

**PENGAPLIKASIAN ALGORITMA *BFS* DAN *DFS* DALAM FITUR  
*PEOPLE YOU MAY KNOW* JEJARING SOSIAL *FACEBOOK***

Laporan Tugas Besar 2 IF2211 Strategi Algoritma  
Semester II Tahun Akademik 2020/2021



Oleh:

Kelompok 26 - *FacePaper*

Muhammad Dzaki Razaan Faza	13519033
Joel Triwira	13519073
Wilson Tandya	13519209

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

## BAB I

### DESKRIPSI TUGAS

Dalam tugas besar ini, kami 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)*, kita dapat menelusuri *social network* pada akun *facebook* untuk mendapatkan rekomendasi teman seperti pada fitur *People You May Know*. Selain untuk mendapatkan rekomendasi teman, kami juga diminta untuk mengembangkan fitur lain agar dua akun yang belum berteman dan tidak memiliki *mutual friends* sama sekali bisa berkenalan melalui jalur tertentu.

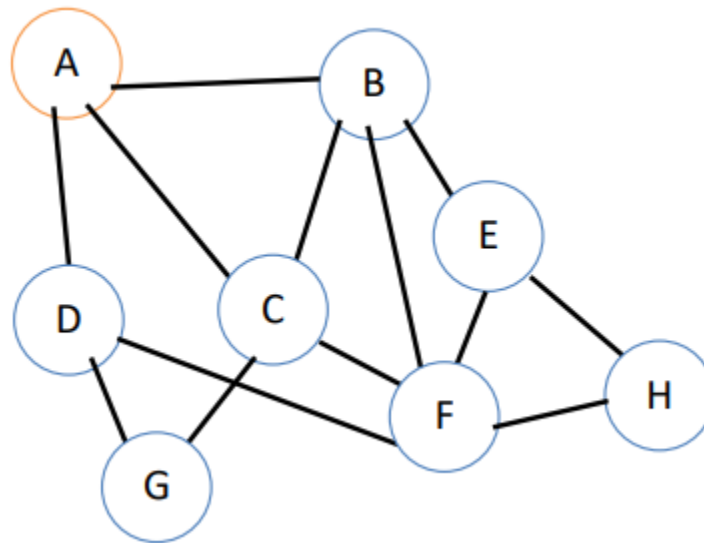
#### **Contoh *Input* dan *Output* Program**

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

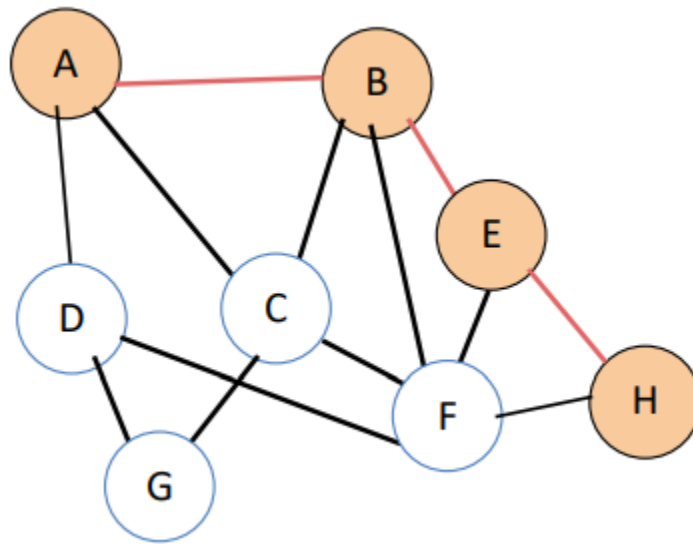
Gambar 1.3. Hasil output yang diharapkan untuk rekomendasi akun A

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. Berikut output yang diharapkan untuk penelusuran menggunakan BFS.

Nama akun: A dan H  
2nd-degree connection  
 $A \rightarrow B \rightarrow E \rightarrow H$

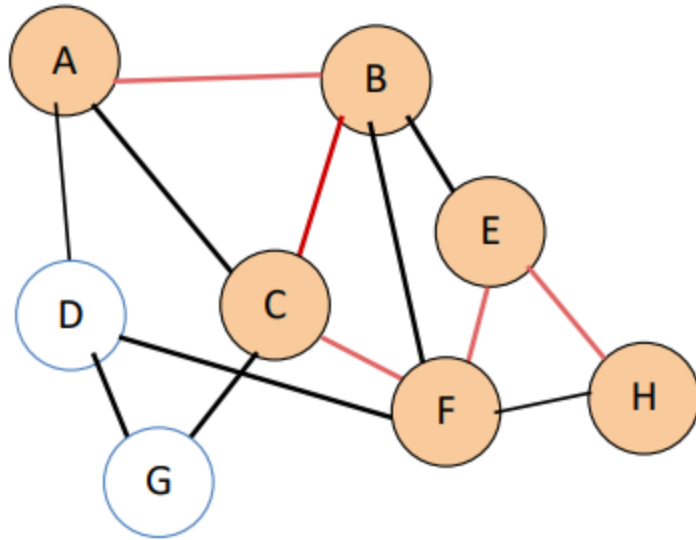
Gambar 1.4. Hasil output akun Nth degree connection akun A dan H menggunakan BFS

Pada busur antara akun A dan H, terbentuk salah satu jalur koneksi sebagai berikut: A-B-E-H (ada beberapa jalur lainnya, seperti A-D-F-H, dll, urutan simpul untuk ekspansi 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 1.5. 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 1.6. 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.

Gambar 1.7. Hasil output tidak ada jalur koneksi antara A dan J

## **BAB II**

### **LANDASAN TEORI**

#### **2.1 Dasar Teori *Graph Traversal***

*Graph traversal* atau yang juga dikenal dengan *graph search* merupakan proses mengunjungi setiap simpul dalam sebuah graf, baik memeriksa dan / atau memperbarui. *Graph traversal* dikategorikan berdasarkan urutan pengunjungan simpul. Algoritma yang paling sederhana digunakan dalam *graph traversal* adalah *Breadth First Search (BFS)* dan *Depth First Search (DFS)*.

*Graph traversal* digunakan untuk melakukan pencarian solusi, pada pencarian solusi algoritma yang digunakan bergantung pada jenis informasi yang dimiliki. Terdapat *blind search* dan *informed search*. Dalam proses pencarian solusi terdapat pula dua buah pendekatan, yaitu graf yang sudah dibentuk sebelum proses pencarian dilakukan (*static graph*) dan graf yang dibentuk saat proses pencarian dilakukan (*dynamic graph*).

#### **2.2 Dasar Teori *Breadth First Search (BFS)***

*Breadth First Search* atau *BFS* merupakan algoritma yang melakukan pencarian secara melebar yang mengunjungi simpul secara *preorder* yaitu mengunjungi suatu simpul kemudian mengunjungi semua simpul yang bertetangga dengan simpul tersebut terlebih dahulu. Selanjutnya, simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya. algoritma *BFS* ini menggunakan graf sebagai media representasi persoalan.

#### **2.3 Dasar Teori *Depth First Search (DFS)***

*Depth First Search* atau *DFS* adalah algoritma pencarian yang dilakukan pada satu simpul dalam setiap *level* dari yang paling kiri atau sesuai urutan numerik maupun alfabetikal. Jika pada *level* yang paling dalam, solusi belum ditemukan, maka pencarian dilanjutkan pada simpul terurut sebelahnyanya dan simpul yang awal dapat dihapus dari memori. Jika pada *level* yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada *level* sebelumnya. Demikian seterusnya sampai ditemukan solusi. Jika solusi ditemukan maka tidak diperlukan proses *backtracking* (penelusuran balik untuk mendapatkan jalur yang diinginkan).

## 2.4 Penjelasan Singkat *C# Desktop Application Development*

*C#* merupakan bahasa pemrograman yang dirancang untuk membuat berbagai aplikasi yang berjalan di *.NET Framework*. Bahasa *C#* ini *powerful*, *type-safe*, dan berparadigma objek (*object-oriented*). *C#* dibangun di atas *.NET Compiler “Roslyn”* yang menyediakan analisis *API* dan semua ini *open source* pada *Github*.

*Windows form application* adalah suatu *desktop application development* yang dapat diprogram menggunakan bahasa pemrograman *C#*. *Windows form application* juga merupakan sebuah aplikasi yang dirancang untuk berjalan di komputer secara lokal, tidak dapat berjalan pada *web browser*.

## BAB III

### ANALISIS PEMECAHAN MASALAH

#### 3.1 Langkah-Langkah Pemecahan Masalah

Masukkan *input file* eksternal (.txt) akan dibaca dan dibentuk graf visualisasinya dengan bantuan *MSAGL* dan ditampilkan pada *panel windows form*. *Drop-down* pada *choose account* dan *explore friends with* diisi dengan nama simpul yang berasal dari *file* eksternal. Bila *drop-down* pada *choose account* telah diisi dan di-submit, akan dipanggil fungsi *Friend Recommendations* dari *Graph.cs* dan menampilkan simpul-simpul yang diperiksa dan dibangkitkan berdasarkan algoritma pencarian *BFS* serta daftar rekomendasi teman terurut berdasarkan jumlah *mutual friends*-nya.

Bila *drop-down* pada *explore friends with* juga diisi dan algoritma *DFS* di *check* (✓), akan dipanggil fungsi *DFS* untuk *explore friends* dan menampilkan simpul-simpul yang ditelusuri, hasil penelusuran, serta derajat koneksi antara kedua akun yang dipilih.

Sama halnya bila *drop-down* pada *explore friends with* juga diisi dan algoritma *BFS* di *check* (✓), akan dipanggil fungsi *BFS* untuk *explore friends* dan menampilkan yang diperiksa, simpul-simpul yang dibangkitkan, hasil penelusuran, dan derajat koneksi antara kedua akun yang dipilih.

#### 3.2 Proses Mapping Persoalan Menjadi Elemen-Elemen Algoritma *BFS* dan *DFS*

##### **Mapping Persoalan dengan Algoritma *DFS* :**

1. Mencari indeks dari simpul awal yang dimasukkan
2. Membuat simpul itu menjadi “telah dikunjungi”
3. Memasukkan simpul awal ke dalam *stack*
4. Mencari simpul berikutnya sesuai urutan abjad
5. Cek apakah simpul berikutnya merupakan solusi atau tidak, jika tidak, ulangi proses dari langkah nomor 2
6. Jika iya, simpul akan dimasukkan ke *stack* dan *stack* akan dimasukkan kedalam *array* hasil sehingga akan ditampilkan ke layar.

##### **Mapping Persoalan dengan Algoritma *BFS* :**

1. Mencari indeks dari simpul awal yang dimasukkan
2. Membuat simpul itu menjadi “telah dikunjungi”

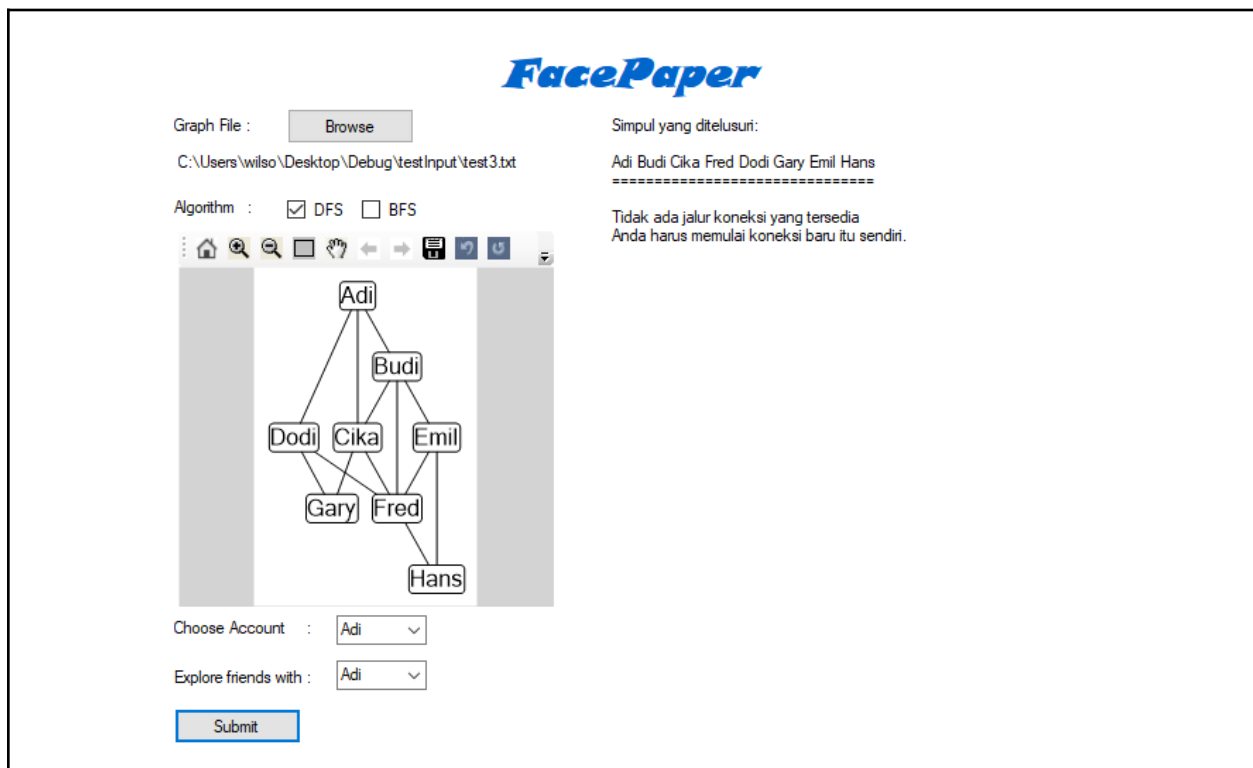


3. Memasukkan simpul awal ke dalam *queue*
4. Mengeluarkan simpul pertama dalam *queue* dan memasukkan semua tetangga dari simpul yang dikeluarkan tersebut
5. Sebelum dimasukkan ke dalam *queue* cek juga apakah simpul tersebut sudah merupakan simpul tujuan
6. Jika sudah, maka telusuri pembangkit simpul tersebut hingga kembali ke simpul awal, masukkan dalam *array*, dan tampilkan ke layar.
7. Jika belum, maka ulangi langkah 4 dan 5 hingga ditemukan simpul tujuan atau *queue* menjadi kosong yang artinya tidak ditemukan jalan

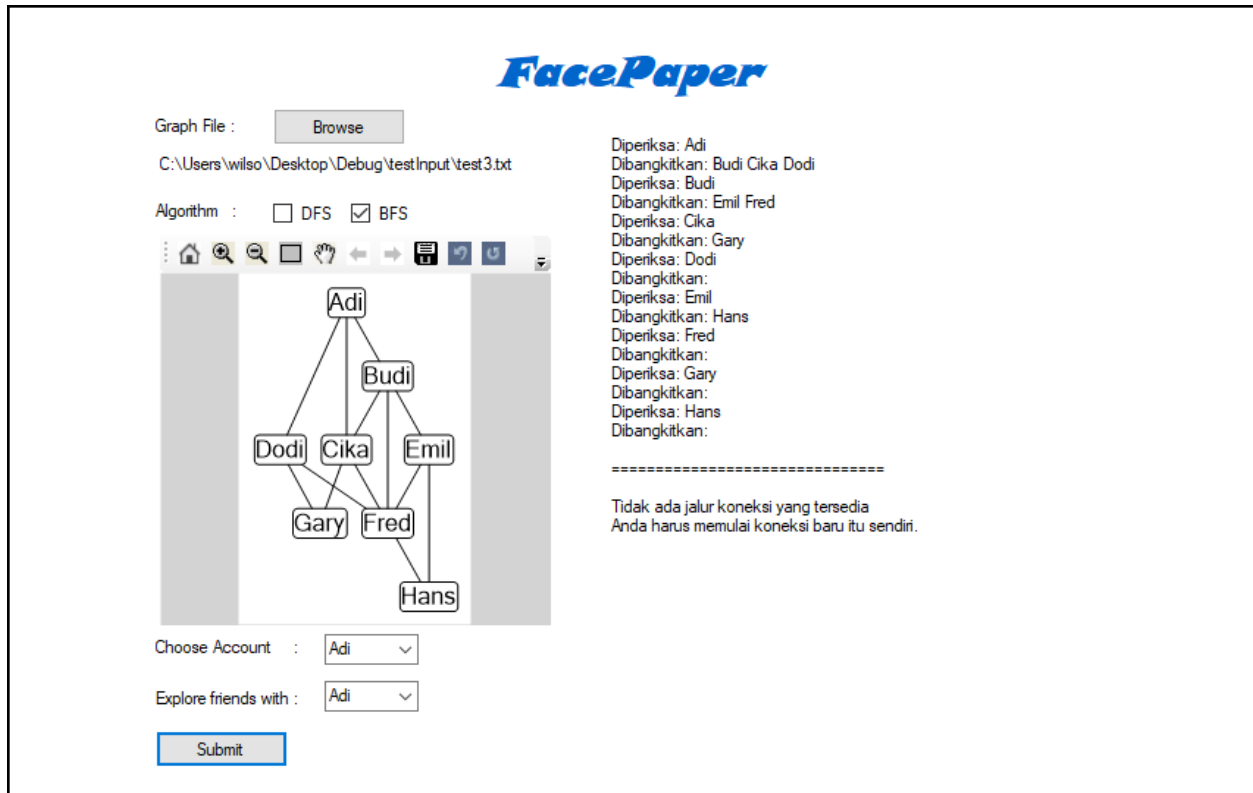
### 3.3 Contoh Ilustrasi Kasus Lain

#### 3.3.1 Ilustrasi Kasus *N-th Degree Connection* dari 2 Akun yang Sama

Pada kasus ini, setiap simpul akun akan ditelusuri baik dengan algoritma *DFS* maupun *BFS*. Namun keduanya akan menampilkan tidak ada jalur koneksi yang tersedia karena memang dari akun itu ke akun sendiri tidak ada sisi yang menghubungkannya.



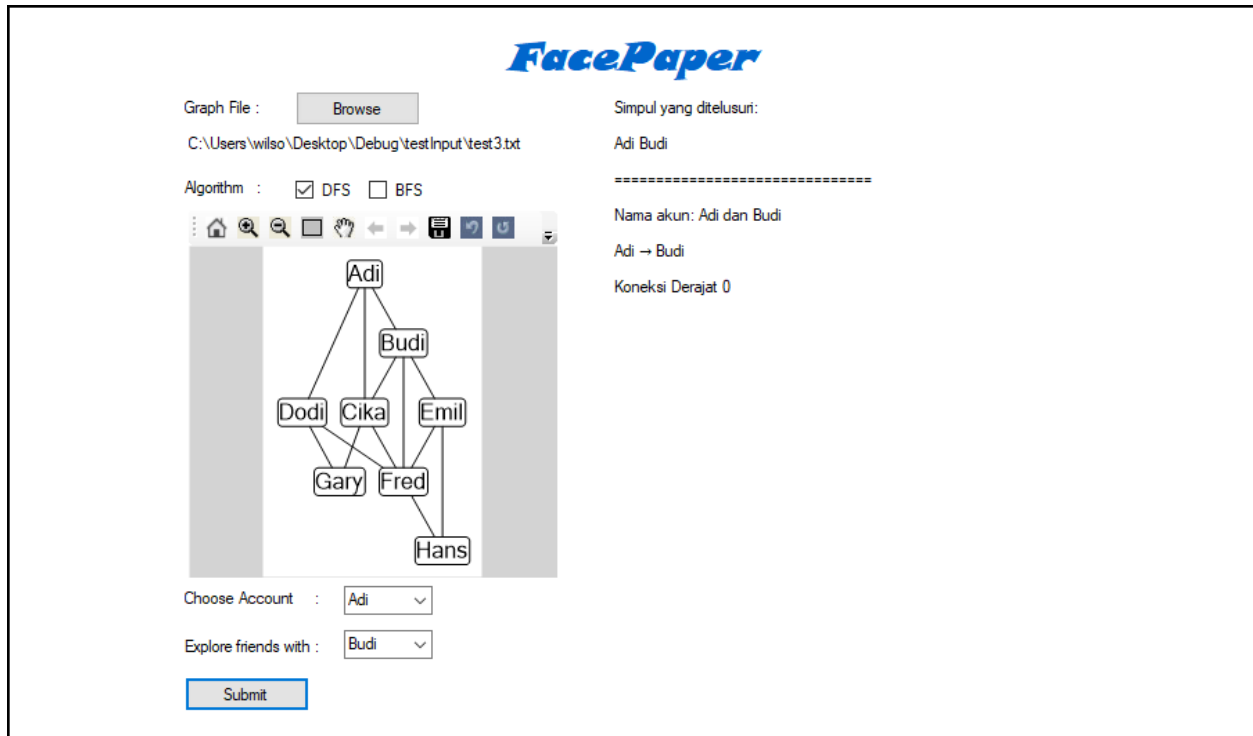
Gambar 3.3.1.1. Ilustrasi Kasus *N-th Degree Connection* dari 2 Akun yang Sama (*DFS*)



Gambar 3.3.1.2. Ilustrasi Kasus *N-th Degree Connection* dari 2 Akun yang Sama (*BFS*)

### 3.3.2 Ilustrasi Kasus *N-th Degree Connection* dari 2 Akun Terhubung Langsung

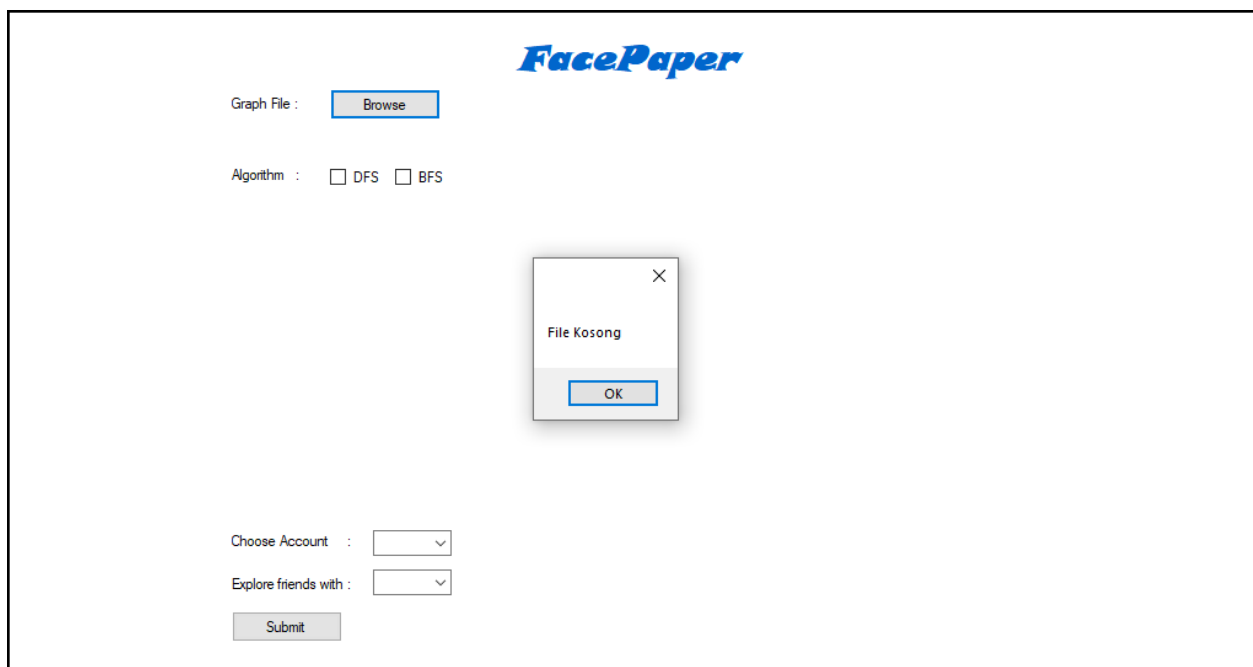
Pada kasus ini, akan ditampilkan bahwa koneksi derajat antara kedua akun yang terhubung langsung adalah 0, baik dengan algoritma *DFS* maupun *BFS*. Hal ini dikarenakan tidak ada akun perantara antara Adi dan Budi pada Gambar 3.3.2.1



Gambar 3.3.2.1. Ilustrasi Kasus *N-th Degree Connection* dari 2 Akun yang Terhubung Langsung (DFS)

### 3.3.3 Ilustrasi Kasus *File Input* Eksternal Kosong

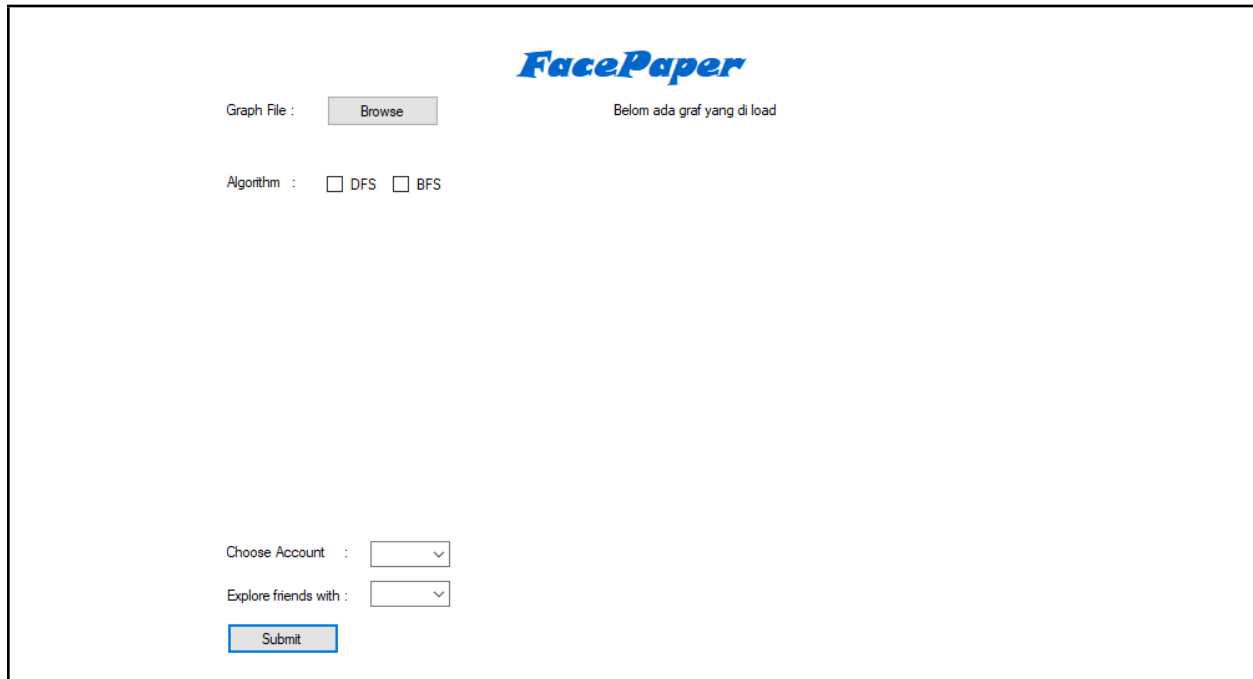
Pada kasus ini, akan muncul *pop-up* bahwa *file* eksternal yang dimasukkan kosong.



Gambar 3.3.3.1. Ilustrasi Kasus *File Input* Kosong

### 3.3.4 Ilustrasi Kasus Menjalankan Fitur Sebelum *Input File* Eksternal

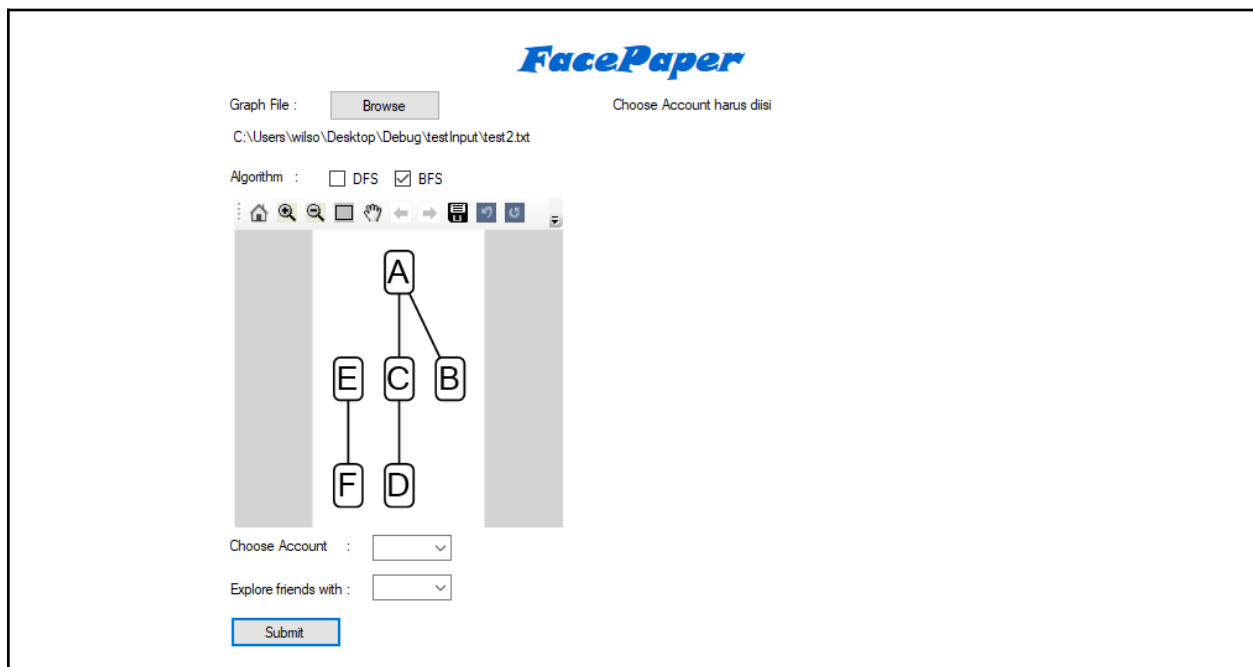
Pada kasus ini, akan muncul pesan bahwa belum ada graf yang di-load.



Gambar 3.3.4.1. Ilustrasi Kasus Menjalankan Fitur Sebelum *Input File* Eksternal

### 3.3.5 Ilustrasi Kasus Menjalankan Fitur saat *Query* Kosong

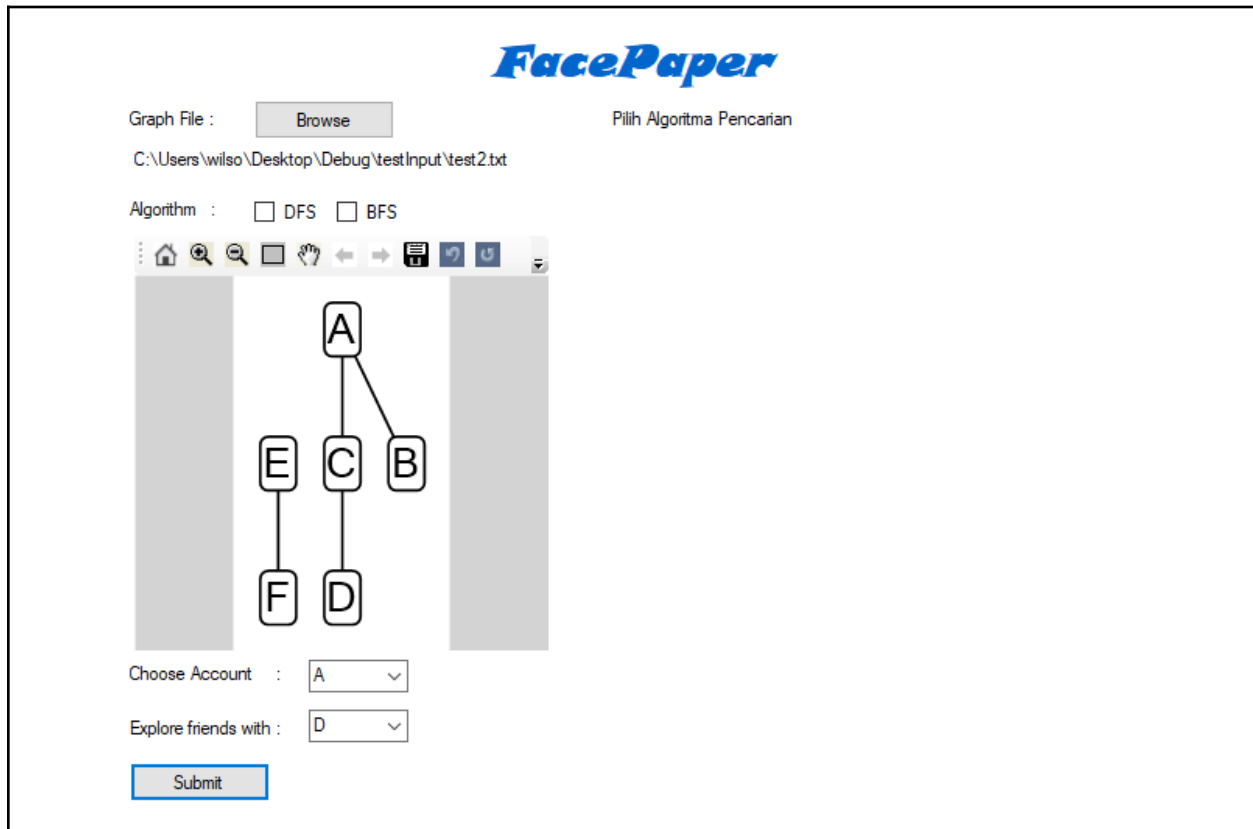
Pada kasus ini, akan muncul pesan bahwa belum ada graf yang di-load.



Gambar 3.3.5.1. Ilustrasi Kasus Menjalankan Fitur saat *Query* Kosong

### 3.3.6 Ilustrasi Kasus Menjalankan Fitur *Explore Friends* Tanpa Memilih Algoritma

Pada kasus ini, akan muncul pesan bahwa algoritma pencarian harus dipilih saat ingin menjalankan fitur *Explore Friends*.



Gambar 3.3.6.1. Ilustrasi Kasus Menjalankan Fitur *Explore Friends* Tanpa Memilih Algoritma

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi Program

##### 4.1.1 Friend Recommendation

**function** FriendRecommendation(input akun: string, output result: array of list)

**KAMUS**

ResetDikunjungi : list of int yang menyimpan indeks array yang sudah diubah status dikunjunginya

**ALGORITMA**

```
v = CariIndexSimpul(akun)
BuatAntian(q)
Dikunjungi[v] ← true
while AdaTetanggaBelumDikunjungi(v) do
    w = CariSimpulTetanggaBelumDikunjungi(v)
    if w > -1 then
        MasukAntrian(q, w)
        Dikunjungi[w] ← true
while not AntriaKosong(q) do
    HapusAntrian(q, v)
    while AdaTetanggaBelumDikunjungi(v) do
        w = CariSimpulTetanggaBelumDikunjungi(v)
        if w > -1 then
            Dikunjungi[w] ← true
            ResetDikunjungi.Add(w)
            Result[w].Add(v)
    for i in ResetDikunjungi
        Dikunjungi[i] ← false
→ result.SortBerdasarkanSize()
```

##### 4.1.2 Explore Friends dengan Algoritma DFS

**function** DFS(input from: string, input to: string, output result: array of string)

**KAMUS**

v : index simpul saat ini yang sedang dicek

q : antrian simpul yang sudah dibangkitkan tetapi belum dicek

S : stack of simpul

**ALGORITMA**

v = CariIndexSimpul(from)

t = CariIndexSimpul(to)

Dikunjungi[v] ← true

JalurKetemu ← false

s.push(v)

while StackTidakKosong(s) and not JalurKetemu do

    w = CariSimpulTetanggaBelumDikunjungi(v)

    if w = -1 then

        s.Pop()

    else

        Dikunjungi[w] ← true

        s.Push(w)

        if w = t then

            JalurKetemu ← true

if not StackKosong(s) then

    result = UbahStackKeArraySimpul(s)

#### 4.1.3 Explore Friends dengan Algoritma BFS

**function** BFS(input from: string, input to: string, output result: array of string)

**KAMUS**

v : index simpul saat ini yang sedang dicek

q : antrian simpul yang sudah dibangkitkan tetapi belum dicek

**ALGORITMA**

v = CariIndexSimpul(from)

t = CariIndexSimpul(to)

BuatAntian(q)

Dikunjungi[v] ← true

JalurKetemu ← false

MasukAntrian(q,v)

while not AntriaKosong(q) do

    HapusAntrian(q,v)

```
while not JalurKetemu and AdaTetanggaBelumDikunjungi(v) do
    w = CariSimpulTetanggaBelumDikunjungi(v)
    if w > -1 then
        MasukAntrian(q, w)
        Dikunjungi[w] ← true
        Pembangkit[w] ← v
        if w = t then
            JalurKetemu ← true
if JalurKetemu then
    result = TelusuriIndexAkhirHinggaAwal(Pembangkit, t)
→ result
```



## 4.2 Penjelasan Struktur Data

Beberapa struktur data yang digunakan dalam program ini adalah:

a. *Queue*

Digunakan dalam algoritma *explore friends* menggunakan metode BFS. *Queue* akan menampung seluruh simpul yang telah dibangkitkan oleh simpul lain. Dengan memanfaatkan sifat FIFO-nya (*First In First Out*) *queue* dapat mengeluarkan setiap simpul yang akan dicek sesuai dengan urutan kebangkitannya.

b. *List*

Digunakan dalam algoritma *friend recommendation*. *List* digunakan untuk menyimpan banyak teman dari sebuah simpul. Jumlah teman setiap simpul tidak dapat ditentukan di awal sehingga *list* sangat cocok karena sangat fleksibel dalam hal ukuran dibandingkan *array* statis.

c. *Stack*

Digunakan dalam algoritma *explore friends* dengan metode DFS. *Stack* digunakan untuk menyimpan simpul yang sedang dan sudah dikunjungi. Simpul akan di-*push* ketika pertama kali dikunjungi dan di-*pop* bila sudah tidak ada simpul tetangganya yang belum dikunjungi.

d. *Graph*

Merupakan representasi dari kasus jejaring sosial. Simpul berperan sebagai akun. Sisi berperan sebagai tanda adanya koneksi antara satu akun dengan akun lainnya atau dengan kata lain berteman. Dalam implementasi *graph* dinyatakan dalam bentuk *array* untuk menampung nama akun dan indexnya dan *adjacency matrix* untuk menampung simpul mana saja yang saling berhubungan.

Program *FacePaper* ini terdiri dari *Graph.cs* yang merupakan program yang berisi kelas *Graph* untuk membuat representasi graf dari *input file* eksternal, dan terdapat fungsi *BFS*, *DFS*, *Friend Recommendation* yang diimplementasikan di dalamnya. Kemudian, terdapat *Form1.cs* yang merupakan program untuk mengaplikasikan fitur-fitur yang telah dibuat sesuai dengan masukkan pengguna pada *GUI Form*. Masih berhubungan dengan *Form* adapula *Form1.Designer.cs* yang digunakan untuk membuat tampilan *GUI* dari *form* itu sendiri. Terakhir ada *Program.cs* yang hanya bertugas untuk melakukan *load* dari *form* (*initialization*) yang dibuat.

## 4.3 Penjelasan Tata Cara Penggunaan Program dan Fitur-Fitur yang Disediakan

### 4.3.1 Tata Cara Penggunaan Program

Untuk menjalankan program *FacePaper*, cukup membuka *folder bin* dan menjalankan *Visualizer.exe*. Jika ingin memasukan tambahan *test-case* dapat memasukan *file .txt* ke dalam *folder testInput* yang terdapat di dalam *folder bin* pula.

### 4.3.2 Fitur Input File Eksternal dan Menampilkan Visualisasi Graf:

Pada saat menjalankan program, hal pertama yang harus dilakukan adalah menekan tombol *Browse* untuk memasukan *file* eksternal (*.txt*). Program kemudian akan melakukan pengecekan apabila *file* kosong atau berisi dan menampilkan nama file serta representasi graf bila *file* memiliki isi.

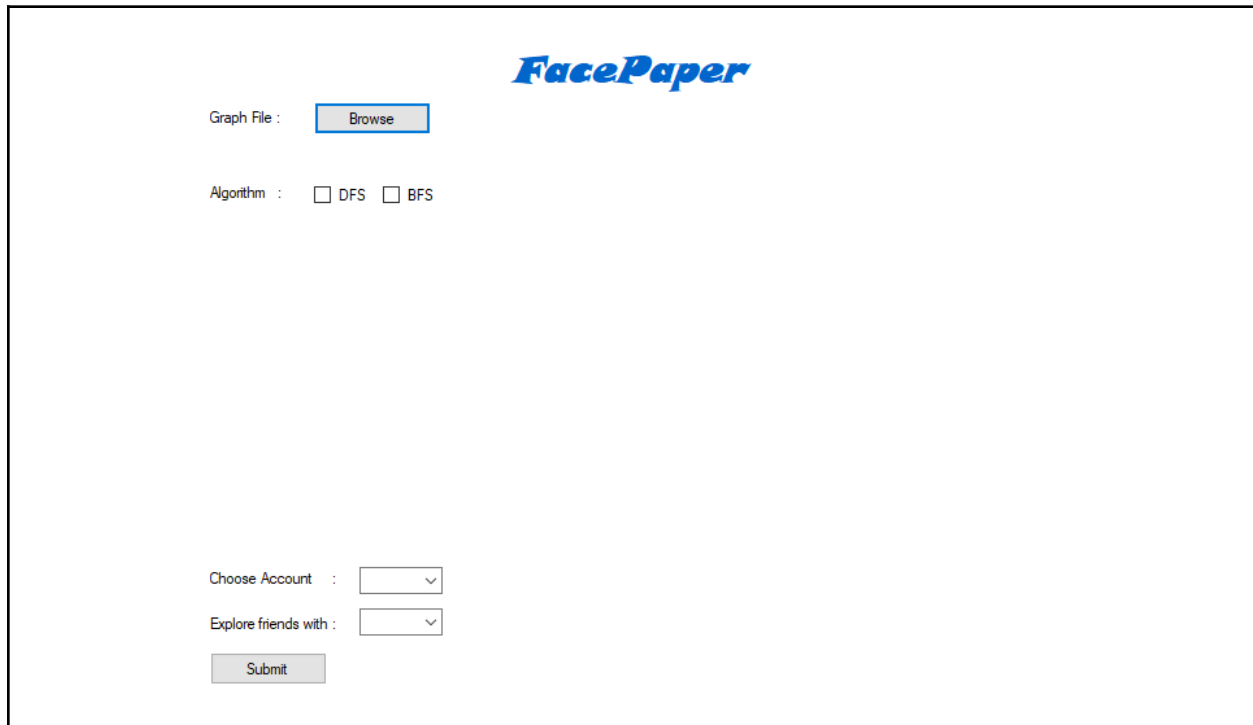
### 4.3.3 Fitur *Friend Recommendation*:

Pengguna dapat memilih sebuah akun yang ada pada graf di *Choose Account*, dan bila menekan tombol *Submit* akan mengeluarkan *Friend Recommendation* untuk akun tersebut. Daftar rekomendasi teman akan ditampilkan berdasarkan urutan *mutual friends* terbanyak, dan akan menampilkan *mutual friend*-nya. Fitur *Friend Recommendation* ini menggunakan algoritma pencarian secara *BFS*.

### 4.3.4 Fitur *Explore Friends*:

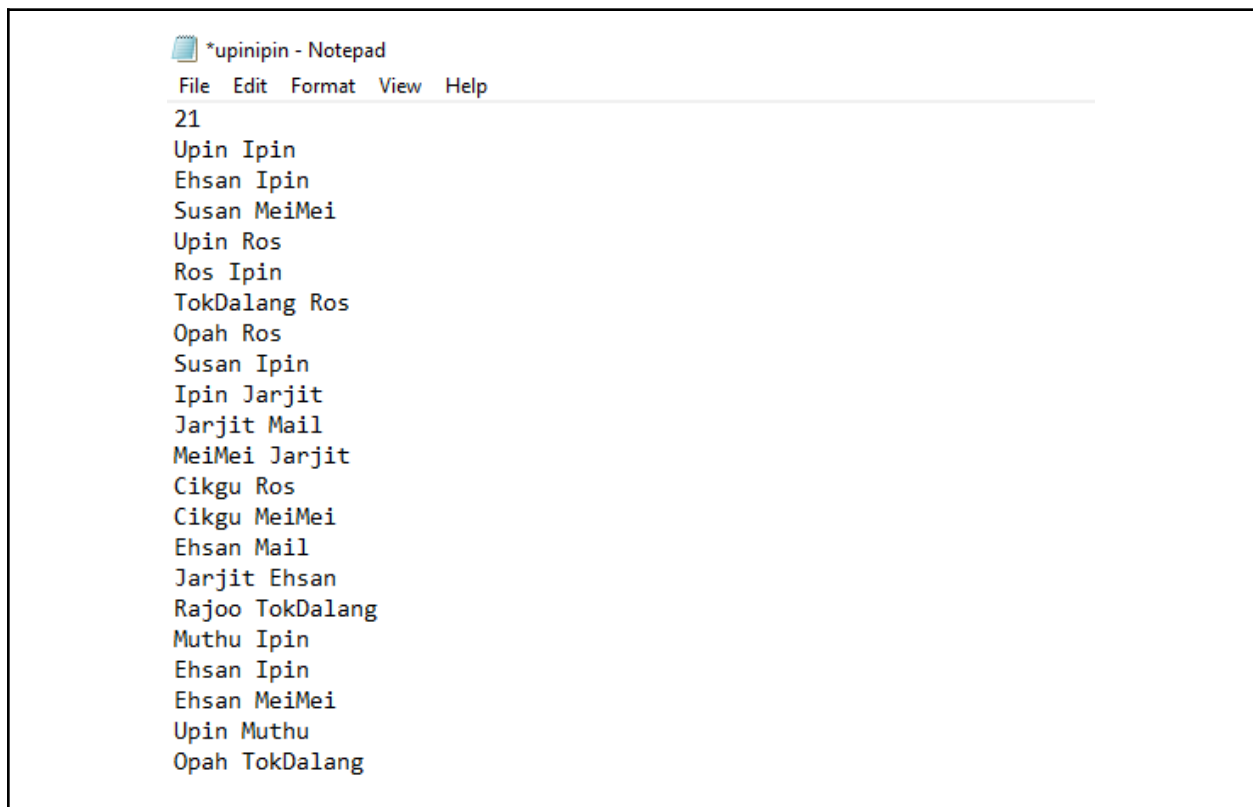
Pengguna dapat memilih akun kedua pada *Explore friends with*, dan harus memilih algoritma yang ingin digunakan pada *checkbox algorithm (DFS atau BFS)*, dan bila menekan tombol *Submit* akan mengeluarkan jalur koneksi dari akun pertama ke akun kedua dan koneksi derajat antara kedua akun. Pada menu algoritma *DFS* akan ditampilkan pula simpul-simpul yang telah ditelusuri selama pencarian di graf. Sedangkan pada menu algoritma *BFS* akan ditampilkan pula simpul-simpul yang diperiksa dan yang dibangkitkan.

## 4.4 Hasil Pengujian



The screenshot shows the initial interface of the FacePaper application. At the top center is the logo "FacePaper" in a blue, stylized font. Below the logo, there is a "Graph File :" label followed by a "Browse" button. Underneath that is the "Algorithm :" label with two radio buttons, "DFS" and "BFS". At the bottom of the interface, there are two dropdown menus labeled "Choose Account :" and "Explore friends with :". Below these dropdowns is a "Submit" button.

Gambar 4.4.1. Tampilan Awal *FacePaper*

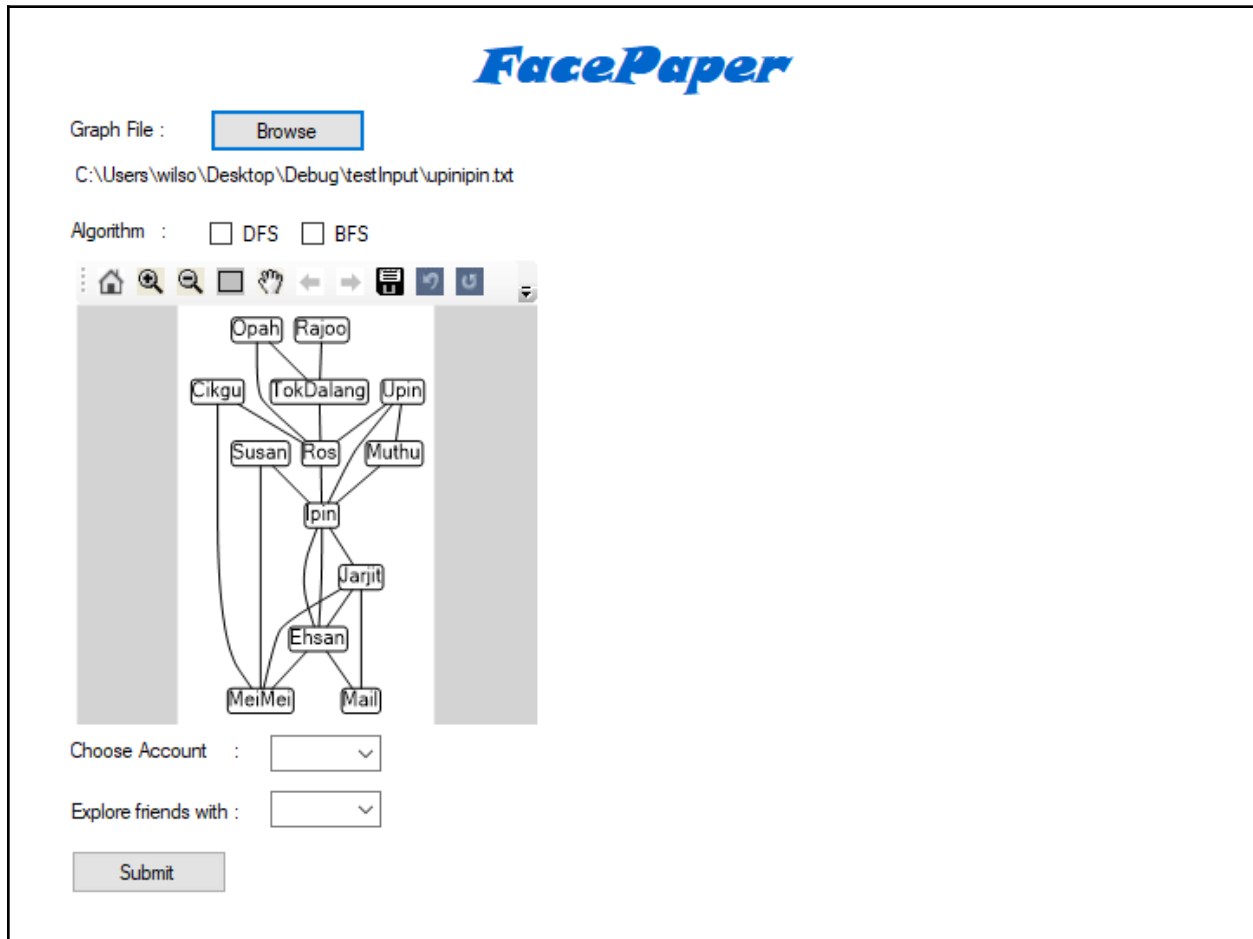


The screenshot shows a Notepad window titled "\*upinipin - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The text content of the file is as follows:

```
21
Upin Ipin
Ehsan Ipin
Susan MeiMei
Upin Ros
Ros Ipin
TokDalang Ros
Opah Ros
Susan Ipin
Ipin Jarjit
Jarjit Mail
MeiMei Jarjit
Cikgu Ros
Cikgu MeiMei
Ehsan Mail
Jarjit Ehsan
Rajoo TokDalang
Muthu Ipin
Ehsan Ipin
Ehsan MeiMei
Upin Muthu
Opah TokDalang
```

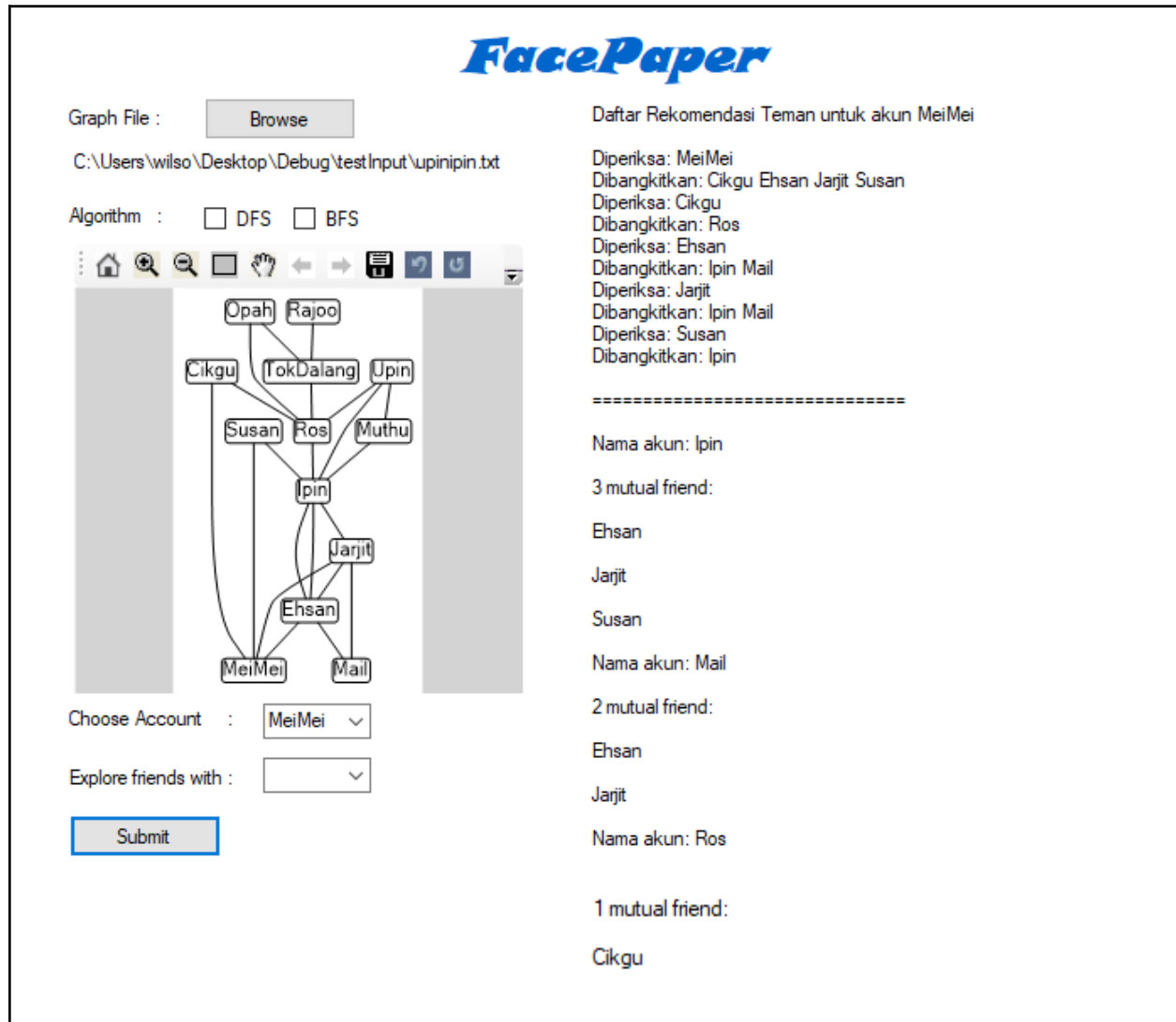
Gambar 4.4.2. *Input File* Eksternal *upinipin.txt*

Pada Gambar 4.4.3 dapat dilihat *file* eksternal yang dimasukkan pengguna yaitu *upinipin.txt* menghasilkan visualisasi graf. Sisi dari graf melambangkan pertemanan antara kedua simpul akun.



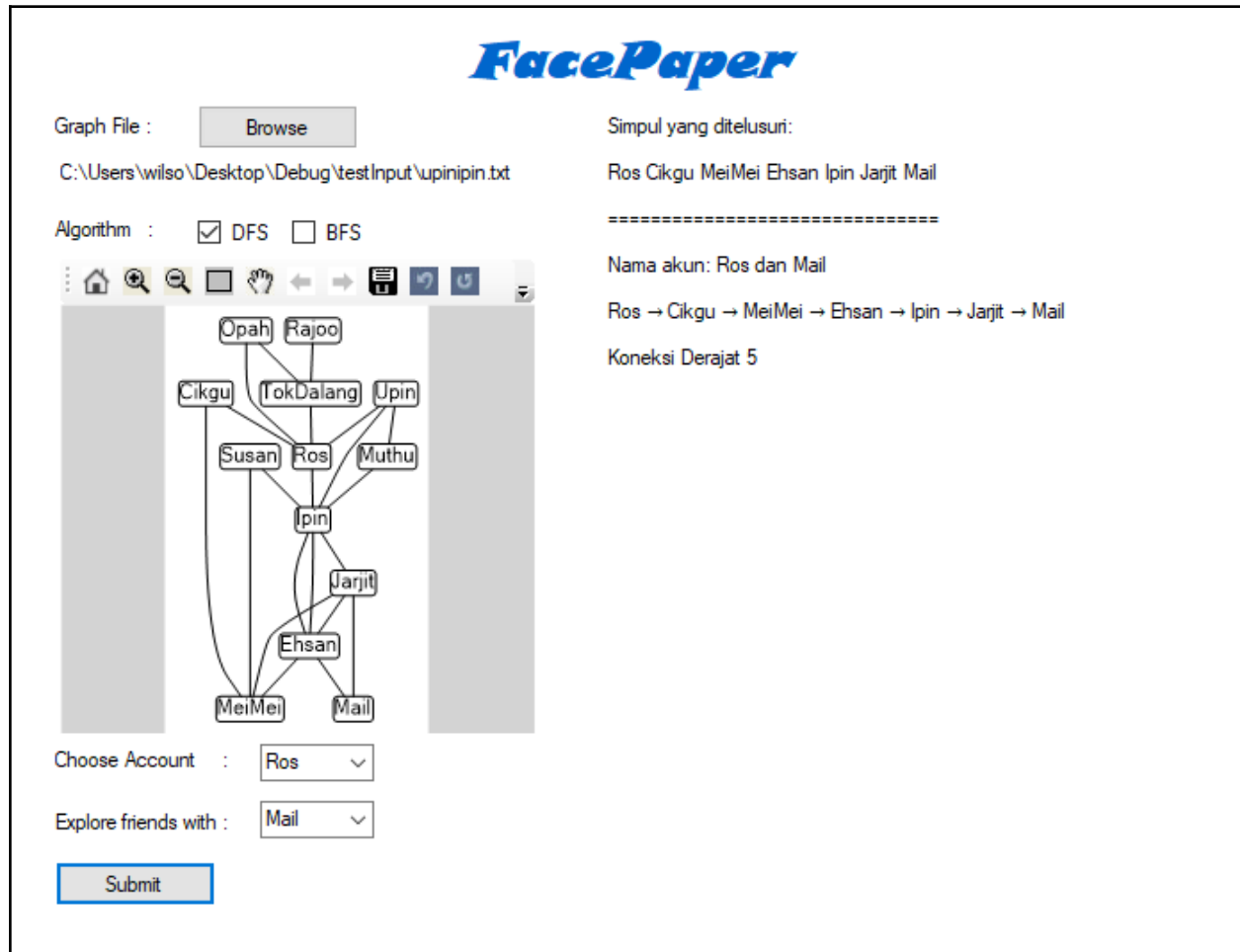
Gambar 4.4.3. Visualisasi Graf untuk *upinipin.txt*

Pada Gambar 4.4.4 *Friend Recommendation* dari akun MeiMei dicari menggunakan algoritma *BFS*. Ditunjukkan pula simpul-simpul yang diperiksa dan dibangkitkan untuk mendapat daftar rekomendasi teman.



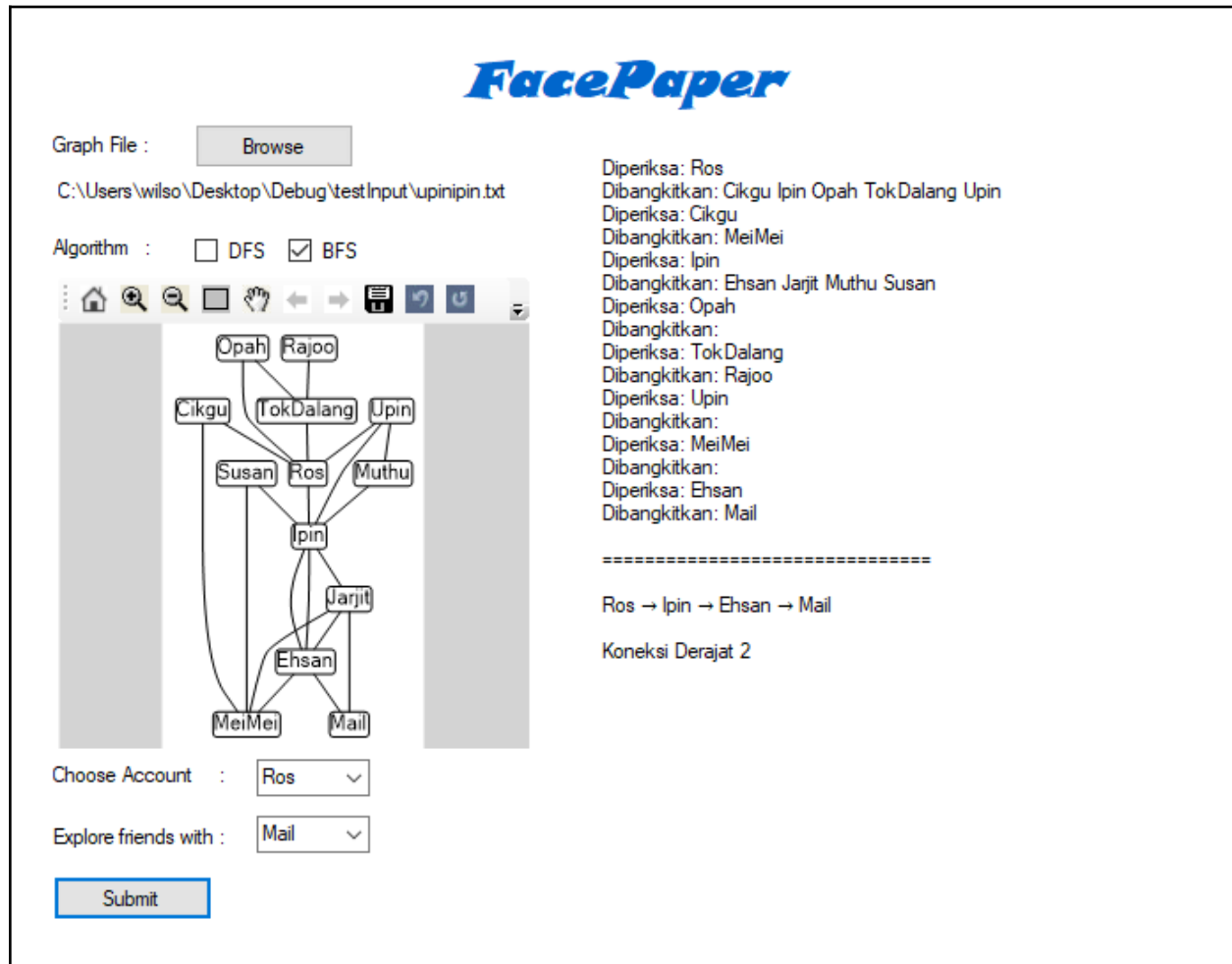
Gambar 4.4.4. Tampilan *Friend Recommendation* Akun MeiMei dari file *upinipin.txt*

Pada pengujian dibawah ini, simpul akun Ros ditelusuri secara *DFS* dengan prioritas abjad yang lebih awal, oleh karena itu urutan penelusurannya adalah Cikgu, MeiMei, Ehsan, Ipin, Jarjit baru ditemukan simpul Mail.



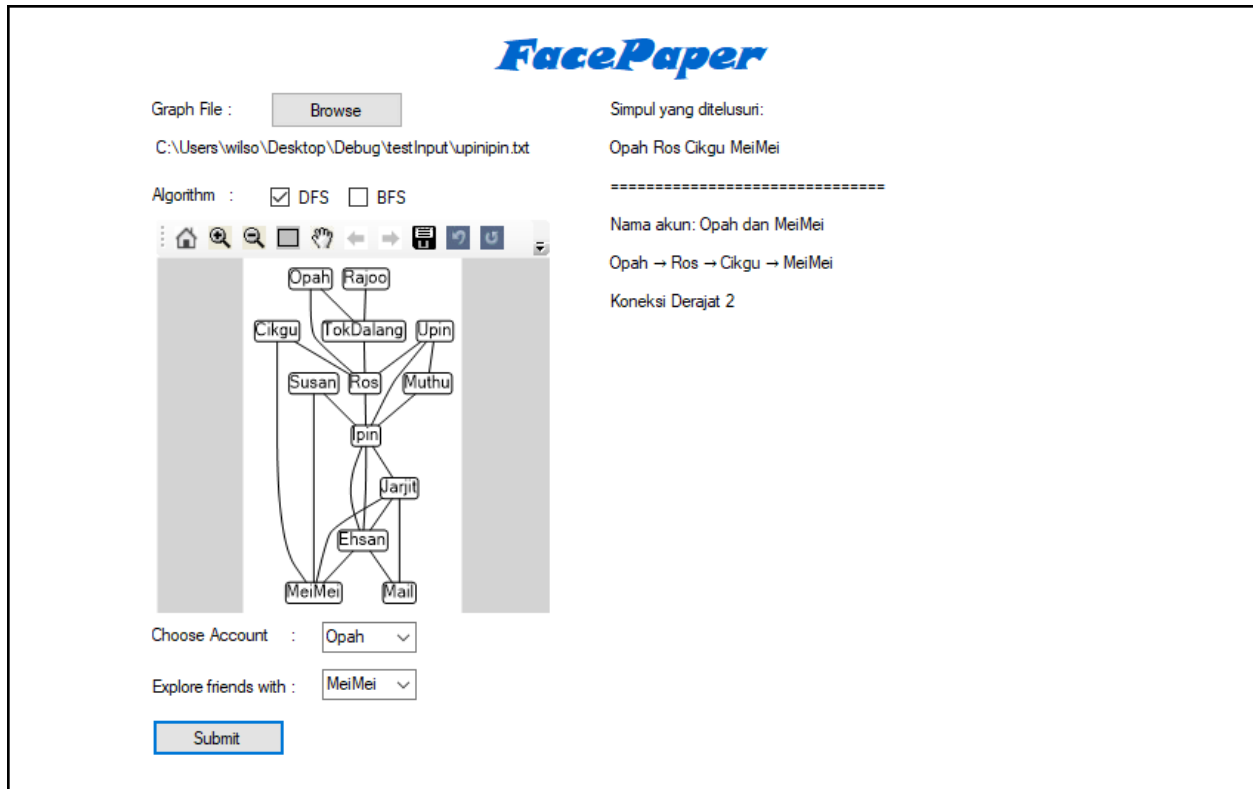
Gambar 4.4.5. Tampilan Fitur *Explore Friends* akun Ros dengan Mail (Algoritma *DFS*)

Dengan kedua akun yang sama pada skenario Gambar 4.4.5, secara *BFS* akan selalu ditemukan koneksi derajat terpendek walaupun membutuhkan penelusuran yang lebih banyak. Dapat dilihat bahwa dengan algoritma *BFS* ditemukan koneksi derajat 2, sedangkan dengan algoritma *DFS* ditemukan koneksi derajat 5 antara akun Ros dan Mail.

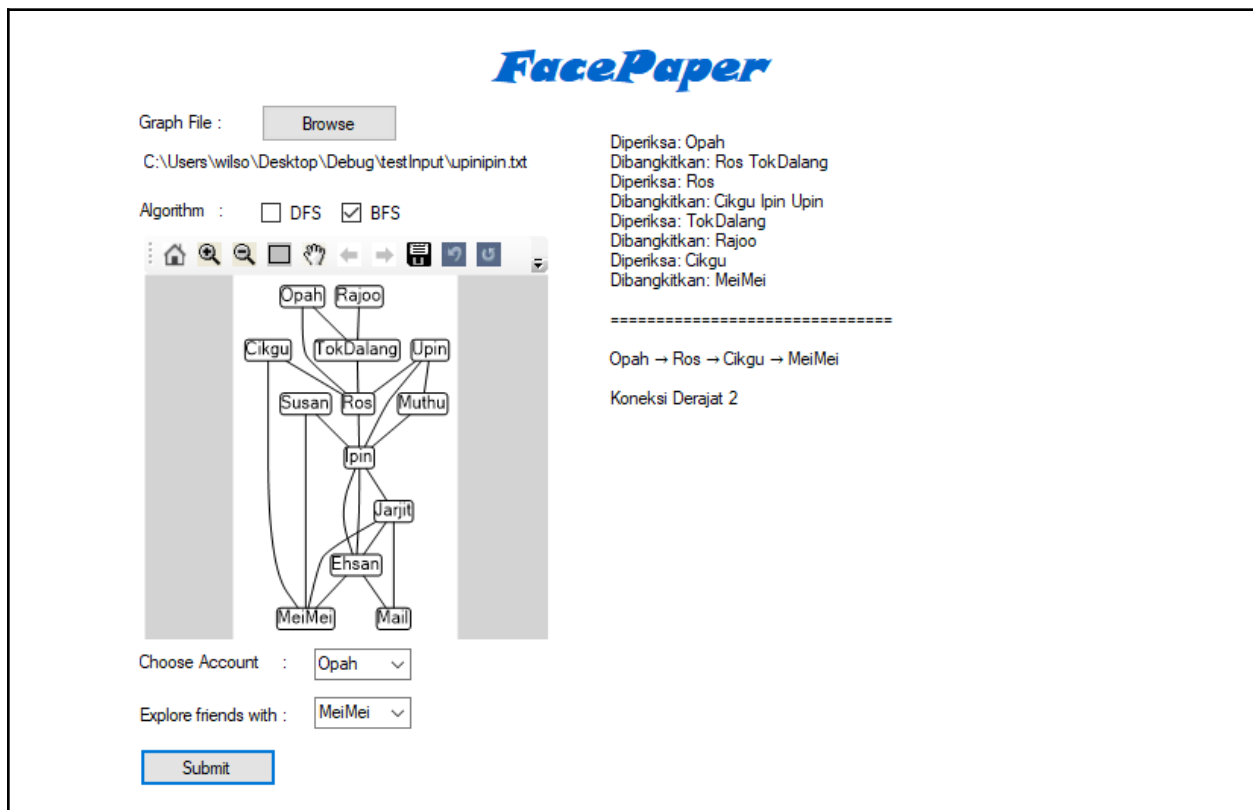


Gambar 4.4.6. Tampilan Fitur *Explore Friends* akun Ros dengan Mail (Algoritma *BFS*)

Namun terdapat juga skenario dimana *DFS* menghasilkan solusi koneksi derajat terpendek sama seperti *BFS* dan *DFS* menelusuri lebih sedikit simpul, seperti pada Gambar 4.4.7 dan 4.4.8.



Gambar 4.4.7. Tampilan Fitur *Explore Friends* akun Opah dengan MeiMei (Algoritma *DFS*)



Gambar 4.4.8. Tampilan Fitur *Explore Friends* akun Opah dengan MeiMei (Algoritma *BFS*)



#### 4.5 Analisis dari Desain Solusi Algoritma *BFS* dan *DFS*

Pada kasus Gambar 4.4.5 dan Gambar 4.4.6, dapat dilihat bahwa pencarian menggunakan *DFS* menghasilkan koneksi derajat 5, sedangkan dengan *BFS* dapat dihasilkan koneksi derajat 2. Hal tersebut disebabkan penelusuran simpul secara *DFS* pada program ini terurut secara abjad. Walaupun pada akhirnya ditemukan simpul solusi, namun tidak mendapatkan jalur terpendeknya. Sedangkan pada pencarian secara *BFS* dapat dipastikan akan mendapat sebuah simpul solusi dengan koneksi derajat terendah.

Pada kasus Gambar 4.4.7 dan Gambar 4.4.8 dapat dilihat bahwa baik pencarian menggunakan algoritma *DFS* dan *BFS* menghasilkan koneksi berderajat 2. Dapat dilihat pula pada pencarian ini, algoritma *DFS* menelusuri simpul-simpul dengan jumlah yang jauh lebih sedikit ketimbang algoritma *BFS*. Hal ini disebabkan, simpul solusi/*goal node* sesuai dengan penelusuran berdasarkan abjad simpul akun. Namun, hal ini tidak selalu terjadi, ada saatnya dimana dengan menggunakan *DFS* simpul solusi tidak ada pada akhir penelusuran dan perlu dilakukan *backtracking* dan akan memakan waktu yang lebih lama ketimbang menggunakan algoritma pencarian *BFS*.

Untuk menyimpulkan, algoritma pencarian secara *DFS* dapat menemukan solusi lebih cepat atau lambat ketimbang *BFS* dan jalur yang ditemukan belum tentu yang terpendek dikarenakan algoritma pencarian secara *DFS* mengurutkan simpul sesuai urutan abjad. Kelebihan dari algoritma pencarian *DFS* adalah menggunakan memori yang jauh lebih sedikit dari *BFS* karena tidak menyimpan simpul-simpul yang ditelusurinya. Pada kasus pencarian secara *BFS*, sudah dijamin akan menemukan jalur terpendek, walau kekurangannya terdapat pada penyimpanan semua simpul yang telah dibangkitkan. Dari adanya kekurangan pada kedua algoritma pencarian tersebut, baik *BFS* dan *DFS*, dikembangkan turunan dari algoritma pencarian ini yang dapat menyelesaikan persoalan lebih baik, seperti *Iterative Deepening Search (IDS)* dan *Depth Limited Search (DLS)*.

## **BAB V**

### **SIMPULAN**

#### **5.1 Kesimpulan**

Kesimpulan dari tugas besar ini adalah fitur *people you may know* pada jejaring sosial *Facebook* dapat diimplementasikan dengan menggunakan algoritma *Breadth First Search (BFS)* maupun *Depth First Search (DFS)* dari graf tak berarah yang dibangun berdasarkan pertemanan satu orang dengan yang lainnya. Masing-masing algoritma memiliki kelebihan dan kekurangannya bergantung pada graf yang dibangun. Namun dengan algoritma *BFS* akan selalu ditemukan jalur terpendek antara dua simpul yang terhubung.

#### **5.2 Saran**

Dalam pengerjaan tugas besar kedepannya dengan menggunakan bahasa pemrograman *C#* dengan kakas *Visual Studio .NET*, penulis memberikan beberapa saran, antara lain:

- a. Melakukan *research* terlebih dahulu terhadap bahasa pemrograman secara mendalam, baik dari *syntax* sampai jenis bahasa pemrograman tersebut.
- b. Mempelajari cara kerja kakas yang diperlukan untuk mendukung keberjalanan program agar dapat memanfaatkan seluruh *tools* yang ditawarkan dengan baik.
- c. Mencari referensi desain *GUI* yang lebih baik.

#### **5.3 Refleksi dan Komentar**

Pada tugas besar 2 strategi algoritma ini kami belajar untuk menggunakan bahasa pemrograman *C#* dengan kakas *Visual Studio .NET* yang belum pernah dipelajari sebelumnya. Kami juga belajar melakukan desain *GUI* yang baik serta fungsional untuk program ini. Komentar kami terhadap tugas besar ini adalah, walaupun cukup menantang karena kami dipaksa untuk menggunakan kakas yang tidak familiar, namun ilmu yang didapat sangat berguna untuk kedepannya.

## DAFTAR PUSTAKA

Munir, R. 2021. *Breadth First Search (BFS) dan Depth First Search (DFS) (2021)*. Bandung: Institut Teknologi Bandung.