Penggunaan BFS dan DFS untuk Pixel Traversing

Fadhil Muhtadin – 13510070
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13510070@std.stei.itb.ac.id

***Abstrak*— Algoritma merupakan konsep sentral dari ilmu computer science. Kualitas suatu program ditentukan kualitas desain algoritma yang dipakainya. Sampai saat ini sudah banyak jenis algoritma yang telah dikembangkan. Dalam makalah ini akan dibahas salah satu implementasi dari algoritma Depth-First Search dan Breadth-First Search dalam image processing. Permasalahan khusus yang dibahas menyangkut pixel traversing untuk sebuah gambar.**

***Kata Kunci*— Breadth-First Search, Depth-First Search, image, pixel**

I. PENDAHULUAN

Dalam dunia computer science, konsep algoritma tidak bisa dipisahkan dari pekerjaan kita sehari-hari, untuk itu konsep desain algoritma yang benar dan baik menjadi sangat penting. Algoritma apapun itu yang kita buat haruslah benar, dengan kata lain *correctness* menjadi suatu keharusan, mengutip dari David Clark yang sekarang dipakai sebagai filosofi IETF (Internet Engineering Task Force) : "We reject kings, presidents and voting. We believe in rough consensus and running code".

Selain benar, sebuah algoritma juga perlu didesain agar memiliki kualitas yang baik. Parameter kualitas ini yang paling utama adalah efisiensi waktu, yang mengindikasikan seberapa cepat algoritma kita berjalan dan efisiensi ruang, yang mengindikasikan berapa banyak memori yang kita butuhkan untuk menjalankannya. Kualitas lainnya termasuk simplisitas dan generalitas.

Seiring perkembangan teknologi konputer, semakin banyak peluang untuk mengembangkan algoritma baru atau menggunakan algoritma yang sudah ada untuk field-field baru dalam dunia komputasi. Untuk itu dalam computer science, dibentuklah semacam taksonomi untuk pengelompokan strategi algoritma yang sudah ada, diantaranya yang populer kita mengenal strategi algoritma brute force, greedy, decrease and conquer, divide and conquer, dsb. Dua diantaranya yang menjadi perhatian pada makalah ini adalah Breadth-First Search (BFS) dan Depth-First Search (DFS).

Metode BFS dan DFS merupakan algoritma traversal pengembangan dari brute force secara exhaustive search. Bedanya dengan exhaustive search adalah DFS dan BFS melakukan traversal secara sistematis sehingga meminimalisasi jumlah kemungkinan solusi yang harus diiterasi. Biasanya kedua metode ini digunakan dalam persoalan graf.

Topik khusus yang akan dibahas untuk kedua algoritma ini menyangkut bidang pengolahan citra. Dalam pengolahan citra, yang menjadi input adalah suatu gambar, dapat berupa foto, video frame, ataupun yang lainnya dan kemudian diproses menghasilkan output yang dapat berupa gamber pula maupun kumpulan karakteristik dan parameter dari gambar tersebut. Aplikasi dari pengolahan citra ini cukup banyak, mulai dari pengeditan gambar, computer vision, sampai survey geologi.

Pengolahan citra dalam *computer vision* menyangkut beberapa tahapan proses, salah satunya yang sangat penting adalah proses pengenalan pola atau objek. Untuk dapat mengenali pola atau objek dalam suatu gambar, komputer harus memiliki suatu cara untuk menelusuri bagian-perbagian dari gambar tersebut dan mengenali warna dari bagian yang sedang ditelusurinya. Hal ini dapat didukung dari definisi struktur data sebuah gambar yang disimpan dalam komputer itu sendiri.

Karena komputer hanya dapat mengenali data yang diskrit (dalam biner 0 dan 1), maka sebuah gambar disimpan dalam komputer juga dalam bentuk yang diskrit. Sebuah gambar tersusun atas pixel, yakni elemen terkecil gambar yang dapat direpresentasikan dalam nilai digital. Dari sudut pandang manusia pixel juga dapat diartikan titik terkecil dari gambar. Pixel dapat dijadikan satuan ukuran gambar, sehingga sebuah gambar memiliki jumlah pixel yang berhingga.

Nilai digital dari pixel ini berupa digit biner yang merepresentasikan warna pixel tersebut. Jadi jumlah bit yang dikandung pixel membatasi jumlah warna yang dapat direpresentasikan. Ukuran yang digunakan untuk hal ini adalah bit per pixel (bpp). Satu bpp artinya tiap pixel hanya mengandung 1 bit. Untuk warna, maka

- 1 bpp, $2^1 = 2$ warna (monokrom)
- 2 bpp, $2^2 = 4$ warna
- 3 bpp, $2^3 = 8$ warna
- 8 bpp, $2^8 = 256$ warna
- 16 bpp, $2^{16} = 65{,}536$ warna ("Highcolor")
- 24 bpp, $2^{24} \approx 16.8$ juta warna ("Truecolor")

II. DASAR TEORI

A. *Depth-First Search*

Algoritma DFS memulai traversal suatu graf dari sembarang node dan menandainya sebagai "visited". Pada setiap iterasi, algoritma ini mengunjungi node tetangganya yang belum dikunjungi dan menandainya sebagai "visited" pula. Jika terdapat lebih dari 1 node tetangga yang memenuhi syarat, maka node mana duluan yang akan dikunjungi dapat

tergantung implementasi oleh desainer algoritma. Biasanya cara umum yang dipakai adalah mengunjungi node secara alfabetik. Hal ini diteruskan sampai ditemukan jalan buntu, yakni tidak ada lagi node tetangga yang memenuhi syarat. Maka penelusuran diteruskan dengan melakukan backtrack ke node sebelumnya dan mencoba mencari node tetangga yang belum dikunjungi dari situ. Algoritma akan selesai bila sudah ditemukan solusi atau tidak ada solusi yang ditemukan. Pada kasus kedua, penelusuran akan sampai ke node awal dengan jalan buntu, artinya tidak ada lagi node yang bisa ditelusuri dari node awal karena semua node sudah dikunjungi. Jika ternyata masih terdapat node yang belum dikunjungi, penelusuran harus dimulai lagi dari sembarang node yang belum dikunjungi tersebut. Hal ini dapat terjadi untuk kasus dimana terdapat 2 graf atau lebih yang tidak terkoneksi.

Penerapan DFS dapat menggunakan struktur data stack untuk mempermudahnya. Saat kita pertama mengunjungi sebuah node, kita push node tersebut ke dalam stack dan melakukan pop terhadapnya bila pada node tersebut ditemukan jalan buntu.

Berikut ini pseudocode algoritma DFS secara kasar :

```
ALGORITMA DFS(G)

Tandai tiap node pada graf sebagai “unvisited”
Count ← 0
For each node v in V do
 If v ditandai “unvisited” do
   Dfs(v)

Prosedur Dfs(v)

Count ← count + 1
Tandai v sebagai “visited”
For each node w in V adjacent to v do\
 If w bertanda “unvisited” do
   Dfs(w)
```

B. Depth-First Search

Pada prinsipnya algoritma BFS juga melakukan traversal dengan mengunjungi tiap node graf dan menandai semua node yang sudah dikunjungi sebagai “visited”. Bedanya dengan DFS, algoritma BFS melakukan traversal graf dengan mengunjungi semua node tetangga dari current node terlebih dahulu, baru mulai mengunjungi node pada level berikutnya, sampai ditemukan jalan buntu lalu dilakukan kembali untuk node lain yang belum dikunjungi (pada kasus 2 graf tidak terkoneksi).

Untuk implementasi BFS kita dapat menggunakan stuktur data queue (berbeda dari DFS yang menggunakan stack) untuk melakukan tracing operasi penelusurannya. Queue ini pertama-tama hanya diisi oleh node pertama graf yang dijadikan current node. Pada tiap iterasi, setiap node tetangga dari current node dicek apakah sudah dikunjungi atau belum, bila belum maka node tersebut ditandai sebagai “visited” dan dimasukkan ke dalam queue. Setelah semua node tetangga dicek, current node yang menempati posisi pertama queue dikeluarkan dari queue, node selanjutnya dalam queue akan menjadi current node yang baru dan proses iterasi diulang kembali sampai elemen queue habis. Karena queue bersifat FILO (First In Last Out), maka node yang masuk duluan ke dalam queue, yakni node yang lebih dekat ke current node (memiliki level lebih rendah) akan diproses terlebih dahulu, mengakibatkan proses penelusuran secara Breadth-First.

Berikut pseudocode algoritma BFS secara kasar :

```
ALGORITMA BFS(G)

Tandai tiap node pada graf sebagai “unvisited”
Count ← 0
For each node v in V do
 If v ditandai “unvisited” do
   Bfs(v)

Prosedur Bfs(v)

Count ← count + 1
Tandai v sebagai “visited”
Inisialisasi queue dengan v
While queue is not empty do
 for each node w in V adjacent to the front node do
   if w bertanda “unvisited”
     count ←count + 1
     tandai w sebagai “visited”
     masukkan w kedalam queue
 Hapus node pertama dalam queue
```

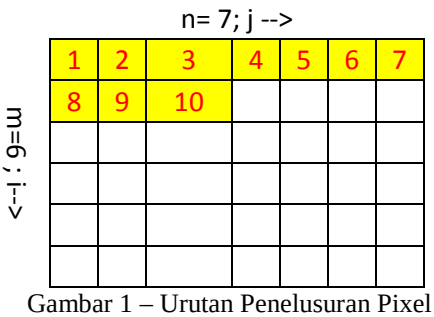
III. Implementasi Dalam Traversal Pixel

Seperti yang sudah dijelaskan sebelumnya, untuk melakukan pengenalan pola atau objek, komputer harus dapat menelusuri tiap pixel dalam gambar tersebut dan mendapatkan informasi mengenai warna yang dikandung dalam pixel tersebut. Informasi mengenai warna ini mudah diambil dari bit pixel tersebut. Yang akan dibahas disini adalah persoalan penelusuran pixel.

Sebuah gambar tersusun atas m x n pixel. Cara yang paling sederhana untuk menelusurinya adalah dengan cara brute force seperti dibawah ini

ALGORITMA BruteforcePixel()

```
i ← 0
while (i<m) do
   j ← 0
   while (j<n) do
      SomeProcedure(Pixel(i,j))
      j ← j +1
   i ← i + 1
```



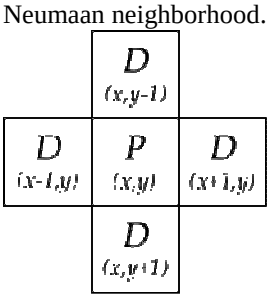
Gambar 1 – Urutan Penelusuran Pixel Secara Brute Force

Cara diatas akan menelusuri tiap pixel mulai dari pixel pada koordinat (0,0) sampai (m,n) secara berurutan. Namun bila yang kita kehendaki adalah pengenalan bentuk dalam gambar maka kita perlu cara penelusuran yang berbeda. Jika persoalan ini kita sederhanakan, yang harus kita lakukan sebenarnya adalah menelusuri pixel-pixel tetangga yang memiliki warna yang sama sampai suatu saat ditemukan pixel yang berbeda warna, pada saat itu kita telah menemukan batasan (border) dari bentuk tersebut. Hal ini dikarenakan pada dasarnya sebuah bentuk adalah sekumpulan pixel yang bertetangga dan memiliki warna yang sama.

Ada satu lagi konsep pixel yang perlu kita ketahui sebelum melanjutkan ke desain algoritma, yakni konsep ketetanggaan pixel. Pada dasarnya kita dapat menggunakan salah satu dari dua konsep ketetanggaan pixel yakni :

- 4-way connectedness

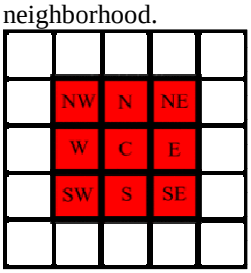
  Tetangga sebuah pixel adalah pixel yang berada di sebelah kanan, atas, kiri, bawah (atau timur, utara, barat, dan selatan). Biasa disebut juga dengan Von Neumaan neighborhood.



Gambar 2 – 4 Way Connectedness

- 8-way connectedness

  Tetangga sebuah pixel adalah pixel yang berada di sebelah kanan, kanan-atas, atas, kiri-atas, kiri, kiri-bawah, bawah, dan kanan-bawah (atau sesuai arah mata angin). Biasa disebut juga dengan Moore neighborhood.



Gambar 3 – 8 Way Connectedness

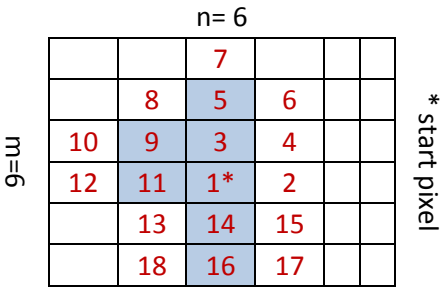
Pada desain algoritma kita, kita akan menggunakan 4-way connectedness.

A. Penggunaan DFS

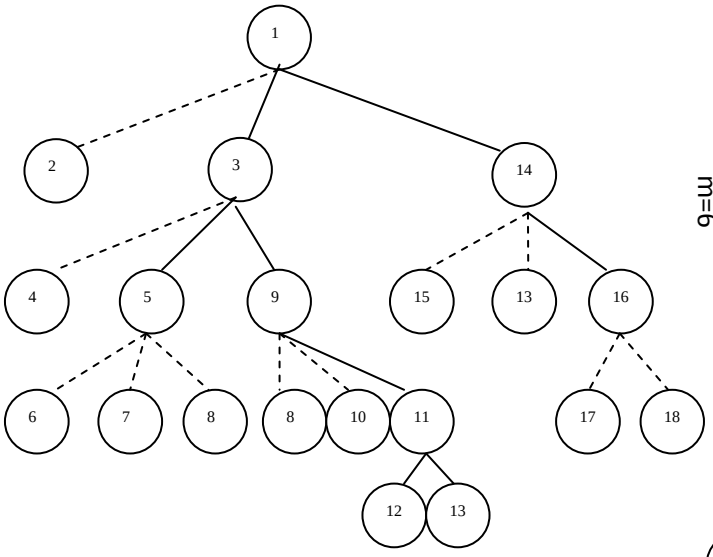
Sesuai scope permasalahan kita, kita dapat mendesain algoritma penelusuran pixel secara DFS seperti dibawah ini

```
ALGORITMA PixelDFS(i,j)
// i dan j adalah koordinat pixel pertama yang akan ditelusuri

Tandai semua pixel dengan “unvisited”
Current_color ← getcolor(getpixel(i,j))
Count ← 0
PixelDfs1(i,j)

Prosedur PixelDfs1(i,j)

Count ← count + 1
Tandai pixel(i,j) sebagai “visited”
If (getcolor(getpixel(i+1,j)) == current_color && pixel(i+1,j)
bertanda “unvisited”) then
   PixelDFS1(i+1,j)
Else if (getcolor(getpixel(i,j+1)) == current_color &&
pixel(i,j+1) bertanda “unvisited”) then
   PixelDFS1(i,j+1)
Else if (getcolor(getpixel(i-1,j)) == current_color && pixel(i-
1,j) bertanda “unvisited”) then
   PixelDFS1(i-1,j)
Else if (getcolor(getpixel(i,j-1)) == current_color &&
pixel(i,j-1) bertanda “unvisited”) then
   PixelDFS1(i,j-1)
```

Algoritma diatas akan mulai menelusuri dari suatu start pixel tertentu yang kita asumsikan berada dalam objek yang mau diamati. Tiap iterasi algoritma akan mulai mengecek dari arah timur dahulu dan menelusuri secara rekursif dengan DFS. Setelah timur, baru dicek utara secara DFS, Barat secara DFS, sampai selatan secara DFS. Berikut contoh iterasinya:



Gambar 4 – Urutan Penelusuran Pixel Secara DFS

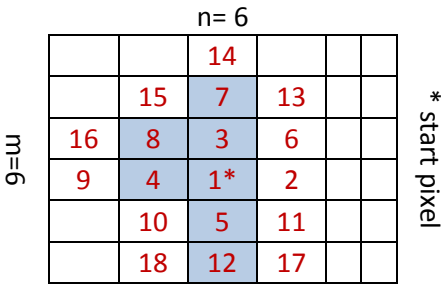


Gambar 5 – DFS Tree Hasil Penelusuran Pixel

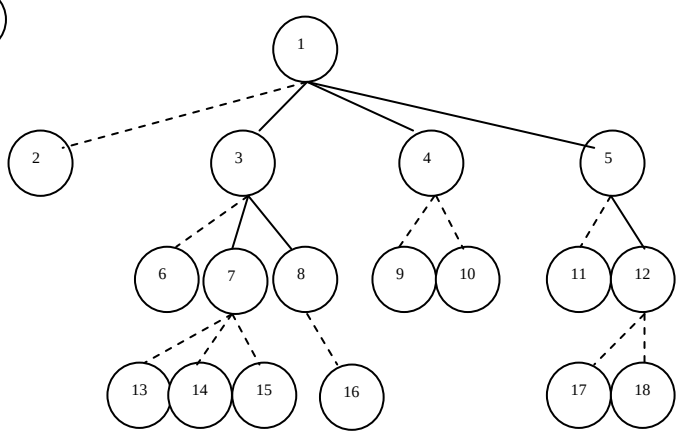
B. Penggunaan BFS

Sekarang, kita lihat bagaimana penggunaan BFS untuk persoalan yang sama

```
ALGORITMA PixelBFS(i,j)
// i dan j adalah koordinat pixel pertama yang akan ditelusuri

Tandai semua pixel dengan “unvisited”
Current_color ← getcolor(getpixel(i,j))
Count ← 0
PixelBfs1(i,j)

Prosedur PixelBfs1(i,j)

Count ← count + 1
Tandai pixel(i,j) sebagai “visited”
Inisialisasi queue dengan pixel(i,j)
While queue is not empty do {
If (getcolor(getpixel(i+1,j)) == current_color && pixel(i+1,j)
bertanda “unvisited”) then
   Masukkan pixel(i+1,j) kedalam queue
   Tandai pixel(i+1,j) sebagai “visited”
Else if (getcolor(getpixel(i,j+1)) == current_color &&
pixel(i,j+1) bertanda “unvisited”) then
   Masukkan pixel(i,j+1) kedalam queue
   Tandai pixel(i,j+1) sebagai “visited”
Else if (getcolor(getpixel(i-1,j)) == current_color && pixel(i-
1,j) bertanda “unvisited”) then
   Masukkan pixel(i-1,j) kedalam queue
   Tandai pixel(i-1,j) sebagai “visited”
Else if (getcolor(getpixel(i,j-1)) == current_color &&
pixel(i,j-1) bertanda “unvisited”) then
   Masukkan pixel(i,j-1) kedalam queue
   Tandai pixel(i,j-1) sebagai “visited”
Hapus elemen pertama queue
}
```



Gambar 6 – Urutan Penelusuran Pixel Secara BFS



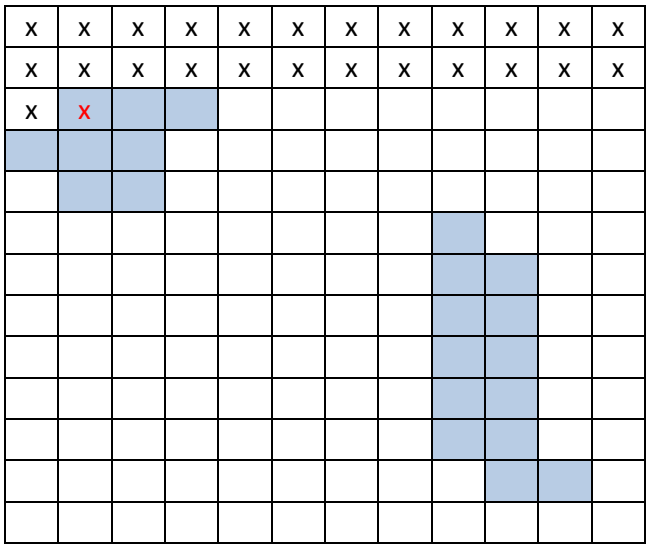
Gambar 7 – BFS Tree Hasil Penelusuran Pixel

C. Pengembangan algoritma

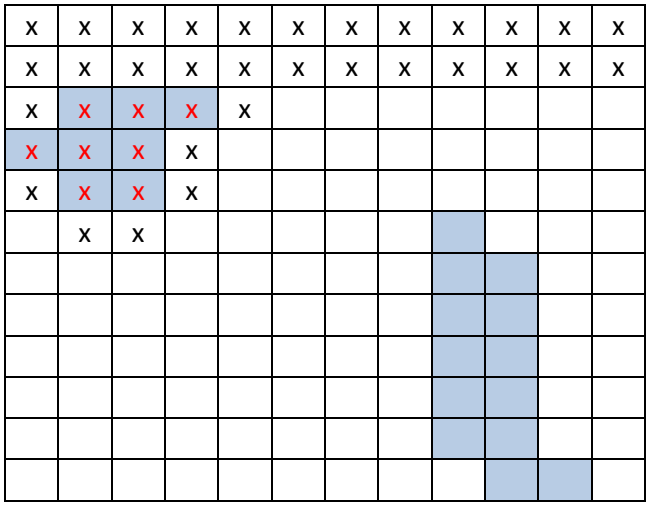
Kedua algoritma sebelumnya berhadapan dengan masalah penelusuran pixel tetangga berdasarkan warna yang sama jika telah diketahui pixel startnya. Untuk masalah yang lebih umum, diberikan sebuah gambar yang terdiri atas latar belakang dan lebih dari satu objek (bentuk) didalamnya. Kita perlu merekognisi semua objek yang berada dalam gambar tersebut. Tentu kita perlu mengiterasi seluruh pixel yang terdapat dalam gambar itu, tidak hanya yang terdapat dalam sebuah objek saja.

Maka dari itu kita dapat menggabungkan algoritma DFS atau BFS dengan traversal secara brute force. Pertama, brute force dilakukan dari pixel paling pojok sampai didapatkan pixel yang berbeda warna dari warna latar belakang. Berarti pixel tersebut termasuk didalam salah satu objek. Pixel ini kita jadikan start pixel. Baru dari start pixel tersebut dapat kita panggil prosedur traversal pixel menggunakan BFS maupun DFS. Setelah BFS atau DFS selesai, sekumpulan pixel yang telah kita tentukan berada dalam objek tersebut dapat kita simpan sebagai satu “shape”. Lalu traversal dilanjutkan secara brute force lagi untuk mencari objek selanjutnya. Satu hal penting yang perlu diperhatikan, pada kedua algoritma sebelumnya inisialisasi semua pixel sebagai “unvisited” dilakukan pada masing-masing algoritma BFS dan DFS, untuk hal ini inisialisasi perlu dilakukan pada saat memulai traversal brute force.

```
ALGORITMA TraversePixel()

Tandai semua pixel sebagai “unvisited”
i ← 0
while (i<m) do {
    j ← 0
    while (j<n) do {
      tandai pixel(i,j) sebagai “visited”
       if (getcolor(getpixel(i,j)) != BACK_COLOR &&
pixel(i,j) bertanda “unvisited”) then {
          InsertNewShape()
          PixelDFS(i,j) // Atau bisa juga pakai PixelBFS}
       j ← j +1
   i ← i + 1
    }
}
```



Gambar 8 – Pixel yang telah Ditelusuri Sampai Pixel Objek Pertama Ditemukan



Gambar 9 – Pixel yang telah Ditelusuri Setelah Objek Pertama Direkognisi

IV. KESIMPULAN

Untuk proses traversal pixel dalam sebuah gambar, kita telah mendesain 2 macam algoritma berdasarkan strategi Depth-First Search dan Breadth-First Search. Keduanya terbukti mampu mengenali kumpulan pixel bertetangga yang direkognisi kedalam sebuah bentuk objek berdasarkan warna yang sama.

Namun yang belum kita lakukan adalah komparasi antara kedua algoritma itu sendiri, manakah diantara keduanya yang lebih efisien secara waktu maupun ruang. Sebuah algoritma tidak selamanya lebih baik dari algoritma yang lain, itu semua tergantung parameter kualitas yang dipakai dan karakteristik spesifik dari kasus persoalan yang dihadapi.

Selain itu, algoritma yang kita pakai baru sebatas untuk kasus sederhana dimana untuk sebuah objek seluruh pixelnya memiliki warna yang persis sama. Bagaimana jika warnanya mirip tapi tidak persis sama? Misalkan objek buah yang memiliki gradasi warna dari merah tua ke merah muda, tentu kita perlu memodifikasi algoritma yang telah kita buat. Bisa saja kita berikan nilai batas toleransi kecocokan warna dalam sebuah objek, dengan asumsi warna latar belakang memang sangat kontras dengan warna pixel dalam objek.

Pada intinya, memang seiring berjalannya waktu dan berkembangnya pemikiran-pemikiran baru, algoritma yang lama bisa saja menjadi *obselete* atau perlu di-*refine* ulang. Dengan demikian algoritma yang ada saat ini dapat menjawab pertanyaan-pertanyaa baru yang mungkin muncul di masa depan.

DAFTAR REFERENSI

[1] Anany Levitin, *Introduction to the Design and Analysis of Algorithms*, 3rd ed., Pearson Education Inc, 2012.

[2] Rinaldi Munir, *Diktat Kuliah IF3051 Strategi Algoritma*, Bandung: Penerbit Informatika, 2009.

[3] T. Theoharis, G. Papaioannou, N. Platis, dan N.M. Patrikalakis, *Graphics and Visualitation Algorithms*, Taylor & Francis Group., 2008.

[4] “Depth First Search of a Graph”
URL : www.csc.lsu.edu/~kundu/dstr/2-**depthFirst**.pdf
diakses tanggal 20-12-2012 pukul 21.00

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, 21 Desember 2012

Fadhil Muhtadin - 13510070