Penerapan *Flood-Fill Concept* dengan Memanfaatkan Algoritma BFS dan DFS dalam Aplikasi Mewarnai

Mohammad Rifqi Farhansyah - 13521166[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
[1]13521166@std.stei.itb.ac.id

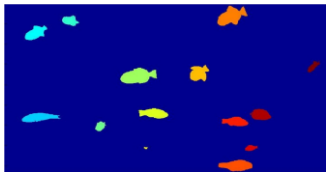
***Abstract*—Dalam bidang *Computer Graphics* atau Grafika Komputer, terdapat berbagai macam *polygon filling algorithms* seperti *inside and outside test, scan line method, boundary fill, flood fill, edge fill,* serta *fence fill algorithm*. Pada makalah ini, akan direalisaikan metode *Flood-Fill* dengan memanfaatkan salah satu konsep strategi algoritma, yaitu Algoritma *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS). *Flood-Fill* merupakan algoritma yang seringkali digunakan untuk menentukan *bounded area* yang terhubung dengan node tertentu pada *multi-dimensional array*. Algoritma BFS dan DFS akan digunakan pada pencarian tetangga terdekat dengan pendekatan *Queue* dan *Stack*. Namun demikian, dalam melakukan sebuah *Flood-Fill* Process, algoritma BFS dan DFS memerlukan beberapa komponen atribut sebagai parameter unik dari setiap representasi *cell* dengan *grid*. Parameter *Flood-Fill* makalah ini akan berbasis pada warna dari *adjacent cell* dalam representasi *grid*. Makalah ini akan berfokus dalam menerapkan *Flood-Fill Concept,* Algoritma BFS dan DFS, serta keseluruhan aspek lainnya dalam pembuatan aplikasi mewarnai sederhana.**

Keywords—BFS; DFS; Flood-Fill; Paint Application.

I. PENDAHULUAN

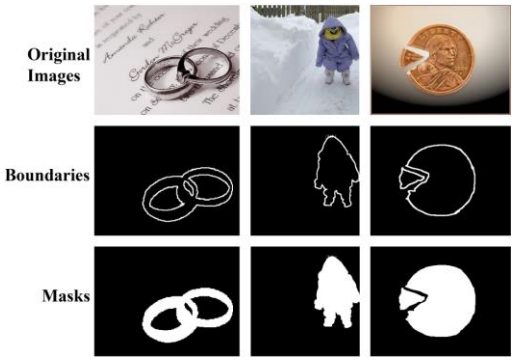
Grafika Komputer atau *Computer Graphics* merupakan salah satu *core technology* dalam industri *digital photography*, film, *video games*, *digital art*, *cell phone*, serta *computer display*. Konsep utama dari grafika komputer adalah menghasilkan sebuah gambar atau citra tertentu dengan bantuan komputer. Salah satu algoritma dasar dalam grafika komputer adalah algoritma *flood-fill* (disebut juga sebagai *seed-fill algorithm*). Algoritma ini digunakan untuk menghubungkan *adjacent neighbour* dan elemen terkait dalam sebuah array dengan parameter tertentu. *Flood-fill algorithm* seringkali digunakan dalam aplikasi pengolahan gambar seperti *Adobe Photoshop* serta *Corel Paintshop*. Selain itu, penerapan lainnya adalah dalam *computer puzzle games*, seperti penyelesaian *mazes* dan sebagainya. Gambar 1.1 menunjukkan salah satu hasil dari penerapan *flood-fill algorithm* pada citra gambar tertentu. Algoritma *flood-fill* diterapkan untuk mendeteksi adanya citra tertentu (dalam hal ini adalah objek) yang terdapat pada suatu gambar. Kumpulan pixel tertentu akan dihubungkan dengan sebuah garis sepanjang warna biru yang terdeteksi oleh program berada di bawah *threshold*. Pewarnaan untuk tiap objek akan didasarkan saturasi dan hue tiap warna dalam gambar serta selisih nilai dengan *threshold* warna dasar. Pada gambar 1.2 diberikan hasil dari proses *flood-fill* berupa tiap grup ikan akan diberi warna yang berbeda. *Flood-fill algortihm* dapat dikembangkan lebih lanjut, misalnya dalam ruang 3 atau 4 dimensi. Contoh kasus yang dapat digunakan, antara lain : pewarnaan sebuah gambar ikan 3D serta melakukan *tracking* pada gambar ikan 3D yang sedang bergerak dalam tiap waktu tertentu.

Gambar 1.1 *Ilustrasi Gambar Awal*
Sumber :
https://ieeexplore.ieee.org/abstract/document/4786975/authors#authors



Gambar 1.2 *Ilustrasi Gambar Akhir*
Sumber :
https://ieeexplore.ieee.org/abstract/document/4786975/authors#authors

Dalam penelitian terbaru terdapat beberapa pengembangan algoritma *flood-fill*, salah satunya adalah *Scan-flood fill* (SCAFF) *algorithm*. SCAFF merupakan salah satu algoritma otomasi efektif yang dapat digunakan untuk melakukan *precise region filling algorithm* untuk *complicated regions*. Salah satu contohnya adalah pada pengolahan citra *LIDC-IDRI dataset* yang seringkali digunakan untuk mendeteksi penyakit dalam hati. Proses tersebut memanfaatkan data ber-ekstensi XML yang akan menyimpan nilai *edge* untuk tiap nodul dari citra hati. Namun, untuk memperoleh performa yang lebih baik dalam sistem syaraf (lebih kompleks), terkhusus dalam *convolution-based neural networks*, biasanya akan memanfaatkan proses *masking* pada citra gambar. Sejalan dengan itu, proses *filling mask* pada batas beranotasi tertentu juga merupakan proses wajib agar dapat menghasilkan akurasi yang lebih baik dalam diagnosis. Secara umum, *region filling* merujuk pada pendekatan untuk melakukan pewarnaan pada *bounded regions* dengan warna yang telah ditentukan. Berdasarkan domain dari citra gambar yang diolah, *region filling algorithm* dapat diklasifikasikan menjadi *raster filling* serta *vector filling*. Gambar 1.3 merupakan salah satu contoh penerapan proses *region fillling* yang dapat digunakan untuk menghasilkan *mask* tertentu pada segmentasi objek yang menonjol. Pada gambar tersebut, *region filling* diterapkan terhadap citra yang terkait dengan *raster filling*. Gambar yang diberikan label dan pewarnaan tertentu hanyalah gambar yang masuk ke dalam piksel batas. *Original images* adalah ilustrasi awal gambar, *Boundaries* adalah citra hasil *masking* pada gambar, serta *Masks* merupakan hasil proses *raster filling*.



Gambar 1.3 *Ilustrasi Region Filling Algorithm*
Sumber : https://openaccess.thecvf.com/content_CVPRW_2019/papers/

Dalam makalah ini penulis akan lebih fokus terhadap pembahasan algoritma *flood-fill* dasar. Sesuai dengan penjelasan di atas, penerapan algoritma *flood-fill* memerlukan sebuah citra gambar tertentu. Namun, pada makalah ini penulis akan menggantinya dengan sebuah representasi *cell* dalam *grid*, sehingga proses *flood-fill* akan dilakukan pada setiap *adjacent cell*. Dengan menerapkan konsep tersebut, penulis dapat merancang sebuah aplikasi mewarnai sederhana.

Selain itu, pada makalah ini, penulis akan menggunakan algoritma BFS dan DFS dalam melakukan proses *flood-fill*. Algoritma tersebut dipilih karena penerapannya yang cukup mudah, tingkat akurasi yang cukup, serta kesesuainnya dengan materi pada mata kuliah IF2211 Strategi Algoritma.

II. LANDASAN TEORI

A. *Definisi Graf Traversal*

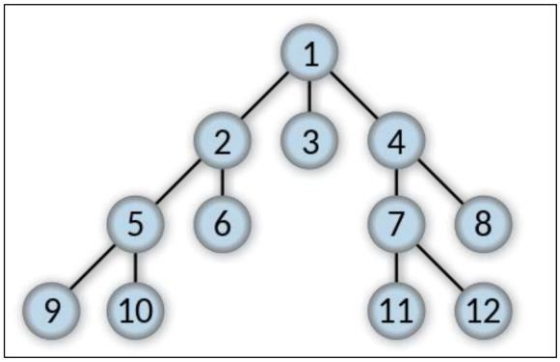
Graf Traversal adalah teknik untuk mengunjungi setiap node dari graf. Teknik ini juga digunakan untuk menghitung urutan simpul dalam proses *traverse*. Secara umum, proses mengunjungi setiap node akan dimulai dari *starting node* serta akan disimpan informasi terkait nodes yang telah dikunjungi. Hal ini bertujuan untuk meminimalisir terjadinya *infine loop* karena dalam graf dimungkinkan terjadinya proses mengunjungi suatu node lebih dari ssatu kali. Proses pencatatan *visited node* dapat direkayasa menggunakan notasi biner, seperti jika suatu *vertex* telah dikunjungi maka nilainya akan menjadi nol, jika tidak maka satu.

Terdapat beberapa algortima yang dapat digunakan untuk menyelesaikan permasalahan graf traversal, antara lain:

1. Breadth-First Search (BFS)
2. Depth-First Search (DFS)
3. A* Search
4. Djikstra's Algorithm
5. Bellman-Ford Algorithm

B. *Breadth-First Search (BFS)*

Breadth-First Search atau BFS merupakan salah satu algoritma yang paling sederhana dalam melakukan pencarian pada graf dan pola dasar bagi penerapan algoritma lainnya. Algoritma *minimum-spanning-tree Prim* dan algoritma pencarian jarak terdekat Djikstra merupakan contoh penerapan algoritma lain yang memiliki ide serupa dengan algoritma *Breadth-First-Search* (BFS). Dalam algoritma BFS, graf direpresentasikan dalam bentuk G = (V, E) dengan v merupakan simpul awal penelusuran.



Gambar 2.1 *Ilustrasi Graph Mernggunakan BFS*
Sumber :
https://upload.wikimedia.org/wikipedia/commons/thumb/3/33/Breadth-first-tree.svg/450px-Breadth-first-tree.svg.png

Secara sederhana, algoritma *breadth-first search* akan memulai penelusuran dari simpul v. Kemudian, akan dilakukan penelusuran terhadap semua simpul yang bertetangga dengan simpul v terlebih dahulu. Terakhir, akan dilakukan penelusuran terhadap simpul-simpul lain yang belum dikunjungi dan bertetangga dengan simpul-simpul yang telah dikunjungi. Langkah ini akan dilakukan terus-menerus hingga ditemukan simpul yang dicari atau hingga seluruh simpul telah ditelusuri.

Pada prosedur BFS, algoritma ini dapat digunakan dengan menerapkan adjacency lists dalam merepresentasikan graf G = (V, E). Selain itu, untuk mengetahui simpul yang akan diperiksa atau tidak, akan digunakan struktur data Queue. Terakhir, untuk mengetahui suatu simpul telah diperiksa atau belum, akan digunakan strktur data array atau *hash-table* yang bertipe boolean. Berikut adalah *pseudocode* umum utuk prosedur *breadth-first search*.

```
procedure BFS(input G: graph, input v: vertice)
{ Traversal Graf dengan algoritma penelususran BFS
  I.S. G dan v terdefinisi dan tidak sembarang
  F.S. Semua simpul yang dilalui tercetak pada layar }
KAMUS
  { Variabel }
  q : queue
  w : vertice
  visited : hashTable
  { Fungsi dan Prosedur antara }
  procedure createQueue(input/output q: queue)
  { Inisialisasi queue
    I.S. q belum terdefinisi
    F.S. q terdefinisi dan kosong }
  procedure enqueue(input/output q: queue, input v: vertice)
  { Memasukkan v ke dalam q dengan aturan FIFO
    I.S. q terdefinisi
    F.S. v masuk ke dalam q dengan aturan FIFO }
  function dequeue(q: queue) → vertice
  { Menghapus simpul dari q dengan aturan FIFO
    dan mengembalikan simpul yang dihapus}
  function isQueueEmpty(q: queue) → boolean
  { Mengembalikan True jika q kosong dan sebaliknya }
  procedure createHashTable(input/output T: hashTable)
  { Inisialisasi hashTable
    I.S. T belum terdefinisi
    F.S. T terdefinisi dan terisi False sebanyak simpul }
   procedure setHashTable(input/output T: hashTable, input v:
vertice, input val: boolean )
  { Mengubah value dari T
    I.S. T sudah terdefinisi
    F.S. Elemen T dengan key v sudah diubah menjadi val }
```

```
   function getHashTable(T: hashTable, input v: vertice) → boolean
  { Mengembalikan elemen T pada key v }

ALGORITMA
  { Inisialisasi visited }
  createHashTable(visited)
  { Inisialisasi q }
  createQueue(q)

  { Mengunjungi simpul pertama (akar) }
  output(v)
  enqueue(q, v)
  setHashTable(T, v, True)

  { Mengunjungi semua simpul }
  while (not isQueueEmpty(q)) do
      v ← dequeue(q)
      for setiap simpul w yang bertetangga dengan simpul v do
          if (not getHashTable(visited, w)) then
              output(w)
              enqueue(q, w)
              setHashTable(T, w, True)

  { q kosong }
```

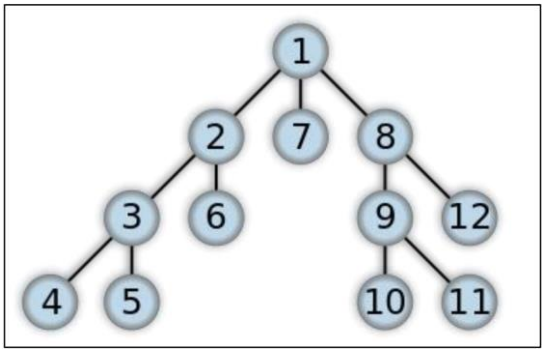
Pseudocode umum untuk prosedur *Breadth-First Search* di atas akan menghasilkan seluruh input yang ada. Untuk pencarian satu simpul, algoritma dapat ditambahkan menjadi seperti berikut:

```
procedure BFS(input G: graph, input v: vertice, input x: vertice)
{ Traversal Graf dengan algoritma penelususran BFS
   I.S. G dan v terdefinisi dan tidak sembarang
   F.S. Semua simpul yang dilalui tercetak pada layar }
KAMUS
  { Variabel }
  { IDEM }
  found : boolean
  { Fungsi dan Prosedur antara }
  { IDEM }

ALGORITMA
  { Inisialisasi found }
  found ← False
  { Inisialisasi visited }
  createHashTable(visited)
  { Inisialisasi q }
  createQueue(q)

  { Mengunjungi simpul pertama (akar) }
  output(v)
  enqueue(q, v)
  setHashTable(T, v, True)

  { Memeriksa simpul pertama (akar) }
  if (v = x) then
      found ← True

  { Mengunjungi semua simpul }
  while (not isQueueEmpty(q) and not found) do
      v ← dequeue(q)
      for setiap simpul w yang bertetangga dengan simpul v do
          if (not getHashTable(visited, w) and not found) then
          output(w)
          enqueue(q, w)
          setHashTable(T, w, True)
          if (w = x) then
              found ← True

  if (not found) then
      output("Tidak ditemukan")
```

C. Depth-First Search (DFS)

Berbeda dengan *Breadth-First Search*, algoritma pencarian *Depth-First Search* melakukan penelusuran terlebih dahulu ke-"dalam" ketika kondisi pencarian memungkinkan. Algoritma *Depth-First Search* akan menelusuri simpul tetangga dari simpul yang ditelusuri sekarang. Apabila sudah tidak ada simpul tetangga yang tersedia, algoritma melakukan proses *backtracking* untuk menelusuri simpul-simpul tetangga dari simpul sebelumnya.

Secara sederhana, algoritma *Depth-First Search* (DFS) akan memulai penelusuran dari simpul v. Kemudian, akan dilanjutkan penelusuran ke simpul w yang bertetangga dengan simpul v. Setelah itu, akan dilakukan pemanggilan kembali algoritma DFS yang dimulai dari simpul w. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga telah dikunjungi, akan dilakukan pencarian runut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya, yaitu simpul yang melakukan pemanggilan algoritma DFS. Pencarian selesai jika tidak ada lagi *unvisited node* yang dapat dicapai dari *visited node*.



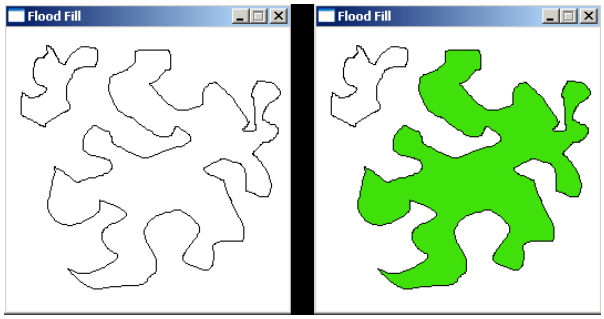
***Gambar 2.2** Ilustrasi Graph Mernggunakan DFS*
Sumber : https://upload.wikimedia.org/wikipedia/commons/thumb/1/1f/Depth-first-tree.svg/375px-Depth-first-tree.svg.png

Pada prosedur *Depth-First Search* (DFS), representasi graf dan strukur data hampir serupa seperti yang digunakan pada prosedur *Breadth-First Search*. Akan tetapi, pada *Depth-First Search* (DFS), tidak akan digunakan struktur data Queue, melainkan lebih mirip dengan struktur data Stack. Berikut adalah *pseudocode* dari prosedur *Depth-First Search* (DFS).

```
procedure DFS(input G: graph, input/output visited: hashTable, input v: vertice)
{ Traversal Graf dengan algoritma penelususran DFS
  I.S. G, v, dan visited terdefinisi dan tidak sembarang
  F.S. Semua simpul yang dilalui tercetak pada layar }
KAMUS
  { Variabel }
  w : vertice
  { Fungsi dan Prosedur antara }
   procedure setHashTable(input/output T: hashTable, input v: vertice, input val: boolean )
  { Mengubah value dari T
    I.S. T sudah terdefinisi
    F.S. Elemen T dengan key v sudah diubah menjadi val }
   function getHashTable(T: hashTable, input v: vertice) → boolean
  { Mengembalikan elemen T pada key v }

ALGORITMA
  { Mengunjungi simpul sekarang }
  output(v)
  setHashTable(T, v, True)
  { Mengunjungi semua simpul }
  for setiap simpul w yang bertetangga dengan simpul v do
      if (not getHashTable(visited, w)) then
          DFS(G, visited, w)
```

D. *Flood Fill Algorithm*

Algoritma *flood-fill* atau yang biasa disebut sebagai *seed fill*, merupakan sebuah *flooding* algoritma yang menentukan serta mengubah suatu area terhubung dari node tertentu dalam *multidimensional array* dengan beberapa parameter atribut pencocokan. *Flood-fill* algoritma telah diterapkan cukup luas pada industri-industri yang berkaitan dengan pengolahan citra gambar, seperti fitur "*bucket*" yang terdapat pada aplikasi pengolahan gambar (*Paint*, *Adobe Photoshop*, *Corel Paintshop*, dll). Pada algoritma *flood-fill* konvensional, terdapat beberapa parameter yang diperlukan, antara lain: *start node, target color*, serta *replacement color*. Algoritma akan mencari keseluruhan nodes yang terhubung *dengan start node* melalui *path* tertentu, kemudian mengganti warna-nya dengan *replacement color*.



***Gambar 2.3** Ilustrasi Flood-Fill*
Sumber : https://lodev.org/cgtutor/floodfill.html

Pada makalah ini akan diterapkan algoritma *flood fill* yang berbasis stack *recursive*. Hal ini diimplementasikan agar dapat diterapkannya algoritma BFS dan DFS. Berikut ini adalah pseudocode dari program *flood-fill* yang hendak diterapkan.

```
function floodFill(input _node : node)
{ Traversal Graf dengan algoritma flood-fill
  I.S. _node terdefinisi
  F.S. kondisi tiap node akan berubah setelah dilalui }
KAMUS
  { Variabel }
  Q : stack / queue
  n : node

ALGORITMA
  { Melakukan set Q menjadi stack atau queue sesuai kebutuhan }
  addLast(Q, node)
  { Proses looping }
  while Q is not empty:
    { Set n agar sama dengan element Q }
    Q ← setFirst(n)
    { Remove elemen pertama jika menggunakan Queue dan elemen
terakhir jika menggunakan stack }
    pop(Q)
    if n is Inside:
      { set the n }
        Add the node to the west of n to the end of Q.
        Add the node to the east of n to the end of Q.
        Add the node to the north of n to the end of Q.
        Add the node to the south of n to the end of Q.
 { Loop hingga seluruh node dikunjungi }
 return
```

III. IMPLEMENTASI

A. *Struktur Data Program*

Directory Tree dari *source code* implementasi program adalah sebagai berikut:

```bash
  changes.txt
  README.md
  requirements.txt

──doc
    13521166_MohammadRifqiFarhansyah.doc
    13521166_MohammadRifqiFarhansyah.pdf

──img
    FoodFill.gif
    icon.png
    SS1.png
    SS2.png
    SS3.png
    SS4.png

──src
  main.py
  visualization.py
```

***Gambar 3.1** Directory Tree dari Source Code*
Sumber : Dokumen Pribadi

Struktur repository program terdiri atas 3 folder utama serta 3 dokumen di luar folder, yaitu: *README.md*, *changes.txt*, serta *requirements.txt*. `*changes.txt*` merupakan file berekstensi TXT yang akan berisi catatan setiap perubahan warna terakhir pada *canvas* (dimanfaatkan dalam visualisasi *pie chart*). Sementara `*requirements.txt*` berisi *library* atau pustaka yang diperlukan pada implementasi makalah ini. Folder pertama merupakan folder `*doc*` yang digunakan untuk menyimpan dokumen-dokumen terkait Tugas Makalah IF2211 Strategi Algoritma 2022/2023. Folder kedua merupakan folder `*img*` yang berisi gambar-gambar yang diperlukan pada *project* ini. *SS1.png*, *SS2.png*, *SS3.png*, serta *SS4.png* merupakan screenshot hasil percobaan program, dimana *SS1.png* merupakan gambar tampilan awal program, *SS2.png* merupakan tampilan gambar sebelum diwarnai, *SS3.png* merupakan tampilan setelah gambar diwarnai., serta *SS4.png* merupakan hasil visualisasi percobaan. *FoodFIll.gif* merupakan representasi langkah kerja program dalam bentuk video berekstensi GIF. Sementara *icon.png* merupakan *favicon* yang digunakan ketika menjalankan program.

Folder terakhir adalah folder `*src*` yang berisi 2 program utama dalam bentuk *python* (versi 3.9.4) *source code file*, bernama `*main.py*` dan `*visualization.py*`. Pada *source code* tersebut akan dijalankan program 'Aplikasi Mewarnai Sederhana'. Program akan ditampilkan menggunakan *library PyGame* sementara algoritma *Flood-Fill* diterapkan dengan algoritma BFS dan DFS. Untuk lebih lengkapnya, *source code* dapat diakses melalui pranala berikut ini.

B. *Implementasi Fungsi Food-Fill*

Pada makalah ini, fungsi *Food-Fill* berada pada *source code* `*main.py*` yang terdapat dalam folder `*src*`. Fungsi penerapan *flood-fill* dalam kode program ini bernama `*fillDfsBfs*`, yang akan menerima *parameter grid*, *start*, *color*, serta algoritma yang hendak dipakai. Berikut ini adalah kutipan kode program yang digunakan:

```
def fillDfsBfs(grid, start, newColor=(150, 150, 0), algoritma="dfs", fileName=None):
    oldColor = grid[start[0]][start[1]]
    if oldColor == newColor:
        return grid
    stack = [start]
    removeIndex = -1 if algoritma == "dfs" else 0
    while len(stack) > 0:
        x, y = stack.pop(removeIndex)
        if grid[x][y] != oldColor:
            continue
        grid[x][y] = newColor
        if fileName:
            recordChanges(grid, fileName)
        if x > 0:
            stack.append((x - 1, y))
        if y > 0:
            stack.append((x, y - 1))
        if x < len(grid) - 1:
            stack.append((x + 1, y))
        if y < len(grid[0]) - 1:
            stack.append((x, y + 1))
        yield gridget_pos()
```

Gambar 3.2 *Ilustrasi Fungsi Flood-Fill*
Sumber : Dokumen Pribadi

Program tersebut nantinya akan menerima 4 parameter sesuai yang telah disampaikan sebelumnya. Pada implementasi tersebut, apabila algoritma yang dipilih pengguna adalah algoritma DFS, maka penjelajahan akan dilakukan menggunakan stack. Dengan demikian, elemen yang ditambahkan ke stack akan dihapus dari bagian paling belakang stack dengan perintah *pop(-1)*. Oleh karena itu, ketika algoritma DFS yang dipilih, maka variabel `*removeIndex*` (bertipe *integer*) akan di-set menjadi -1. Sedangkan jika algoritma yang digunakan adalah BFS, maka penjelajahan akan dilakukan menggunakan queue dan elemen yang ditambahkan akan dihapus dari bagian depan queue dengan perintah *pop(-0)*. Dengan kata lain, apabila algoritma yang dipilih adalah BFS, maka variabel `*removeIndex*` akan di-set menjadi 0.

Pada kode program tersebut, setiap titik yang dikunjungi selama penjelajahan warnanya akan diubah menjadi *newColor* dan tetangganya akan ditambahkan ke stack atau queue. Pada implementasi ini, setiap kali ada sel yang berhasil diisi dengan warna, maka fungsi *yield* akan dipanggil dan *grid* yang baru diisi warna akan dikembalikan sebagai *generator*. Proses pengisian sel dalam *grid* akan terus dilakukan hingga seluruh area yang terhubung dengan sel awal (*start*) telah diisi dengan warna. Proses pengecekan bahwa pengisian sel telah selesai diketahui berdasarkan panjang stack yang telah sama dengan 0.

C. *Implementasi Fungsi Perantara*

Pada implementasi program, terdapat 4 fungsi perantara, yaitu : *display_grid*, *postToGrid*, *coloring*, dan *recordChanges*. Fungsi *display_grid* digunakan untuk menampilkan kotak-kotak *grid* yang dibuat untuk menampilkan warna-warna pada layar ‘Aplikasi Mewarnai Sederhana’. Fungsi *display_grid* akan menerima 4 parameter, antara lain : *screen*, *grid*, *width*, serta *height*. Untuk setiap kotak dalam *grid* akan diset dengan warna putih sebagai warna awal. Sementara pembuatan *grid* akan menggunakan 2 *loop*, yaitu : *outer loop* dan *inner loop*. *Outer loop* ditandai dengan *iterator* `*i*` sepanjang *grid* yang telah ditentukan. Sementara, *inner loop* ditandai dengan *iterator* `*j*` sepanjang kolom *grid* yang telah ditentukan. Penentuan koordinat `*x*` dan `*y*` untuk setiap kotak *grid* menggunakan nilai `*width*` dan `*height*`. Koordinat ini diperoleh dengan mengalikan nilai `*i*` dan `*width*` untuk `*x*` dan mengalikan nilai `*j*` dan `*height*` untuk `*y*`. Setelah itu, fungsi mengambil warna (*color*) dari setiap sel *grid* `*cellColor*` dan menggunakan perintah `*pygame.draw.rect*` untuk menggambar persegi panjang dengan warna tersebut pada layar `*screen*`. Berikut adalah kutipan implementasi dari fungsi `*display_grid*`:

```
def display_grid(screen, grid, width=2, height=2) -> None:
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            x, y = i * width, j * height
            cellColor = grid[i][j]
            pygame.draw.rect(
                screen,
                cellColor,
                pygame.Rect(x, y, width, height),
            )
```

Gambar 3.3 *Ilustrasi Fungsi display_grid*
Sumber : Dokumen Pribadi

Fungsi perantara yang kedua adalah fungsi `*postToGrid*`, bertujuan untuk mengubah posisi *mouse* dalam bentuk piksel menjadi posisi baris dan kolom dalam *grid*. Variabel `*cellWidth*` dan `*cellHeight*` merupakan ukuran satu sel dalam grid, sedangkan variabel `*rows*` dan `*cols*` merupakan jumlah baris dan kolom dalam *grid*. Berikut ini adalah kutipan implementasi dari fungsi `*postTogrid*`:

```
def postToGrid(pos):
    row, col = pos[0] // cellWidth, pos[1] // cellHeight
    if row >= rows or col >= cols:
        return None, None
    return row, col
```

Gambar 3.4 *Ilustrasi Fungsi postToGrid*
Sumber : Dokumen Pribadi

Langkah kerja dari fungsi tersebut, antara lain:

1. Fungsi ini akan menghitung nilai `*row*` dan `*col*` dengan cara membagi posisi *mouse* dengan `*cellWidth*` dan `*cellHeight*` menggunakan operator `*//*`. Operator `*//*` digunakan untuk melakukan pembagian bilangan bulat sehingga nilai hasil bagi selalu merupakan bilangan bulat.
2. Fungsi ini kemudian melakukan pengecekan apakah nilai `*row*` dan `*col*` yang diperoleh sudah melebihi jumlah baris atau kolom dalam *grid*. Apabila terpenuhi, maka fungsi akan mengembalikan nilai `*None*` dan `*None*` sebagai *output*.
3. Apabila nilai `*row*` dan `*col*` masih dalam batas *grid*, maka fungsi mengembalikan nilai `*row*` dan `*col*` dalam bentuk *tuple* sebagai *output*. Kedua nilai ini kemudian dapat digunakan dalam fungsi lain untuk melakukan operasi pada sel yang dipilih oleh *mouse*.

Selain itu, terdapat fungsi perantara yang ketiga, yaitu : `*coloring*`. Kode program dalam fungsi `*coloring*` akan menerima 2 parameter, yaitu `*pos*` dan `*color*`. Parameter `*pos*` berisi koordinat posisi dalam bentuk *tuple* (x,y) yang merupakan posisi di dalam grid dan diwarnai dengan `color`. Proses pertama kode ini akan sama dengan tahapan pada fungsi `*postToGrid*` (menghitung nilai `*row*` dan `*col*`). Langkah fungsi ini untuk mewarnai kotak dalam *grid* antara lain:

1. Fungsi akan mengawali dengan melakukan pengecekan apakah nilai `*row*` dan `*col*` yang diperoleh sudah melebihi jumlah baris atau kolom dalam *grid*. Apabila terpenuhi, maka fungsi akan langsung keluar dengan menggunakan perintah `*return*` tanpa melakukan apapun.
2. Apabila nilai `*row*` dan `*col*` masih dalam batas gris, maka fungsi ini mengubah warna sel pada posisi yang diberikan dengan warna `*color*`. Hal ini dilakukan dengan mengakses elemen `*grid[row][color]*` dan mengisinya dengan nilai `*color*`.

Berikut ini adalah kutipan dari implementasi fungsi `*coloring*`:

```
def coloring(pos, color):
    row, col = pos[0] // cellWidth, pos[1] // cellHeight
    if row >= rows or col >= cols:
        return
    grid[row][col] = color
```

Gambar 3.5 *Ilustrasi Fungsi coloring*
Sumber : Dokumen Pribadi

Sementara fungsi perantara terakhir, `*recordChanges*` merupakan fungsi yang digunakan untuk mencatat tiap perubahan warna yang terjadi di dalam *grid*. Perubahan warna akan selalu dicatat tiap ada perubahan dalam fungsi `*fillDfsBfs*`. Berikut ini adalah kutipan dari implementasi fungsi `*recordChanges*`:

```
def recordChanges(grid, fileName):
    with open(fileName, 'w') as f:
        for row in grid:
            for cell in row:
                f.write(f"{cell[0]} {cell[1]} {cell[2]} ")
```

Gambar 3.6 *Ilustrasi Fungsi recordChanges*
Sumber : Dokumen Pribadi

D. Pustaka PyGame

Pada makalah ini, visualiasi serta manipulasi hasil implementasi program diterapkan menggunakan *library PyGame* (versi 2.4.0) yang tersedia pada bahasa pemrograman *Python*. Pustaka *PyGame* merupakan pustaka *Python* yang seringkali digunakan untuk mengembangkan *game* 2-D dan merupakan sebuah *platform* yang dapat dimanfaatkan untuk menggunakan satu set modul *Python* dalam pembuatan game. Beberapa fungsi yang digunakan pada kode program, antara lain:

1. *pygame.init()*: menginisialisasi *library Pygame*.
2. *pygame.display.set_mode()*: membuat *window game* dengan ukuran dan mode yang telah ditentukan.
3. *pygame.display.set_caption()*: mengatur judul *window*.
4. *pygame.image.load()*: memuat gambar pada *window*.
5. *pygame.time.Clock()*: mengatur waktu *frame*.
6. *pygame.font.SysFont()*: mengatur font teks.
7. *pygame.draw.rect()*: membuat persegi pada layar.
8. *pygame.gfxdraw*: membuat grafik pada layar.
9. *pygame.locals*: memperoleh konstanta *Pygame*.
10. *pygame.constants*: memperoleh konstanta *Pygame*.

E. Pustaka Eksternal Lain

Implementasi program melakukan impor 3 pustaka eksternal, antara lain: `*sys*`, `*os*`, serta `*random*`. Pustaka `*sys*` digunakan untuk memanipulasi *command-line arguments* yang diberikan saat menjalankan program. Selain itu, terdapat pustaka `*random*` yang digunakan untuk menghasilkan angka acak dengan menggunakan pemanggilan fungsi `*randint*`. Pustaka terakhir adalah `*os*`, yang digunakan untuk mengakses sistem operasi dan melakukan operasi seperti membaca/menulis file. Pada implementasi program, terdapat pemanggilan fungsi `*os.path.isFile*` yang digunakan untuk memeriksa apakah file dengan nama tersebut ada atau tidak. Selain itu, terdapat pula pemanggilan fungsi `*os.rename*` yang digunakan untuk mengganti nama file dengan nama baru yang diberikan.

F. Langkah Kerja Hasil Implementasi Program

Program memiliki beberapa fitur yang dapat digunakan oleh pengguna, antara lain:

1. Fitur untuk mengganti algoritma yang akan digunakan.
2. Fitur untuk mengganti warna dengan warna tertentu hasil *generate random*.
3. Fitur untuk menggambar pada *canvas* yang disediakan.
4. Fitur untuk menghapus hasil gambar di *canvas*.
5. Fitur untuk mewarnai suatu gambar yang ber-boundaries pada *canvas*.

Fitur mengganti algoritma dapat dilakukan dengan menekan tombol `*c*` pada *keyboard* (algoritma yang tersedia adalah BFS dan DFS). Fitur untuk mengganti warna dapat dilakukan dengan `*left click*` pada *icon* persegi panjang berwarna di sebelah tulisan `Pilih Warna:`. Fitur untuk menggambar pada *canvas* dapat dilakukan dengan menekan `*left click*` pada *canvas* yang disediakan. Sementara itu, fitur untuk menghapus hasil gambar di *canvas* dapat dilakukan dengan `*right click*` pada *canvas*. Serta fitur yang terakhir, mewarnai suatu gambar pada *canvas* dapat dilakukan dengan menekan tombol `*f*` pada keyboard.

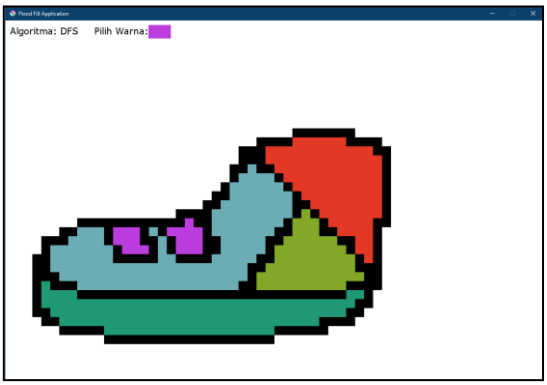
IV. Pengolahan Data Dan Uji Coba

A. Ujicoba Gambar Pada Canvas

Pada Makalah ini, proses pengujian akan dilakukan sebanyak 3 kali. Variasi percobaan yang dilakukan adalah terkait jumlah warna yang digunakan dalam *canvas*, termasuk

warna dasar *canvas* (putih). Berikut ini adalah hasil percobaan yang dilakukan:

Gambar 4.1 *Ilustrasi Hasil Gambar 4 Warna*
Sumber : Dokumen Pribadi



Gambar 4.2 *Ilustrasi Hasil Gambar 7 Warna*
Sumber : Dokumen Pribadi

Gambar 4.3 *Ilustrasi Hasil Gambar 10 Warna*
Sumber : Dokumen Pribadi

B. *Pengolahan Data*

Dari 3 Gambar hasil ujicoba di atas, telah dicatat jumlah *cell* untuk tiap *grid* yang menandakan frekuensi kemunculan warna. Hal ini akan mengindikasikan bahwa proses *flood-fill* berhasil dilakukan, karena proses pewarnaan dilakukan hanya dalam *boundaries* yang ditentukan (dalam hal ini warna yang berbeda dengan *adjacent cell color*). Berikut ini adalah tabel hasil 3 percobaan di atas :

Tabel 4.1 *Data Gambar Pertama*
Sumber : Dokumen Pribadi

No	Warna	Jumlah Cell
1	(255, 255, 255)	1520
2	(0, 0, 0)	210
3	(145, 122, 24)	545
4	(71, 163, 20)	125
Total		2400

Tabel 4.2 *Data Gambar Kedua*
Sumber : Dokumen Pribadi

No	Warna	Jumlah Cell
1	(255, 255, 255)	1723
2	(0, 0, 0)	218
3	(31, 153, 117)	123
4	(108, 172, 181)	152
5	(188, 61, 221)	20
6	(130, 168, 40)	62
7	(226, 56, 37)	102
Total		2400

Tabel 4.3 *Data Gambar Ketiga*
Sumber : Dokumen Pribadi

No	Warna	Jumlah Cell
1	(52, 69, 198)	310
2	(0, 0, 0)	440
3	(124, 177, 203)	444
4	(8, 185, 35)	465
5	(149, 162, 141)	585
6	(196, 166, 27)	19
7	(42, 22, 31)	55
8	(215, 152, 83)	67
9	(166, 74, 155)	11
10	(252, 4, 14)	4
Total		2400

C. *Visualisasi Data*

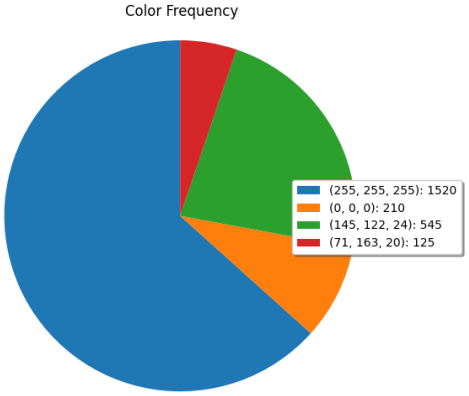
Pada program kali ini, visualisasi pengolahan data akan dilakukan dengan menggunakan pustaka `*matplotlib*` pada `*visualization.py*`. Berikut ini adalah kode visualisasi yang digunakan:

```python
import matplotlib.pyplot as plt
from collections import Counter

# Membaca file txt dan mengubahnya menjadi list of tuples
with open('changes.txt', 'r') as f:
    colors_str = f.read().split()
colors = [(int(colors_str[i]), int(colors_str[i+1]), int(colors_str[i+2])) for i in range(0, len(colors_str), 3)]

# Menghitung frekuensi kemunculan warna
color_counts = Counter(colors)

# Memvisualisasikan frekuensi kemunculan warna dalam pie chart
fig, ax = plt.subplots(figsize=(8, 6))
ax.pie(list(color_counts.values()), startangle=90)
ax.axis('equal')
ax.set_title('Color Frequency')
labels = []
for c in color_counts:
    labels.append(f'({c[0]}, {c[1]}, {c[2]}): {color_counts[c]}')
ax.legend(labels, loc='center left', bbox_to_anchor=(0.7, 0.5), fancybox=True, shadow=True)
plt.show()
```

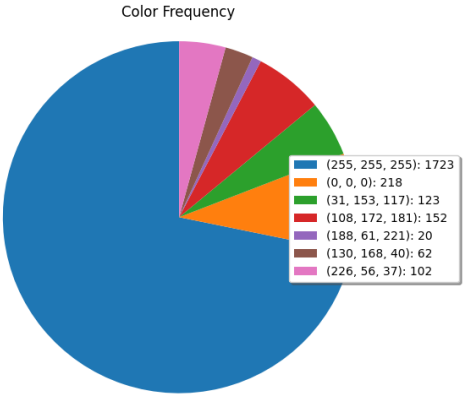
Gambar 4.4 *Ilustrasi Fungsi Visualisasi Data*
Sumber : Dokumen Pribadi

Kode tersebut akan memulai dengan membaca hasil file `*changes.txt*` yang mencatat warna-warna pada *grid*. Kemudian akan dihasilkan sebuah *pie chart* yang berisi frekuensi

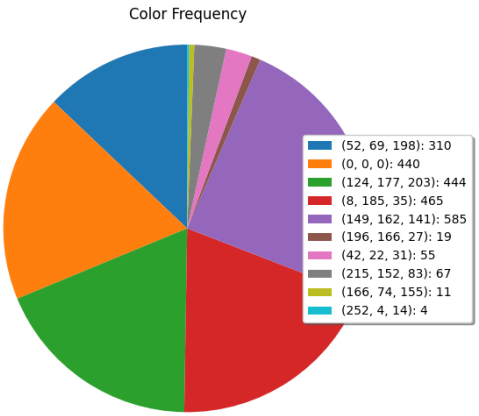
kemunculan untuk tiap warna. Selain itu, akan dimunculkan pula *index* yang berisi kode warna dan frekuensi kemunculan untuk tiap warnanya. Berikut ini adalah hasil visualisasi dari 3 percobaan tersebut:



Gambar 4.5 *Ilustrasi Visualisasi Gambar Pertama*
Sumber : Dokumen Pribadi



Gambar 4.6 *Ilustrasi Visualisasi Gambar Kedua*
Sumber : Dokumen Pribadi



Gambar 4.7 *Ilustrasi Visualisasi Gambar Ketiga*
Sumber : Dokumen Pribadi

V. KESIMPULAN

Flood-Fill Algorithm merupakan salah satu metode yang seringkali digunakan dalam proses-proses terkait pengolahan citra gambar. Penerapan *flood-fill* seringkali memanfaatkan algoritma *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS) dengan menggunakan prinsip penelusuran graf traversal secara rekursif. BFS memfokuskan penelusuran graf secara melebar disertai implementasi menggunakan struktur data Queue, sedangkan DFS memfokuskan penelusuran graf secara mendalam disertai implementasi menggunakan struktur data Stack. Algoritma *flood-fill* dapat diimplementasikan untuk membuat sebuah "Aplikasi Mewarnai Sederhana" seperti yang dilakukan pada makalah ini. Penerapan konsep-konsep memanfaatkan beberapa pustaka untuk dapat mencapai tujuan makalah ini, antara lain : *PyGame*, *random*, *os*, dan lain sebagainya. "Aplikasi Mewarnai Sederhana" berhasil diimplementasikan dalam makalah ini, dibuktikan dengan data pada hasil ujicoba dan visualisasi data pada Bab 4. Proses *flood-fill* dengan algoritma BFS dan DFS pada makalah ini berhasil dibuat dengan *limitasi boundary* yang didasarkan pada perbedaan warna yang dimilikinya.

LINK VIDEO YOUTUBE

Link video youtube yang berisi penjelasan program, cara penggunaan program, serta pembahasan makalah dapat diakses melalui pranala berikut ini.

UCAPAN TERIMA KASIH

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas segala rahmat dan kasih karunia-Nya yang telah memberikan kesehatan dan kesempatan kepada penulis sehingga penulis dapat menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih sebesar-besarnya kepada seluruh pihak yang telah membantu penulis dalam menyelesaikan makalah ini, antara lain:

1. Dr. Nur Ulfa Maulidevi, S.T, M.Sc. sebagai dosen pengampu dalam mata kuliah IF2211 Strategi Algoritma K02.
2. Dr. Ir. Rinaldi Munir, M.T. atas kontribusi buku dan materi yang penulis kutip pada makalah ini.
3. Seluruh pihak yang telah membuat source code dan modul secara *open-source* serta penulis kutip pada makalah ini.

REFERENSI

[1] Yixuan He, Tianyi Hu, dan Delu Zeng. 2019. *Scan-flood Fill (SCAFF) : an Efficient Automatic Precise Region Filling Algorithm for Complicated Regions*. https://openaccess.thecvf.com/content_CVPRW_2019/papers/CEFRL/He_Scan-Flood_FillSCAFF_An_Efficient_Automatic_Precise_Region_Filling_Algorithm_for_CVPRW_2019_paper.pdf, diakses pada 12 Mei 2023.

[2] Bhawnesh Kumar, Umesh Kumar Tiwari, Santosh Kumar, Vikas Tomer, dan Jasmeet Kalra. 2019. *Comparison and Performance Evaluation of Boundary Fill and Flood Fill Algorithm*. https://www.researchgate.net/profile/Jasmeet-Kalra/publication/343809379_Comparison_and_Performance_Evaluation_of/links/5f40c2b4299bf13404df8c2a/Comparison-and-Performance-Evaluation-of.pdf, diakses pada 12 Mei 2023.

[3] George Law. 2013. *Quantitative Comparison of Flood Fill and Modified Flood Fill Algorithms*. https://marsuniversity.github.io/ece387/FloodFill.pdf, diakses pada 12 Mei 2023.

[4] Phuong Minh Chu, Seoungjae Cho, Yong Woon Park, dan Kyungeun Cho. 2017. *Fast Point Cloud Segmentation Based on Flood-Fill Algorithm*. https://ieeexplore.ieee.org/abstract/document/8170397, diakses pada 12 Mei 2023.

[5] R. Khudeev. 2005. *A New Flood-Fill Algorithm for Closed Contour*. https://ieeexplore.ieee.org/abstract/document/1611214, diakses pada 12 Mei 2023.

[6] Eva-Marie Nosal. 2008. *Flood-Fill Algorithms Used for Passive Acoustic Detection and Tracking*. https://ieeexplore.ieee.org/abstract/document/4786975/authors#authors, diakses pada 12 Mei 2023.

[7] Semuil Tjiharjadi dan Erwin Setiawan. 2016. *Design and Implementation of Path Finding Robot Using Flood-Fill Algorithm*. https://www.researchgate.net/publication/310816644_Design_and_Implementation_of_a_Path_Finding_Robot_Using_Flood_Fill_Algorithm, diakses pada 12 Mei 2023.

[8] Semuil Tjiharjadi. 2019. *Design and Implementation of Flood Fill and Pledge Algorithm for Maze Robot*. https://www.researchgate.net/publication/340579894_Design_and_Implementation_of_Flood_Fill_and_Pledge_Algorithm_for_Maze_Robot, diakses pada 12 Mei 2023.

[9] Jukka Arvo, Mika Hirvikorpi, and Joonas Tyystjarvi. 2004. *Approximate Soft Shadows win an Image-Space Flood-Fill Algorithm*. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2004.00758.x, diakses pada 12 Mei 2023.

[10] FreeCodeCamp. 2020. *Flood-Fill Algorithm Explained*. https://www.freecodecamp.org/news/flood-fill-algorithm-explained/, diakses pada 12 Mei 2023.

[11] Rinaldi Munir. 2023. *Diktat Algoritma BFS dan DFS*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf, diakses pada 12 Mei 2023.

[12] AIForEveryOne. 2020. *What is the best way to traverse graph.*. https://www.aiforanyone.org/glossary/graph-traversal, diakses pada 12 Mei 2023.

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 2023

Mohammad Rifqi Farhansyah 13521166