fungsi print findprefix. Apabila program dijalankan, makan akan keluar output
seperti pada gambar berikut.



Gambar 6.
Hasil keluaran program.

Ambil contoh pada baris pertama, program mengeluarkan hasil "(True, 2)" yang merupakan hasil dari fungsi find_prefix(root, 'hac'). True karena ada string "hac" dalam Trie, dan ada 2 kata yang bisa dibentuk dari string "hac" tersebut. Lalu ambil contoh keluaran program pada baris ketiga (True, 1), yang merupakan hasil dari fungsi find_prefix(root, 'hackathon'), True karena ada kata hackathon pada Trie dan tidak hanya ada 1 kata yang mengandung string "hackathon" pada Trie. Terakhir pada keluaran program baris kelima (False,0), False karena tidak ada kata hammer pada Trie dan 0 adalah jumlah kata yang ada pada Trie.

V. PENGEMBANGAN

1. Penghapusan pada node Trie.

Penghapusan pada node Trie pada fitur*autocomplete* ini mungkin kurang terpakai, karena apabila suatu kata dihapus dari trie, dan ada seorang pengguna yang ingin mencari suatu kata yang baru dihapus tadi, maka fitur *autocomplete* ini tidaklah lengkap lagi. Namun apabila pada suatu waktu dibutuhkan, maka ada cara sederhana untuk melakukannya.

Apabila ada suatu kata, yaitu bid, pada gambar 4, yang ingin dihapus. Caranya cukup mudah, yaitu dengan melihat node.counter dari setiap node. Jadi, pertama program akan pergi ke node dengan huruf terakhir dari suatu string yang ingin dihapus, pada kasus ini huruf 'd'. Lalu akan dilihat counter di huruf d, apabila counter = 1, maka pointer akan menunjuk ke node orang tuanya dan simpul dengan huruf 'd' tersebut akan di-dealokasikan. Proses ini akan berlanjut sampai ditemukan sebuah simpul yang memiliki counter >1, maka counter simpul itu akan berkurang 1, namun tidak akan dihapus.

Ini adalah salah satu keuntungan struktur data Trie, dimana sangat mudah untuk menambahkan dan menghapus suatu elemen, karena semua elemennya independen, tidak berhubungan dengan elemen lain.

2. Pemunculan saran kata.

Aplikasi Trie pada program ini hanya memunculkan berapa saran kata dari string yang dimasukkan pengguna, bukan kata apa saja yang disarankan. Untuk itu, harus dilakukan sedikit modifikasi dan penambahan pada struktur data Trie tersebut.

Pertama, untuk menampilkan semua kemungkinan kata dari sebuah string. Program harus melakukan traversal sampai karakter terakhir string, dan melihat berapa counter yang ada di simpul yang ditunjuk. Lalu program melakukan traversal lagi semua kemungkinan string dari 1 sampai counter, dan mengeprint setiap kemungkinan kata yang muncul. Misalkan pada gambar 4, ketika user mengetikkan string"be", maka counter akan berpindah ke simpul 'e', dan mencair memunculkan ke layar kata "bear" dan "bell".

Kedua, penggabungan dengan frekuensi banyaknya akses, sehingga tampilan saran pencarian adalah yang paling sering diakses. Untuk fitur ini, kita perlu menambahkan frekuensi akses pada setiap simpul dengan huruf terakhir pada suatu kata.Sebagai contoh, ketika pengguna mengetikkan kata "be", maka yang urutan muncul kata pada program yaitu "bear", kemudian "bell" (karena kemunculan kata diatur secara traversal). Namun apabila user lebih banyak mengakses kata "bell", maka frekuensi kata "bell" akan semakin besar dan kata ini akan menjadi kata pertama yang muncul pada fitur *autocomplete*. Frekuensi kata ini juga berguna ketika sudah terlalu banyak data yang ada pada Trie, sehingga akan terlalu banyak saran pencarian yang dapat diberikan, namun dengan frekuensi kata dapat dibatasi berapa saran pencarian yang akan dikeluarkan, contohnya saran pencarian yang akan dikeluarkan hanyalah 10 string dengan freukensi tertinggi.

3. Koreksi Kata

Dengan adanya fitur frekuensi yang sudah dibahas pada poin kedua, maka dapat diketahui dari data apakah pengguna mengetikkan kata yang benar atau salah. Misalkan, seorang pengguna mengetikkan kata "payyng", yang bukan merupakan sebuah kata. Namun karena ada data frekuensi, sehingga dapat ditelusuri bahwa sebenarnya apa yang ingin diakses oleh pengguna adalah kata "payung", dan program bisa menampilkan saran pencarian tersebut.

4. Dari segi dunia periklanan.

Untuk mengoptimalkan fitur frekuensi kata dari yang sudah dibahas pada poin 2, dapat dikembangkan ke arah bisnis periklanan. Sebagai contoh, banyak orang yang mengakses sebuah string "*how to play piano for beginners"*. Dari sini kita mempunyai data bahwa banyak orang yang ingin bermain piano namum tidak tahu mau belajar darimana. Sebaagi pemilik data dari frekuensi pada Trie, akan sangat baik apabila data yang dimiliki dijual ke pihak ketiga, dalam hal ini *courses* piano online, atau tempat-tempat les privat, untuk memasang iklannya pada website mesin pencari. Sehingga dari sini akan didapatkan keuntungan tambahan bagi pemilik, begitu juga pihak ketiga diuntungkan karena bisa memasang iklan di tempat orang yang membutuhkan jasanya.

VI. KESIMPULAN

Trie bukanlah suatu struktur data yang banyak dipakai, namun penggunaanya sangat tepat sebagai fitur saran pencarian/*autocomplete* yang ditawarkan oleh *search engine.*

Trie memiliki banyak kelebihan, yaitu dapat mengakses data dengan cepat, karena setiap datanya berdiri secara independen, dapat menghapus data dengan mudah, dan proses pembuatan program juga tidak terlalu sulit.

Namun, kekurangan Trie terletak pada borosnya memori yang dipakai untuk menyimpan data, karena setiap ada data baru yang berbeda dari data lain, akan dialokasikan memori untuk data terbaru tersebut dan hal ini sangat memakan banyak memori, terlebih jika semakin banyak data yang dimasukkan kedalam Trie.

Pada intinya, secara garis besar, struktur data Trie menjanjikan cepatnya mengakses data, walaupu mengorbankan banyaknya memori yang terpakai.

VII. UCAPAN TERMIA KASIH

Terima kasih saya ucapkan pada seluruh pihak yang telah membantu saya dalam pembuatan makalah ini secara langsung maupun tidak langsung.

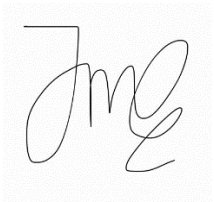
REFERENSI

[1] Munir, Rinaldi, Matematika Diskrit, Bandung: Informatika Bandung,2010.
[2] https://medium.com/basecs/trying-to-understand-tries-3ec6bede0014 diakses pada 8 Desember 2018 pukul 14:40
[3] https://towardsdatascience.com/implementing-a-trie-data-structure-in-python-in-less-than-100-lines-of-code-a877ea23c1a1 diakses pada 8 Desember pukul 17:12

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, 3 Desember 2017



Johanes Boas Badia
13517009