Optimalisasi Algoritma Pencarian Data Memanfaatkan Pohon Biner Terurut

Mohammad Rizky Adrian [1)]
1) Jurusan Teknik Informatika ITB, Bandung 40132, email: if17108@students.if.itb.ac.id

Abstract – *Makalah ini membahas tentang pohon, dan penerapannya dalam usaha mengoptimalisasi algoritma pencarian data serta . Dalam struktur data seringkali kita membutuhkan pencarian satu atau beberapa data dari sekelompok data. Hal ini mungkin seringkali kita mengabaikan waktu yang dipakai dengan melakukan traversal sekelompok data tersebut secara biasa. Namun jika Anda membutuhkan fungsi pencarian tersebut terus menerus dalam ruang lingkup program yang besar, maka pencarian data bisa berlangsung cukup lama. Oleh karena itu, diperlukan metode- metode tertentu untuk mengoptimalisasi pencarian data tersebut. Dalam kasus kali ini, pencarian data akan dioptimalisasi secara rekursif dengan pohon biner.*

Kata Kunci*: Kompleksitas Algoritma, Maksimalisasi, Pohon, Metode Pencarian.*

1. PENDAHULUAN

Sejak masuk ke abad 21, internet mulai diminati orang di seluruh penjuru dunia. Hal tersebut ditunjang oleh berbagai hal: kemajuan teknologi komputasi, penemuan-penemuan penting seperti semi konduktor ataupun transistor, dan rasa ingin tahu manusia untuk terus maju mengembangkan ilmu pengetahuannya.

Kita berada di era informasi dimana saat ini semua informasi yang kita butuhkan sudah tersedia di dunia maya. Informasi disajikan dalam bentuk data. Umumnya data tersebut dibuat dalam suatu paket.

Dalam jaringan, dikenal yang namanya OSI Layer yaitu tahapan-tahapan dalam pengiriman data. OSI Layer merupakan model ideal dari koneksi logis yang harus terjadi agar komunikasi data dalam jaringan dapat berlangsung.

Dalam prakteknya perpindahan informasi data berlangsung dengan sangat cepat, namun tak dapat disangkal jika segala sesuatu pasti ada batasnya. Jika data sangat besar pemrosesan akan semakin rumit dan membutuhkan waktu yang lebih lama.

Dalam kasus pencarian data misalnya, jika data yang tersedia adalah semua komputer yang ada di seluruh dunia, maka jika kita harus mencari satu persatu masuk ke komputer tersebut, tidak dapat dibayangkan berapa lama Anda akan menemukan data tersebut.

Tidak semua prosedur harus dilakukan secara *brute force* yang memiliki pengertian metode menemukan data dari semua kemungkinan yang ada. Misalnya mesin pencari data, mesin pelacak sandi lewat, atau dalam kehidupan sehari-hari: Anda mencari buku di rak perpustakaan yang tidak terorganisir dengan baik dengan mengecek satu persatu rak yang ada.

Oleh karena itu diperlukan metode yang dapat meningkatkan kemudahan menemukan data yang dicari. Ada dua hal yang dapat dilakukakan. Pertama manajemen penyimpanan data harus dilakukan dengan baik. Misalnya pemberian label pada setiap buku di perpustakaan lalu menyimpannya dalam katalog buku atau komputer, pemberian kata kunci pada artikel di blog Anda. Kedua, dengan mengoptimalkan metode atau media pencarian data yang digunakan.

Peningkatan penggunaan internet juga tidak terlepas dari keberadaan media untuk menemukan data. Mesin pencari data atau yang sering dikenali oleh kita sebagai *search engine* banyak membantu pengguna untuk menemukan data dari seluruh belahan dunia. Google, Astalavista, dan Yahoo, adalah salah satu contohnya.

Apa yang mesin pencari lakukan. Pertama melakukan manajemen dalam menerima masukan data dengan kaidah tertentu. Kedua mengoptimalkan algoritma pencarian data. Umumnya jika data sudah ditata dengan rapi dalam media penyimpanan kita, maka semakin mudah kita memperoleh data tersebut.

Dalam kasus ini metode yang akan kita bahas adalah metode pencarian dengan pohon biner. Metode ini sangat terkenal di dunia struktur data. Kemudahan dan kecepatan ditawarkan oleh metode ini. Seperti yang telah disampaikan sebelumnya bahwa selain metode pencarian data, manajemen memasukkan atau mengeluarkan data juga diperlukan agar pencaria data semakin optimal.

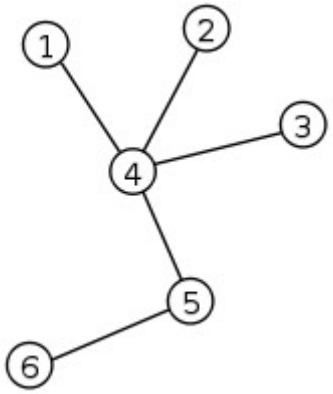
2. POHON[1]

2.1. Definisi Pohon

Pohon merupakan salah satu penerapan dari teori graf. Konsep pohon sangat penting posisinya baik dalam dunia informasi ataupun dalam bidang-bidang lainnya. Penerapan pohonpun banyak diterapkan pada kehidupan kita sehari-hari misalnya silsilah keluarga, struktur organisasi, bagan sistem gugur pada pertandingan, dan lain-lain.

Pohon merupakan graf yang memiliki sifat-sifat khusus, sehingga dapat memiliki definisi sebagai berikut:

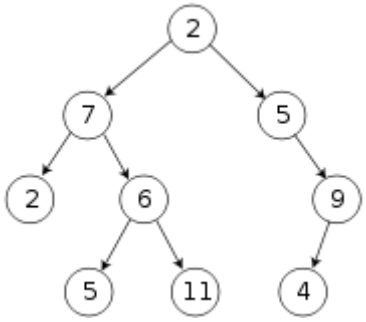
Pohon adalah graf tak berarah yang tidak mengandung sirkuit.



Gambar 1: Contoh pohon

2.2. Pohon Berakar dan Beberapa Terminologinya

Pohon dimana satu nodenya diperlakukan sebagai akar(*root*) disebut pohon berakar. Definisi Pohon jenis ini yang akan terus digunakan pada keseluruhan sisa pembahasan di makalah ini.



Gambar 2: terminologi pohon

Anak dan Orang Tua

Jika node a=2 merupakan akar dari suatu pohon, dan dihubungkan dengan sisi (a,b) dan (a, c), dimana b= 7 dan c= 5, maka b dan c adalah anak dari a sedangkan a adalah orang tua dari b dan c.

Lintasan

Lintasan antara simpul v1 dan simpul vk adalah runtutan simpul-simpul dari v1 hingga vk yang memiliki aturan sedemikian rupa sehingga vi adalah orang tua dari v(i+1). Contoh: lintasan dari simpul 2 ke simpul 11 adalah 2, 7, 6, 11.

Keturunan

Jika terdapat lintasan dari x ke y maka x disebut leluhur dari y dan y keturunan dari x

Saudara Kandung

Simpul yang memiliki orang tua yang sama disebut saudara kandung. Contoh : 5 dan 11.

Upapohon

jika simpul selain akar dijadikan akar baru, maka simpul-simpul keturunan hingga daun di bawahnya merupakan upapohon dari pohon.

Derajat

definisi derajat untuk pohon berakar dan pada graf berbeda. Pada graf derajat menunjukkan banyaknya sisi yang bersisian dengan simpul yang ingin ditunjukkan derajatnya, sedangkan pada pohon berakar derajat menunjukkan banyak anak pada suatu simpul.

Daun

Simpul yang berderajat nol atau dalam arti lain tidak mempunyai anak disebut daun. Dalam gambar 2, 5,11,4 merupakan daun dari pohon tersebut.

Simpul Dalam

Simpul yang bukan daun dan bukan akar disebut simpul dalam. Pada gambar 2 ditunjukkan dengan simpul 7,6,5, dan 9.

Aras(Level) atau Tingkat

Dengan mengacu kepada akar, simpul selain akar nilai arasnya adalah panjang lintasan dari Akar menuju simpul tersebut. Ada beberapa referensi mengenai aras akar. Satu diantaranya Akar memiliki aras=0. Ada juga yang menganggap aras akar =1.

Tinggi atau Kedalaman

aras maksimum dari suatu pohon disebut tinggi. Pada gambar 2 pohon tersebut memiliki tinggi 3.

2.3. Pohon Biner beserta variasinya

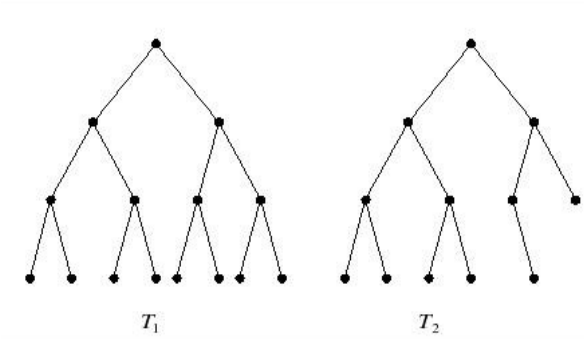
Pohon n-ary memiliki makna bahwa setiap simpul akan memiliki anak maksimal n. Pohon Biner adalah pohon n-ary dengan nilai n=2, yang berarti paling banyak pohon biner memiliki dua buah anak.

Di buku-buku referensi[1][3] anak dari pohon sering

disebut sebagai anak kanan dan anak kiri. Addanya perbedaan anak/upapohon menyebabkan maka pohon biner termasuk ke dalam pohon terurut.

Pohon yang simpulnya semua berada di kiri mulai dari akar disebut pohon condong kiri, sebaliknya jika berada di kanan disebut pohon condong kanan.

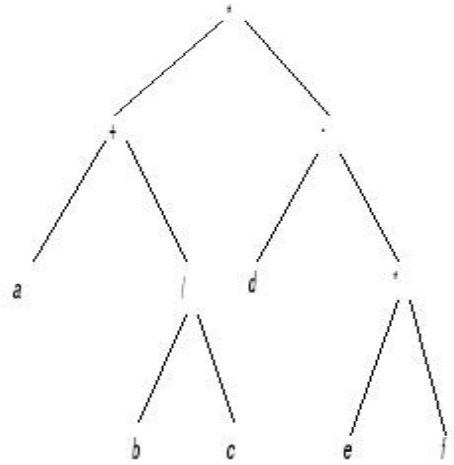
Terkadang dibutuhkan pohon yang seimbanng tinggi setiap upapohonnya. Pohon yang maksimal memiliki beda tinggi = 1 adalah pohon biner seimbang.



Gambar 3: Pohon seimbang

2.4. Penelusuran Pohon Biner

Ada tiga macam skema penelusuran pohon biner. Misalkan T adalah pohon biner, R adalah akarnya, T1 adalah upapohon yang ada di kiri, dan T2 adalah upapohon yang kanan.



Gambar 4: Penelusuran pohon biner melalui 3 cara

1. Pre Order
   a) kunjungi R
   b) telusuri T1 secara pre order
   c) telusuri T2 secara pre order

2. In Order
   a) telusuri T1 secara in order
   b) kunjungi R
   c) telusuri T2 secara in order

3. Post Order
   a) telusuri T1 secara post order
   b) telusuri T2 secara post order
   c) kunjungi R

Agar lebih mudah memahaminya coba lihat pohon pada gambar 4 di atas.

Preorder : * + a / b c – d * e f (prefix)
In order : a + b / c * d – e * f (infix)
Post order : a b c / + d e f * - * (post fix)

2.4. Pohon Biner Terurut(Binary Search Tree/BST)[3]

Pohon biner berurut adalah suatu metode yang sering digunakan dalam dunia komputasi struktur data yang memiliki ketentuan sebagai berikut.

- Setiap Simpul pohon biner terurut memiliki nilai yang berbeda.
- Baik upapohon(anak) kiri ataupun kanan dari simpul tersebut haruslah merupakan pohon biner terurut juga.
- Nilai dari anak kiri dari sebuah simpul harus lebih kecil dari simpul tersebut.
- Nilai dari anak kanan sebuah simpul harus lebih besar dari simpul tersebut.

Proses yang umumnya diperlukan pada pohon biner terurut ini adalah Insersi atau memasukkan data, pencarian sebuah data, penghapusan data, dan penelusuran data.

Pohon biner terurut merupakan salah satu penerapan pohon yang sangat penting dalam ilmu komputasi.

3. OPTIMALISASI ALGORITMA PENCARIAN DATA[2]

Optimalisasi algoritma pencarian data yang akan dibahas menggunakan metode Pohon Biner terurut(*Binary Search Tree(BST)*).

Dalam manajemen masukan dan keluaran data tidak bisa sembarangan dan harus mengikuti kaidah pohon biner. Selain menggunakan Pohon Biner terurut algoritma juga akan dibuat secara rekursif.

Data yang akan kita oleh cukuplah sederhana yaitu data-data bertipe integer. Untuk pengembangan lebih lanjut info dari integer tersebut bisa iubah menjadi sebuah tipe bentukan atau tipe-tipe lain yang lebih kompleks.

3.1. Tipe data pohon dan beberapa fungsi

```
{deklarasi tipe }
type simpul :
<info : integer, kiri : address, kanan:
address>
```

```
type  Pbiner : address
type elmSimpul : < info: integer, next :
address>
type daftarSimpul : address

function kosongkah(P: Pbiner) → boolean
procedure BuatPohon(Akar: integer, Kiri:
Pbiner, Kanan: Pbiner, output P:Pbiner)
```

Tipe Pbiner merupakan tipe yang berisi address yang akan menunjuk pada alamat akar dari pohon biner. Sedangkan tipe daftarSimpul merupakan sebuah list linear tempat menampung hasil penelusuran simpul-simpul pada pohon biner baik secara *post order, in order,* ataupun *pre order.* Kedua tipe tersebut memakai representasi fisik dengan pointer.

Fungsi Kosongkah(P) akan mengembalikan nilai 1 jika ternyata benar bahwa P adalah Pohon kosong atau Nil. Sedangkan prosedur BuatPohon akan mengalokasikan alamat di memori untuk info simpul, alamat ke anak kiri dan anak kanan.

3.2. Manajemen memasukkan dan membuang informasi data.

Karena kita menggunakan struktur data pohon biner pada algoritma mesin pencari, maka dalam memasukkan data dan membuang data harus sesuai kaidah konsep penyimpanan data pada phon biner berurut. Konsepnya adalah sebagai berikut.

Setiap simpul baik akar, daun, maupun simpul dalam merupakan sebuah address yang didalamnya terdapat info dari simpul tersebut, alamat yang mengacu pada anak kiri, dan alamat yang mengacu pada anak kanan.

Aturan masuknya sebuah data seperti ini.
Info simpul di upapohon sebelah kiri(anak kiri) harus lebih kecil dari info di simpul tersebut. Sebaliknya upapohon di sebelah kanan(anak kanan) harus lebih besar nilai infonya daripada info di simpul tersebut.

```
Procedure insersiXBST(input x: integer,
input/output P: Pbiner)
-----------------------------------------
{menambahkan sebuah simpul x ke pohon biner
pencarian P
Basis: Pohon Kosong
Rekurens: Jika pohon tidak kosong masukkan
simpul baru ke dalam anak kiri jika x lebih
kecil dari info simpul tersebut, atau ke
dalam anak kanan jika x lebih besar dari info
simpul tersebut. Jika sama x tidak dimasukkan
ke dalam pohon}
Algoritma
if (kosongkah(P)) then {sebagai basis}
       BuatPohon(x,Nil,Nil, P)
else {rekurens}
       if (x = Akar (P)) then
              output(“x sudah pernah
terisi”)
       else
              depend on x
                x< Akar(P) :
insersiXBST(x,Kiri(P))
                x> Akar(P) :
insersiXBST(x,Kanan(P)
```

Prosedur selanjutanya adalah prosedur untuk membuang simpul yang memiliki nilai x.

```
Procedure hapusXBST(input x: integer,
input/output P: Pbiner)
---------------------------------------
{membuang sebuah simpul yang berinfo x dari
pohon biner berurut P.}
Kamus Lokal
     q : address
     procedure hapusSimpul(input/output P:
Pbiner)
     procedure dealokasi(P: Pbiner)

Algoritma
depend on x
  x< Akar(P) : hapusXBST(x,Kiri(P))
  x> Akar(P) : hapusXBST(x,Kanan(P)
  x = Akar(P) :
        q ← P
        if Kanan(q) = Nil then p ← Kiri(q)
        else if Kiri (q) = Nil then p ←
Kanan(q)
              else
                  hapusSimpul(Kiri(q))
                  dealokasi(q)
```

Dalam struktur data memang selalu harus ada yang dikorbankan. Jika ingin keluaran mudah dicari(dalam kasus ini keluarannya adalah hasil pencarian pada sub bab berikutnya), pasti manajemen untuk memasukkan datanya sangat rumit. Sebaliknya jika manajemen dilakukan secara biasa, tentunya hasil pencarian tidak akan lebih mudah dibandingkan dengan cara pohon biner terurut.

Metode yang kita pilih sebenarnya tergantung kebutuhan kita. Semua ini pilihan pemrogram, apakah ingin rumit di awal namun pencarian data lebih mudah, atau memasukkan data semaunya, tapi akan kerepotan saat pencarian data.

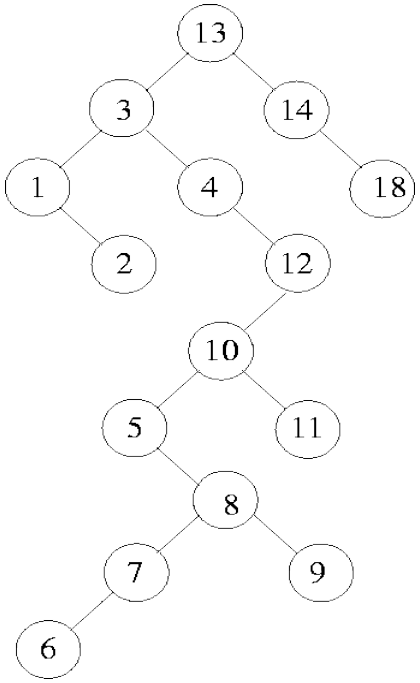
3.3. Pencarian Data dengan Pohon Biner Terurut

```
function cariXBST(P:Pbiner, x:integer →
booolean
{mengembalikan nilai true jika x ditemukan
pada pohon biner, dan mengambalikan nilai
false jika x tidak ditemukan.
Basis 1: x ditemukan
Basis 2: x tidak ditemukan
rekurens: pencarian secara rekursif pada
upapohon kiri ataupun kanan sekaligus}

Kamus Lokal
     Temu : boolean
```

```
Algoritma
if (Akar(P) = x) then
  Temu = true
else
  if (Akar(P) = Nil) then
     Temu = false
  else
     if (Akar(P) > x) then
        Temu = cariXBST(Kiri(P),x)
     else {Akar(P) < x}
        Temu = cariXBST(Kanan(P),x)

→ Temu
```

3.4. Optimalisasi lebih lanjut

Untuk kasus terbaik, x mungkin akan ditemukan di awal-awal pencarian atau bahkan pada akar pohon biner terurut tersebut. Namun ada kemungkinan data tidak ditemukan sehingga ia harus menelusuri pada lintasan terpanjang pada pohon biner tersebut.



Gambar 4: contoh sebuah pohon biner terurut yang tidak seimbang

Misalkan kita akan mencari nilai 13, dengan algoritma di atas. Tentu data akan masuk pada basis algoritma dan menghasilkan nilai true. Itu kasus terbaik.

Bagaimana jika dalam kasus terburuk Anda diminta untuk mencari nilai 6? Tentu anda akan melakukan rekursif hingga mesin sampai pada simpul tersebut.

Hal tersebut karena pada algoritma di atas hanya dirancang untuk membentuk pohon biner terurut yang sesuai kaidah tetapi bukan merupakan pohon seimbang. Walaupun ada kemungkinan pohon seimbang terbentuk dari algoritma tersebut.

Oleh karena itu, algoritma di atas masih dapat dioptimalkan lebih lanjut dengan merancang algoritma yang selain membentuk pohon biner terurut yang sesuai kaidah sekaligus terbentuk pohon seimbang.

Jika itu dipenuhi, maka setiap pencarian akan membutuhkan waktu pencarian yang relatif sama karena panjang lintasan memiliki tinggi yang relatif sama.

Namun, Hasil yang baik akan mengakibatkan dampak lain yang tidak bisa dihindarkan. Manajemen untuk memasukkan data dan membuang data akan lebih rumit dari pada algoritma membentuk pohon biner terurut biasa.

Adalah pilihan Anda dalam menentukan mana algoritma yang tepat untuk program yang dibuat. Segala sesuatu harus ditimbang dari segi dampak yang diakibatkan, apa yang dibutuhkan oleh program Anda, maupun apa yang perlu diutamakan.

4. MEMPERKIRAKAN KOMPLEKSITAS ALGORITMA PENCARIAN DATA[1]

4.1. Kompleksitas Algoritma

Memori komputer kita menyediakan ruang untuk melakukan pekerjaan seperti program yang kita buat. Karena adanya keterbatasan memori komputer, peran para pemrogramlah untuk membuat algoritma yang dibuatnya menjadi mangkus. Algoritma yang mangkus akan menghemat kebutuhan pemakaian ruang dan waktu memori.

Kompleksitas waktu dapat diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n.

Kita mendefinisikan kompleksitas waktu berdasarkan berapa kali algoritma dijalankan. Karena kalau setiap baris algoritma kita perhitungkan kompleksitas waktunya, hal tersebut tidaklah akan berlaku sama pada komputer satu dengan yang lainnya. Hal tersebut yang menjadi dasar penggunaan nilai T(n) sebagai kompleksitas waktu. T(n) dibedakan atas kompleksitas waktu rata-rata, terbaik, dan terburuk.

Jika sebuah program memerlukan satu kali jalan tanpa perlu mengulanginya kita dapat menganggapnya T = 1. sebaliknya jika pada sebuah program menggunakan looping hingga n kali, kompleksitas waktu algoritma tersebut adalah T(n) = n.

Notasi O-Besar

T(n) = O(f(n)) dibaca dengan T(n) berorde paling besar f(n) jika terdapat konstanta C dan No sedemikian sehingga T(n) <= C(f(n)). Dengan kata lain T(n) tidak akan berorde lebih besar dari f(n).

Sehingga untuk menunjukkan bahwa T(n) = O(f(n)) kita hanya perlu menemukan pasangan C dan No sedemikian sehingga T(n) <= C(f(n)).

4.2. Kompleksitas Algoritma Pencarian Data dengan Pohon biner Terurut

Dari algoritma pencarian data dengan pohon biner terurut kita dapat mencari berapa orde dari algoritma tersebut. Nilai n disini menandakan panjang lintasan yang ditempuh.

Pada kasus terbaik nilai T(n) dari algoritma tersebut adalah 1, dengan c=1 dan No =1.

Dengan demikian pada kasus terbaik algoritma ini memiliki O(n) = 1

Sedangkan pada kasus T(n) rata-rata(saat pohon biner terurut mendekati bentuk pohon biner seimbang) sebagai berikut.

Misalkan ada n buah simpul dan dibagi merata ke upapohon kiri dan kanan di setiap simpulnya.
Tinggi dari pohon ini adalah 2 pangkat n/2. Nilai tersebut setara dengan log(n)
T(n) untuk n>1 adalah $1+T(2^{n/2})$
T(n) <= $2^2\log(n+1)$
Nilai C = 2 dan No = 1
Dengan demikian pada kasus dengan T(n) rata-rata algoritma ini memiliki O(n) = O(log n).

Pada kasus terburuk, pohon biner terurut membentuk list linear, sehingga panjang lintasan pencarian paling buruk sama dengan tinggi.
T(n) = n, dengan c =1 dan No = 0 , maka ditemukan f(n) = n

Dengan demikian pada kasus terburuk algoritma ini memiliki O(n) = O(n).

5. KESIMPULAN

Pohon Biner sangat penting manfaatnya di segala bidang kehidupan terutama di dalam dunia komputer dan informasi. Sistem pencarian data menggunakan metode ini lebih baik dibandingkan dengan metode pencarian secara brute force ataupun secara traversal mengecek satu-persatu data karena dalam dunia informasi pengguna sangat menghargai waktu.

Optimalisasi akan lebih baik jika pembentukan pohon biner terurut juga mengusahakan terbentuknya pohon biner yang seimbang sehingga kasus terburuk denga kompleksitas O(n) tidak akan terjadi.

DAFTAR REFERENSI

[1]Munir, Rinaldi, Herianto, Diktat Kuliah IF 2151, Matematika Diskrit, Edisi Keempat, Program Studi Teknik Informatika, STEI, ITB, 2004

[2]Liem, Inggriani, Diktat Struktur Data Edisi 2008, Progran Studi Teknik Informatika, ITB, 2008.

[3] Wikipedia, Wikipedia – Free Encyclopedia, www.wikipedia.com, 2006. Tanggal akses : 4 Januari 2009, pukul 9.00 WIB.