# LAPORAN TUGAS BESAR
# IF2211
# STRATEGI ALGORITMA


**Disusun oleh:**
**13519194 Rezda Abdullah Fachrezzi**
**13519196 Rayhan Asadel**
**13519221 Allief Nuriman**

**PROGRAM STUDI TEKNIK INFORMATIKA**
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**
**INSTITUT TEKNOLOGI BANDUNG**
**2021**

# Daftar Isi

# BAB I

# DESKRIPSI TUGAS

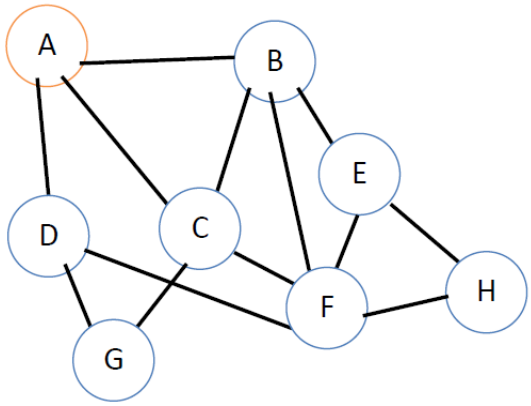
Dalam tugas besar ini, penulis diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan beberapa fitur dari People You May Know dalam jejaring sosial media (Social Network). Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), penulis dapat menelusuri social network pada akun facebook untuk mendapatkan rekomendasi teman seperti pada fitur People You May Know. Selain untuk mendapatkan rekomendasi teman, Anda juga diminta untuk mengembangkan fitur lain agar dua akun yang belum berteman dan tidak memiliki mutual friends sama sekali bisa berkenalan melalui jalur tertentu.

**Contoh Input dan Output Program**
Contoh berkas file eksternal:

```
13
A B
A C
A D
B C
B E
B F
C F
C G
D G
D F
E H
E F
F H
```

Gambar 1.1. : Contoh input berkas file eksternal

Visualisasi graf pertemanan yang dihasilkan dari file eksternal:




*Gambar 1.2. Contoh visualisasi graf pertemanan dari file eksternal*

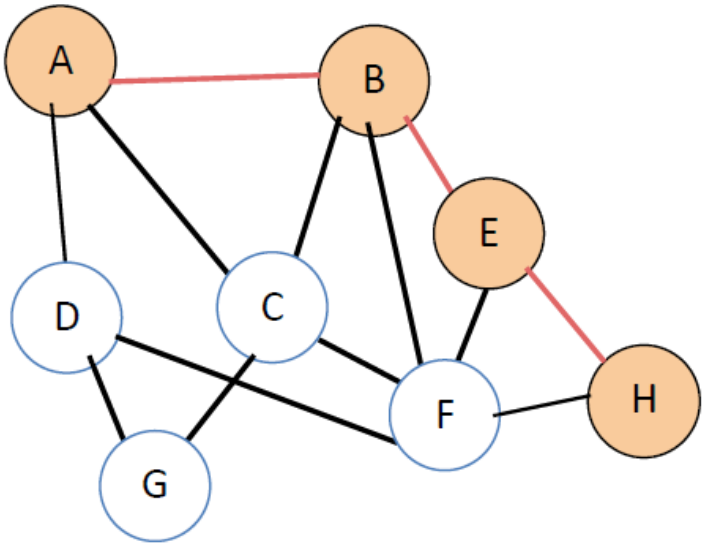
Untuk fitur friend recommendation, misalnya pengguna ingin mengetahui daftar rekomendasi teman untuk akun A. Maka output yang diharapkan sebagai berikut.

```
Daftar rekomendasi teman untuk akun A:
Nama akun: F
3 mutual friends:
B
C
D

Nama akun: G
2 mutual friends:
C
D

Nama akun: E
1 mutual friend:
B
```

Untuk fitur explore friends, misalnya pengguna ingin mengetahui seberapa jauh jarak antara akun A dan H serta bagaimana jalur agar kedua akun bisa terhubung. Berikut output graf dengan penelusuran BFS yang dihasilkan.

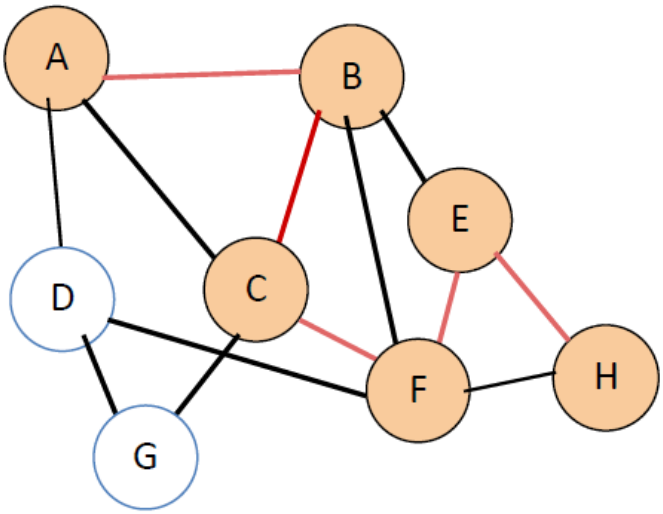
```
Nama akun: A dan H
2nd-degree connection
A → B → E → H
```

Perhatikan busur antara akun A dan H, terbentuk salah satu jalur koneksi sebagai berikut: A-BE-H (ada beberapa jalur lainnya, seperti A-D-F-H, dll, urutan simpul untuk ekspan diprioritaskan berdasarkan abjad). Akun A dan H tidak memiliki mutual friend, tetapi kedua akun merupakan 2nd-degree connection karena di antara A dan H ada akun B dan E yang saling berteman. Sehingga akun H dapat terhubung sebagai teman dengan jalur melalui akun B dan akun E. Jalur koneksi dari A ke H menggunakan BFS digambarkan dalam bentuk graf sebagai berikut.




Gambar 1.3. : Hasil visualisasi jalur koneksi menggunakan BFS

Sedangkan untuk penggunaan algoritma DFS, diperoleh jalur lainnya, yaitu A-B-C-F-E-H yang digambarkan dalam bentuk graf sebagai berikut




Gambar 1.4. : Hasil visualisasi jalur koneksi menggunakan DFS

Pada fitur explore friends, apabila terdapat dua buah akun yang tidak bisa saling terhubung (tidak ada jalur koneksi), maka akan ditampilkan bahwa akun tersebut tidak bisa terhubung melalui jalur koneksi

yang sudah dimilikinya sekarang sehingga orang tersebut memang benar-benar harus memulai koneksi baru dengan orang tersebut. Misalnya terdapat dua orang baru, yaitu J dan I yang hanya terhubung antara J-I. Maka jalur koneksi yang dibentuk dari A ke J adalah.

```
Nama akun: A dan J
Tidak ada jalur koneksi yang tersedia
Anda harus memulai koneksi baru itu sendiri.
```

# BAB II

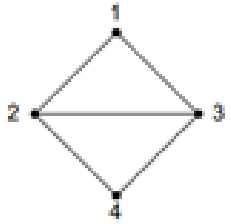
# LANDASAN TEORI

## 2.1. Graf

Graf merupakan salah satu representasi data yang dapat dipakai pada pemrograman. Graf digunakan untuk mempresentasikan objek-objek dan hubungan antar objek-objek tersebut. Graf dapat didefinisikan sebagai berikut.

$$G = (V,E)$$

Dari definisi tersebut, G merupakan Graf, V merupakan himpunan tidak kosong dari simpul-simpul, dan E merupakan himpunan sisi yang menghubungkan simpul-simpul.




Gambar 2.1.1. Contoh graf sederhana.

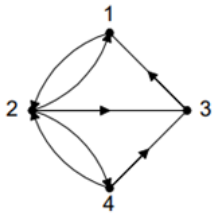
Contoh graf pada gambar 2.1.1 memiliki simpul V = {1,2,3,4} dan sisi E = {(1,3),(3,4),(4,2),(2,1),(2,3)}. Graf dapat dibagi menjadi dua berdasarkan arahnya.

1. Graf tak berarah

   Graf yang tidak memiliki arah pada sisinya (Gambar 2.1.1).

2. Graf berarah

   Graf yang memiliki arah pada sisinya.




Gambar 2.1.2. Graf berarah.

Graf tersebut dapat ditelusuri untuk dicari simpul target dari suatu simpul awal dengan suatu metode atau algoritma. Algoritma yang dapat digunakan untuk mencari suatu simpul pada graf salah satunya menggunakan metode traversal. Metode yang digunakan ini dibagi menjadi dua, yaitu algoritma Breadth-first Search dan Depth-first Search.

## 2.2. Algoritma Breadth-first Search (BFS)

Algoritma BFS pada dasarnya mencari tetangga dari suatu node (Sebutlah sebagai node "Start"), kemudian memasukkan tetangga dari Start ke suatu Queue (Setelah semua tetangga dimasukkan, Queue diurutkan secara *ascending*) dan menandai tiap tetangga dari Start bahwa node-nya sudah dikunjungi. Kemudian akan mengambil setiap elemen di Queue & mengulangi proses tersebut sampai Queue kosong.

## 2.3. Algoritma Depth-first Search (DFS)

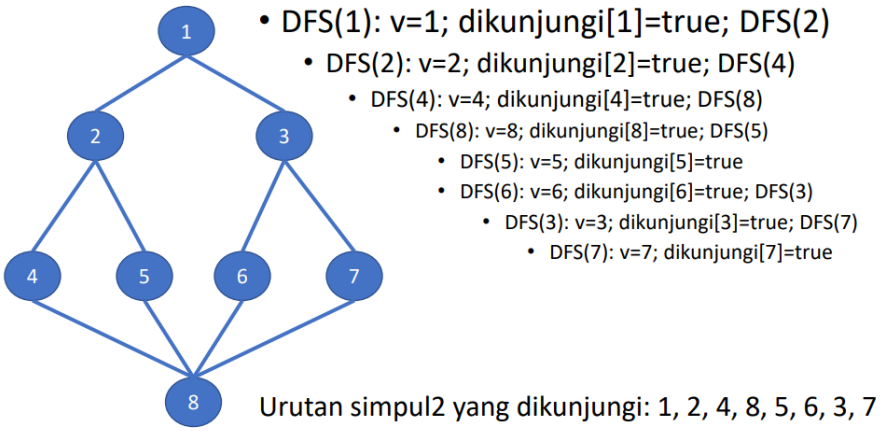
Algoritma Depth-first Search, kemudian disebut DFS, merupakan algoritma pencarian solusi dengan metode traversal. Algoritma ini biasa disebut juga sebagai pencarian secara mendalam. Solusi yang dimaksud adalah sebuah persoalan yang direpresentasikan dalam bentuk Graf. Pencarian dengan algoritma DFS ini dapat digunakan untuk dua jenis graf, yaitu graf statis dan graf dinamis.

a. Pencarian Graf Statis

```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS

Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang dikunjungi ditulis ke layar
}
Deklarasi
   w : integer

Algoritma:
   write(v)
   dikunjungi[v]←true
   for w←1 to n do
      if A[v,w]=1 then {simpul v dan simpul w bertetangga }
          if not dikunjungi[w] then
             DFS(w)
          endif
      endif
   endfor
```

Gambar 2.2.1. *Pseudocode* Algoritma DFS Graf Statis

Pencarian graf statis dengan algoritma DFS ini akan menelusuri simpul-simpul secara mendalam. Maksud dari pencarian secara mendalam adalah pencarian dengan algoritma ini akan menelusuri suatu simpul dan anak-anaknya terlebih dahulu sampai menemukan titik buntu, lalu setelahnya akan dilakukan *backtracking* untuk mencari simpul lainnya yang belum dikunjungi dan akan diulang proses yang sama. Berikut adalah contoh dari pencarian graf secara DFS.

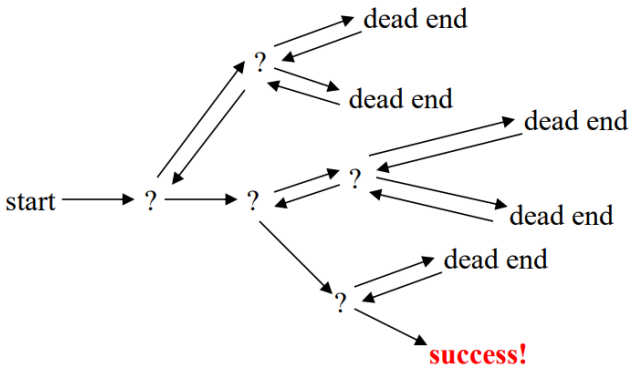



Gambar 2.2.2. Contoh pencarian dengan Algoritma DFS

b. Pencarian Graf Dinamis

Algoritma DFS untuk pencarian graf dinamis sama seperti pencarian graf statis, tetapi perbedaannya dengan graf statis adalah pencarian graf dinamis akan membangkitkan simpul-simpul graf seiring dengan pencariannya. Graf dinamis ini memiliki beberapa komponen, yaitu pohon ruang status yang merepresentasikan grafnya, simpul graf yang merepresentasikan masalah dengan akar adalah initial state dan daun adalah solution / goal state, cabang adalah langkah pencarian, ruang status adalah himpunan simpul dan ruang solusi adalah himpunan solusi. Solusinya sendiri merupakan sebuah jalur dari akar ke daun. Algoritma ini juga memanfaatkan *backtracking* dalam aplikasinya.

Algoritma DFS untuk pencarian graf dinamis ini banyak dipakai untuk mencari solusi persoalan pada permainan, seperti *tic-tac-toe*, *maze problem,* catur, sudoku, dan banyak lainnya.




20 *) Sumber: *www.cis.upenn.edu/.../35-**backtracking**.ppt*

Gambar 2.2.3. Ilustrasi Algoritma DFS pada graf dinamis.

## 2.4. Graphical User Interface (GUI)

Sebuah GUI atau kependekan dari *Graphical User Interface*, adalah sebuah antarmuka yang menampilkan komponen visual pada sebuah program komputer. Sebuah GUI menampilkan berbagai objek yang mendukung proses fungsionalitas dan pertukaran informasi antara user dan program. Pembuatan GUI pada proyek kali ini menggunakan bahasa pemrograman C# beserta kakas .Net Framework.

Dari GUI yang kami buat, kami menggunakan beberapa objek GUI untuk menunjang fungsionalitas program kami. Berikut adalah penjelasan dari objek-objek yang kami gunakan.

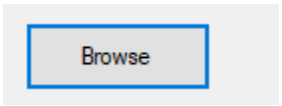
1. Label

   Objek label, sesuai namanya berfungsi untuk memberi label dan menuliskan string statis untuk ditampilkan.


Gambar 2.3.1 : Contoh Label

2. Button

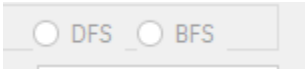
   Objek ini berfungsi sebagai tombol yang menghubungkan suatu aksi jika tombol tersebut diklik oleh pengguna. Program ini menggunakan 3 button, yaitu button untuk browse files, submit input, dan reset pilihan.




Gambar 2.3.2 : Contoh Button

3. Radio Button

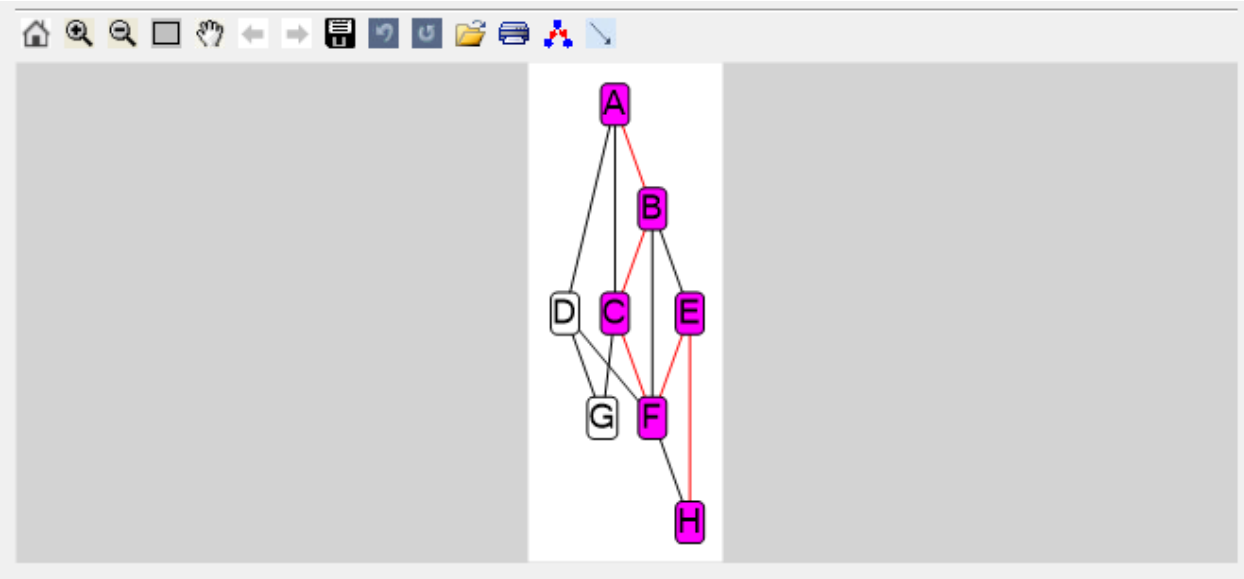
   Objek ini memberikan opsi pilihan majemuk untuk item tertentu. Pada program ini radio button digunakan untuk pemilihan metode algoritma yang digunakan, yaitu antara DFS dan BFS.




Gambar 2.3.2 : Contoh Radio Button

4. Panel

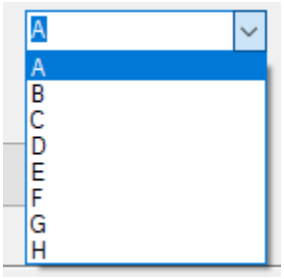
   Objek panel merupakan objek kontrol, yang dapat menampilkan sekelompok objek didalamnya. Pada program ini panel digunakan untuk menampilkan hasil visualisasi graph dengan kakas MSAGL.




Gambar 2.3.3 : Contoh Panel yang menampilkan visualisasi graph

5. Combo Box

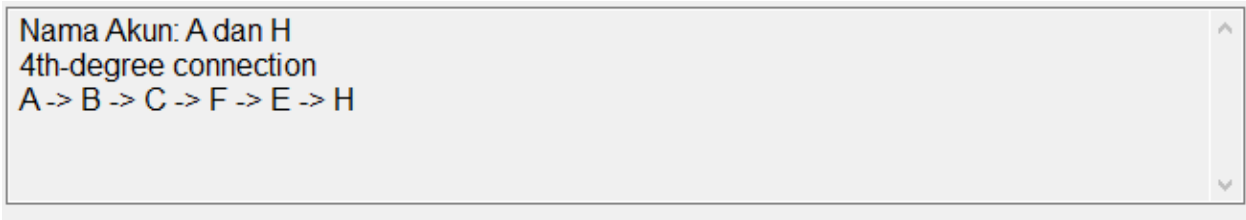
   Merupakan objek kombinasi dari text box dan list box. Pada combo box user dapat memilih berbagai macam pilihan yang tersedia. Opsi pilihan dapat ditampilkan dengan menggunakan menu *dropdown*. Pada program ini combo box digunakan untuk opsi pemilihan simpul akar dan simpul tujuan.




Gambar 2.3.4 : Contoh Combo Box pemilihan simpul

6. Textbox

Merupakan suatu objek yang digunakan untuk menampilkan text, baik text statis yang sudah ditentukan sebelumnya, atau text hasil komputasi yg akan ditampilkan setelah komputasi terjadi. Pada program ini text box digunakan untuk menuliskan hasil Friend Recommendation dan Explore Friends.




Gambar 2.3.5 : Contoh Text box yang menampilkan hasil komputasi

# BAB III

# ANALISIS PEMECAHAN MASALAH

## 3.1. Langkah-langkah Pemecahan Masalah

Pada tugas besar kali ini, sebelum dilakukan pengerjaan, penulis melakukan dekomposisi masalah terlebih dahulu. Masalah-masalah didekomposisi berdasarkan fungsionalitasnya, yaitu sebagai berikut.

1. Penentuan struktur data graf yang akan digunakan.
2. Penerapan algoritma (BFS dan DFS).
3. Aplikasi *Graphical User Interface* (GUI) serta komponen-komponen yang terkait.

Setelah itu, program diimplementasikan, implementasi dari program adalah sebuah aplikasi berbasis *user-interface*, program ini akan dijalankan lalu menunggu pengguna mengunggah berkas yang berisi sesuai deskripsi tugas. Kemudian, file tersebut akan di*parsing* ke struktur data matriks yang merepresentasikan suatu graf. Pengguna dapat memilih fitur yang ada, dan melihat hasil setelah eksekusi fitur.

## 3.2. Mapping Persoalan Menjadi Elemen Algoritma BFS dan DFS

Pada persoalan, graf pertemanan yang dimaksud merupakan sebuah graf tanpa arah. Sehingga struktur data yang akan digunakan adalah berupa matriks berukuran NxN dengan N merupakan jumlah simpul pada graf persoalan. Matriks ini mewakili ketetanggan setiap simpul dengan angka 1 yang berarti kedua simpul bertetangga dan 0 berarti kedua simpul tidak bertetangga.

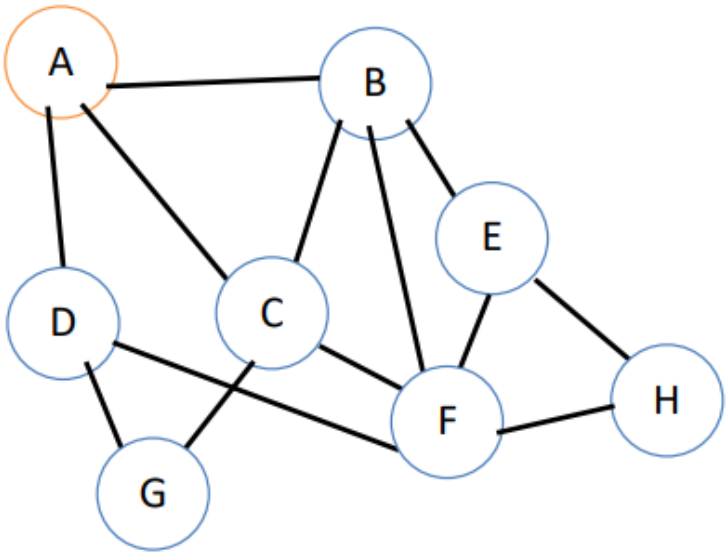

Gambar 3.2.1. Contoh matriks ketetanggaan

Dari gambar 3.2.1, masing-masing baris dan kolom mewakili nomor urut dari suatu simpul. Sebagai contoh, pada baris pertama dengan kolom kedua bernilai 1, berarti simpul dengan nomor urut 1 dan 2 saling bertetangga.

Selain mendefinisikan matriks ketetanggaan, didefinisikan pula sebuah larik atau *list of string* yang berisi nama-nama dari simpul. Masukan pengguna nantinya akan menghasilkan matriks

ketetanggaan juga sebuah *list of string* yang sesuai. Sebagai contoh, untuk matriks tersebut, didefinisikan lariknya adalah Simpul = { A, B, C, D, E, F, G, H } dengan urutan simpul pada larik menandakan nomor indeks pada matriks ketetanggaan.

Matriks ketetanggaan dan larik simpul tersebut akan menghasilkan graf pada gambar 3.2.2.




Gambar 3.2.2. Graf yang dihasilkan dari matriks pada gambar 3.2.1.

Karena graf sudah disediakan sebelum program dijalankan, maka akan digunakan metode pencarian graf statis dengan Algoritma BFS dan DFS.

### 3.2.1. Elemen Algoritma BFS

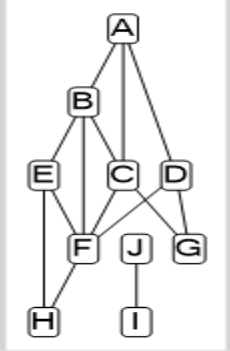
Algoritma BFS akan mengambil graf dalam representasi matriks yang sudah dibuat, *list of string* yang berisi nama-nama dari simpul, kemudian algoritma BFS akan membentuk suatu larik bertipe *boolean*. Larik bertipe *boolean* ini digunakan untuk menandai apakah suatu elemen sudah dikunjungi. Larik ini berukuran sebanyak jumlah *node* (Jumlah elemen pada *list of string*). Kemudian, graf dalam representasi matriks dan *list of string* yang berisi nama-nama dari simpul akan digunakan pada algoritma BFS.

### 3.2.2. Elemen Algoritma DFS

Dari komponen graf yang sudah disebut, kemudian akan di-*map* ke aplikasi algoritma DFS. Algoritma DFS memerlukan suatu larik lain yang bernama *visited* yang berukuran N simpul. Larik ini menandakan suatu simpul dengan indeks *i* pada larik *visited* bahwa sudah dikunjungi atau belum. Isi dari larik ini adalah nilai 0/1 atau false/true yang merepresentasikan suatu simpul sudah dikunjungi atau belum (false/0 = belum, true/1 = sudah). Komponen-komponen tersebut, yaitu matriks ketetanggaan dan larik *visited*, kemudian digunakan pada algoritma DFS.

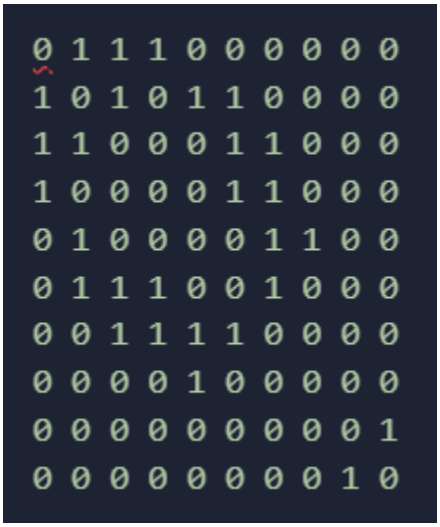
## 3.3. Ilustrasi Penyelesaian Masalah

Sebagai ilustrasi, akan digunakan graf berikut.




Gambar 3.3.1. Graf

Graf tersebut dibuat dari masukan pengguna berupa fail yang berisi daftar simpul yang terhubung. Kemudian diproses menjadi sebuah matriks ketetanggaan dan larik simpul-simpul. Larik simpulnya adalah {A,B,C,D,E,F,G,H,I,J} dengan matriks ketetanggaannya pada gambar 3.3.2.




Gambar 3.3.2. Matriks ketetanggaan pada bab 3.3.

Dapat dilihat bahwa graf sudah ditentukan dari awal. Sehingga graf yang digunakan untuk dicari solusinya nanti adalah sebuah graf statis. Setelah dihasilkan matriks ketetanggaan tersebut, barulah graf tersebut bisa dicari solusinya sesuai dengan persoalan yang sudah dideskripsikan.

Jika ingin mencari *friend recommendation* pada suatu simpul, maka akan digunakan sebuah algoritma BFS sederhana untuk mencari *mutual friend* dari suatu simpul dengan cara mencari simpul yang tidak berteman atau terhubung dengan simpul tadi dengan maksimal 1st-degree connection. Sebagai satu contoh, untuk simpul A akan memiliki salah satu simpul yang tak terhubung dengannya, yaitu simpul E, memiliki *mutual friend* adalah simpul B.

Jika ingin mencari eksplorasi teman dari suatu simpul ke simpul lain, maka diperlukan salah satu dari dua algoritma yang tersedia, yaitu DFS atau BFS. Hasil dari kedua algoritma ini adalah berupa jalur dari suatu simpul awal ke simpul yang dituju. Sebagai contoh, misal simpul A ingin mencari jalur pertemanan ke simpul C, maka nantinya algoritma DFS atau BFS akan

menghasilkan jalur yang sesuai dengan cara kerja algoritmanya masing-masing. Untuk algoritma DFS akan menghasilkan A->B->C sedangkan algoritma BFS akan menghasilkan A->C.

# BAB IV

# IMPLEMENTASI DAN PENGUJIAN

## 4.1. Implementasi Program

Berikut adalah pseudocode sederhana, yang menggambarkan bagaimana alur berjalan program.

**Psedocode Program Utama**

```
// KAMUS
// asumsi semua prosedur, variabel dan object sudah terdefinisi

/** Pseodocode Program Utama */

if (ButtonBrowse_Clicked==True) // ada input masukan file
{
   parseFile(); //Memparsing input file txt
   if (ChooseFeature==FriendsRecommendation){  //memilih fitur friendsrecommendation
      if (ButtonSubmit_Clicked==True){    //menekan tombol submit
         print(FriendRecommendation());  //menampilkan hasil friendrecommendation berdasarkan input
      }
      else if (ButtonReset_Clicked==True){
         ResetAllObjectOption();    //Mengosongkan semua nilai di objek yang sudah dipilih (combobox dan radio button)
      }

   }
   else { // ChooseFeature == ExploreFriends
      if (ButtonSubmit_Clicked==True){    //menekan tombol submit
         if (InputRadioButton="BFS"){    //Memilih BFS pada radio button
            DoBFS();                 // menjalankan algoritma BFS untuk mencari jalur explore friends
            GraphColoringBFS();        // memberi warna pada visualisasi graph sesuai dengan hasil BFS
            PrintBFSResult();          // menampilkan hasil explore friends pada textbox
         }
         else{ //InputRadioButton="DFS"
            DoDFS();                 // menjalankan algoritma DFS untuk mencari jalur explore friends
            GraphColoringDFS();        // memberi warna pada visualisasi graph sesuai dengan hasil DFS
            PrintDFSResult();          // menampilkan hasil explore friends pada textbox
         }
         }

      }
      else if (ButtonReset_Clicked==True){
         ResetAllObjectOption();        //Mengosongkan semua nilai di objek yang sudah dipilih (combobox dan radio button)
      }
   }
}
else // tidak ada input masukan file
{
   DisableAllCombobox();
   DisableAllRadioButton();
   DisableButtonSubmit();
}
```

**BFS**

```
bool BFS(string str, List<string> orang_dist, int[,] matrix_adj, string target, int[] pred, int[] dist)
{
   /* KAMUS
      idx_target, v, start

   // Cari dulu index dari target
   int idx_target ← orang_dist.IndexOf(target)
   int v ← orang_dist.Count

   // Array boolean, representasi yg udah divisit
   bool[] visited = new bool[v]

   start ← orang_dist.IndexOf(str)

   SetSeluruhVisitedKeFalse()
   SetSeluruhMatrixDistke9999()
   SetSeluruhMatrixPredke-1()

   for (int i = 0; i < visited.Length; i++)
      visited[i] = false
      dist[i] = 99999
      pred[i] = -1

   List<int> q = new List<int>();
   q.Add(start);

   // Source sudah dikunjungi
   visited[start] = true;
   dist[start] = 0;
   int[,] matrix_adj_new = new int[orang_dist.Count, orang_dist.Count];

   int vis;

   while (q.Count != 0)
   {
      vis = q[0];

      // cetak node sekarang
      Console.Write(orang_dist[vis] + " ");
      q.Remove(q[0]);

      // Untuk setiap tetangga nya node sekaranggit
      for (int i = 0; i < v; i++)
      {
         if (matrix_adj[vis, i] == 1 && (!visited[i]))
         {
            // Push tetangga ke node
            q.Add(i);
            q.Sort();

            // Set kalo telah dikunjungi
            visited[i] = true;
            dist[i] = dist[vis] + 1;
            pred[i] = vis;
            matrix_adj_new[vis, i] = 1;
            matrix_adj_new[i, vis] = 1;
            if (visited[idx_target])
```

```
                {
                    return true;
                }
            }
        }
    }
    return false;
}

void printBFS(string str, List<string> orang_dist, int[,] matrix_adj, string target)
{
    int[] pred = new int[orang_dist.Count];
    int[] dist = new int[orang_dist.Count];

    if (!BFS(str, orang_dist, matrix_adj, target, pred, dist)) then
        print_Bahwa_Str_target_tidak_terhubung

    else
    {

        List<int> path = new List<int>();
        int crawl = orang_dist.IndexOf(target);
        path.Add(crawl);
        while (pred[crawl] != -1)
        {
            path.Add(pred[crawl]);
            crawl = pred[crawl];
        }
        string stndrd = "";
        int deg = path.Count() - 2;
        outputBFS += deg + stndrd + "-degree connection\r\n";

        for (int i = path.Count - 1; i >= 0; i--)
        {
            Console.Write(orang_dist[path[i]] + " ");
            if (i == 0)
            {
                outputBFS += orang_dist[path[i]];
            }
            else
            {
                outputBFS += orang_dist[path[i]] + " -> ";
            }
        }
        this.textBox1.Text = outputBFS;

        int[,] matrix_adj_new = new int[orang_dist.Count, orang_dist.Count];
        for (int i = 0; i < path.Count - 1; i++)
        {
            matrix_adj_new[path[i], path[i + 1]] = 1;
            matrix_adj_new[path[i + 1], path[i]] = 1;
        }
        addColorFromMatrix(matrix_adj_new);
    }
}
```

**DFS**

{ fungsi pencarian traversal graf metode dfs }

```
{ idawal dan idakhir merupakan indeks simpul pada larik simpul }
dfs(idawal: integer, idtarget: integer): array of array of integer
  { mencari tetangga idawal }
  ttg = tetangga(idawal)


  { menandai simpul sudah dikunjungi }
  visited[idawal] = 1

  { mengecek apakah simpul awal dan target bersebelahan,
    apabila iya, dicek lagi tetangga dari simpul awal apakah memiliki
    simpul lain yang belum dikunjungi juga memiliki prioritas tertinggi, yaitu
    abjad yang lebih rendah. Jika iya, akan mengunjungi simpul tsb terlebih dahulu }

  { basis }
  if isadjacent(idawal, idtarget) then
    for i = 0 to jumlah(ttg)
      if (visited(ttg[i]) then
                    if ttg[i] == idtarget then { jika tetangga yg blum dikunjungi hanya target }
                                tambahJalur(node[idtarget])
                                tambahJalur(node[idawal])
                                return BuatMatriksSisi(idawal, idtarget) {merupakan fungsi untuk membuat matriks ketetanggaan tetapi
hanya 1 sisi}
                    else        { jika ada simpul lain yg sesuai komentar di atas }
                                otw = dfs(ttg[i], idtarget) { rekursi }
                                if otw not null then
                                            tambahJalur(node[idawal])
                                            return otw + BuatMatriksSisi(idawal,idtarget) { menambahkan sisi }
  { rekurens }
  else
          for i=0 to jumlah(ttg)
                    if visited(ttg[i]) then
                                otw = dfs(ttg[i], idtarget) { rekursi secara dfs }
                                if otw not null then
                                            tambahJalur(node[idawal])
                                            return otw + BuatMatriksSisi(idawal,idtarget) { menambahkan sisi }

  return null { jika tidak ada kondisi yang memenuhi }

printDFS(input: string account, target)
          { menuliskan hasil pencarian dengan dfs ke aplikasi
            serta mewarnai jalur dan warna simpul yang dilewati }

          { mencari id node }
          accidx = findIdxNode(account)
          targetidx = findIdxNode(target)

          result = dfs(accidx, targetidx) { proses pencarian dfs }

          if result not null then
                    output = makeOutput(result)
                    addColorFromMatrix(result)
          else
                    output = "Tidak ada jalur koneksi yang tersedia, Anda harus membuat koneksi itu sendiri"

          printToLayar(output)
```

## 4.2. Struktur Data dan Spesifikasi Program

#### 4.2.1. Struktur Data

Struktur data yang dipakai pada program menyatu dengan struktur data dan program dari GUI yang dipakai. Terdapat beberapa variabel global pada program, seperti graf, simpul graf, indikator visited berupa list, jalur solusi berupa list, dan graf untuk GUI. Struktur data graf untuk diproses yang dipakai yaitu berupa matriks ketetanggaan yang sudah dijelaskan pada bab 3. Sedangkan graf untuk GUI merupakan tipe data graf yang terdapat pada library MSAGL yang digunakan untuk menggambar graf pada GUI.

Metode-metode pada program terdapat dua bagian, yaitu metode untuk memproses interaksi dari GUI dan metode program itu sendiri, yaitu algoritma BFS dan DFS dan metode-metode pendukungnya.

#### 4.2.2. Spesifikasi

Program yang dibuat dalam tugas ini memiliki detail spesifikasi sebagai berikut.

a. Spesifikasi GUI.
    1. Program dapat menerima input berkas file eksternal dan menampilkan visualisasi graph.
    2. Program dapat memilih algoritma yang digunakan.
    3. Program dapat memilih akun pertama dan menampilkan friends recommendation untuk akun tersebut.
    4. Program dapat memilih akun kedua dan menampilkan jalur koneksi kedua akun dalam bentuk visualisasi graf dan teks bertuliskan jalur koneksi kedua akun.
    5. GUI dapat dibuat sekreatif mungkin asalkan memuat 4 spesifikasi di atas.

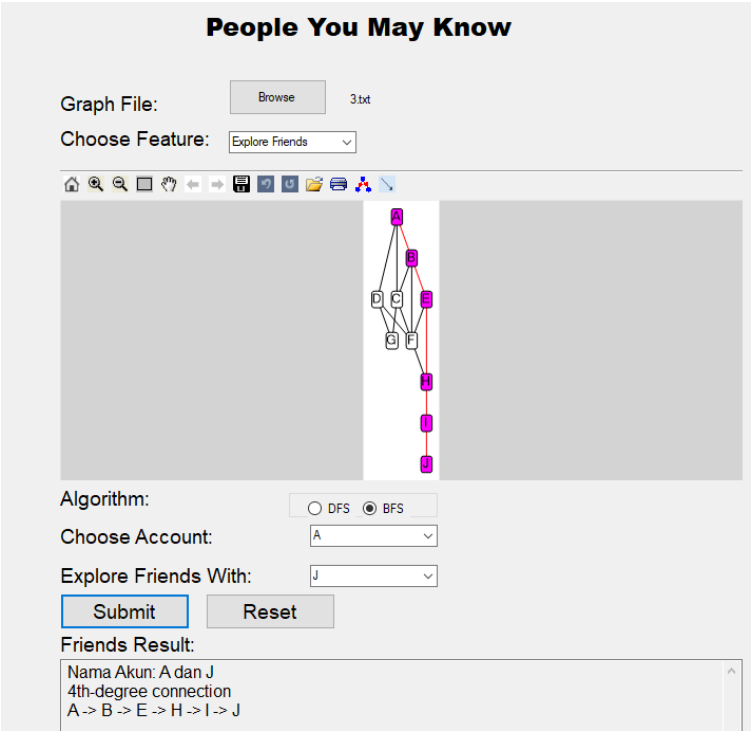
b. Spesifikasi Program
    1. Program dibuat dalam bahasa C# dan memanfaatkan algoritma DFS dan BFS
    2. Program menerima file eksternal dengan Baris pertama merupakan sebuah integer N yang adalah banyaknya pertemanan antar akun di facebook. Sebanyak N baris berikutnya berisi dua buah string (A, B) yang menunjukkan akun A dan B sudah berteman
    3. Program dapat menampilkan visualisasi graf pertemanan berdasarkan informasi dari file eksternal tersebut. Graf pertemanan ini merupakan graf tidak berarah dan tidak berbobot. Setiap akun facebook direpresentasikan sebagai sebuah node atau simpul pada graf. Jika dua akun berteman, maka kedua simpul pada graf akan dihubungkan dengan sebuah busur.

### 4.3. Tata Cara Penggunaan Program

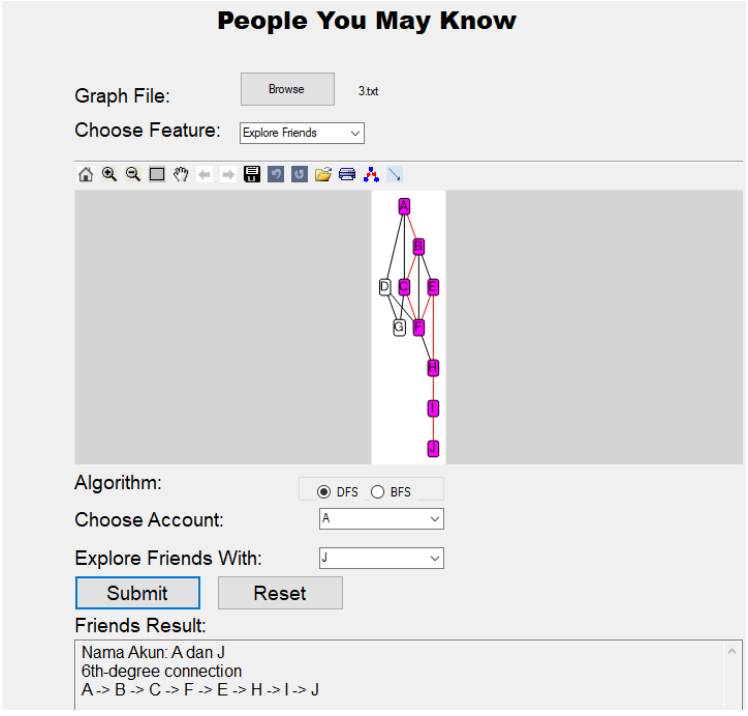
1. Jalankan bin/PrototypeUI.exe
2. Klik *browse*, gunakan salah satu file dari bin/testcase/
3. File telah di*parsing* dan seharusnya suatu graf sudah tampil di layar.
4. Gunakan salah satu dari dua fitur yang ada, yaitu Friend Recommendation & Explore Friends.
5. Pilih akun sumber & akun target (Jika pengguna memilih opsi Explore Friends)
6. Tekan tombol submit & lihat hasil graf & output pada panel teks
7. Jika ingin mengganti file & meng-*clear* panel teks, tekan tombol reset.

## 4.4. Hasil Pengujian Program

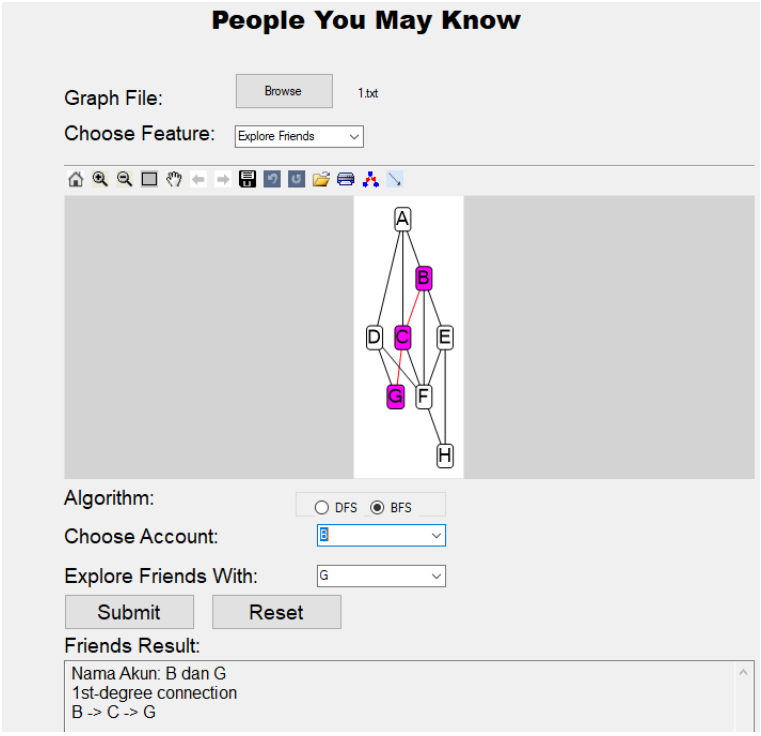
Berikut merupakan tangkapan layar dari antarmuka program beserta beberapa graf yang diujikan.



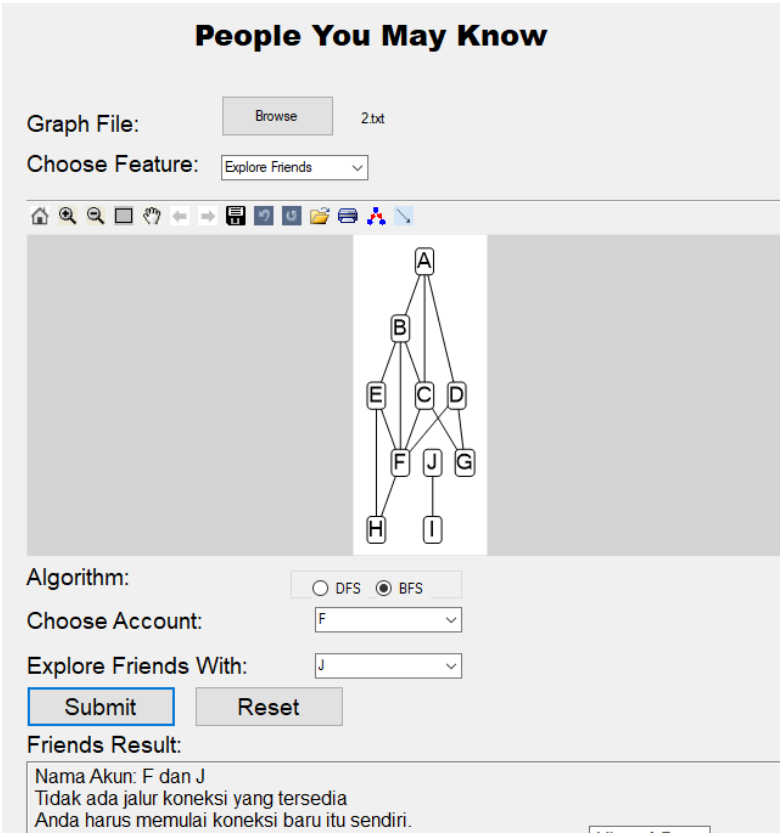

Gambar 4.4.1. : Hasil pengujian 1




Gambar 4.4.2. : Hasil pengujian 2




Gambar 4.4.3. : Hasil pengujian 3




Gambar 4.4.4. : Hasil pengujian 4

## 4.5. Analisis Algoritma yang Digunakan

### 4.5.1. Analisis Algoritma DFS

Pada pencarian solusi menggunakan algoritma DFS, hasil akhir dari algoritma tersebut tidak selalu menjadi *shortest path* atau jalur terpendek. Hal ini dikarenakan algoritma yang dibuat memprioritaskan pencarian simpul dengan urutan abjad terkecil. Contoh dari kasus ini adalah pada gambar 4.4.2. Pada gambar tersebut, dapat dilihat urutan dari simpul A ke simpul J adalah {A,B,C,F,E,H,I,J}, jika pencarian tidak diprioritaskan terhadap urutan abjad, maka hasil akhirnya nanti akan sama seperti pencarian dengan algoritma BFS (gambar 4.4.1).

Pada kasus lain, algoritma DFS juga dapat menemukan solusi dengan hasil akhir adalah *shortest path* atau jalur terpendek. Sebagai contoh, untuk graf yang sama dengan gambar 4.4.2, apabila dicari jalur dari simpul A ke simpul C, maka akan didapat jalur terpendeknya yaitu {A,B,C}.

Algoritma DFS ini akan lebih baik atau sama daripada algoritma BFS dalam hal pencarian jalur terpendek apabila suatu graf saling terhubung satu sama lain dengan masing-masing nama simpul terurut menaik berdasarkan abjad, solusi ini termasuk ke kategori Kasus Skenario Terbaik atau *Best Case Scenario*. Kasus Skenario Terburuk atau *Worst Case Scenario* dari pencarian dengan algoritma DFS ini adalah jika graf yang saling terhubung tetapi nama simpul-simpulnya tidak terurut menaik berdasarkan abjad.

### 4.5.2. Analisis Algoritma BFS

Pada pencarian solusi menggunakan algoritma BFS, hasil akhir dari algoritma selalu menjadi suatu *shortest path*, hal ini dikarenakan terjadinya pengecekan & “pemangkasan” jalur yang tidak perlu

Pada kasus lain, algoritma DFS juga dapat menemukan solusi dengan hasil akhir adalah *shortest path* atau jalur terpendek. Sebagai contoh, untuk graf yang sama dengan gambar 4.4.2, apabila dicari jalur dari simpul A ke simpul C, maka akan didapat jalur terpendeknya yaitu {A,B,C}.

Algoritma DFS ini akan lebih baik atau sama daripada algoritma BFS dalam hal pencarian jalur terpendek apabila suatu graf saling terhubung satu sama lain dengan masing-masing nama simpul terurut menaik berdasarkan abjad, solusi ini termasuk ke kategori Kasus Skenario Terbaik atau *Best Case Scenario*. Kasus Skenario Terburuk atau *Worst Case Scenario* dari pencarian dengan algoritma DFS ini adalah jika graf yang saling terhubung tetapi nama simpul-simpulnya tidak terurut menaik berdasarkan abjad.

# BAB V

# KESIMPULAN, SARAN, REFLEKSI, DAN KOMENTAR

## 5.1 Kesimpulan

Algoritma DFS dan BFS adalah algoritma yang mencari node pada suatu graf, dari suatu node *root* ke suatu node *destination*.

Pada algoritma DFS pencarian pada node dilakukan secara mendalam, dengan menyimpan node-node aktif yang belum ditelusuri dalam suatu Stack yang bersifat *last in first out*, karena karakteristik ini pada level implementasinya algoritma DFS juga menggunakan cara *backtracking*.

Berbeda dengan algoritma DFS, algoritma BFS merupakan pencarian yang dilakukan secara melebar, dan memanfaatkan *queue* untuk menelusuri node-node yang belum ditelusuri.

Kedua algoritma tersebut dapat juga dimanfaatkan, selain untuk mencari node pada suatu graf, namun juga untuk mencari dan menelusuri *path* atau jalur dari suatu node *root* ke node *destination*. Hal inilah yang menjadi dasar dari aplikasi fitur rekomendasi teman, dan *people you may know* yang sering kita temui pada jejaring sosial.

## 5.2 Saran

Saran yang dapat kami berikan untuk pengembangan aplikasi *people you may know* dengan algoritma BFS dan DFS ini adalah dengan mempercantik dan menambah kreativitas lagi dalam desain GUI dan pewarnaan graf, agar meningkatkan *user experience* dari pengguna. Selain itu kedepannya agar tugas-tugas seperti ini dipertahankan karena menuntut mahasiswa untuk bereksplorasi dan belajar dengan mandiri tentang teknologi yang berkaitan dengan bidang studinya.

## 5.3 Refleksi

Pengerjaan Tugas Besar ini terkendala oleh waktu dan *timeline* yang bersamaan dengan UTS, Milestone Sistem Operasi, Mileston Rekayasa Perangkat Lunak, dan Tugas Besar Pemrograman Berorientasi Objek. Awalnya terasa sangat berat, dan muncul pikiran, apakah semua tugas-tugas tersebut bisa dikerjakan dengan baik dan tepat waktu. Namun seiring berjalannya waktu tugas-tugas tersebut gugur satu persatu (berhasil dikumpulkan) dan pada akhirnya dengan kerja keras dan kerjasama yang baik dengan rekan sekelompok, semua tugas-tugas tersebut dapat kami kerjakan dengan baik.

# Daftar Pustaka


- Munir, Rinaldi. 2020. Graf 2020 untuk IF2120 Matematika Diskrit. http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf (diakses tanggal 22 Maret 2021).
- Munir, Rinaldi. 2021. Tugas Besar 2 (Tubes 2): Pengaplikasian Algoritma BFS dan DFS dalam Fitur People You May Know Jejaring Sosial Facebook. http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Tugas-Besar-2-IF2211-Strategi-Algoritma-2021.pdf (diakses tanggal 24 Maret 2021).
- Munir, Rinaldi. 2021. Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1). http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf (diakses tanggal 24 Maret 2021).
- Knuth, Donald E. (1997), The Art of Computer Programming Vol 1. 3rd ed., Boston: Addison-Wesley, ISBN 978-0-201-89683-1