Perbandingan Kecepatan Algoritma *Flood Fill* dengan Algoritma *Boundary Fill*

Prisila Michelle, 13516129
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
prisilamichelle@yahoo.com

***Abstrak*—BFS dan DFS merupakan algoritma pencarian graf/pohon secara traversal. Algoritma Flood Fill merupakan salah satu penerapan BFS dan DFS pada larik multidimensi. Algoritma Boundary Fill merupakan salah satu penerapan DFS. Kedua algoritma diatas umumnya diaplikasikan untuk mengisikan warna tertentu ke dalam suatu bentuk gambar. Makalah ini membahas mengenai perbandingan kecepatan algoritma Flood Fill BFS, Flood Fill DFS, dan Boundary Fill.**

Kata kunci—BFS, DFS, Flood Fill, Boundary Fill.

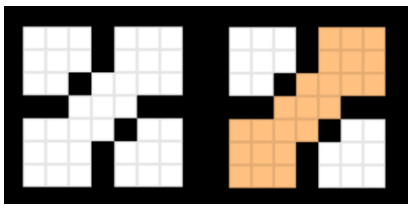
I. PENDAHULUAN

Algoritma *Seed Fill* merupakan algoritma yang berfungsi untuk menentukan daerah yang terhubung dengan sebuah titik pada sebuah larik multidimensi. Algoritma ini dapat dibagi menjadi dua, yaitu algoritma *Flood Fill* dan algoritma *Boundary Fill*.

A. *Flood Fill*

Algoritma *Flood Fill* dapat diimplementasikan secara rekursif dengan menggunakan *Depth First Search* (DFS) atau alternatifnya dengan menggunakan *Breadth First Search* (BFS). Algoritma *Flood Fill* yang diimplementasikan dengan BFS memiliki nama lain yaitu "*Forest Fire Algorithm*". Algoritma *Flood Fill* dapat digunakan pada daerah yang memiliki batas lebih dari satu warna.

Pada umumnya, algoritma *Flood Fill* digunakan pada *bucket fill tool* yang terdapat dalam berbagai aplikasi olah gambar seperti Adobe Photoshop, Microsoft Paint, GIMP (*GNU Image Manipulation Program*), dan sebagainya. *Bucket fill tool* berfungsi untuk mengisikan warna tertentu ke dalam suatu bentuk gambar, contoh penggunaannya seperti gambar di bawah ini.



Gambar 1.1 Efek dari *bucket fill tool* (kiri: kondisi awal, kanan: kondisi akhir) Sumber : https://commons.wikimedia.org/wiki/User:Aka

Algoritma ini juga digunakan pada permainan Minesweeper untuk menentukan sel mana saja yang dibuka ketika pemain membuka sebuah sel. Variasi dari algoritma *Flood Fill* juga dapat digunakan oleh robot Micro-Mouse untuk menentukan arah pada labirin.

B. *Boundary Fill*

Algoritma *Boundary Fill* dapat diimplementasikan secara rekursif dengan menggunakan *Depth First Search* (DFS). Algoritma ini hanya dapat diterapkan pada daerah yang memiliki batas satu warna. Pada umumnya, *Boundary Fill* digunakan untuk mengisi warna pada poligon.

II. DASAR TEORI

A. *Depth First Search* (DFS)

DFS adalah algoritma pencarian mendalam secara traversal pada graf/pohon. Variasi dari DFS pertama kali ditemukan pada abad ke-19 oleh seorang matematikawan Perancis bernama Charles Pierre Trémaux. Trémaux menggunakan temuannya sebagai salah satu cara untuk memecahkan labirin secara lebih efektif.

Berikut adalah pseudocode dari algoritma DFS.

```
procedure DFS(G,v):
  label v as discovered
  for all edges from v to w in G.adjacentEdges(v) do
    if vertex w is not labeled as discovered then
       recursively call DFS(G,w)
```

Pada algoritma ini, pencarian dimulai pada sebuah simpul v, lalu dilanjutkan ke simpul w yang bertetangga dengan simpul v. Setelah itu, pencarian diulang lagi secara rekursif dari simpul w, hingga suatu saat mencapai simpul yang tidak memiliki tetangga lagi. Selanjutnya, dilakukan runut-balik ke simpul yang memiliki tetangga yang belum dikunjungi. Pencarian berakhir jika tidak ada lagi simpul yang belum

dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

DFS diimplementasikan secara rekursif dengan menggunakan *stack* yang bersifat *Last In First Out* (LIFO). Algoritma DFS membutuhkan sebuah tabel *boolean* untuk menyimpan simpul-simpul yang telah dikunjungi agar tidak akan dikunjungi dua kali.

B. Breadth First Search (BFS)

BFS adalah algoritma pencarian melebar secara traversal pada graf/pohon. BFS dan aplikasinya telah ditemukan pada tahun 1945 oleh Konrad Zuse dan Michael Burke, namun penemuan mereka baru dipublikasikan pada tahun 1972. Algoritma ini ditemukan kembali pada tahun 1959 oleh Edward F. Moore untuk menemukan *shortest path* dari sebuah labirin.

Berikut adalah pseudocode dari algoritma BFS.

```
procedure BFS(G,v):
  let Q be a queue
  Q.enqueue(v)
  label v as discovered
  while Q is not empty
    v ← Q.dequeue()
    process(v)
    for all edges from v to w in G.adjacentEdges(v)
do
    if vertex w is not labeled as discovered then
       Q.enqueue(w)
       label w as discovered
```

Pada algoritma BFS, pencarian dimulai pada sebuah simpul v, lalu dilanjutkan ke tetangga-tetangganya terlebih dahulu sebelum dilanjutkan ke *level* yang lebih dalam. Pencarian berakhir jika semua simpul telah dikunjungi.

BFS diimplementasikan dengan menggunakan *queue* yang bersifat *First In First Out* (FIFO). *Queue* ini digunakan untuk menyimpan simpul yang telah dikunjungi. Simpul yang telah dikunjungi perlu disimpan sebagai acuan untuk mengunjungi simpul-simpul yang bertetangga dengannya. Sama seperti algoritma DFS, algoritma ini juga membutuhkan sebuah tabel *boolean* untuk menyimpan simpul-simpul yang telah dikunjungi agar tidak akan dikunjungi dua kali.

C. Flood Fill

Pixel adalah unsur gambar atau representasi sebuah titik terkecil dalam sebuah gambar grafis yang dihitung per inci. Pada implementasi algoritma *Flood Fill*, ditetapkan konvensi bahwa yang merupakan tetangga dari sebuah *pixel* adalah *pixel* yang terletak tepat di sebelah *pixel* tersebut.

a. *Flood Fill* DFS

Berikut adalah pseudocode untuk implementasi *Flood Fill* delapan arah dengan DFS.

```
Flood-fill (node, target-color, replacement-color):
 1. If target-color is equal to replacement-color,
return.
 2. If the color of node is not equal to target-
color, return.
 3. Set the color of node to replacement-color.
 4. Perform Flood-fill (one step to the south of
node, target-color, replacement-color).
    Perform Flood-fill (one step to the north of
node, target-color, replacement-color).
    Perform Flood-fill (one step to the west of
node, target-color, replacement-color).
    Perform Flood-fill (one step to the east of
node, target-color, replacement-color).
    Perform Flood-fill (one step to the southeast of
node, target-color, replacement-color).
    Perform Flood-fill (one step to the southwest of
node, target-color, replacement-color).
    Perform Flood-fill (one step to the northeast of
node, target-color, replacement-color).
    Perform Flood-fill (one step to the northwest of
node, target-color, replacement-color).
 5. Return.
```

Parameter dari fungsi *Flood Fill* adalah *node* (titik yang sedang diperiksa), *target-color* (warna yang ingin diganti), dan *replacement-color* (warna pengganti). Jika target-color sudah sesuai dengan *replacement-color* atau warna *node* tidak sama dengan *target-color*, maka tidak perlu dilakukan *Flood Fill*, fungsi langsung kembali. Jika warna *node* sama dengan *target-color*, maka warna *node* diganti dengan *replacement-color*, lalu panggil lagi fungsi *Flood Fill* dan lakukan *Flood Fill* ke setiap arah.

b. *Flood Fill* BFS

Berikut adalah pseudocode untuk implementasi *Flood Fill* delapan arah dengan BFS.

```
Flood-fill (node, target-color, replacement-color):
  1. If target-color is equal to replacement-color,
return.
```

```
  2. If color of node is not equal to target-color,
return.
  3. Set Q to the empty queue.
  4. Set the color of node to replacement-color.
  5. Add node to the end of Q.
  6. While Q is not empty:
  7.     Set n equal to the first element of Q.
  8.     Remove first element from Q.
  9.     If the color of the node to the west of n
is target-color,
            set the color of that node to
replacement-color and add that node to the end of Q.
 10.     If the color of the node to the east of n
is target-color,
            set the color of that node to
replacement-color and add that node to the end of Q.
 11.     If the color of the node to the north of n
is target-color,
            set the color of that node to
replacement-color and add that node to the end of Q.
 12.     If the color of the node to the south of n
is target-color,
            set the color of that node to
replacement-color and add that node to the end of Q.
13.      If the color of the node to the southeast
of n is target-color,
            set the color of that node to
replacement-color and add that node to the end of Q.
14.      If the color of the node to the southwest
of n is target-color,
            set the color of that node to
replacement-color and add that node to the end of Q.
15.      If the color of the node to the northeast
of n is target-color,
            set the color of that node to
replacement-color and add that node to the end of Q.
16.     If the color of the node to the northwest of
n is target-color,
            set the color of that node to
replacement-color and add that node to the end of Q.

17. Continue looping until Q is exhausted.
18. Return.
```

Parameter dari fungsi ini sama dengan parameter pada fungsi *Flood Fill* pada DFS. Jika *target-color* sudah sesuai dengan *replacement-color* atau warna *node* tidak sama dengan *target-color*, maka tidak perlu dilakukan *Flood Fill*, fungsi langsung kembali. Jika warna titik sama dengan *target-color*, maka warna *node* diganti dengan *replacement-color*, lalu tambahkan *node* pada akhir *queue*. Selama masih ada *node* di dalam *queue*, jadikan *node* perama dari *queue* sebagai patokan. Hapus elemen pertama dari *queue*, lalu cek setiap tetangga dari *node* tersebut apakah warnanya sama dengan *target-color*. Jika warnanya sama, maka warna *node* tersebut diganti dengan *replacement-color* dan *node* tersebut ditambahkan pada akhir *queue.*

D. Boundary Fill

Pada algoritma *Boundary Fill*, warna dari *boundary* (batas) dijadikan sebagai patokan untuk mengisi warna. Proses mewarnai dimulai dari suatu titik didalam poligon. *Pixel* yang warnanya sama dengan *boundary* tidak akan diwarnai.

Berikut adalah pseudocode dari algoritma *Boundary Fill* delapan arah yang diterapkan secara DFS.

```
Boundary-fill (node, boundary-color,
replacement-color):
 1. If color of node is not equal to replacement-
color and color of node is not equal to boundary-
color then
 2. Set the color of node to replacement-color.
 3. Perform Boundary-fill (one step to the south of
node, boundary-color, replacement-color).
    Perform Boundary-fill (one step to the north of
node, boundary-color, replacement-color).
    Perform Boundary-fill (one step to the west of
node, boundary-color, replacement-color).
    Perform Boundary-fill (one step to the east of
node, boundary-color, replacement-color).
    Perform Boundary-fill (one step to the southeast
of node, target-color, replacement-color).
    Perform Boundary-fill (one step to the southwest
of node, boundary-color, replacement-color).
    Perform Boundary-fill (one step to the northeast
of node, boundary-color, replacement-color).
    Perform Boundary-fill (one step to the northwest
of node, boundary-color, replacement-color).
```

```
4. Return.
```

Parameter dari fungsi ini adalah *node* (titik yang sedang diperiksa), *boundary-color* (warna dari batas poligon), dan *replacement-color* (warna pengganti). Jika warna *node* tidak sama dengan *replacement-color* dan warna *node* tidak sama dengan *boundary-color*, maka warna *node* diganti dengan *replacement-color*. Setelah itu, panggil lagi fungsi *Boundary Fill* dan lakukan *Boundary Fill* ke delapan arah.

III. Pengujian Algoritma Flood Fill dan Boundary Fill

Pada seluruh pengujian yang dilakukan, digunakan *Flood Fill* dan *Boundary Fill* delapan arah. Program dijalankan pada web rextester.com. Digunakan larik dua dimensi (matriks) sebagai representasi dari gambar dan pixel di dalamnya. Setiap kotak pada matriks melambangkan sebuah *pixel*. Huruf yang berada di dalamnya melambangkan warna dari *pixel*. Untuk menghitung waktu dari setiap percobaan, digunakan *library chrono* yang dapat menampilkan informasi waktu dalam nano detik. Dilakukan masing-masing tiga kali pengujian dengan ukuran matriks yang berbeda-beda. Berikut adalah hasil pengujian yang telah dilakukan.

A. Matriks berukuran 10x10

Test case yang digunakan :

```
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y
Y Y C C C C C C Y Y
Y Y C C C C C C Y Y
Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
```

Replacement-color : A
Target-color : C
Boundary-color : Y

- Hasil *Flood Fill* dengan DFS

```
Compilation time: 0.52 sec, absolute running time: 0.16 sec, cpu time: 0.11 sec, memory peak: 3 Mb, absolute service time: 0,7 sec
```

```
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y A A Y Y Y Y
Y Y Y Y A A Y Y Y Y
Y Y A A A A A A Y Y
Y Y A A A A A A Y Y
Y Y Y Y A A Y Y Y Y
Y Y Y Y A A Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Waktu yang dihabiskan : 1229 nano detik
```

- Hasil *Flood Fill* dengan BFS

```
Compilation time: 0.62 sec, absolute running time: 0.16 sec, cpu time: 0.1 sec, memory peak: 3 Mb, absolute service time: 0,79 sec
```

```
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y A A Y Y Y Y
Y Y Y Y A A Y Y Y Y
Y Y A A A A A A Y Y
Y Y A A A A A A Y Y
Y Y Y Y A A Y Y Y Y
Y Y Y Y A A Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Waktu yang dihabiskan : 18607 nano detik
```

- Hasil *Boundary Fill*

```
Compilation time: 0.53 sec, absolute running time: 0.17 sec, cpu time: 0.11 sec, memory peak: 3 Mb, absolute service time: 0,71 sec
```

```
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y A A Y Y Y Y
Y Y Y Y A A Y Y Y Y
Y Y A A A A A A Y Y
Y Y A A A A A A Y Y
Y Y Y Y A A Y Y Y Y
Y Y Y Y A A Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y
Waktu yang dihabiskan : 2025 nano detik
```

B. Matriks berukuran 15x15

Test case yang digunakan :

```
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y C Y Y Y Y Y Y Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y Y
Y Y C C C C C C Y Y Y Y Y Y Y
Y Y C C C C C C Y Y Y Y Y Y Y
Y Y Y Y C C Y C C C Y Y Y Y Y
Y Y Y Y C C Y C C C C Y Y Y Y
Y Y Y Y Y Y Y C C C C Y Y Y Y
Y Y Y Y Y Y Y C Y Y C Y Y Y Y
Y Y Y Y Y Y Y C Y Y C Y Y Y Y
Y Y Y Y Y Y Y Y Y Y C Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
```

Replacement-color : A
Target-color : C
Boundary-color : Y

- Hasil *Flood Fill* dengan DFS

```
Compilation time: 0.63 sec, absolute running time: 0.18 sec, cpu time: 0.12 sec, memory peak: 3 Mb,
absolute service time: 0,83 sec
```

```
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y A Y Y Y Y Y Y Y Y Y Y
Y Y Y Y A A Y Y Y Y Y Y Y Y Y
Y Y A A A A A A Y Y Y Y Y Y Y
Y Y A A A A A A Y Y Y Y Y Y Y
Y Y Y Y A A Y A A A Y Y Y Y Y
Y Y Y Y A A Y A A A A Y Y Y Y
Y Y Y Y Y Y Y A A A A Y Y Y Y
Y Y Y Y Y Y Y A Y Y A Y Y Y Y
Y Y Y Y Y Y Y A Y Y A Y Y Y Y
Y Y Y Y Y Y Y Y Y Y A Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Waktu yang dihabiskan : 1866 nano detik
```

- Hasil *Flood Fill* dengan BFS

```
Compilation time: 0.62 sec, absolute running time: 0.16 sec, cpu time: 0.1 sec, memory peak: 3 Mb,
absolute service time: 0,79 sec
```

```
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y A Y Y Y Y Y Y Y Y Y Y
Y Y Y Y A A Y Y Y Y Y Y Y Y Y
Y Y A A A A A A Y Y Y Y Y Y Y
Y Y A A A A A A Y Y Y Y Y Y Y
Y Y Y Y A A Y A A A Y Y Y Y Y
Y Y Y Y A A Y A A A A Y Y Y Y
Y Y Y Y Y Y Y A A A A Y Y Y Y
Y Y Y Y Y Y Y A Y Y A Y Y Y Y
Y Y Y Y Y Y Y A Y Y A Y Y Y Y
Y Y Y Y Y Y Y Y Y Y A Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Waktu yang dihabiskan : 46675 nano detik
```

- Hasil *Boundary Fill*

```
Compilation time: 0.53 sec, absolute running time: 0.2 sec, cpu time: 0.13 sec,
memory peak: 3 Mb, absolute service time: 0,75 sec
```

```
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y A Y Y Y Y Y Y Y Y Y Y
Y Y Y Y A A Y Y Y Y Y Y Y Y Y
Y Y A A A A A A Y Y Y Y Y Y Y
Y Y A A A A A A Y Y Y Y Y Y Y
Y Y Y Y A A Y A A A Y Y Y Y Y
Y Y Y Y A A Y A A A A Y Y Y Y
Y Y Y Y Y Y Y A A A A Y Y Y Y
Y Y Y Y Y Y Y A Y Y A Y Y Y Y
Y Y Y Y Y Y Y A Y Y A Y Y Y Y
Y Y Y Y Y Y Y Y Y Y A Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
 Waktu yang dihabiskan : 3209 nano detik
```

C. Matriks berukuran 20x20

Test case yang digunakan :

```
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y C C C C C C C C C C C C C C C C Y Y
Y Y C C C C C C C C C C C C C C C C Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y C C C C C C C C C C C C C C C C Y Y
Y Y C C C C C C C C C C C C C C C C Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y Y Y C C Y Y Y Y Y Y Y Y C C Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
```

Replacement-color : A
Target-color : C
Boundary-color : R

- Hasil *Flood Fill* dengan DFS

```
Compilation time: 0.52 sec, absolute running time: 0.18 sec, cpu time: 0.12 sec, memory peak: 3 Mb, absolute service time: 0,72
sec
```

```
R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R R R R R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R A A A A A A A A A A A A A A A A R R
R R A A A A A A A A A A A A A A A A R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R A A A A A A A A A A A A A A A A R R
R R A A A A A A A A A A A A A A A A R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R R R R R R R R
Waktu yang dihabiskan : 9578 nano detik
```

- Hasil *Flood Fill* dengan BFS

```
Compilation time: 0.73 sec, absolute running time: 0.22 sec, cpu time: 0.16 sec, memory peak: 4 Mb, absolute service time: 0,96
sec
```

```
R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R R R R R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R A A A A A A A A A A A A A A A A R R
R R A A A A A A A A A A A A A A A A R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R A A A A A A A A A A A A A A A A R R
R R A A A A A A A A A A A A A A A A R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R R R R R R R R
Waktu yang dihabiskan : 25618758 nano detik
```

- Hasil *Boundary Fill*

```
Compilation time: 0.53 sec, absolute running time: 0.18 sec, cpu time: 0.12 sec, memory peak: 3 Mb, absolute service time: 0,72
sec
```

```
R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R R R R R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R A A A A A A A A A A A A A A A A R R
R R A A A A A A A A A A A A A A A A R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R A A A A A A A A A A A A A A A A R R
R R A A A A A A A A A A A A A A A A R R
R R R R A A R R R R R R R R A A R R R R
R R R R A A R R R R R R R R A A R R R R
R R R R R R R R R R R R R R R R R R R R
R R R R R R R R R R R R R R R R R R R R
Waktu yang dihabiskan : 10192 nano detik
```

Pada pengujian matriks 10x10, waktu yang dimakan oleh algoritma *Flood Fill* DFS adalah 1.229 nano detik, *Flood Fill*

BFS 18.607 nano detik, dan *Boundary Fill* 2.025 nano detik. Sedangkan pada pengujian matriks 15x15, waktu yang dimakan oleh algoritma *Flood Fill* DFS adalah 1.866 nano detik, *Flood Fill* BFS 46.675 nano detik, dan *Boundary Fill* 3.209 nano detik. Pada pengujian matriks 20x20, waktu yang dimakan oleh algoritma *Flood Fill* DFS adalah 9.578 nano detik, *Flood Fill* BFS mengalami peningkatan waktu yang signifikan menjadi 25.618.758 nano detik, dan *Boundary Fill* 10.192 nano detik..

IV. Kesimpulan

Jika dilihat dari waktunya, pengujian yang yang kecepatannya paling tinggi adalah *Flood Fill* DFS, disusul oleh algoritma *Boundary Fill* dan *Flood Fill* BFS. Algoritma *Flood Fill* BFS terlihat tidak cocok digunakan pada gambar yang berukuran besar karena waktunya meningkat secara signifikan seiring dengan bertambahnya ukuran matriks.

Ucapan Terima Kasih

Pertama-tama, Penulis mengucap syukur kepada Tuhan Yang Maha Esa karena berkat dan penyertaan-Nya, Penulis dapat merampungkan makalah ini dengan baik. Penulis berterima kasih kepada orang tua Penulis yang telah memberikan dukungan dalam segala aspek kepada Penulis. Penulis juga berterima kasih kepada Dr. Ir. Rinaldi Munir, M.T., Dr. Masayu Leylia Khodra, S.T., M.T., dan Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku dosen pengajar Strategi Algoritma yang telah membimbing dan mengajarkan materi-materi yang digunakan pada penulisan makalah ini.

Daftar Pustaka

[1] Munir, Rinaldi. 2018. *Diktat Kuliah IF2211 Strategi Algoritma*. Bandung: Institut Teknologi Bandung.

[2] Torbert, Shane. 2016. *Applied Computer Science*, *Second Edition*. Cham: Springer International Publishing.

[3] http://www.codecodex.com/wiki/Implementing_the_flood_fill_algorithm#Pseudocode diakses tanggal 10 Mei 2018, pukul 19.05

[4] http://otfried.org/courses/cs206/notes/floodfill.pdf diakses tanggal 10 Mei 2018, pukul 20.19

[5] https://www.geeksforgeeks.org/boundary-fill-algorithm/ diakses tanggal 11 Mei 2018, pukul 18.15

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, 12 Mei 2018

Prisila Michelle, 13516129