Penerapan Pohon dalam Fitur Autocomplete

Nugroho Satriyanto – 13514038
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514038@std.stei.itb.ac.id

***Abstrak*—Autocomplete atau sering disebut word completion adalah persoalan yang umum terjadi, terutama dalam dunia komputer. Terdapat beberapa algoritma untuk menyelesaikan permasalahan ini, masing-masing memiliki karakteristik tersendiri.**

***Keywords*— autocomplete, binary search, pohon, ternary search.**

I. PENDAHULUAN

Autocomplete atau biasa disebut fitur word completion adalah fitur yang dapat menebak kemungkinan input yang dimasukkan user berdasarkan beberapa huruf yang diketikkan pengguna sebelumnya.

Contoh fitur word completion bisa kita lihat dengan jelas pada web browser saat kita hendak mengisi suatu form, pada aplikasi pengolah kata, pada mesin pencari, atau pada papan ketik ponsel pintar. Hasil dari "tebakan" fitur ini biasanya adalah kata yang pernah diketik pengguna sebelumnya, atau kata yang sering digunakan saat mengetik.

Fitur autocomplete ini berfungsi untuk meningkatkan kecepatan dalam mengetik dan meningkatkan interaksi antara manusia dan komputer. Fitur ini akan bekerja dengan baik saat pengguna hanya dapat mengetikkan masukan dengan domain atau jenis karakter yang terbatas, misalnya hanya huruf a sampai z dan spasi.

Word completion biasanya didefinisikan dalam struktur data pohon prefix, atau biasa disebut trie. Pada beberapa algoritma dalam fitur autocomplete ini dapat menambahkan daftar katanya saat pengguna mengetikkan kata yang sama beberapa kali. Beberapa yang lain bahkan bisa mempelajari kebiasaan mengetik pengguna dan menampilkan saran kata yang belum pernah diketikkan sebelumnya. Pada ponsel pintar, daftar kata yang ada dalam program bahkan disinkronasikan dengan akun daring pengguna dan saat pengguna mengganti ponsel pintarnya, saran yang ditampilkan tidak akan jauh berbeda dengan ponsel lamanya.

II. LANDASAN TEORI

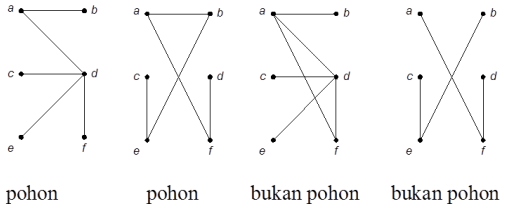
2.1 Matematika Diskrit

Matematika Diskrit adalah cabang matematika yang membahas objek-objek diskrit. Diskrit adalah keadaan dimana benda yang terdiri dari sejumlah elemen yang berbeda dan berhingga atau elemen yang tidak bersambungan. Lawan dari diskrit adalah kontinu. Hal yang dibahas pada matematka diskrit antara lain logika, teori himpunan, probabilitas, graf, pohon, dan lain-lain.

2.2 Pohon

2.2.1 Definisi Pohon

Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit. Dari definisi ini, dapat diketahui bahwa pohon memiliki dua sifat penting yaitu terhubung dan tidak mengandung sirkuit.



Gambar 1 – Pohon

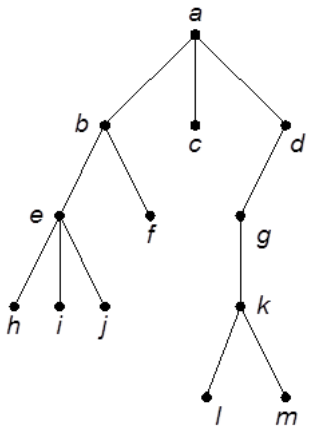
Misalkan G = (V,E) adalah graf tak sederhana dan jumlah simpulnya adalah n. Maka semua pernyataan di bawah ini ekivalen.

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.

2.2.3 Terminologi pada Pohon

Beberapa terminology pada pohon berakaar adalah

1. Anak (child atau children) dan Orangtua (parent) b, c, dan d adalah anak-anak simpul a, a adalah orangtua dari anak-anak itu.



Gambar 2 – Pohon

2. Lintasan (path)
   Lintasan dari a ke j adalah a, b, e, j. Panjang lintasan dari a ke j adalah 3.
3. Saudara kandung (sibling)
   f adalah saudara kandung e, tetapi g bukan saudara kandung e, karena orangtua mereka berbeda.
4. Upapohon (subtree)
   Yaitu simpul b dan e lalu daun f, h, i, j.
5. Derajat (degree)
   Derajat sebuah simpul adalah jumlah upapohon (atau jumlah anak) pada simpul tersebut. Derajat a adalah 3, derajat b adalah 2, Derajat d adalah satu dan derajat c adalah 0. Jadi, derajat yang dimaksudkan di sini adalah derajatkeluar. Derajat maksimum dari semua simpul merupakan derajat pohon itu sendiri. Pohon di atas berderajat 3.

2.2.4 Transversal pada Pohon

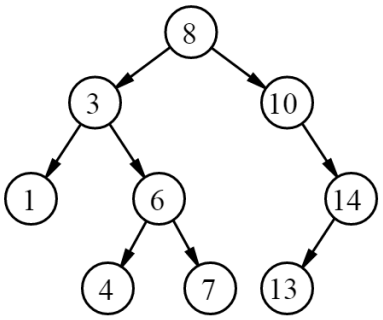
Pada sebuah pohon, transversal atau pencarian dilakukan secara rekursif. Terdapat tiga cara dalam melakukan transversal pada pohon,

1. Pre-order
   1. Lakukan pemrosesan terhadap akar.
   2. Transversal ke upapohon kiri.
   3. Transversal ke upapohon kanan.
2. In-order
   1. Transversal ke upapohon kiri.
   2. Lakukan pemrosesan terhadap akar.
   3. Transversal ke upapohon kanan.
3. Post-order
   1. Transversal ke upapohon kiri.
   2. Transversal ke upapohon kanan.
   3. Lakukan pemrosesan terhadap akar.

Ketiga proses ini akan menghasilkan urutan pemrosesan yang berbeda, hal ini dapat dilihat dengan cara menuliskan akar pohon melalui ketiga cara di atas.

2.2.5 Pohon Pencarian Biner

Pohon pencarian biner adalah pohon biner yang disusun secara terurut untuk mempercepat pencarian.



Gambar 3 – Pohon Pencarian Biner

Segala proses dari pohon pencarian biner harus dilakukan dengan mempertahankan keterurutan elemen pohon.

Cara melakukan pencarian pada pohon pencarian biner adalah secara preorder, yaitu dengan membaca akar dari pohon terlebih dahulu lalu dibandingkan dengan angka yang ingin dicari. Jika yang akan dicari lebih besar, maka yang akan diakses adalah upapohon kanan. Jika lebih kecil, maka yang selanjutnya diakses adalah upapohon kiri.

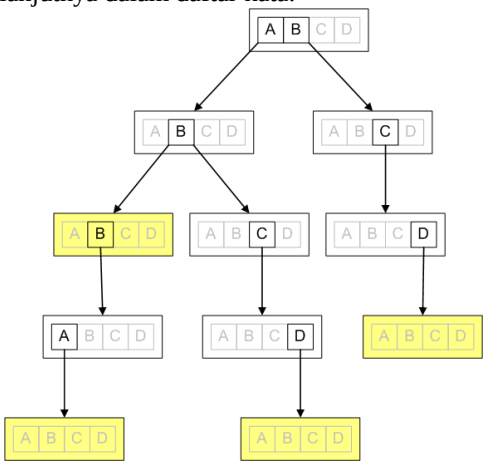
2.2.6 Pohon Pencarian Terner

Pohon pencarian terner adalah salah satu jenis pohon prefiks atau biasa disebut trie. Simpul pada pohon ini disusun dengan cara yang sama seperti pohon pencarian biner. Perbedaan kedua pohon ini adalah pohon pencarian terner mempunyai maksimal derajat tiga.

III. Penerapan Pohon dalam Fitur Autocomplete

3.1 Penggunaan Pohon dan Tabel dalam Autocomplete

Salah satu cara membuat fitur autocomplete adalah dengan membuatnya sebagai pohon dengan tipe elemen akarnya adalah tabel yang berisi alamat dari huruf selanjutnya dalam daftar kata.



Gambar 4 – Pohon dengan elemen akar tabel http://igoro.com/archive/efficient-auto-complete-with-a-ternary-search-tree/

Misalkan pengguna hendak mengetikkan kata “ABBA”, saat pengguna mengetikkan “A” maka program mengakses upapohon yang ditunjuk oleh tabel pada akar pohon dengan indeks A. Setelah itu program akan menampilkan kata yang mungkin dimaksud pengguna, yaitu “ABBA” dan “ABCD”.

Jika pengguna mengetikkan “C” pada awal kata, program akan mengakses tabel pada akar pohon dengan indeks C. Karena isi tabel dengan indeks C masih kosong, program tidak menampilkan saran apapun.

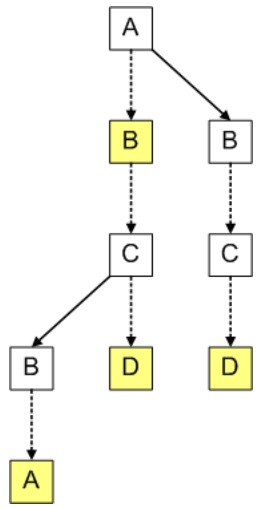
Cara ini sangatlah mudah, namun boros dalam pemakaian memori. Jika kita menginginkan semua abjad termasuk lower case dan upper case, maka diperlukan 52 elemen pada tabel. Padahal mungkin hanya sebagian kecil saja yang digunakan, ini akan menyebabkan pemborosan memori karena kebanyakan elemen tabel mungkin hanya menyimpan alamat kosong. Sementara kompleksitas pencarian algoritma ini adalah O(n) dimana n adalah panjang masukan.

3.2 Autocomplete dengan Pohon Terner

3.2.1 Ilustrasi Operasi Pohon Terner

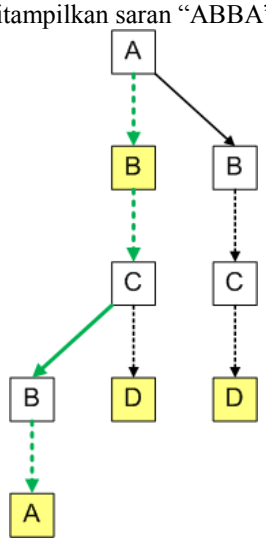
Salah satu cara mengatasi masalah pada autocomplete dengan pohon dengan elemen berupa tabel adalah dengan menggunakan pohon dengan elemennya berupa karakter.

Jika input yang dimasukkan sama dengan akar pohon, maka dilakukan transversal ke upapohon tengah. Jika berbeda, masukan akan dibandingkan dengan elemen akar pohon. Jika biner karakter masukan lebih besar dari elemen akar, dilakukan transversal ke upapohon kanan. Jika lebih kecil, dilakukan transversal ke upapohon kiri. Elemen yang diproses hanyalah elemen yang sama dengan masukan.



Gambar 5 – Pohon Terner

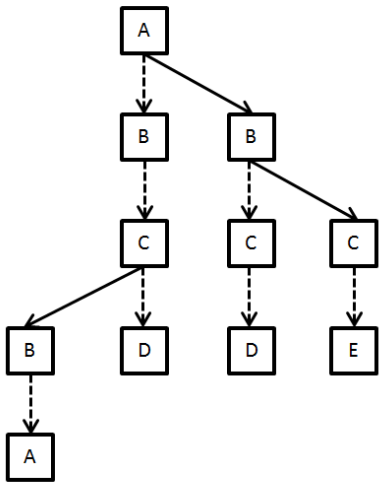
Jika pengguna mengetikkan huruf A, karena akar pohon adalah A, maka dilakukan transversal ke upapohon tengah. Jika masukan selanjutnya adalah B, maka dilakukan transversal ke upapohon tengah, dan akar pohon adalah C. Jika masukan selanjutnya adalah B, karena biner B lebih kecil dari C, dilakukan transversal ke upapohon kiri. Akar pohon selanjutnya adalah B yang sama dengan masukan ketiga pengguna, maka dilakukan transversal ke upapohon tengah. Selanjutnya akan ditampilkan saran “ABBA”.



Gambar 6 – Ilustrasi pencarian kata “ABBA” dalam pohon terner

Jika pada masukan pertama pengguna mengetikkan karakter C, maka dilakukan transversal ke upapohon kanan dengan akar B. Karena masih tidak sama, seharusnya program akan melakukan transversal ke upapohon kanan lagi, tetapi karena alamat upapohon kanan kosong, maka pencarian dihentikan dan program tidak memberikan saran apapun.

Apabila pengguna ingin menambahkan kata “CE” pada daftar kata, maka pertama dilakukan transversal untuk huruf pertama, C. Transversal ini akan berhenti pada simpul B paling kiri. Karena C berada setelah B, maka C dialokasikan sebagai upapohon kanan dari B. Setelah itu E dialokasikan sebagai upapohon tengah dari B.



Gambar 7 – Penambahan kata pada pohon terner.

Kompleksitas pencarian algoritma ini adalah O(n log(n)) dimana n adalah panjang masukan.

3.2.2 Implementasi Sederhana Autocomplete dengan Pohon Terner

Semua contoh implementasi sederhana berikut penulis buat menggunakan bahasa C.

Berikut adalah definisi dari pohon.

```
/*** Definisi Type Pohon Terner ***/
typedef struct tNode *AddressTree;
typedef struct tNode {
      Infotype Info;
      AddressTree Left;
      AddressTree Right;
      AddressTree Center;
} Node;
typedef AddressTree BinTree;
```

Untuk selanjutnya, penulis menggunakan:

Left(P) untuk menggantikan P->Left
Right(P) untuk menggantikan P->Right
Center(P) untuk menggantikan P->Center
Akar(P) untuk menggantikan P.Info

Untuk mengimplementasikan prosedur add, digunakan fungsi InsSearch.

```
BinTree InsSearch (BinTree P, Infotype X)
{
      BinTree T,Ptemp;
      if (IsTreeEmpty(P))
            return Tree(X,Nil,Nil,Nil);
      else
      {
            if(Akar(P) < X){
                  Ptemp = Left(P);
            }
            else{
                  Ptemp = Right(P);
            }
            T = InsSearch(Ptemp,X);
            Ptemp = P;
            while (!(IsOneElmt(Ptemp))){
                  if(X<Akar(Ptemp))
                        Ptemp = Left(Ptemp);
                  else
                        Ptemp = Right(Ptemp);
            }
            if (X<Akar(Ptemp))
                  Left(Ptemp) = T;
            else
                  Right(Ptemp) = T;
            return P;
      }
}
```

Dimana fungsi antara Tree mengalokasikan X sebagai akar pohon dan parameter kedua sampai keempat secara berurutan adalah alamat yang akan dialokasikan sebagai Left, Center, dan Right pohon.

Untuk mencari apakah suatu string terdapat dalam daftar kata, digunakan fungsi Contains.

```
boolean Contains(char* s, BinTree T)
{
      int pos = 0;
      BinTree P = T;
      if (strlen(s) == 0)
      {
            return false;
      }
      else
      {
            while (P != NULL)
            {
                  if (s[pos] < Akar(P))
                        P = Left(P);
                  else if (s[pos] > Akar(P))
                        P = Right(P);
                  else
                  {
                        if (++pos == strlen(s))
                              return true;
                  }
            }
      }
      return false;
}
```

Selanjutnya untuk menampilkan saran kata, dapat dengan cara menulis ulang masukan pengguna diikuti tebakan dari program yang dapat ditulis dengan cara menulis hanya akar upapohon yang diperoleh dengan cara seperti pada fungsi Contains diikuti dengan center dari suatu simpul dari upapohon tersebut.

IV. KEKURANGAN

Kekurangan dari algoritma ini adalah algoritma ini belum memuat perhitungan dari seberapa sering suatu kata dalam daftar diketikkan. Dengan kata lain, saran yang diberikan oleh algoritma ini masih acak sesuai dengan urutan kata tersebut ditambahkan.

V. KESIMPULAN

Dalam beberapa kasus, modifikasi dari algoritma yang telah ada penting untuk meningkatkan efektifitas dari proses pencarian, meminimalisir waktu pencarian dari suatu algoritma pencarian, dan menghemat pemakaian memori. Pada proses autocomplete yang dilakukan dengan pohon terner jauh lebih hemat dalam hal pemakaian memori daripada dengan pohon yang elemennya berupa tabel, walaupun kompleksitas dalam pencariannya lebih tinggi daripada dengan pohon yang elemennya berupa tabel.

VII. UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan kepada Tuhan YME karena atas izin-Nya makalah ini dapat selesai. Penulis mengucapkan terima kasih kepada bapak Rinaldi Munir atas bimbingannya selaku dosen matematika diskrit, serta kepada pendahulu-pendahulu yang telah memberikan

karya-karya terkait algoritma penyelesaikan berbagai masalah terutama dalam masalah pencarian dan autocomplete sehingga membantu dalam menyelesaikan makalah ini.

REFERENCES

[1] http://igoro.com/archive/efficient-auto-complete-with-a-ternary-search-tree/ , 8 Desember 2015 12.29 WIB.

[2] Munir, Rinaldi. Diktat Kuliah IF2120 Matematika Diskrit. Bandung : Penerbit Informatika. 2008

[3] Tam, C. & Wells, D. 2009. Evaluating the Benefits of Displaying Word Prediction Lists on a Personal Digital Assistant at the Keyboard Level.

[4] https://autohotkey.com/board/topic/49517-ahk-11typingaid-v2220-word-autocompletion-utility/, 9 Desember 2015 19.00 WIB.

[5] http://www.reputationstation.com.au/changing-autocomplete-suggested-terms-in-google-searches/, 12 Desember 2015 18.50 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, 8 Desember 2015

ttd

Nugroho Satriyanto - 13514038