Studi Digital Tree dan Aplikasinya pada Kamus

Anis Istiqomah - NIM : 13505116

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if115116@students.if.itb.ac.id

Abstrak

Fasilitas pengolah kata (pemeriksa ejaan, *thesaurus*, dan lain-lain) elektronis adalah sebuah fasilitas yang memungkinkan pengguna aplikasi pengolah kata memeriksa ejaan dari dokumen yang ia ketik. Di negara-negara maju penggunaan fasilitas pengolah kata elektronis sangat umum, sehingga menjadi salah satu indikator pemilihan terhadap pengolah kata yang hendak dipakai. Penggunaan kamus elektronis dalam aplikasi pemrosesan teks merupakan hal yang tidak dapat dihindarkan.

Dari beberapa struktur data yang dapat digunakan untuk pengolahan kamus salah satunya adalah digital tree atau yang bisa disebut juga dengan trie. Struktur data trie adalah model struktur data yang paling baik untuk pemrosesan kamus.

Makalah ini merupakan studi tentang struktur data pada digital tree, bagaimana cara penyimpanan datanya dan skema pencariannnya yang efektif.

***Kata Kunci** : digital tree, trie, patricia trie*

1. Pendahuluan

Dengan semakin berkembangnya kemajuan teknologi, penggunaan komputer sebagai fasilitas untuk mempermudah pekerjaan sudah dilakukan di mana-mana. Misalnya untuk menulis. Tetapi, para pengguna komputer di Indonesia belum dapat menggunakan pemeriksa ejaan bahasa Indonesia karena fungsi tersebut belum ada pada aplikasi pengolah kata yang mereka gunakan .

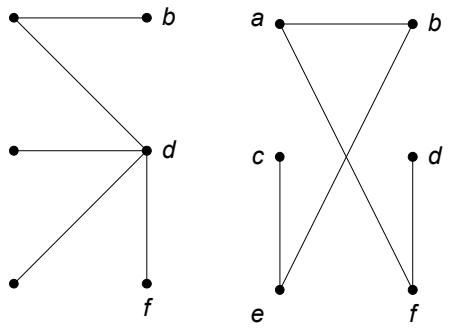
Fasilitas pengolah kata (pemeriksa ejaan, *thesaurus*, dan lain-lain) elektronis adalah sebuah fasilitas yang memungkinkan pengguna aplikasi pengolah kata memeriksa ejaan dari dokumen yang ia ketik. Hal ini dapat meminimumkan kemungkinan salah eja atau salah ketik. Di negara-negara maju penggunaan fasilitas pengolah kata elektronis sangat umum, sehingga menjadi salah satu indikator pemilihan terhadap pengolah kata yang hendak dipakai. Penggunaan kamus elektronis dalam aplikasi pemrosesan teks merupakan hal yang tidak dapat dihindarkan.

Saat ini kamus berbasis web sudah banyak dibuat. Ketika akan membuat kamus, berarti kita akan membuat sebuah database berisi kumpulan kata-kata. Sekumpulan informasi yang terdapat di database nantinya akan diambil kembali. Dalam hal ini, terjadi dua masalah yaitu penyimpanan informasi *(Information storage)* dan pengambilan kembali *(Information Retrieval*). Proses pengambilan informasi harus dilakukan secepat mungkin dan struktur data penyimpanannya harus mampu mengakomodasi hal ini.

Terdapat tiga struktur data untuk pengorganisasian data, yakni search trees, digital trees, dan hashing. Untuk keperluan kamus elektronis, struktur data digital tree, atau sering disebut dengan trie, merupakan salah satu struktur data yang efisien. Baik dalam pencarian kata maupun penghematan memori untuk penyimpanan huruf-hurufnya, baik pada saat berada di main memori maupun pada saat disimpan di file.

2. Tree

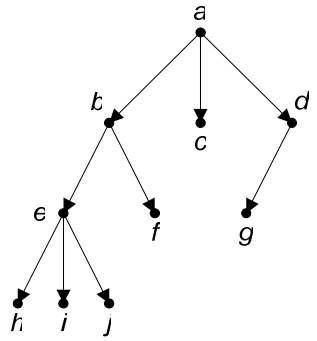
Pohon (tree) adalah graf tak berarah terhubung yang tidak mengandung sirkuit. Pohon juga didefinisikan sebagai graf tak berarah dengan yang sifat hanya terdapat sebuah lintasan unik antara setiap pasang simpul.



(a) . Contoh Pohon

2.1. Pohon Berakar

Untuk representasi struktur data pada kamus, pohon yang digunakan adalah pohon berakar (*rooted tree*). Pohon berakar berarti pohon yang satu buah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah.



(b). Contoh pohon berakar (*rooted tree*)

Beberapa istilah pada pohon berakar yang penting dan akan banyak digunakan pada trie adalah :

- Orang tua (*parent*)

  a adalah orangtua dari *b, c, d*

- Anak (*child*)

  b, c, d anak dari *a*

- Saudara kandung (*sibling*)

  f adalah saudara kandung *e*, tetapi *g* bukan saudara kandung *e*, karena orangtua mereka berbeda.

- Akar (*root*)

  Simpul *a* adalah akar dari pohon di atas

- Daun (*leaf*)

  Simpul yang tidak mempunyai anak (terletak di paling bawah)

- Simpul dalam (*internal node*)

  Simpul yang mempunyai anak disebut simpul dalam. Simpul *b, d*, dan *e* adalah simpul dalam.

2.2. Pohon Biner

Pohon biner juga merupakan struktur yang penting karena banyak diaplikasikan di dunia komputer. Yang disebut dengan pohon biner adalah :

- pohon *n-ary* dengan $n = 2$.
- Setiap simpul di dalam pohon biner mempunyai paling banyak 2 buah anak.
- Dibedakan antara anak kiri (*left child*) dan anak kanan (*right child*)
- Karena ada perbedaan urutan anak, maka pohon biner adalah pohon terurut.

2.3 Contoh-contoh Terapan Pohon Biner

Beberapa contoh struktur yang menggunakan pohon biner adalah pohon ekspresi (*expression tree*), pohon keputusan (*decision tree*), kode prefix (*prefix code*), kode Huffman (Huffman code), pohon pencarian biner (*binary search tree*), *digital tree (trie).*

Contoh-contoh pohon tersebut banyak digunakan untuk berbgai keperluan.

Pohon pencarian biner dan digital tree dapat dijadikan model struktur data suatu kamus elektronis. Masing-masing model memiliki kelebihan dan kekurangannya masing-masing. Dalam tulisan ini yang akan dibahas adalah struktur data kamus dengan model digital tree.

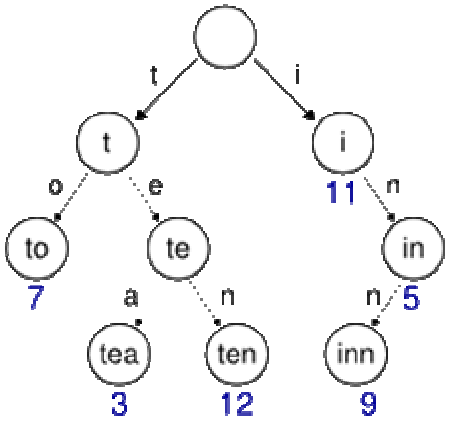
Model struktur data kamus selain kedua model yang telah disebutkan antara lain :

- Model Hash
- Model hibrida antara Hash dan pohon pencari biner

3. Struktur Data Digital Tree (Trie)

Tries pertama kali diperkenalkan pada tahun 1960-an oleh Fredkin. Trie atau *digital tree* berasal dari kata *retrival* (pengambilan kembali) sesuai dengan fungsinya. Secara etimologi kata ini diucapkan sebagai 'tree'. Meskipun mirip dengan penggunaan kata 'try' tetapi hal ini bertujuan untuk membedakannya dari general tree**.**

Dalam ilmu komputer, trie, atau prefix tree adalah sebuah struktur data dengan representasi *ordered tree* yang digunakan untuk menyimpan *associative array* yang berupa string. Berbeda dengan *binary search tree* (BST) yang tidak ada node di tree yang menyimpan elemen yang berhubungan dengan node sebelumnya dan, posisi setiap elemen di tree sangat menentukan. Semua keturunan dari suatu node mempunyai prefix string yang mengandung elemen dari node itu, dengan root merupakan string kosong. Values biasanya tidak terkandung di setiap node, hanya di daun dan beberapa node di tengah yang cocok dengan elemen tertentu.



(c). Contoh digital tree (trie)

Digital tree juga dapat digunakan untuk :

- Mengimplementasikan kamus suatu ADT dimana operasi standarnya seperti MakeEmpty, Search, Insert, Delete, dapat ditampilkan
- Encoding dan kompresi data
- Regular expression search dan aproksimasi pencocokan string

Regular expression adalah suatu cara untuk mendefinisikan suatu pola karakter. Contoh dari regular expression adalah [aB]*[cd][efg] dimana * adalah indikator pengulangan.

Ekspresi di atas merepresentasikan string apa saja dengan properties :

* String dimulai dari a, b, atau nol
* Diikuti dengan c atau d
* Diikuti dengan e, atau f, atau g

Sebagai contoh, ‘aaace’ and ‘bdg’ adalah string yang dapat direpresentasikan dengan regular expression tersebut.

Contoh lain regular expresion adalah ‘^’, ’.*’, ‘$’ . Tanda ‘^’ menandakan dimulainya baris baru, titik dapat merepresentasikan apa saja, dan $ menandakan akhir dari sebuah baris.

Kita dapat mengetik command '[aB]*[cd][efg]' <filename> dan semua substring yang didefinisikan dengan regular expression yang terdapat di <filename> akan ditunjukkan.

Trie dibentuk dari input string. Misalnya input adalah sebuah himpunan sejumlah n kata yaitu $S_1,S_2,..S_n$, dan setiap S_i terdiri dari simbol-simbol alfabet dan mempunyai simbol terminasi yang unik misalnya $.

Alfabet dalam hal ini bisa berupa :

* {0,1} untuk file biner
* {seluruh karakter ASCII yang berjumlah 256}
* {a,b,c,d,...,x,y,z}

Trie tidak hanya digunakan untuk mencari string pada text biasa tetapi juga dapat digunakan untuk mencari pola gambar.

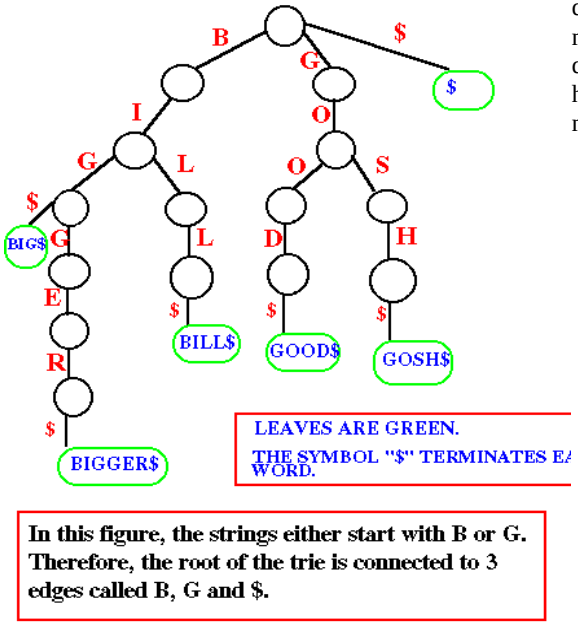
Jenis-jenis trie :

- *Non compact tries*
- *Compact tries*
- Tries yang disebut “PATRICIA” yang lebih *compact*
- *Suffix tries*
- *Suffix trees*

3.1. Non Compact Tries

Non Compact Tries adalah trie yang di setiap sisinya merepresentasikan simbol alfabet. Untuk seterusnya, yang merupakan simbol alfabet akan dituliskan dalam huruf kapital a A sampai Z dengan simbol terminasi ‘$’.

Misalnya 5 buah string : BIG, BIGGER, BILL, GOOD, GOSH dibuat dalam bentuk *non compact tries*.



(d). Non Compact Tries

Pada proses pencarian, ketika mencari kata GOOD, kita akan mulai dengan root, kemudian mengikuti sisi G, yang diikuti O, O yg berikutnya, dan terakhir D.

Jika kita mencari kata BAD, kita akan mulai dari root, mengikuti sisi B dan menemukan bahwa A tidak ada di sisi setelahnya. Dengan demikian, BAD tidak ada dalam text.

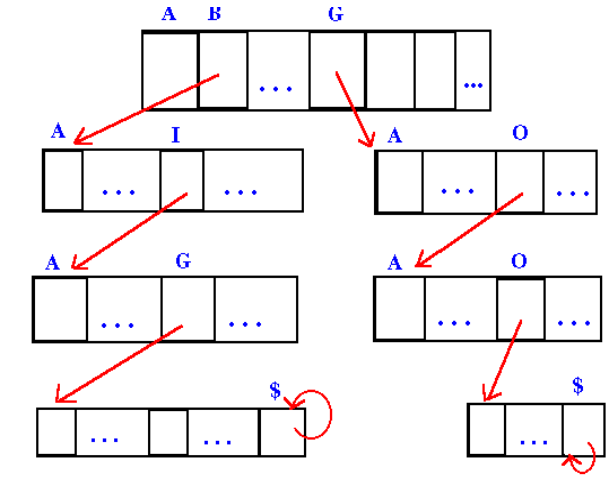
Struktur di atas boros karena setiap sisi merepresentasikan satu buah simbol. Untuk text yang besar, hal ini akan menimbulkan banyak pemborosan. Sebagai gantinya, kita harus mencari representasi bentuk trie yang lebih padat.

Dua jenis implementasi trie :

1. Implementasxi yang pertama menggunakan array of pointer

Setiap cell array mempunyai mempunyai index. Jika alfabet berisi huruf a sampai z, maka index array adalah rentang dari a sampai z ditambah dengan simbol terminasi '$'. Jadi ukuran array sama dengan jumlah alfabet ditambah satu.

Di bawah ini adalah sebuah representasi array untuk string BIG dan GOO. String dimulai dengan huruf B atau G. Dengan demikian pada array pertama, cell B dan G punya anak yang diberi pointer ke array berikutnya. Cell lain mempunyai pointernya nil. Pada contoh yang diberikan, tidak ada kata yang diawali dengan huruf A, sehingga pointer pada huruf A menunjuk nil.



Trie representation for words BIG and GOO using array of pointers

(e). Trie dengan array of pointer

Implementasi ini boros karena sebagian besar pointernya menunjuk ke nil.

2. Implementasi kedua adalah dengan linked list

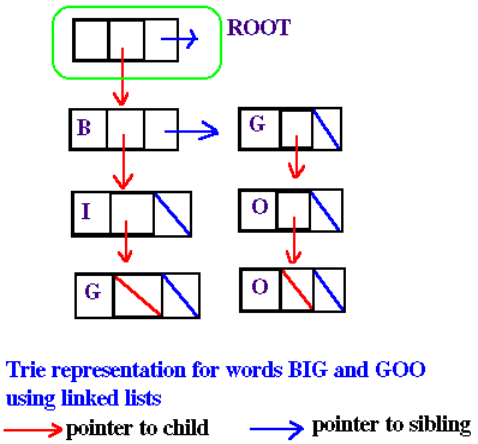
Setiap simpul linked list merepresentasikan sebuah simbol dan mempunyai pointer ke sibling dan ke child.

Contoh :

$[s_1...s_{i-1}s_i...s_n\$\ s_1...s_{i-1}\ t_1...t_m\$]$

Anak dari S_k adalah S_{k+1} untuk setiap k yang memenuhi $1< k < n$ (dimana n+1 = $)

Sibling dari S adalah t_1. Tidak ada pointer ke Sibling lain.

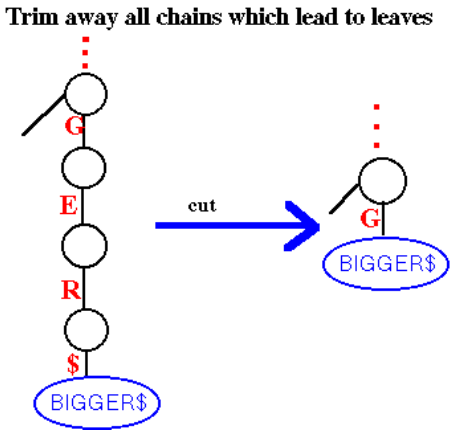


(f). Trie dengan Linked List

3.2. Compact Tries

Jenis trie ini menggunakan gambar (d) tetapi rangkaian yang mengarah ke daun disusun kembali.

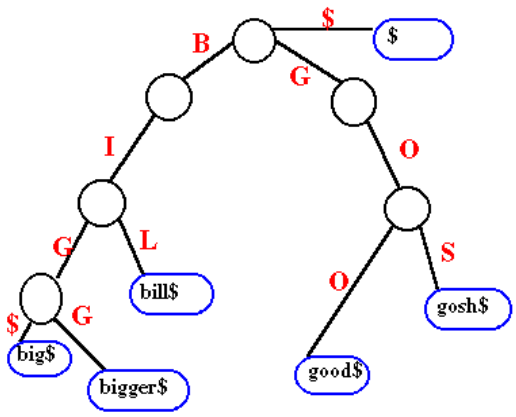
Proses pemotongannya seperti ditunjukkan pada gambar (g).



(g). Proses pemadatan Simpul

Pada *non compact tries* setiap huruf dari string BIGGER menempati satu buah sisi. Meskipun setelah huruf G kedua sudah dapat dipastikan string apa yang akan dicapai, setiap huruf tetap diberi tempat pada satu buah sisi. Pada *non compact tries*, karena setelah huruf G kedua, string sudah dapat dipastikan maka satu buah simpul terakhir berisi lebih dari satu karakter.

Hasil akhir simpul-simpul yang dipadatkan dari gambar (d) menghasilkan gambar di bawah ini :



(h). Compact Tries

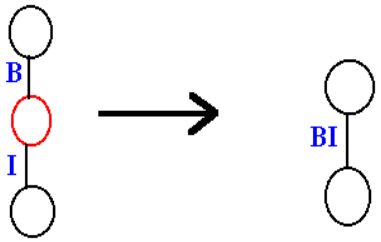
Jumlah daun adalah n+1, dimana n adalah jumlah input string. Lebih jauh lagi, pada daun, kita dapat menyimpan string itu sendiri atau pointer ke string itu.

Compact Tree seperti ini masih dapat lebih dimampatkan lagi yang akan dibahas pada subbab berikutnya.

3.3. PATRICIA Tries

PATRICIA adalah singkatan dari *Practical Algorithm to Retrieve Information Code in Alpha Numeric*. Tries ini akan berbeda dari dari trie sebelumnya karena sebuah simpul dapat ditempati oleh lebih dari 1 karakter. Oleh karena itu, semua unary node akan dihapus.

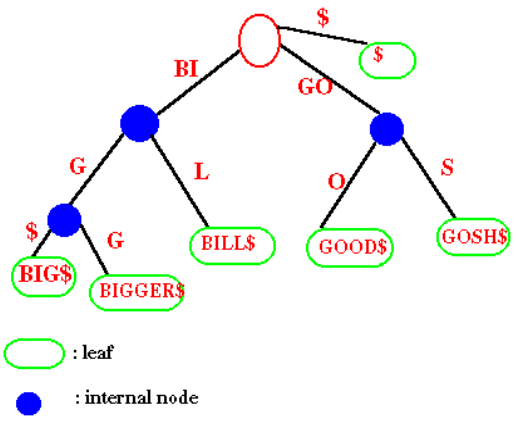
Gambar dibawah ini menunjukkan proses pemotongan :



Before, one edge is used to represent a character. Now, the red node (unary node) is collapsed and one edge can hold more than one character.

(i). Pemotongan simpul untuk membentuk PATRICIA Trie

Hasil dari pemotongan di atas ditunjukkan pada gambar di bawah ini :



(j). PATRICIA Trie

Mirip dengan compact trie, pada Patricia Trie juga dilakukan pemotongan simpul. Perbedaannya adalah pada Patricia Trie pemotongan simpul tidak hanya dilakukan pada simpul yang sudah tidak mempunyai dua cabang lagi. Pada Patricia Trie,pemotongan dapat dilakukan di tengah simpul yang anak-anaknya masih mungkin mempunyai dua anak. Hasil dari pemotongan sebuah simpul pada Patricia Tries tidak menghasilkan simpul yang merupakan kata yang lengkap. Pemotongan simpul hanya menghasilkan gabungan beberapa huruf.

Pemotongan seperti ini dapat dilakukan di inernal node sehingga akan semakin banyak jumlah node yang dipadatkan. Dengan demikian, akan semakin menghemat memori.

Pada Binary Patricia Trie, jumlah internal node sama dengn jumlah daun dikurangi satu.

3.3. Suffix Tries

Ide suffix tries adalah mengisi suatu nilai ke setiap simbol yang terdapat pada sebuah text. Setiap index disesuaikan deengan posisinya dalam text.

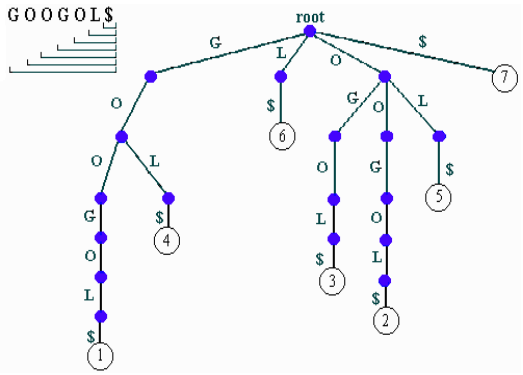
Misalnya, simbol pertama punya index 1, simbol terakhir punya index n pada text. Untuk membangun suffix trie, kita menggunakan index ini menggantikan objek nyata.

Contoh :

TEXT : G O O G O L $

POSITION : 1 2 3 4 5 6 7

Hal tersebut akan menghasilkan trie seperti yang ditunjukkan di bawah ini :



(k). Contoh Suffix Trie

Pada suffix trie, root memiliki anak sejumlah jumlah huruf atau index. Pada ujung daun di atas, untuk memudahkan ditunjukkan index dari awal sampai akhir. Anak dari root akan mengandung akhiran huruf dimulai dari index.

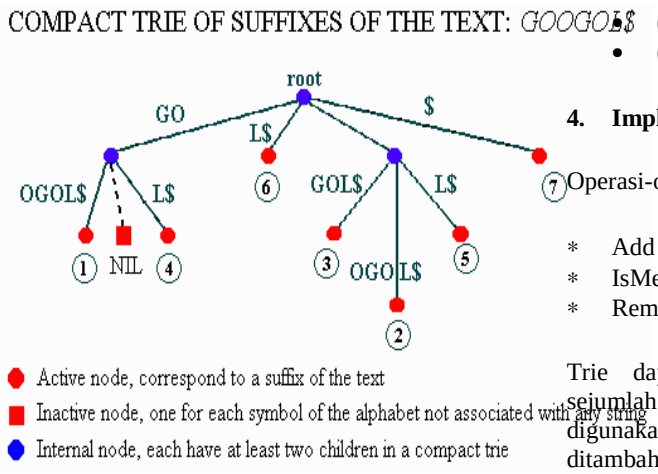
Struktur ini mempunyai beberapa keuntungan :

* Membutuhkan space yg lebih sedikit
* Text dapat direpresentai sebagai bin, ASCII, dan lain-lain
* Tidak perlu menyimpan objek yang sama dua kali.

3.4. Suffix Tree

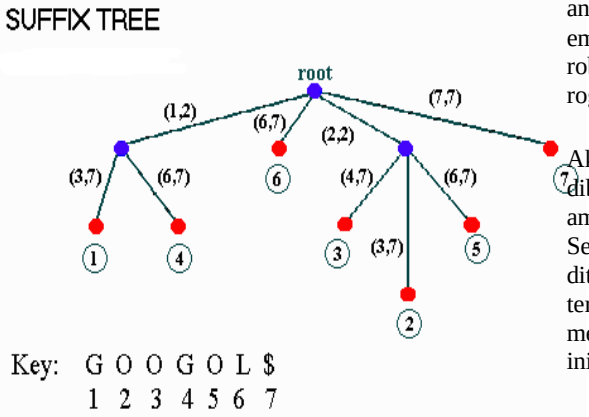
Suffix Tree dibuat dari Trimming (pemadatan dan pemotongan) dari suffix trie.

Gambar dibawah ini menunjukkan suffix trie yang dipadatkan.



(l). Compact Trie dari Suffix Trie

Setelah melalui proses trimming maka suffix trie di atas menjadi suffix tree.



(m). Contoh Suffix Tree

We store pointers rather than words in the leaves. And we replaced every string by a pair of indices, (a,b), where a is the index of the beginning of the string and b the index of the end of the string. i.e: We write

- (3,7) for OGOL$
- (1,2) for GO
- (7,7) for $

Pointer disimpan di daun menggantikan kata. Dan setiap string tempatnya digantikan dengan sepasang index (a,b). a adalah index awal string dan b adalah index akhir string. Contoh :

- (3,7) menggantikan OGOL$
- (1,2) menggantikan GO
- (7,7) menggantikan $

4. Implementasi Trie dalam bahasa C

Operasi-operasi pada trie :

* Add
* IsMember
* Remove

Trie dapat digunakan untuk menyimpan sejumlah data yang mempunyai key (sesuatu digunakan untuk mengenali data) dan dapat ditambah sebuah nilai yang menyimpan data tambahan lain yang bersangkutan dengan key.

Misalnnya kita akan menyimpan nama dan usia orang-orang di bawah ini

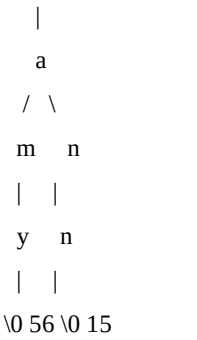
amy 56
ann 15
emma 30
rob 27
roger 52

Akan dimulai dengan amy. Pohon akan dibangun dengan setiap karakter pada nama amy yang akan dipisahkan dengan node. Setelah huruf terakhir di namanya, ditambahkan sebuah node lagi. Pada node yang terakhir ini, kita simpan nul karakter '\0' untuk menunjukkan akhir dari nama. Node terakhir ini juga tempat menyimpan usia amy.

```
.    <- level 0 (root)
|
a    <- level 1
|
m    <- level 2
|
y    <- level 3
|
\0 56  <- level 4
```

Untuk menambahkan nama ann, kita harus melakukan prosedur yang sama. Karena huruf a pada level 1 sudah tersimpan, tidak perlu menyimpan huruf a lagi dari karakter ann. Huruf a yang sudah ada, digunakan lagi sebagai karakter pertama. Di bawah huruf a, hanya ada huruf m, tetapi karena huruf kedua dari ann adalah n mka harus ditambahkan sebuah cabang baru setelah a yaitu n. setelah

akhir dari string ann ditambahkan juga karakter nul dan usianya.



Cara ini dilakukan terus untuk menambahkan string baru. Dan setiap tidak ditemukan huruf pada trie yang sama dengan huruf dari kata yang akan disimpan maka tambahkan cabang yang baru.

Trie yang akan terbentuk nantinya adalah non compact trie.

Mencari sebuah key dapat dilakukan dengan iterasi. Garis besar algoritma ismember :

a. Cari top level di key untuk node yang sama dengan first char
b. Jika tidak ada return false else
c. Jika karakter yang sesuai adalah '\0' return true else
d. Pergi ke subtrie yang sama dengan current karakter
e. Advance ke next character di key
f. Pergi ke step 1

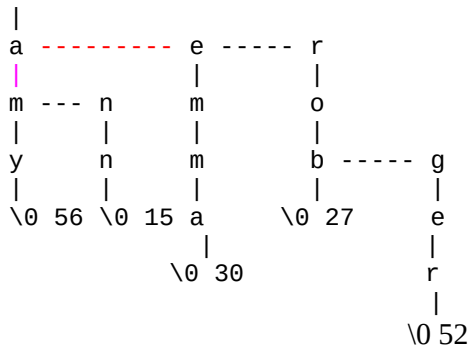
Struktur trie seperti ini dapat diimplementasikan ke dalam bahasa C. Dapat dibuat suatu ADT (*Abstract Data Type*) trie.

Tipe dasar

```
typedef int trieValueT;
typedef struct trieNodeTag
{
    char key;
    trieValueT value;
    ...
} trieNodeT;
```

Harus dibuat struktur trie yang simpulnya punya beberapa anak. salah satu caranya adalah dengan memiliki anak dari simpul menjadi bagian dari *linked list of nodes.*

Jika kita melihat sibling pada sebuah level sebagai linked list, kita akan melihat struktur di atas akan terlihat seperti :



Untuk mengimplementasikan struktur ini, dibutuhkan dua pointer pada node yang diberikan :

```
typedef struct trieNodeTag
{
 char key;
 trieValueT value;
 struct trieNodeTag
*next,*children;
} trieNodeT;
```

Karena pointer harus diimplementasi dengan trie, maka struct trieCDT:

```
typedef struct trieCDT
{
  trieNodeT *root;
}  trieCDT;
```

Dengan demikian, akan terbentuk file trie.h yang terdiri dari :

```
typedef int trieValueT;

typedef struct trieCDT *trieADT;
```

Sedangkan file trie.c akan berisi :

```
typedef struct trieNodeTag
{
  char key;
  trieValueT value;
  struct trieNodeTag *next,
*children;
}  trieNodeT;

typedef struct trieCDT

{
```

```
  trieNodeT *root;
} trieCDT;
```

Dibawah ini adalah implementasi algoritma pencarian yang telah dibahas sebelumnya

```
int TrieIsMember(trieADT trie, char keys[])
{
 /* Mulai dari top level. */
 trieNodeT *level = trie->root;
 /* Mulai pada awal key. */
 int i = 0;
 for (;;)
 {
   trieNodeT *found = NULL;
   trieNodeT *curr;

   for (curr = level; curr != NULL; curr = curr->next)
   {
     /*
      * menginginkan simpul pada level ini
      * sesuai dengan current character di key
      */
      if (curr->key == keys[i])
      {
         found = curr;
         break;
      }
   }
   /*
   * If either no nodes at this level or none
   * with next character in key, then key not
   * present.
   */
   if (found == NULL)
     return 0;

   /* If we matched end of key, it's there! */
   if (keys[i] == '\0')
     return 1;

   /* Go to next level. */
   level = found->children;

   /* Advance in string key. */
   i++;
 }
}
```

5. Penerapan Digital Trie pada Kamus

Seperti telah dijelaskan sebelumnya, digital tree atau trie seringkali dipakai untuk menyimpan data yang berupa string.

Aplikasi trie yang sudah umum adalah untuk menyimpan kamus seperti yang sudah diterapkan pada telepon selular. Aplikasi seperti ini memanfaatkan kemampuan trie untuk melakukan pencarian dengan cepat.

Metode ini juga berhasil menghemat pemakaian memori untuk penyimpanan huruf-hurufnya, baik pada saat berada di main memori maupun pada saat disimpan di file.

Digital tree sebagai struktur data pada kamus selain memiliki keuntungan yang telah disebutkan di atas, juga memiliki kerugian. Struktur data lain yang juga dapat diterapkan pada kamus adalah Binary Search Tree (BST).

Keuntungan tries dibandingkan dengan Binary Search Tree :

- Mencari sebuah kata akan lebih cepat. mencari sebuah kata dengan panjang m, ambil worst case, akan memakan waktu O(m) time. Dengan BST akan memakan waktu O(log n) time, dimana n adalah jumlah elemen di tree, karena mencari tergantung pada ketinggian pohon yang setara dengan logaritma
- Trie meminta space yg lebih sedikit ketika harus mengandung kata-kata pendek dalam jumlah banyak, karena kata tidak disimpan secara eksplisit dan kata dibagi dengan node yang lain
- Trie tidak harus selalu seimbang, yang merupakan keharusan bagi BST, karena hal tersebut akan sangat berpengaruh

Kerugian :

- Trie dapat memberikan elemen yang berurutan, tetapi harus sesuai dengan urutan kamus

- Trie dapat menjadi sangat besar dalam suatu keadaan, seperti trie yang mengandung beberapa string yang sangat panjang
- Algoritma trie lebih kompleks daripada BST

6. Kesimpulan

Digital tree atau bisa disebut juga dengan trie terdiri dari beberapa jenis, yaitu *non compact tries*, *compact tries*, *Patricia Tries*, *Suffix tries*, dan *Suffix trees.*

Struktur data digital tree tidak semuanya hemat memori. Salah satu jenis trie yang paling hemat memori adalah Patricia Tries dan Suffix Tree.

Dalam pemrosesan kamus elektronis, digital tree adalah struktur data yang paling baik untuk digunakan dibandingkan dengan struktur data yang lain. Namun, tidak berarti struktur data ini tidak memiliki kekurangan.

7. Daftar Pustaka

[1] *http://www.answers.com*
Tanggal akses : 31 Desember 2006 pukul 19.50

[2] *http:// www.cs.mcgill.ca/~cs251/*
Tanggal akses : 3 Januari 2007 pukul 0.12

[3] *http://www.cs.bu.edu/teaching/c/tree/trie/*
Tanggal akses : 3 Januari 2007 pukul 0.12

[4] *http://informatika.org/~rinaldi*
Tanggal akses : 2 Januari 2007 pukul 22.59

[5] *http://mail.its-sby.edu/~agusza/*
Tanggal akses : 30 Desember 2006 pukul 13.08

[6] *http://www.cs.ui.ac.id/WebKuliah/*
Tanggal akses : 31 Desmber 2006 pukul 19.49