Implementasi Algoritma DFS pada Pewarnaan Gambar Sederhana Menggunakan *Bucket tool*

Sharon Loh (13510086) [1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13510086@std.stei.itb.ac.id

***Abstraksi*—Flood fill merupakan sebuah algoritma pewarnaan yang terkenal dan banyak digunakan pada berbagai aplikasi dalam mengimplementasi pewarnaan gambar, misalnya pada *bucket tool*. Salah satu implementasi algoritma flood fill yang dapat dilakukan adalah dengan menerapkan algoritma DFS. Makalah ini akan membahas bagaimana algoritma DFS diterapkan pada pewarnaan gambar, khususnya dengan menggunakan *bucket tool* pada aplikasi Paint.**

***Kata kunci*—flood fill, DFS, *bucket tool*.**

I. PENDAHULUAN

Saat ini, *drawing application* banyak sekali digunakan oleh desainer maupun orang awam. Mulai dari *drawing tool* yang mempunyai fitur yang lengkap seperti Photoshop maupun *drawing tool* sederhana seperti Paint yang sudah tersedia pada sistem operasi Windows. Banyak orang yang membutuhkan aplikasi ini untuk mendesain sebuah gambar atau hanya sekedar untuk mengubah atau menambahkan sesuatu pada gambar yang telah ada. Salah satu fitur dasar yang paling berguna pada sebuah *drawing application* adalah *bucket tool*. *Bucket tool* merupakan sebuah *tool* yang terdapat pada hampir semua *drawing application* saat ini yang digunakan untuk melakukan pewarnaan pada suatu area dimana pewarrnaan hanya dilakukan pada area itu saja tanpa menyentuh area di luar batas area tersebut.

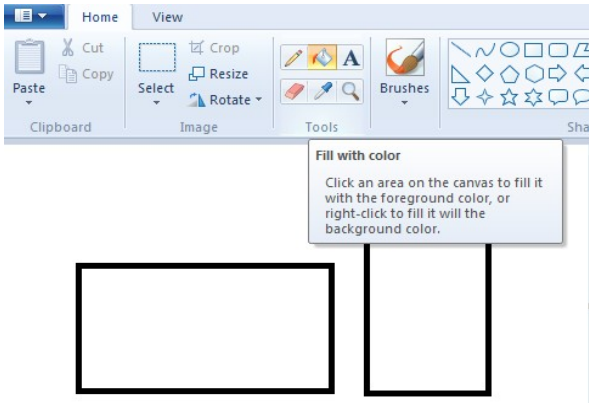

Figure 1- Bucket tool

Figure 1 merupakan contoh *bucket tool* yang ada pada aplikasi Paint. *Bucket tool* mengadopsi prinsip menuangkan cat, yaitu pewarnaan dilakukan pada area dimana pengguna mengklik kursornya dan menyebar ke area lain di sekitar area itu selama belum ada pembatas. Pada contoh *figure* 2, Jika pengguna menekan *bucket tool* pada area di dalam persegi panjang sebelah kiri, maka area pada persegi panjang tersebut akan diwarnai sesuai warna yang dipilih tanpa menyentuh garis hitam yang merupakan pembatas area persegi panjang dan area di luar persegi panjang.

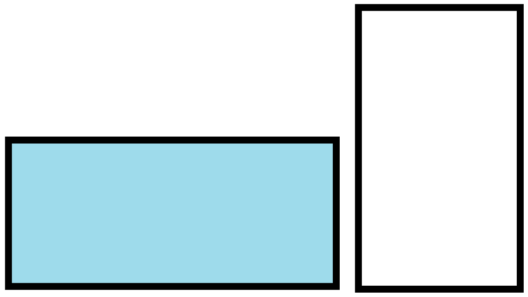

***Figure 2* - Contoh Pewarnaan Persegi**

Sedangkan jika pengguna mengklik garis hitam pada persegi pertama, maka garis hitam tersebut akan diwarnai dengan warna lain sesuai dengan warna yang dipilih oleh pengguna. Pewarnaan dilakukan pada area tersebut tanpa menyentuh area berwarna biru dan area berwarna putih.

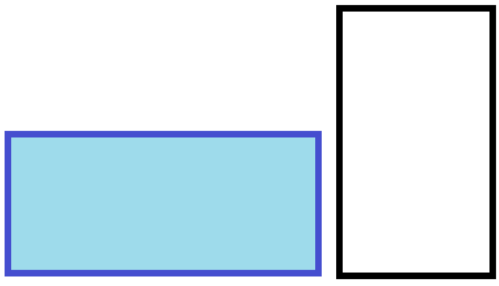

***Figure 3* - Contoh Pewarnaan Garis Hitam**

II. DASAR TEORI

Algoritma DFS (*Depth First Search*) merupakan suatu algoritma pencarian traversal pada sebuah graf atau pohon yang yang dilakukan dengan mengunjungi sebuah simpul awal dan kemudian melakukan eksplorasi dengan mengunjungi simpul-simpul tetangga dari simpul saat ini pada suatu graf sebelum akhirnya melakukan runut balik / *backtracking*. Algoritma ini disebut dengan algoritma pencarian mendalam (*depth first search*) karena pencarian terhadap suatu simpul dilakukan secara rekursif terhadap simpul-simpul tetanggannya.

Secara umum, cara kerja algoritma DFS dijelaskan pada Figure 4.

Misalkan terdapat sebuah graf dimana pencarian akan dilakukan.
Langkah – langkah yang dilakukan dalam melakukan pencarian dengan algoritma DFS adalah :

1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v
3. Ulangi DFS mulai dari simpul w
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi

***Figure 4* - Algoritma DFS**

Proses rekursif yang diterapkan pada algoritma DFS dilakukan pada langkah ke-3, yaitu dengan memanggil algoritma DFS itu sendiri terhadap simpul w yang dikunjungi. Pseudocode algoritma DFS digambarkan pada Figure 5.

```
procedure DFS (input v:integer)
Deklarasi
  w : integer
Algoritma:
  Proses(v)
  visited[v] ← true
  for w ← 1 to n do
   if (v dan w bertetangga)
      if (not visited[w]) then
         DFS(w)
      endif
   endif
  endfor
```

***Figure 5 - Pseudocode* Algoritma DFS**

Graf yang didefinisikan pada *pseudocode* pada *Figure* 5 merupakan graf dimana suatu simpul dapat berhubungan dengan n buah simpul lainnya. Karena itu, algoritma DFS akan mengunjungi n buah simpul tetangga tersebut. Contoh urutan pengecekan simpul graf yang dilakukan berdasarkan algoritma DFS dijelaskan pada *figure* 6.

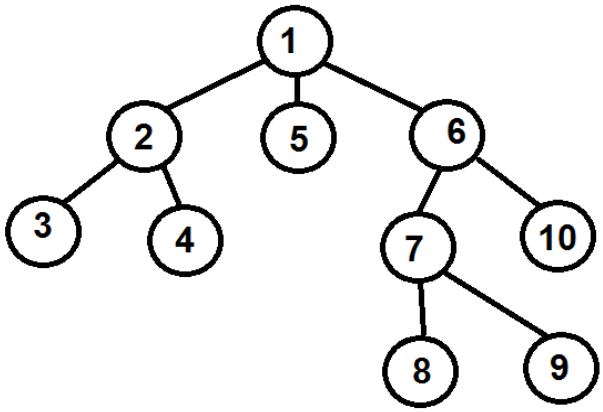

***Figure 6* - Pohon Pengecekan Status DFS**

Pada *figure* 6, dapat dilihat bahwa algoritma DFS melakukan pengecekan secara mendalam mulai dari simpul awal ke simpul tetangganya. Dan dari simpul tetangga tersebut, dilakukan pengecekan ke simpul tetangganya lagi dan begitu seterusnya. Setelah pengecekan dilakukan hingga tidak ada simpul tetangga yang dapat dikunjungi lagi, program akan melakukan *backtracking* atau runut balik untuk melakukan pengecekan terhadap simpul tetangga yang belum pernah dikunjungi.

Selain dengan menggunakan struktur data berupa graf atau pohon, algoritma DFS juga dapat dapat digunakan pada struktur data berupa matriks yang menggambarkan sebuah peta sebagai kumpulan dari *grid*. Pada struktur data ini, sebuah *grid* berhubungan dengan *grid* lainnya dalam 4 arah mata angin yaitu :

- Atas
- Kanan
- Bawah
- Kiri

	Grid Atas	
Grid Kiri	**Current Grid**	Grid Kanan
	Grid Bawah	

***Figure 7* – Gambaran Struktur Data Matriks**

Jika struktur data yang digunakan adalah struktur data matriks, maka pengecekan *grid* tetangga dilakukan terhadap 4 buah *grid* tetangga sesuai dengan arah mata angin seperti pada *figure* 7, yaitu *grid* atas, *grid* kanan, *grid* bawah, dan *grid* kiri. Urutan pengecekan terhadap masing-masing *grid* tergantung pada algoritma yang dirancang. Selain itu, diperlukan juga suatu variabel sementara yaitu `visited[][]` yang digunakan untuk menyimpan suatu nilai penanda (*true* / *false*) untuk mengecek apakah suatu *grid* atau simpul sudah pernah dikunjungi sebelumnya. Algoritma DFS hanya boleh mengunjungi simpul / *grid* yang belum pernah dikunjungi sebelumnya.

Algoritma DFS terkenal untuk memecahkan berbagai masalah graf, seperti penyelesaian permainan 8 puzzle, melakukan penjelajahan web pada search engine, pencarian rute untuk keluar dari suatu labirin, dll.

III. IMPLENTASI ALGORITMA DFS

Salah satu implementasi algoritma DFS yang dapat diterapkan pada aplikasi nyata yaitu impelmentasi DFS pada proses pewarnaan menggunakan *bucket tool*. Algoritma ini dikenal dengan istilah *flood fill*.

Algoritma *Flood Fill* merupakan sebuah algoritma pewarnaan yang dilakukan seperti menumpahkan seember cat pada daerah tertentu. Cat tersebut akan menyebar ke daerah sekitar tempat dimana cat pertama kali ditumpahkan. Algoritma *flood fill* menggunakan 3 parameter utama yaitu *node* awal, warna awal, dan warna pengganti. Warna awal dapat diperoleh dari *node* awal.

Pada kasus pewarnaan ini, pengguna menentukan titik awal dimana proses pewarnaan akan dimulai. Selain itu, pengguna juga menentukan warna yang akan digunakan dalam proses pewarnaan.

Struktur data yang digunakan dalam merepresentasikan sebuah gambar merupakan struktur data matriks dimana tiap *grid* dari matriks tersebut merepresentasikan sebuah pixel pada gambar.

```
Inisialisasi(startX)
Inisialisasi(startY)
Inisialisasi(originalColor)
Inisialisasi(newColor)

procedure FloodFill (x, y, newColor)
Algoritma

   if ((grid[x][y] = newColor) and
      (grid[x][y] != originalColor))
       or (visited[x][y])
      return;
   else
      visited[x][y] ← true

      grid[x][y] ← newColor

      if (grid[x+1][y] valid)
         FloodFill(x+1, y, newColor)

      if (grid[x][y+1] valid)
         FloodFill(x, y+1, newColor)

      if (grid[x-1][y] valid)
         FloodFill(x-1, y, newColor)

      if (grid[x][y-1] valid)
         FloodFill(x, y-1, newColor)
```

***Figure 8* - Algoritma Flood Fill dengan DFS pada Fungsi Pewarnaan Gambar**

Pada algoritma *flood fill* dengan DFS yang dijelaskan pada Figure 8, pengecekan dilakukan terhadap 4 buah *grid* tetangga (*grid* atas, *grid* kanan, *grid* kiri, *grid* bawah) dari suatu *current grid*. Suatu *grid* tetangga dikategorikan sebagai *grid* yang valid jika *grid* tetangga tersebut memenuhi kriteria berikut :

- *Grid* tetangga memiliki warna yang sama dengan warna dari titik awal dimana pewarnaan dimulai.
  ```
  grid[x][y]=originalColor
  ```
- *Grid* tetangga belum pernah dikunjungi sebelumnya.
  ```
  visited[x][y]= false
  ```
- *Grid* tetangga tidak berwarna yang sama dengan warna pengganti.
  ```
  grid[x][y] != newColor
  ```
- *Grid* tetangga berada dalam area berukuran *width* *x height* dari gambar yang bersangkutan. Hal ini dilakukan untuk menghindari *logical error* pada program yang dibuat.
  ```
  x+1 <= height
  y+1 <= width
  x-1 > 0
  y-1 > 0
  ```

Pada program sederhana yang telah dibuat untuk mensimulasikan proses pewarnaan ini melakukan pengecekan *grid* tetangga dengan urutan *grid* bawah, *grid* kanan, *grid* atas, lalu *grid* kiri.

Proses pewarnaan dari algoritma yang dirancang disimulasikan pada *figure* 9 sampai *figure* 17. *Grid* awal pewarnaan dimulai dari *grid* yang bertanda x.

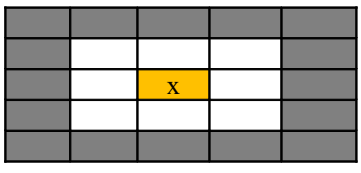

***Figure 9* - Langkah 1**

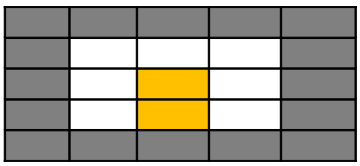

***Figure 10* - Langkah 2**

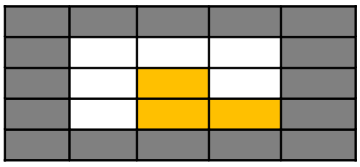

***Figure 11* - Langkah 3**

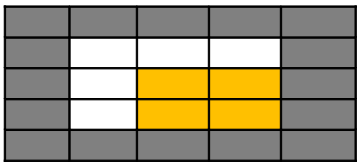

***Figure 12* - Langkah 4**

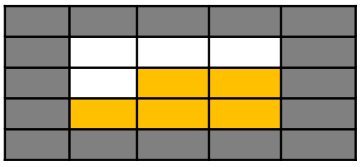

***Figure 13* - Langkah 5**

***Figure 14* - Langkah 6**

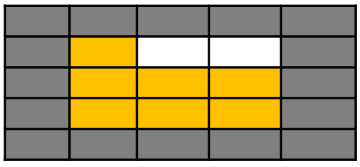

***Figure 15* - Langkah 7**

***Figure 16* - Langkah 8**

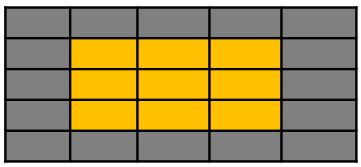

***Figure 17* - Langkah 9**

IV. HASIL SIMULASI

Dengan algoritma *flood fill* menggunakan DFS yang telah diimplementasikan untuk mensimulasikan proses pewarnaan gambar dengan prinsip *bucket tool*, hasil pewarnaan dapat dilakukan dengan hasil yang sesuai dengan apa yang diharapkan, yaitu seluruh area gambar yang seharusnya diwarnai berhasil diubah warnanya dan area gambar yang seharusnya tidak diwarnai tidak berubah warnanya.

Kita misalkan sebuah gambar terdiri dari *grid* yang berisi *character* untuk merepresentasikan sebuah warna.

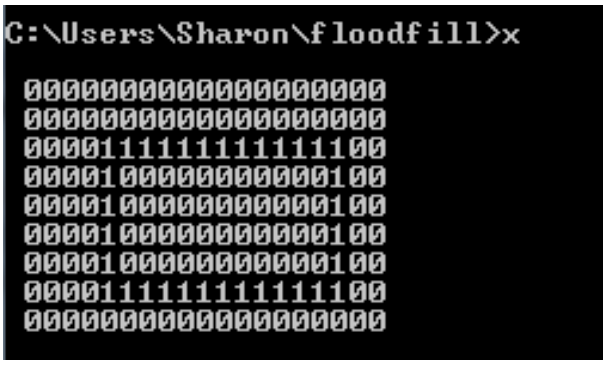

***Figure 18* - Gambar Awal**

Pada *Figure* 18, dapat dilihat bahwa warna latar dari gambar tersebut direpresentasikan sebagai karakter ‘0’, sedangkan warna ‘1’ digunakan untuk menggambar batas persegi panjang.

Jika proses pewarnaan dilakukan dengan mengklik grid[1][1] yaitu grid yang berkarakter ‘0’ dengan warna baru yaitu warna ‘2’, maka semua latar yang berwarna ‘0’ akan diubah menjadi warna ‘2’. Proses pewarnaan tidak dilakukan terhadap warna ‘0’ yang terletak di dalam persegi panjang. Hasil dari pewarnaan ini ditunjukkan pada *Figure* 19.

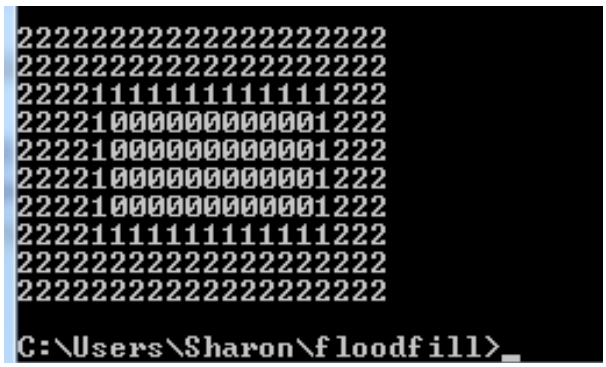

Figure 19 **- Hasil Pewarnaan Latar**

Jika proses pewarnaan dilakukan terhadap garis pembatas persegi panjang, yaitu gambar yang berwarna '1', maka garis pembatas yang berwarna '1' akan berubah warna menjadi warna baru yang ditentukan, yaitu pada kasus ini berupa warna '2'. Hasil proses pewarnaan garis berwarna '1' dari gambar awal pada *Figure* 18 ditunjukkan pada *Figure* 20.

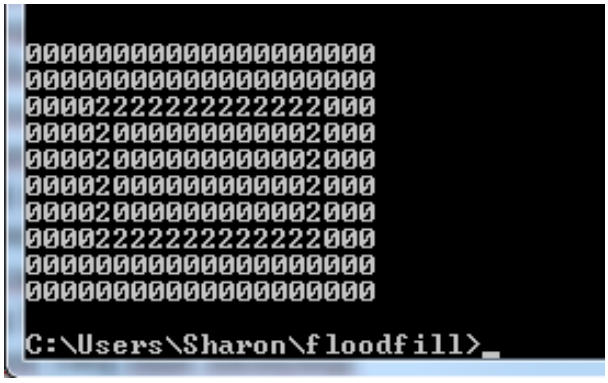

Figure 20 **- Hasil Pewarnaan Garis**

V. KESIMPULAN

Algoritma DFS (*Depth First Search*) merupakan algoritma yang dapat digunakan untuk mengimplementasikan *bucket tool* dalam pewarnaan sebuah gambar. Namun algoritma ini tidak optimal untuk digunakan sebagai algoritma pewarnaan. Algoritma pewarnaan yang lebih optimal adalah algoritma BFS (*Breadth First Search*) yang melakukan pewarnaan secara menyebar langsung ke seluruh arah pada simpul tetangga.

REFERENSI

[1] Munir, Rinaldi, "Diktat Kuliah IF3051 Strategi Algoritma", Program Studi Teknik Informatika, 2009

[2] http://ww3.algorithmdesign.net/handouts/DFS.pdf
Akses : 15 Desember 2012, 08.15 WIB

[3] http://www.programmerinterview.com/index.php/data-structures/dfs-vs-bfs/
Akses : 15 Desember 2012, 08.23 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, 15 Desember 2012

Sharon Loh - 13510086