Penerapan Algoritma BFS dan DFS pada *Directory Traversal* Sistem File Komputer

Adityo Jiwandono [1)]
[1)] Program Studi Teknik Informatika ITB, Bandung, email: jiwandono@students.itb.ac.id

Abstrak *– Makalah ini membahas penerapan algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS) pada directory traversal dalam berbagai struktur direktori sistem file seperti FAT32 dan NTFS. Directory traversal sangat sering dilakukan pada sistem file komputer untuk mengetahui isi dari sebuah direktori. Struktur direktori sistem file FAT32 dan NTFS merupakan sebuah pohon, oleh karena itu teori BFS dan DFS dapat diterapkan pada persoalan ini.*

Kata Kunci*: BFS, DFS, duplikasi file, sistem file, struktur direkori,* directory traversal.

1. PENDAHULUAN

Dalam penggunaan komputer, *directory traversal* adalah hal yang sangat penting dan sangat sering dilakukan untuk mengetahui isi dari sebuah direktori. Penggunaan *directory traversal* yang lebih intensif terjadi saat komputer melakukan pencarian file, misalnya pencarian file yang terduplikasi.

Pada saat menyimpan data dalam sebuah file seringkali pengguna komputer, baik secara sengaja maupun tidak sengaja, melakukan penyimpanan pada lebih dari satu tempat sehingga terjadilah duplikasi file. Jika memang hal ini diharapkan, tentu tidak akan menjadi masalah. Namun jika ternyata duplikasi file ini terjadi lebih dari satu kali dan file yang terduplikasi berjumlah besar, tentu penyimpanan file menjadi tidak efektif dan tidak efisien.

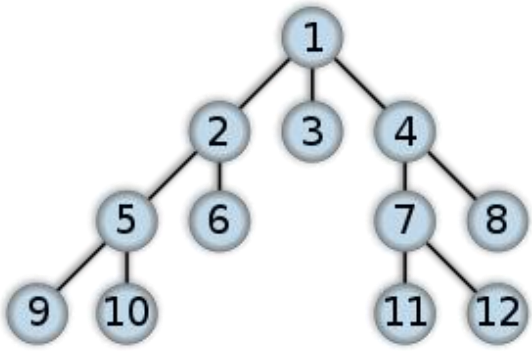
Struktur penyimpanan file pada sistem file yang menjadi kasus dalam makalah ini adalah berupa hirarki yang berbentuk pohon direktori. Struktur direktori sistem file akan dibahas pada bab selanjutnya.

Algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS) merupakan algoritma untuk melakukan kunjungan pada simpul-simpul graf dengan cara yang sistematis. Karena pohon juga merupakan sebuah graf, algoritma BFS dan DFS dapat diterapkan pada persoalan ini.

2. ALGORITMA BFS DAN DFS

2.1. Breadth-First Search

Dalam teori graf, BFS adalah algoritma pencarian graf yang mulai mencari dari simpul akar dan kemudian mencari ke semua simpul yang bertetangga. Untuk setiap simpul-simpul tetangga yang telah dikunjungi, simpul-simpul lain yang belum dikunjungi yang bertetangga dengan simpul tersebut akan ditelusuri. Begitu seterusnya sampai pencarian berakhir. Contoh kasus persoalan graf yang diselesaikan menggunakan algoritma BFS adalah pada gambar 1.



Gambar 1: Contoh urutan kunjungan algoritma BFS

Dari sudut pandang algoritma ini, semua simpul tetangga atau simpul anak yang didapat dengan mengembangkan sebuah simpul akan dimasukkan ke dalam FIFO *queue*. Pada implementasinya, informasi simpul-simpul akan disimpan pada sebuah kontainer yang berupa *queue* atau *linked list*.

Secara umum, algoritma BFS adalah sebagai berikut:

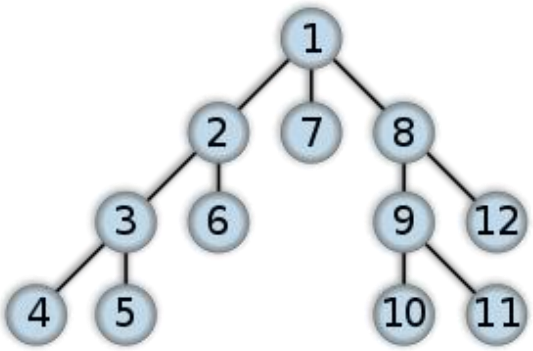
1. Masukkan simpul akar ke dalam antrian.
2. Ambil sebuah simpul pada antrian kemudian diperiksa.
    a. Jika elemen yang dicari ditemukan pada simpul ini, hentikan pencarian.
    b. Jika tidak, masukkan simpul-simpul anak simpul ini ke dalam antrian.
3. Jika antrian kosong, semua simpul graf telah diperiksa dan elemen tidak ditemukan. Hentikan pencarian.
4. Ulangi mulai langkah 2.

2.2. Depth-First Search

DFS adalah algoritma untuk melakukan traversal atau pencarian pada sebuah graf atau pohon. Dimulai dari simpul akar, pencarian akan dilakukan dan mengeksplorasi sejauh mungkin pada tiap cabang sebelum akhirnya melakukan *backtracking*.

DFS bekerja dengan mengembangkan simpul anak pertama pada pohon kemudian mencari lagi lebih

dalam pada anak simpul tersebut sampai simpul yang dicari ditemukan atau sampai tidak ada lagi simpul anak yang bisa dikunjungi. Setelah itu, *backtrack* dilakukan, kembali ke simpul terakhir yang masih memiliki simpul anak yang belum dikunjungi, dan seterusnya. Pada implementasi non-rekursif, simpul-simpul yang baru dikembangkan dimasukkan pada sebuah *stack* untuk eksplorasinya.



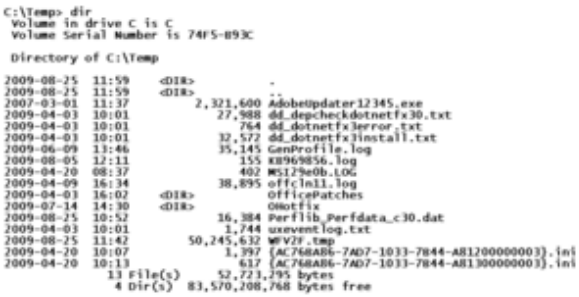
Gambar 2: Contoh urutan kunjungan algoritma DFS

Secara umum, algoritma DFS adalah sebagai berikut:

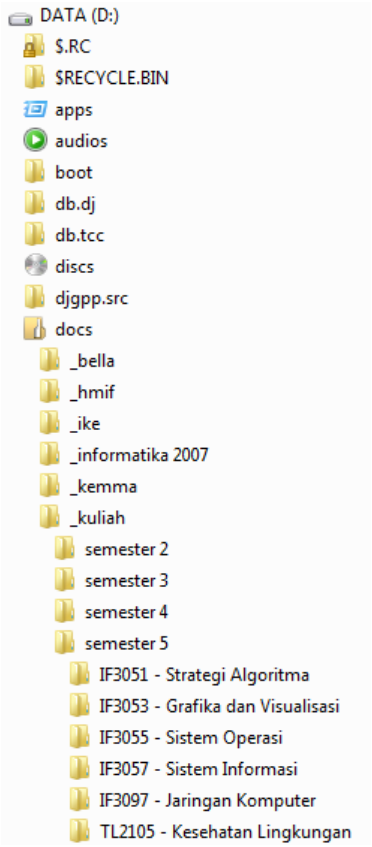
1. Masukkan simpul akar ke dalam *stack*.
2. Ambil sebuah simpul pada antrian kemudian diperiksa.
    a. Jika elemen yang dicari ditemukan pada simpul ini, hentikan pencarian.
    b. Jika tidak, masukkan simpul-simpul anak simpul ini ke dalam antrian.
3. Jika antrian kosong, semua simpul graf telah diperiksa dan elemen tidak ditemukan. Hentikan pencarian.
4. Ulangi mulai langkah 2.

3. STRUKTUR DIREKTORI

Istilah direktori dalam komputer merujuk pada sebuah wadah maya dalam sebuah sistem file digital. Kumpulan file dalam sistem file dapat disimpan dalam sebuah direktori dengan tujuan mempermudah pengelompokan.



Gambar 3: Contoh tampilan file-file dalam sebuah direktori



Gambar 4: Contoh tampilan direktori pada Explorer

Pada sistem operasi Windows, direktori akar adalah "namadrive:\", contohnya adalah "C:\", "D:\", dan sebagainya. Separator direktori yang digunakan adalah "\", namun sistem operasi juga mengenali "/" sebagai separator. Gambar 3 dan 4 adalah contoh-contoh penggunaan direktori dalam sistem operasi Windows.

4. ALGORITMA DIRECTORY TRAVERSAL

Struktur direktori sistem file dapat dimodelkan sebagai sebuah pohon sehingga algoritma BFS dan DFS dapat diterapkan untuk menyelesaikan permasalahan *directory traversal*. Tinjau struktur direktori berikut (indentasi menunjukkan level direktori):

```
G:\
  A
    file.txt
  B
    BA
      musik.mp3
    film.mpg
  C
    data1.dat
    data2.dat
  D
    DA
      DAA
        musik.mp3
        testing.tes
    makalah.doc
  kuliah.txt
  nilai.txt
```

Struktur direktori tersebut akan dapat ditelusuri menggunakan BFS maupun DFS. Berikut adalah algoritma BFS yang disesuaikan dengan persoalan ini:

1. Masukkan direktori akar ke dalam antrian.
2. Ambil sebuah direktori atau file pada antrian kemudian diperiksa.
    a. Jika elemen yang dicari ditemukan pada direktori atau file ini, hentikan pencarian.
    b. Jika tidak, masukkan isi direktori yang sekarang diproses ke dalam antrian.
3. Jika antrian kosong, semua direktori atau file telah diperiksa dan file tidak ditemukan. Hentikan pencarian.
4. Ulangi mulai langkah 2.

Dengan algoritma tersebut, urutan kunjungan direktori dan file adalah sebagai berikut:

```
G:\
A
B
C
D
kuliah.txt
nilai.txt
file.txt
BA
film.mpg
data1.dat
data2.dat
DA
makalah.doc
musik.mp3
DAA
musik.mp3
testing.tes
```

Berikut adalah algoritma BFS yang disesuaikan dengan persoalan ini:

1. Masukkan direktori akar ke dalam *stack*.
2. Ambil sebuah direktori atau file pada *stack* kemudian diperiksa.
    a. Jika elemen yang dicari ditemukan pada direktori atau file ini, hentikan pencarian.
    b. Jika tidak, masukkan isi direktori yang sekarang diproses ke dalam *stack*.
3. Jika *stack* kosong, semua direktori atau file telah diperiksa dan file tidak ditemukan. Hentikan pencarian.
4. Ulangi mulai langkah 2.

Dengan algoritma tersebut, urutan kunjungan direktori dan file adalah sebagai berikut:

```
G:\
A
file.txt
B
BA
musik.mp3
film.mpg
C
data1.dat
data2.dat
D
DA
DAA
musik.mp3
testing.tes
makalah.doc
kuliah.txt
nilai.txt
```

Berikut adalah implementasi algoritma pencarian dalam bahasa pemrograman PHP:

```
function getdirBFS($outerDir){
  $dirs = array_diff(scandir($outerDir),
Array(".", ".."));
  $dir_array = Array();
  $dir_array2 = Array();
  $dir_array = array_merge($dirs,
$dir_array);
  foreach($dirs as $d) {
    if(is_dir($outerDir."/".$d)) {
      $dir_array2 = array_merge($dir_array2,
getdirBFS($outerDir."\\".$d));
    }
  }

  $dir_array = array_merge($dir_array,
$dir_array2);
  return $dir_array;
}

function getdirDFS($outerDir){
  $dirs = array_diff(scandir($outerDir),
Array(".", ".."));
  $dir_array = Array();
  foreach($dirs as $d) {
    $dir_array = array_merge($dir_array,
(array) ($outerDir."\\".$d));
    if(is_dir($outerDir."/".$d)) {
      $dir_array = array_merge($dir_array,
getdirDFS($outerDir."\\".$d));
    }
  }
  return $dir_array;
}

print_r(getdirBFS("G:"));
print_r(getdirDFS("G:"));
```

Dalam contoh program ini, daftar file hasil traversal yang diperoleh disimpan di sebuah *array*. Pada fungsi getdirBFS, *array* diperlakukan sebagai *stack* sedangkan pada fungsi getdirDFS, *array* diperlakukan sebagai *queue*.

Array hasil fungsi getdirBFS dan getdirDFS dapat digunakan untuk melakukan pencarian. Tergantung pada implementasi sistem file, kecepatan pencarian akan berbeda antara pencarian menggunakan

algoritma BFS dan algoritma DFS.

Berikut adalah contoh keluaran untuk fungsi getdirDFS.

```
Array
(
    [0] => G:\A
    [1] => G:\A\file.txt
    [2] => G:\B
    [3] => G:\B\BA
    [4] => G:\B\BA\musik.mp3
    [5] => G:\B\film.mpg
    [6] => G:\C
    [7] => G:\C\data1.dat
    [8] => G:\C\data2.dat
    [9] => G:\D
    [10] => G:\D\DA
    [11] => G:\D\DA\DAA
    [12] => G:\D\DA\musik.mp3
    [13] => G:\D\DA\testing.tes
    [14] => G:\D\makalah.doc
    [15] => G:\kuliah.txt
    [16] => G:\nilai.txt
)
```

5. KESIMPULAN

Directory traversal dapat dilakukan secara BFS maupun DFS. Untuk mementukan algoritma apa yang optimal, perlu dilakukan pengujian lebih lanjut karena kinerja *directory traversal* sangat bergantung pada implementasi sistem file yang bersangkutan.

DAFTAR REFERENSI

[1] Directory Structure, http://averstak.tripod.com/fatdox/dir.htm. Diakses pada 2 Januari 2010 15:30.

[2] Wikipedia, Breadth-First Search , http://en.wikipedia.org/wiki/Breadth-first_search. Diakses pada 3 Januari 2010 11:00.

[3] Wikipedia, Depth-First Search, http://en.wikipedia.org/wiki/Depth-first_search. Diakses pada 3 Januari 2010 11:05.

[4] Wikipedia, Directory Structure, http://en.wikipedia.org/wiki/Directory_structure. Diakses pada 3 Januari 2010 11:10.

[5] Wikipedia, Hierarchical Tree Structure, http://en.wikipedia.org/wiki/Hierarchical_tree_structure. Diakses pada 3 Januari 2010 11:15.