TUGAS BESAR II IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2020/2021

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

Disusun oleh:

Kelompok Hope Peak's

Shaffira Alya Mevia

13519083

Febriawan Ghally Ar Rahman

13519111

Karina Imani

13519166

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

26 MARET 2020

BAB I
DESKRIPSI TUGAS

A. Deskripsi Persoalan

Facebook adalah salah satu pelopor jejaring sosial yang berdiri sejak tahun 2004. Kini, layanan tersebut sudah memiliki lebih dari 2,6 miliar pengguna. Karena banyaknya jumlah pengguna, fitur *friend recommendation* "People You May Know" menjadi sangat penting untuk merekomendasi orang-orang yang dapat ditambahkan sebagai teman berdasarkan jumlah *mutual friends* yang dimiliki.

Menggunakan prinsip *mutual friends*, akun A dan C yang belum saling kenal dapat direkomendasikan kepada satu sama lain melalui teman-teman yang dikenal oleh kedua akun tersebut, misalkan B. Semakin banyak *mutual friend* yang dimiliki kedua akun, maka semakin tinggi rekomendasi akun tersebut untuk ditambahkan sebagai teman.

Selain itu, layaknya semua platform media sosial lainnya, pengguna juga dapat mengeksplorasi akun-akun pengguna lainnya yang tidak memiliki *mutual friends* dengan akun pengguna tersebut sama sekali. Menelusuri graf keterhubungan antar akun dapat memberikan kita bayangan "jarak" berapa akun yang harus dilalui seseorang untuk berteman dengan akun lain yang tidak memiliki *mutual friend*.

Dalam tugas ini, akan diimplementasikan dua fitur utama yaitu *Friend Recommendation* dan *Explore Friends*. Input yang diperoleh berasal dari file .txt yang menyatakan keterhubungan antar node dalam graf, yang kemudian diolah menjadi representasi tertentu. Adapun, contoh file yang dimasukkan adalah sebagai berikut:

```
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
```

Angka 13 adalah jumlah baris yang dibaca, dan A B memiliki arti node A dan B terhubung dengan satu sisi. Graf kemudian diolah untuk memperoleh rekomendasi teman dan

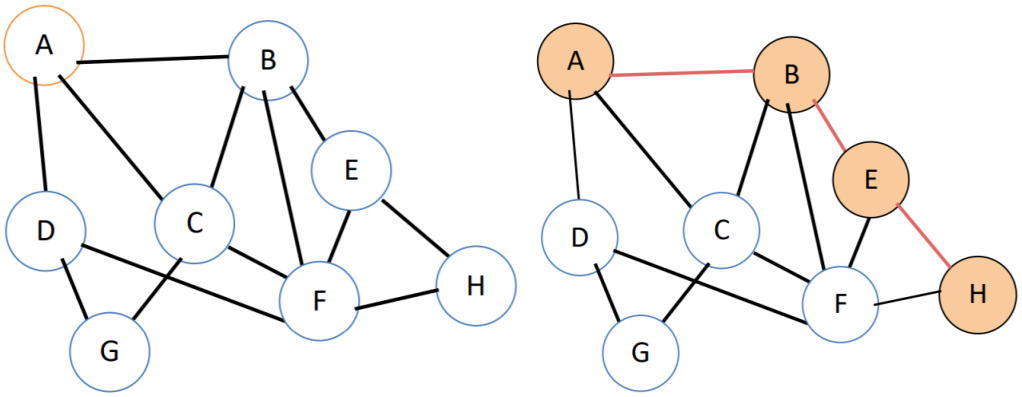
jarak antara dua akun dalam eksplorasi teman. Berdasarkan input suatu nama akun, keluaran yang diharapkan untuk *Friend Recommendation* terurut berdasarkan *mutual friend* terbanyak, yaitu sebagai berikut:

```
Daftar rekomendasi teman untuk akun A:

F - 3 mutual friends : B C D
G - 2 mutual friends : C D
E - 1 mutual friend  : B
```

Kemudian, untuk fitur *Explore Friends* akan ditelusuri graf untuk menemukan *Nth degree connection* antara dua akun yang tidak saling terhubung. Program menerima input dua nama akun yang terdapat dari graf dan pilihan apabila graf ingin ditelusuri secara BFS atau DFS. Graf dibangkitkan berdasarkan urutan abjad. Adapun, keluaran yang diharapkan:

```
Nama akun: A dan H

2nd-degree connection
A → B → E → H
```

Aplikasi yang dibangun berbasis GUI, dengan spesifikasi dapat menerima berkas file untuk graf dan pilihan-pilihan pengguna seperti akun yang ingin dicari dan metode pencarian (BFS/DFS). Selain itu, juga terdapat visualisasi graf berupa graf awal, dan graf yang sudah dicari jalur keterhubungan dua akunnya, yaitu sebagai berikut:

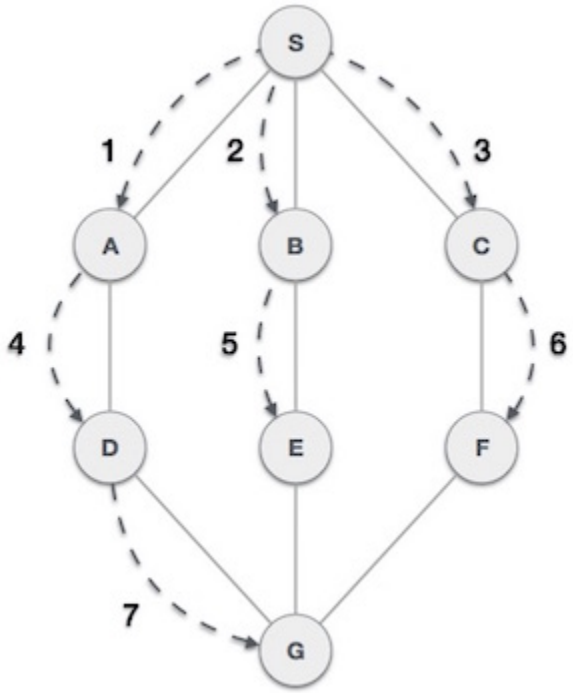


Gambar 1.1.1 Contoh visualisasi graf (kiri) dan jalur antara dua akun pada graf (kanan).

BAB II
LANDASAN TEORI

1. Algoritma Breadth First Search (BFS)

Breadth First Search (BFS atau disebut juga pencarian melebar) adalah salah satu metode traversal pohon atau graf yang dilaksanakan dengan mengunjungi terlebih dahulu semua tetangga dari suatu node sebelum melanjutkan ke semua tetangga tiap node dalam "kedalaman" yang lebih besar. Ilustrasinya adalah sebagai berikut:



Gambar 2.1.1 Urutan pembangkitan BFS (Sumber: *Tutorials Point*).

Dalam implementasinya, algoritma ini memanfaatkan queue pencarian, dimana daftar seluruh tetangga suatu node dimasukkan ke dalam queue pada saat node tersebut dikunjungi. Karena sifat queue yang first in, first out (FIFO), pencarian dilakukan secara melebar sesuai konsep BFS. Adapun, algoritmanya adalah sebagai berikut:

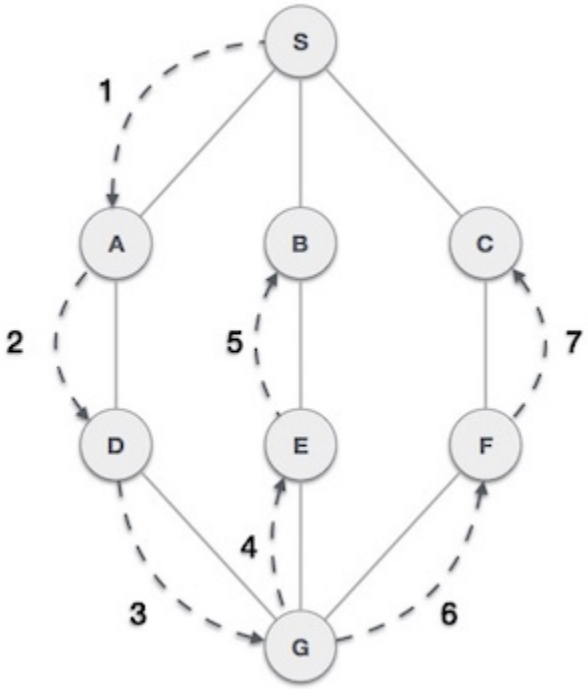
```
queue q { Antrian kosong }
array dikunjungi { Inisialisasi 0 }

enqueue(q, v) { Masukkan simpul awal kunjungan ke dalam antrian }
dikunjungi[v] = true { Catat simpul awal sebagai sudah dikunjungi }

while not IsEmpty(q) do
    dequeue(q, v)
    for each w yang bertetangga dengan v do
```

```
        if not dikunjungi[w] then
            enqueue(q, w)
            dikunjungi[w] = true
        endif
    endfor
endwhile
```

Selain itu, untuk implementasi BFS dalam graf digunakan juga array tuple yang berisi nama tiap node dan bilangan 0/1 yang menyatakan apakah node tersebut sudah dikunjungi sebelumnya, karena dalam graf terdapat siklus-siklus.

2. Algoritma Depth First Search (DFS)

Depth First Search (DFS atau disebut juga pencarian mendalam) adalah satu lagi metode traversal pohon atau graf. Bedanya, metode ini menelusuri satu tetangga tiap node secara menyeluruh sampai obyek pencarian ditemukan atau tidak ada lagi tetangga yang dapat dikunjungi. Jika hal kedua terjadi, pencarian akan mundur satu node dan mencari tetangga lain yang belum dikunjungi. Ilustrasinya adalah sebagai berikut:



Gambar 2.2.1 Urutan pembangkitan DFS (Sumber: *Tutorials Point*).

Dalam implementasinya, algoritma ini memanfaatkan stack, dimana setiap node yang dikunjungi didorong (push) ke dalam stack. Ketika obyek pencarian belum ditemukan dan tidak

ada lagi tetangga yang dapat dikunjungi, node terakhir yang dikunjungi akan dikeluarkan (pop) dari stack untuk menyatakan node yang dikunjungi mundur selangkah. Algoritmanya:

```
array dikunjungi { Inisialisasi 0 }

dikunjungi[v]

w traversal [1..n]
    if w bertetangga dengan v then
        if not dikunjungi[w] then
            DFS(w)
        endif
    endif
endfor
```

Implementasi DFS untuk graf juga menggunakan array tuple seperti BFS untuk menyimpan data node yang sudah dikunjungi. Ketika node yang dikunjungi mundur selangkah, DFS mengingat node yang sudah dikunjungi agar tidak mengunjunginya lagi.

3. C# Desktop Application Development

C# adalah bahasa pemrograman yang berorientasi pada objek dan komponen yang dibuat oleh Microsoft pada tahun 2000. Namun, penggunaan C# masih prevalen hingga sekarang, dengan stable version terakhir dirilis pada tanggal 10 November 2020, sekitar 4 bulan yang lalu. Bahasa pemrograman ini dijalankan pada framework .NET, sebuah virtual execution system yang disebut common language runtime (CLR). CLR sendiri merupakan implementasi Microsoft terhadap CLI (common language infrastructure) yang digunakan dalam standar internasional.

Bahasa C# memiliki beberapa fitur yang berguna dalam pemrograman dan pembuatan aplikasi perangkat lunak. Beberapa di antaranya adalah garbage collection, yang langsung mengembalikan memori dari obyek-obyek yang tidak digunakan atau tidak dapat dicapai oleh program; nullable types, yang menanggulangi variabel yang tidak menunjuk ke obyek yang sudah dialokasikan sebelumnya; exception handling, yang memberikan pendekatan terstruktur untuk deteksi dan perbaikan berbagai macam error. Selain itu, masih banyak lagi fitur-fitur yang ditawarkan C# yang dapat mempermudah programming.

Bahasa ini cocok digunakan untuk mengerjakan proyek-proyek yang memungkinkan untuk dikembangkan di masa mendatang. Bahasa ini memiliki readability dan kemudahan pengaksesan yang cukup baik, sehingga dapat mempermudah developer dalam mengembangkan proyek-proyeknya.

BAB III
ANALISIS PEMECAHAN MASALAH

A. Langkah Pemecahan Masalah

Pemecahan masalah diawali dengan memecah deskripsi persoalan yang diberikan menjadi beberapa modul terpisah yang dapat diimplementasikan satu-persatu, yakni:

- Input file dan representasi graf
- Fitur *friend recommendation*
- Fitur *explore friends* menggunakan BFS
- Fitur *explore friends* menggunakan DFS
- Implementasi GUI

1. Input file dan representasi graf

Representasi graf memiliki pertimbangan pemilihan struktur data yang paling memudahkan implementasi fitur-fitur lainnya. *Dictionary* dipilih karena kemudahan akses, menambah dan mengurangi elemen, serta dipisahkannya tiap elemen menjadi *key* dan *value*. Pada implementasi graf program ini, *key* adalah nama *vertice* (*string*), dan *value* adalah *hashmap* dari setiap nama *vertice* yang bertetangga dengan *vertice* tersebut.

2. Fitur *friend recommendation*

Method Friend Recommendation sebenarnya merupakan implementasi sederhana dari Algoritma BFS dan memiliki batasan pencarian hingga pada *depth* kedua untuk mendapatkan semua kemungkinan himpunan penyelesaian. Kemudian dilakukan pengecekan kembali untuk mengeliminasi *node-node* yang merupakan himpunan *adjacent* dari *node* yang dicari.

3. Implementasi GUI

Dalam mengimplementasikan *Graphical User Interface* atau GUI yang didesain, diperlukan tiga buah Windows Form yang terdiri dari halaman utama, halaman untuk *Explore Friends*, dan halaman untuk *Friend Recommendation.* Untuk menampilkan bentuk graf, digunakan *library* tambahan Microsoft Auto Graph Layout.

B. Elemen Algoritma BFS dan DFS

Algoritma BFS dan DFS yang digunakan keduanya diimplementasikan dengan *loop*. Fungsi menerima input berupa *string* a (*vertice* awal pencarian) dan *string* b (*vertice* yang dicari jalurnya), kemudian mengembalikan *List* <*string*> yang berisi nama-nama *vertice* yang dilalui saat menelusuri graf dari a ke b.

Selain itu, fungsi juga mengimplementasikan satu lagi *Dictionary* yang berisi *tuple (string, integer)*. *Integer* diinisialisasi dengan 0, dan akan diubah menjadi 1 ketika *vertice* sudah dikunjungi. Pengulangan dari algoritma mengunjungi tiap *vertice* sendiri menggunakan *while loop* yang tidak akan berhenti sampai b ditemukan atau semua *vertice* sudah dikunjungi.

Perbedaan utama algoritma BFS dan DFS yang digunakan adalah penyimpanan *path* yang dicari. Dalam BFS, diimplementasikan *queue vertice* yang akan dikunjungi berikutnya dalam t*uple (string, List<string>*), di mana *List<string>* digunakan untuk menyimpan jalur sejauh ini antara *vertice* awal dan *vertice* yang sedang dikunjungi. Oleh karena itu, *Current Vertice* tidak hanya berisi nama *vertice* yang sedang dikunjungi, tapi juga *path* menuju *vertice* tersebut. *Queue tuple* ini sangat berguna karena dalam BFS yang akan mengembalikan jalur, harus mengingat jalur dari semua *vertice* yang dikunjungi dalam pencarian melebar. Ketika pencarian berakhir, fungsi akan mengembalikan *item* kedua dalam *tuple* yang berisi *path*.

Sementara itu, dalam DFS diimplementasikan *List<array>* yang sudah berfungsi sebagai *path*. Oleh karena itu, *Current Vertice* pada DFS hanya berupa *string* yang menyatakan nama *node* yang sedang dikunjungi. Karena urutan pembangkitan pada graf sesuai abjad, dan *dictionary* pada graf sudah diatur sehingga terurut menaik, fungsi akan langsung menelusuri *vertice* pertama yang merupakan tetangga *Current Vertice* dan belum dikunjungi. Ketika b belum ditemukan dan tidak ada lagi tetangga yang belum dikunjungi, *Current Vertice* akan bergerak mundur selangkah ke node terakhir yang dikunjungi untuk mencari jalur lain. Gerakan mundur ini serupa dengan *pop* pada *stack* yang bersifat *last in, first out (LIFO)*.

Pencarian pada BFS dan DFS juga dihentikan pada kondisi yang berbeda. Pada BFS, pencarian dihentikan ketika *queue* sudah kosong, yang berarti semua *node* sudah dikunjungi. Pada DFS, pencarian dihentikan ketika sudah kembali ke node a (tidak dapat mundur lagi) dan semua tetangga a telah dikunjungi secara DFS.

C. Contoh Kasus

Berikut ini adalah beberapa contoh kasus yang akan dimasukkan ke dalam file-file .txt untuk divisualisasi sebagai graf, serta digunakan dalam pencarian *Friend Recommendation* dan *Explore Friends*.

1. Graf 1 (sesuai contoh pada spesifikasi)

```
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
```

2. Graf 2

```
14
A B
B C
B E
C D
D E
E F
F G
F H
F I
G J
H I
H J
J K
K J
```

3. Graf 3

```
10
A B
A C
B E
C D
D H
D I
E F
E G
F G
H I
```

4. Graf 4

```
15
0 1
0 3
0 4
1 2
1 3
1 7
```

```
1 8
2 3
2 6
3 4
3 5
3 6
4 5
5 6
7 8
```

5. Graf 5

```
10
V1 V2
V2 V3
V2 V5
V2 V6
V3 V4
V3 V6
V4 V5
V4 V6
V5 V6
```

6. Graf 6

```
11
Andy Benny
Andy Charlie
Andy David
Benny David
Benny Evan
Charlie David
Charlie Fred
David Evan
David Gavin
Evan Gavin
Fred Gavin
```

BAB IV
IMPLEMENTASI DAN PENGUJIAN

A. Pseudocode

Berikut ini adalah pseudocode yang digunakan untuk fitur *friend recommendation*.

```
function FriendRec (input string a, input Dictionary<string,HashSet<string>
map) → Dictionary<String, List<String>> result
{Mencari Rekomendasi teman dengan menggunakan BFS}

      HashSet<string> mutual {Mutual dari a dan juga recommended friend
      Dictionary<string,Hashset<string>> recommend
      foreach(val in map[a]) do
            mutual.Add(val)  {Assign mutual}
      foreach (test in mutual)
            foreach(val in map[test])
                  if(val not in mutual and val!=a)
                      recommend.Add(val,{})
      foreach (val in recommend)
            foreach(name in map[val.key])
                  foreach(test in mutual)
                        if(name==test) then
                              recommend[val].Add(name)

      Dictionary<String, List<String>> result
      foreach (val in recommend sortByValue,decreasing)
            List<String> mutuals
            foreach(name in recommend[val])
                  mutuals.Add(name)
            result.Add(val.Key, mutuals)
```

Pada fitur *explore friends* memanggil dua buah fungsi tergantung pilihan pengguna, yaitu pencarian menggunakan BFS maupun DFS. Adapun, pseudocode kedua fungsi tersebut adalah sebagai berikut.

```
function exploreBFS (input string a, input string b) → List <string> path
{ Mencari path dari a ke b menggunakan BFS }

      Queue q { Buat queue baru, akan diisi tuple (string, List <string>) }
      Boolean found = true { Menyatakan apakah pencarian belum selesai }

      Array check { Buat array baru, akan diisi tuple (string, int) }

      { Masukkan semua vertice dengan value 0 (belum dikunjungi) }
      for each (node dalam graf) do
            check.Add(node, 0)

      { Inisialisasi vertice awal (a dengan path kosong) }
      Tuple <string, List <string>> CurrentVertice = (a, {})
```

```
      while (CurrentVertice[1] != b and found) do
            check[CurrentVertice] = 1
            for each (node tetangga CurrentVertice[1]) do
                  if check[node] == 0
                        { Buat item queue baru berisi node yang belum
dikunjungi dan path ke node tersebut dari a }
                        Tuple <string, List <string>> tpl = (node,
CurrentVertice[2].Add(CurrentVertice[1]))
                        q.Enqueue(tpl)
            if q.Count = 0 then
                  found = true
            else
                  CurrentVertice = q.Dequeue()

      { Jika tidak ditemukan, kembalikan path awal saja }
      if not found then
            path = {a}
      else
            path = CurrentVertice[2].Add(CurrentVertice[1])
```

```
function exploreDFS (input string a, input string b) → List <string> path
{ Mencari path dari a ke b menggunakan DFS }

      Boolean found = true { Menyatakan apakah pencarian belum selesai }

      Array check { Buat array baru, akan diisi tuple (string, int) }

      { Masukkan semua vertice dengan value 0 (belum dikunjungi) }
      for each (node dalam graf) do
            check.Add(node, 0)

      { Inisialisasi vertice awal, string a }
      string CurrentVertice = a

      while (CurrentVertice != b and found) do
            if (CurrentVertice == a and
allNeighborsVisited(CurrentVertice)) then
                  found = false
            else if (allNeighborsVisited(CurrentVertice)) then
                  path.Remove(CurrentVertice)
                  CurrentVertice = path[path.Count - 1]
            else
                  for each (node tetangga CurrentVertice) do
                        if check[node] != 1 then
                              CurrentVertice = node
                              path.Add(CurrentVertice)
                              break
```

B. Struktur Data

Berikut ini adalah daftar *class* dan *method* yang digunakan dalam program.

1. Class Program

Class program mencakup *method main* atau program utama. Pada *class* ini berisi *method-method* untuk inisialisasi dan memulai *graphical user interface* untuk aplikasi.

2. Class Functions

a. Atribut

```
protected List<string> file = new List<string>();
{ Tempat penyimpanan file }

Dictionary<string, HashSet<string>> graf = new
Dictionary<string, HashSet<string>>();
{ Kamus penyimpanan vertice dan daftar tetangga tiap vertice }
```

b. Method

```
public void BacaFile(string location)
{ Membaca file dari path location }

public List<string> getFile()
{ Getter untuk atribut file }

public void AddGraphIfNotExist(string key, string val)
{ Menambahkan vertice dan tetangganya ke atribut graf }

public Dictionary<string, HashSet<string>> getGraf()
{ Getter untuk atribut graf }

public Dictionary<String, List<String>>
friendRecommendation(string chosenAccount)
{ Function untuk mencari rekomendasi teman berdasarkan jumlah
mutual friends yang dimiliki }
{ Input: nama akun yang ingin dicari rekomendasi temannya }
{ Output: Dictionary nama akun yang direkomendasi dan list nama
akun mutual friend yang dimiliki }

public List<string> exploreFriends(string chosenAccount, string
exploreFriendAccount, Boolean useDFS)
{ Function utama untuk melakukan branching ke BFS dan DFS }
{ Input: chosenAccount → vertice awal, exploreFriendAccount →
vertice goal, useDFS untuk pemilihan algoritma }
{ Output: jalur yang menghubungkan chosenAccount dan
exploreFriendAccount dalam bentuk array of string }

public List<string> exploreBFS(string a, string b)
{ Mencari jalur dari a ke b menggunakan BFS }

public List<string> exploreDFS(string a, string b)
{ Mencari jalur dari a ke b menggunakan DFS }
```

```
public bool allNeighborsVisited(string vertice,
Dictionary<string, int> check)
{ Mengecek apakah semua tetangga vertice sudah dikunjungi }

public List<string> deepCopyIsh(List<string> reference)
{ Mengembalikan list baru hasil duplikasi dari list reference }
```

C. Tata Cara Penggunaan Program

Untuk menjalankan aplikasi e-Handbook : Socials Tab dapat langsung membuka *executable file*-nya pada *folder* bin. Penggunaan aplikasi ini sendiri sangatlah mudah karena yang perlu dilakukan adalah untuk mengunggah file berekstensi .txt yang berisi graf sesuai dengan format, kemudian memilih algoritma yang akan digunakan, memilih akun pertama dan kedua untuk menggunakan fitur.

Gambar 4.3.1 Tampilan awal e-Handbook : Socials Tab saat pertama kali dibuka

Pada tampilan terlihat ada tombol *browse*, yang ketika ditekan akan mengeluarkan dialog yang hanya bisa membaca file dengan ekstensi .txt. Saat file berhasil dibaca oleh aplikasi, secara otomatis akan muncul gambar graf dari file tersebut pada *viewer* di bagian kanan. Kedua *dropdown* untuk memilih akun juga secara otomatis terisi sehingga pengguna dapat memilih akun dengan mudah.

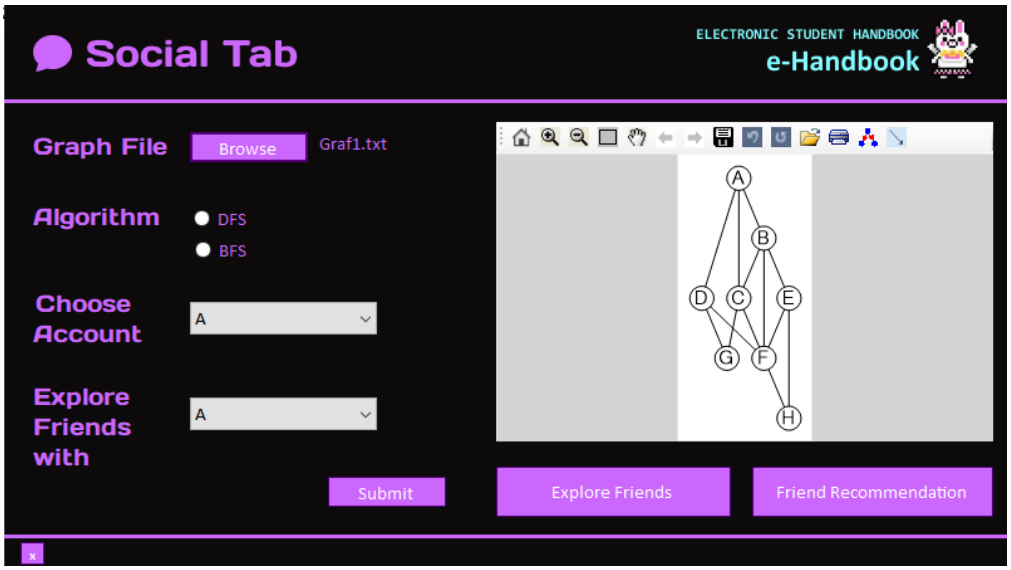
Setelah memilih algoritma dan kedua akun, data akan dikirim dengan menekan tombol *submit*. Apabila langsung menekan tombol *Explore Friends* atau *Friend Recommendation* akan mengeluarkan pesan *error* karena belum ada proses yang terjadi. Namun, apabila pengguna sudah menekan *submit*, maka hasil dari kedua fitur dapat langsung dilihat ketika menekan salah satu tombol tersebut. Apabila pengguna ingin keluar dari aplikasi, terdapat tombol *exit* pada bagian kiri bawah.

D. Pengujian dan Analisis

Berikut ini adalah beberapa graf dan input yang digunakan untuk pengujian, berikut tangkapan layar hasil yang diperoleh. Adapun nama graf yang dicantumkan sesuai nama graf pada contoh kasus bab sebelumnya.

1. Graf 1

   a. Halaman utama

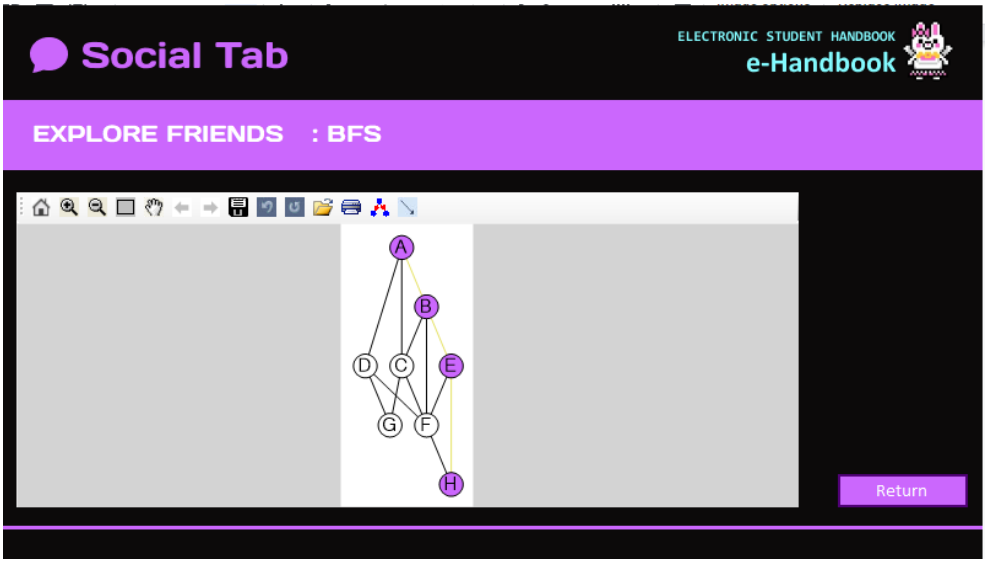


Gambar 4.4.1.1 Halaman Utama Aplikasi

   b. *Friend recommendation* untuk A

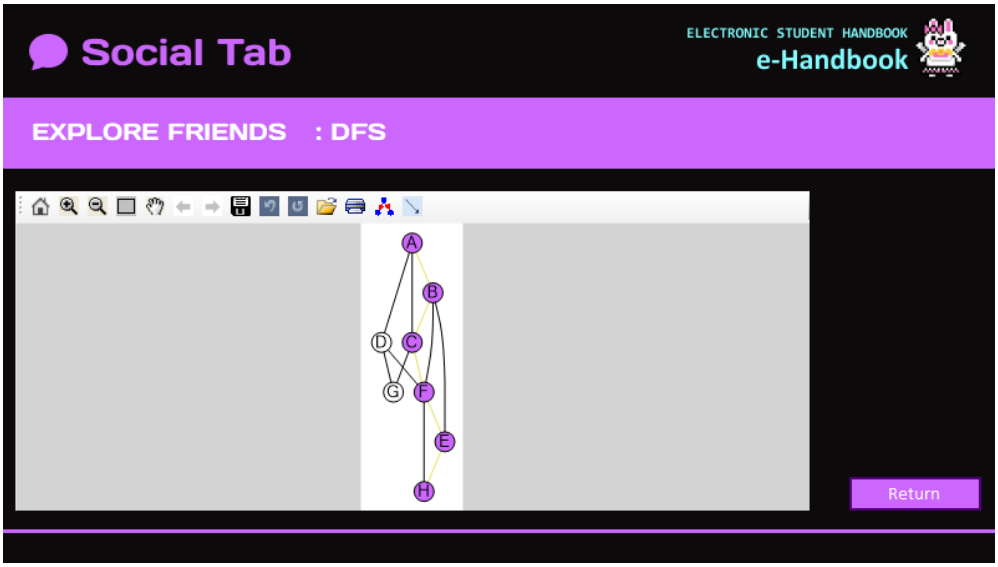
Gambar 4.4.1.2 Tampilan hasil *friend recommendation*

c. Jalur A ke H secara BFS



Gambar 4.4.1.3 Tampilan hasil *explore friends* dari A ke H dengan BFS

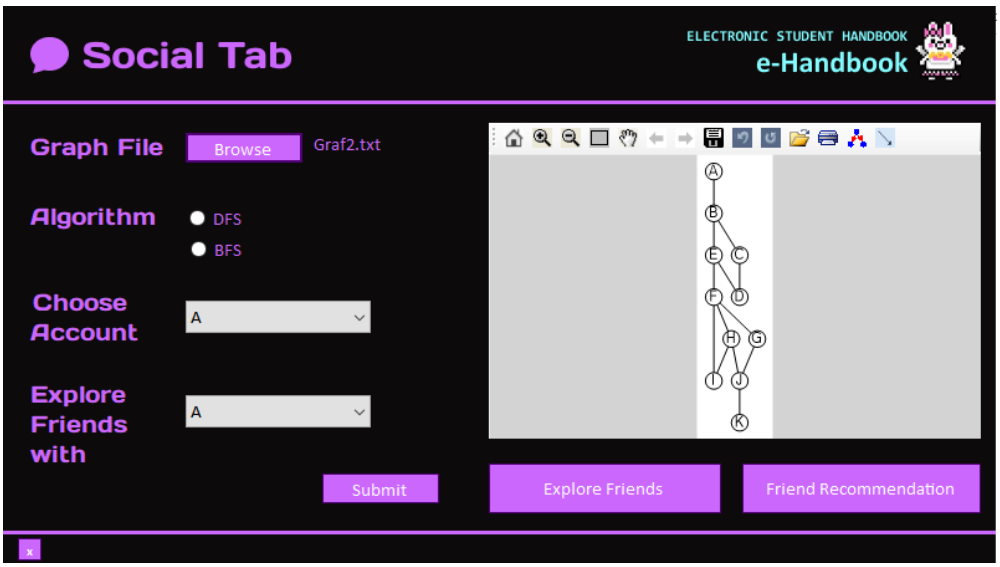
d. Jalur A ke H secara DFS



Gambar 4.4.1.4 Tampilan hasil *explore friends* dari A ke H dengan DFS

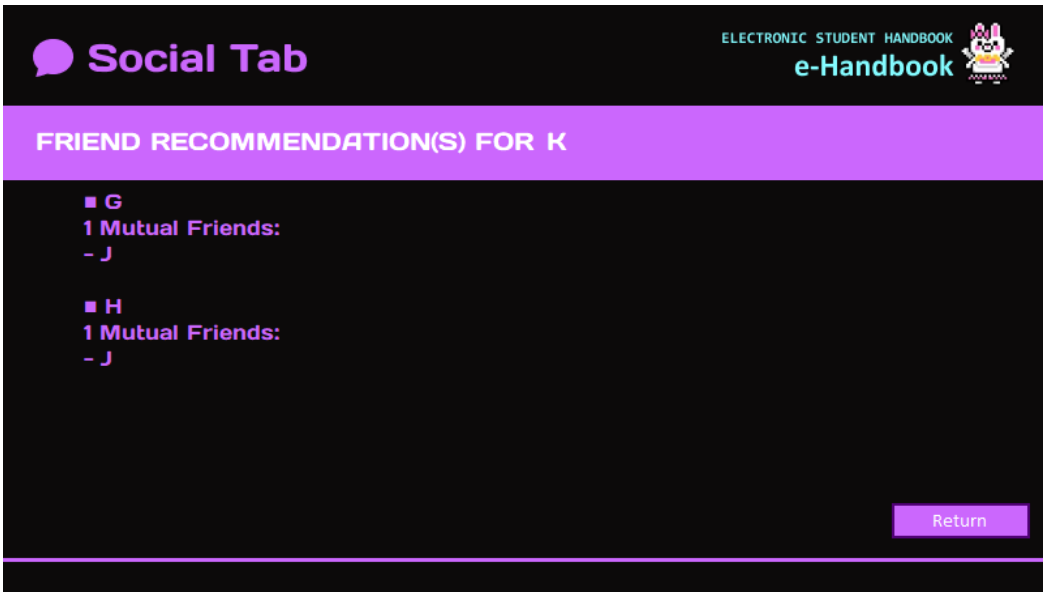
2. Graf 2

a. Halaman utama



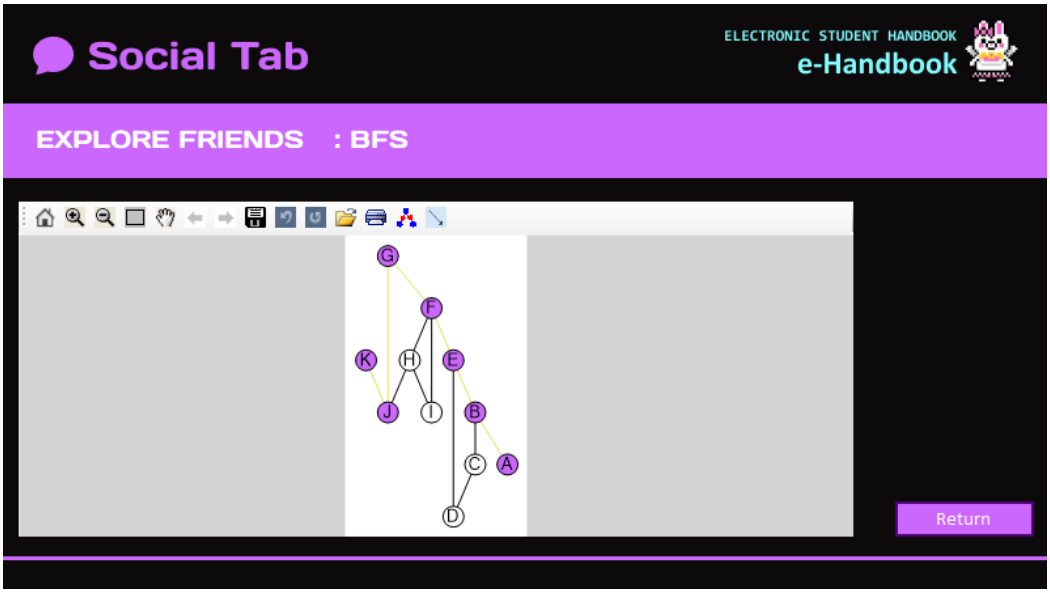
Gambar 4.4.2.1 Halaman Utama Aplikasi

b. *Friend recommendation* untuk K



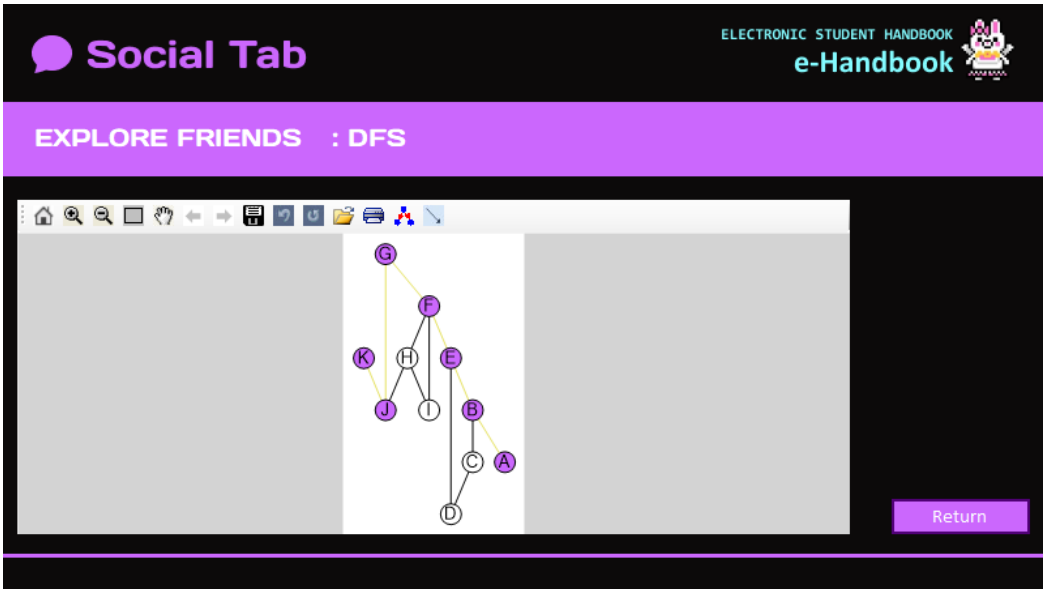
Gambar 4.4.2.2 Tampilan hasil *friend recommendation*

c. Jalur K ke A secara BFS



Gambar 4.4.2.3 Tampilan hasil *explore friends* dari K ke A dengan BFS

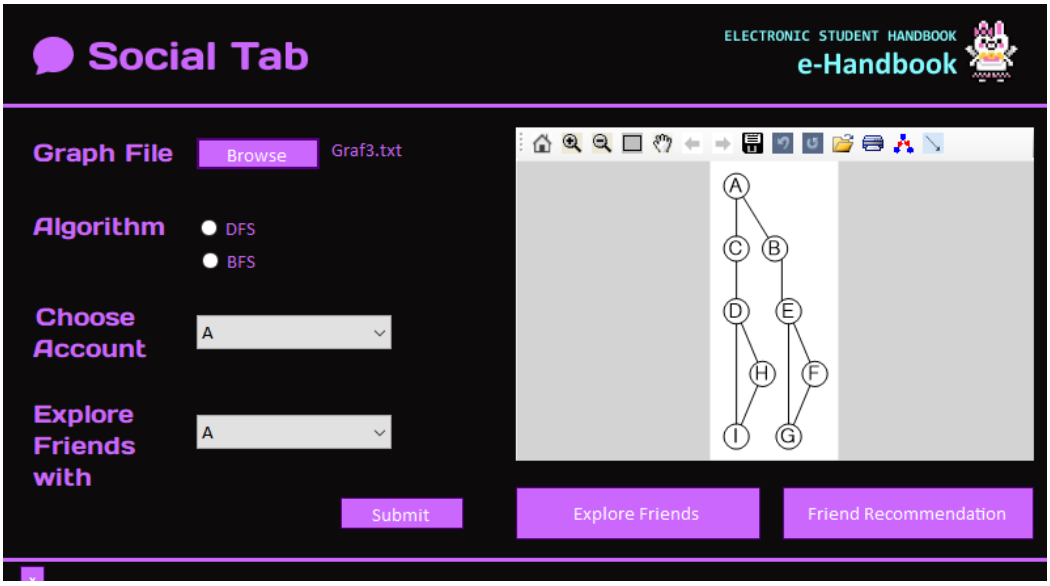
d. Jalur K ke A secara DFS



Gambar 4.4.2.4 Tampilan hasil *explore friends* dari K ke A dengan DFS

3. Graf 3

a. Halaman utama

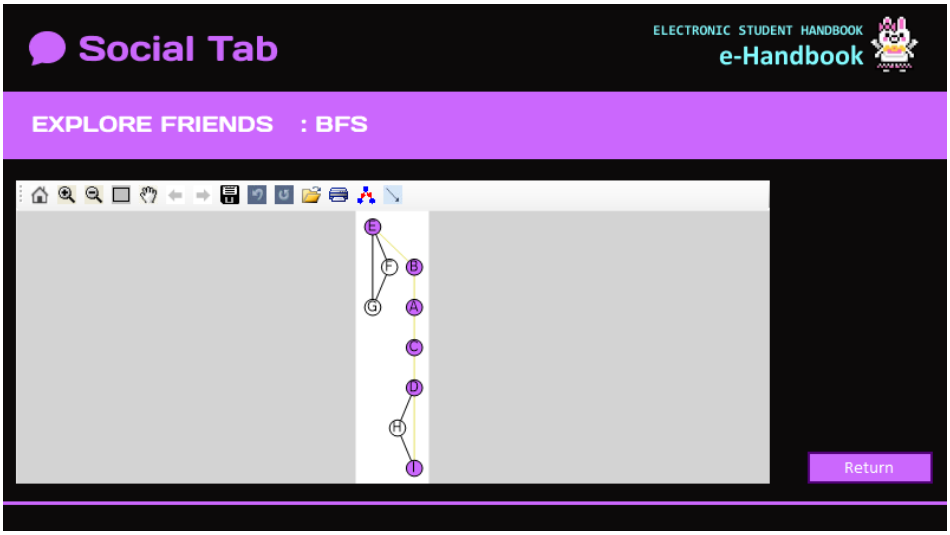


Gambar 4.4.3.1 Halaman Utama Aplikasi

b. *Friend recommendation* untuk E

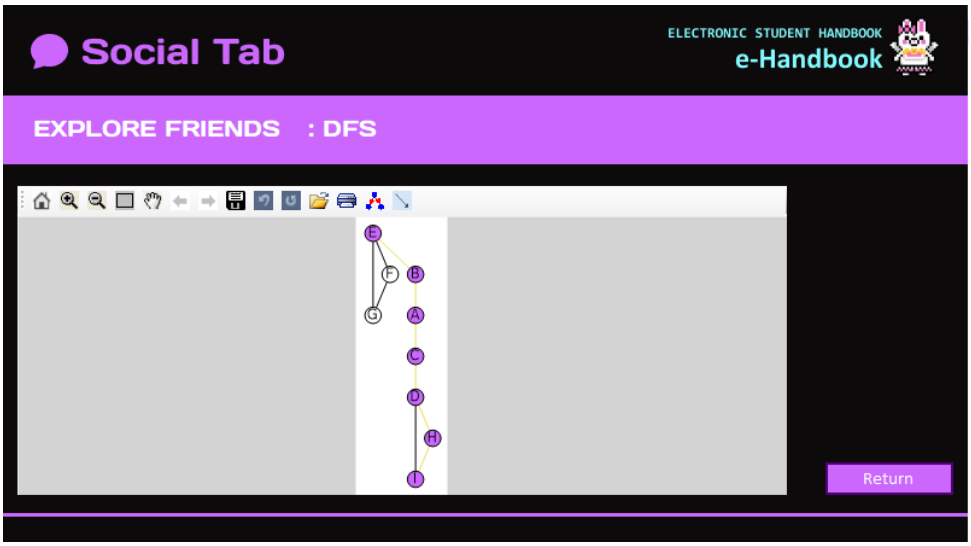
Gambar 4.4.3.2 Tampilan hasil *friend recommendation*

c. Jalur E ke I secara BFS



Gambar 4.4.3.3 Tampilan hasil *explore friends* dari E ke I dengan BFS

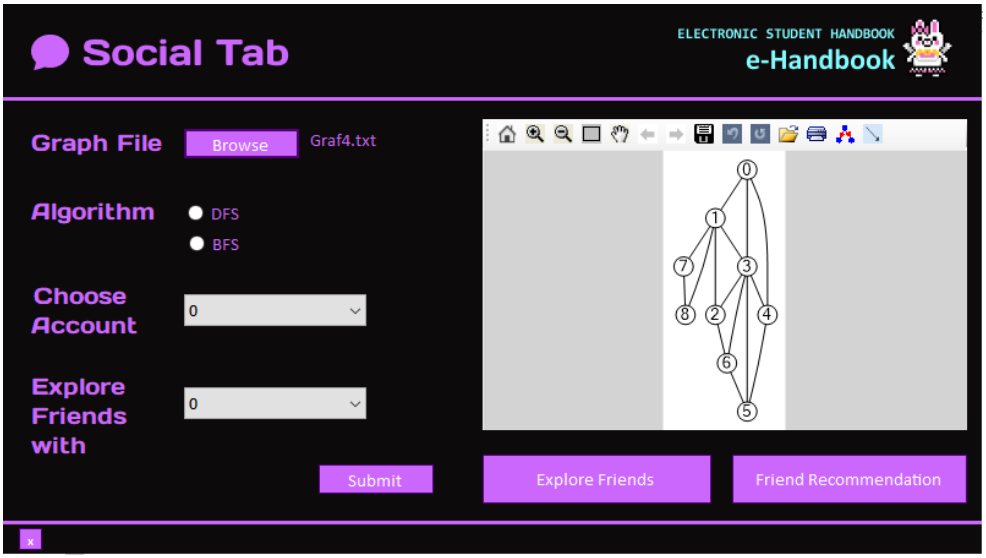
d. Jalur E ke I secara DFS



Gambar 4.4.4.4 Tampilan hasil *explore friends* dari E ke I dengan BFS

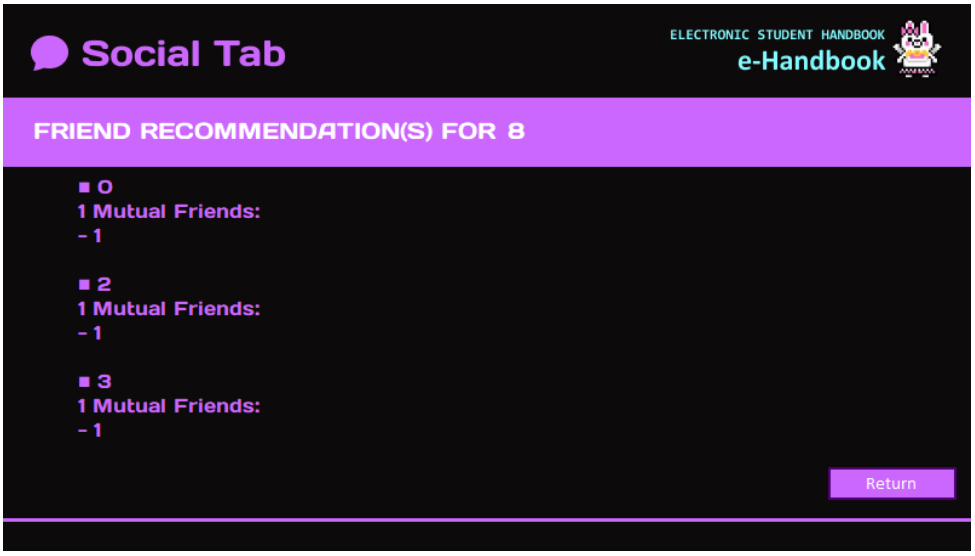
4. Graf 4

a. Halaman utama



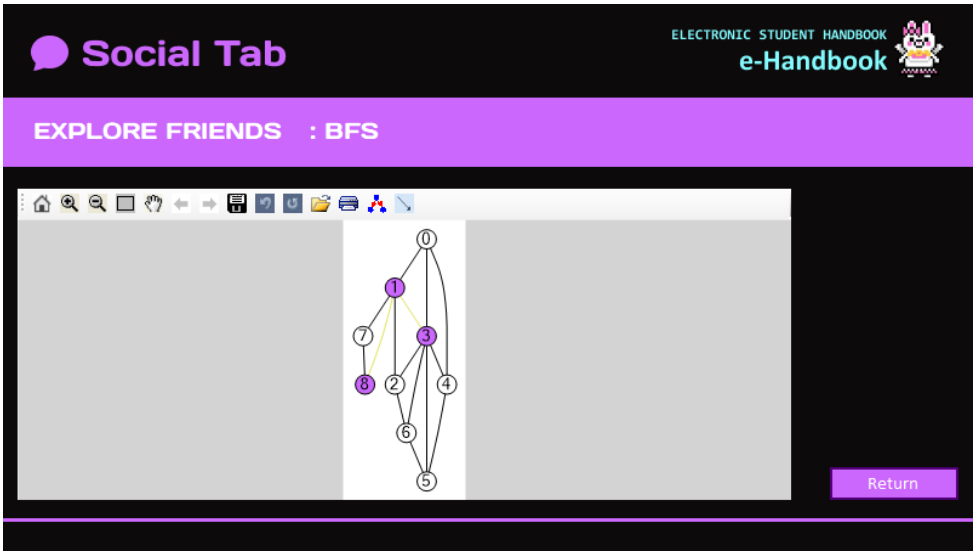
Gambar 4.4.4.1 Halaman Utama Aplikasi

b. Friend recommendation untuk 8



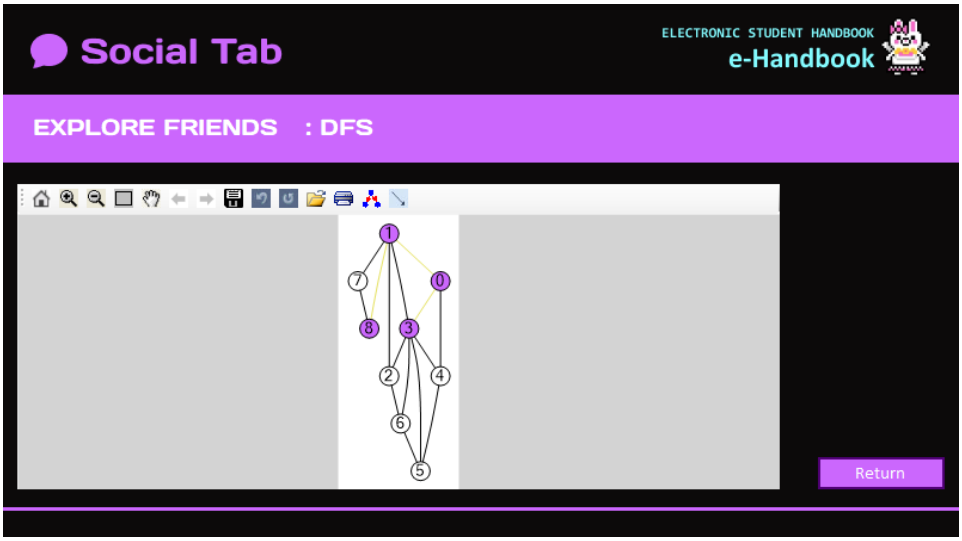
Gambar 4.4.4.2 Tampilan hasil *friend recommendation*

c. Jalur 8 ke 3 secara BFS



Gambar 4.4.4.4 Tampilan hasil *explore friends* dari 8 ke 3 dengan BFS

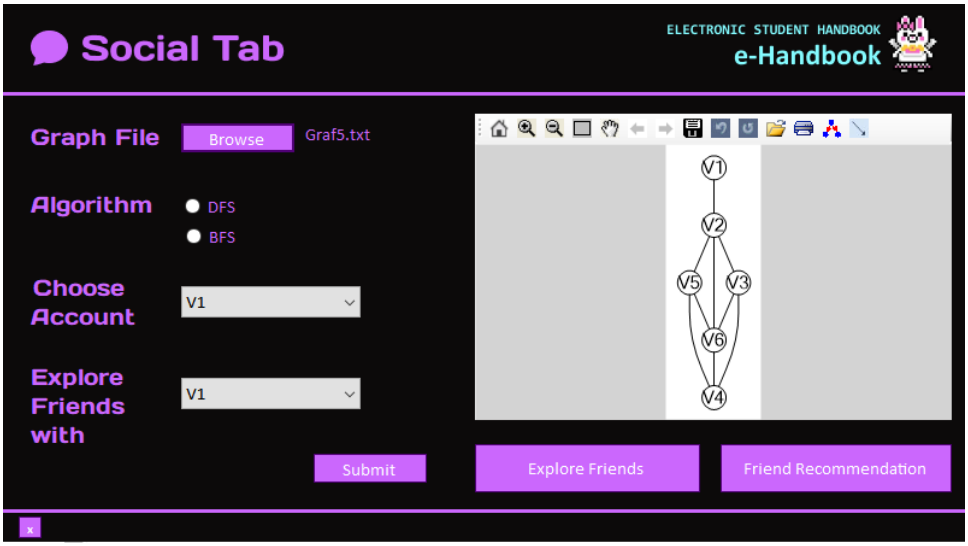
d. Jalur 8 ke 3 secara DFS



Gambar 4.4.4.4 Tampilan hasil *explore friends* dari 8 ke 3 dengan DFS

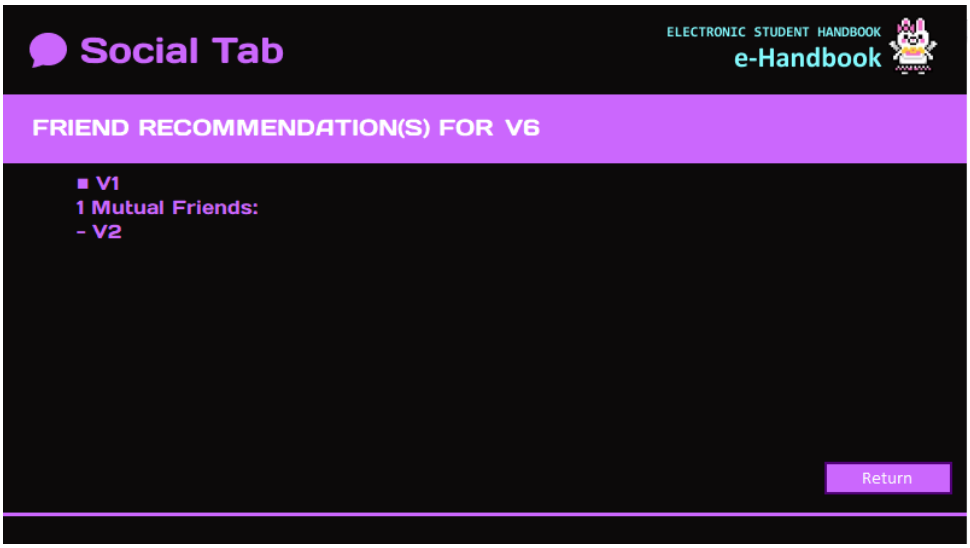
5. Graf 5

a. Halaman utama



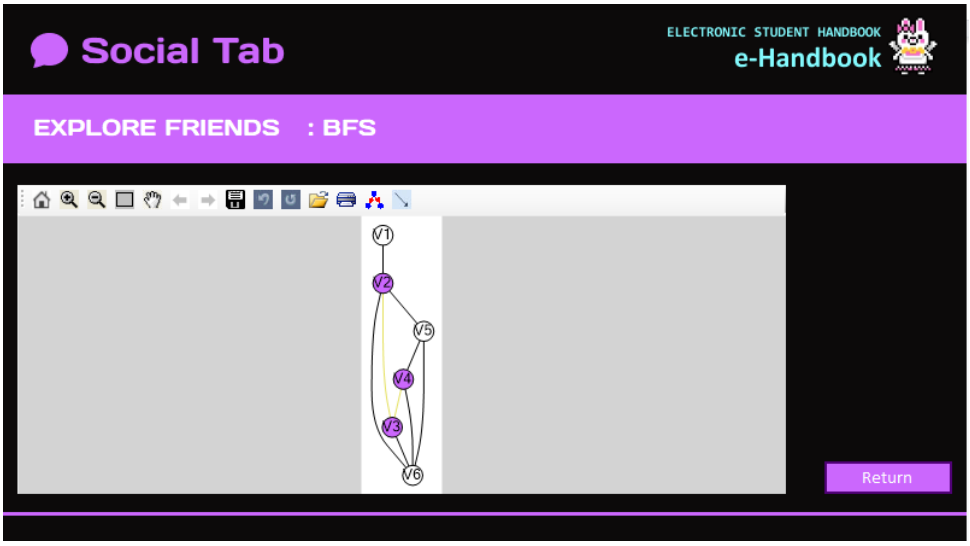
Gambar 4.4.5.1 Halaman Utama Aplikasi

b. *Friend recommendation* untuk V6



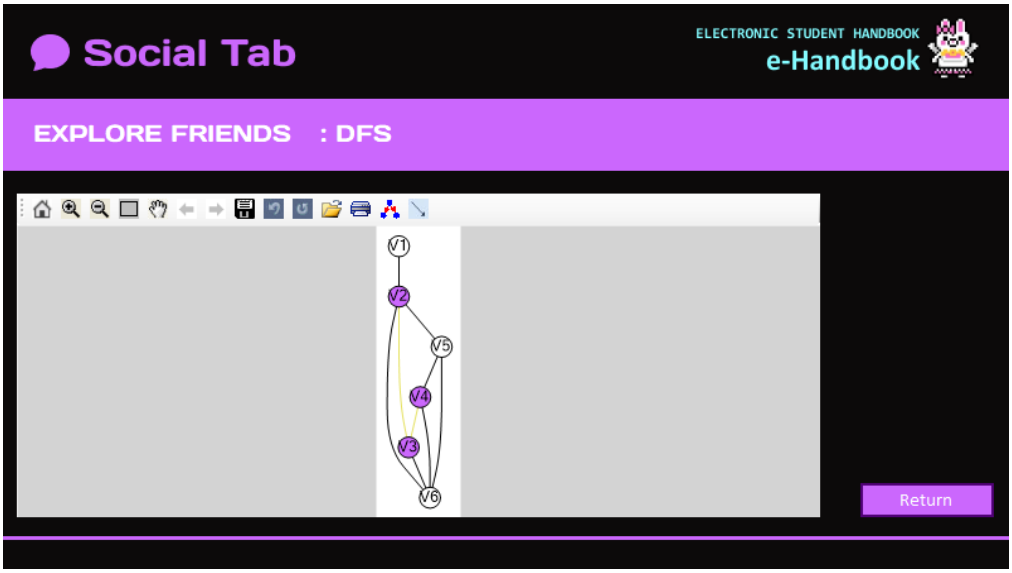
Gambar 4.4.5.2 Tampilan hasil *friend recommendation*

c. Jalur V4 ke V2 secara BFS



Gambar 4.4.5.3 Tampilan hasil *explore friends* dari V4 ke V2 dengan BFS

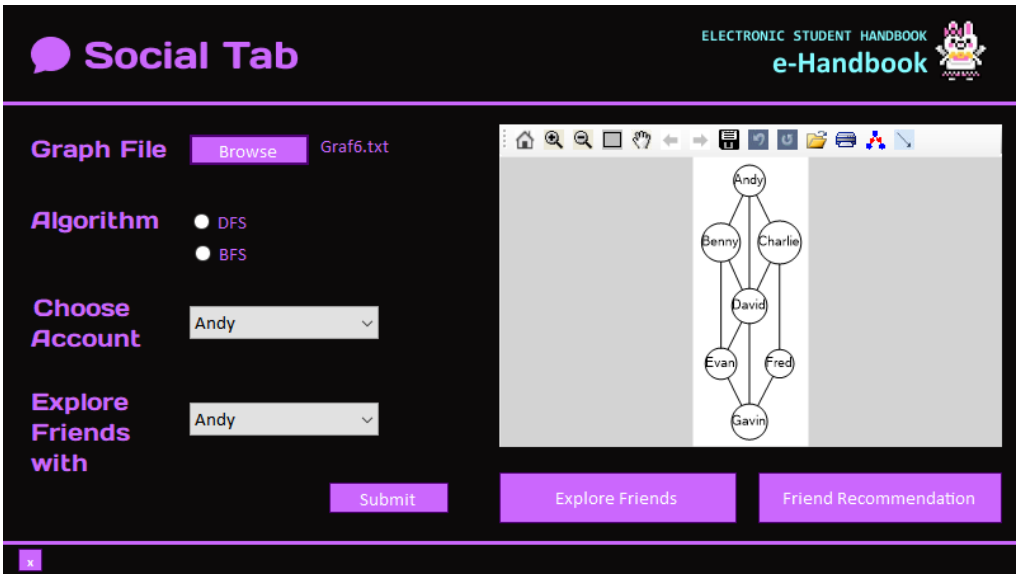
d. Jalur V4 ke V2 secara DFS



Gambar 4.4.5.4 Tampilan hasil *explore friends* dari V4 ke V2 dengan DFS

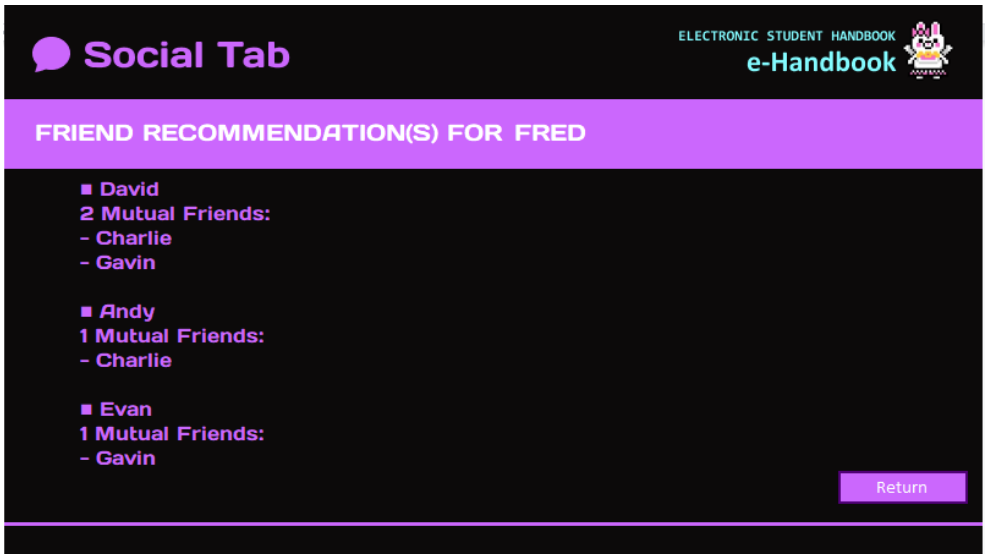
6. Graf 6

a. Halaman utama



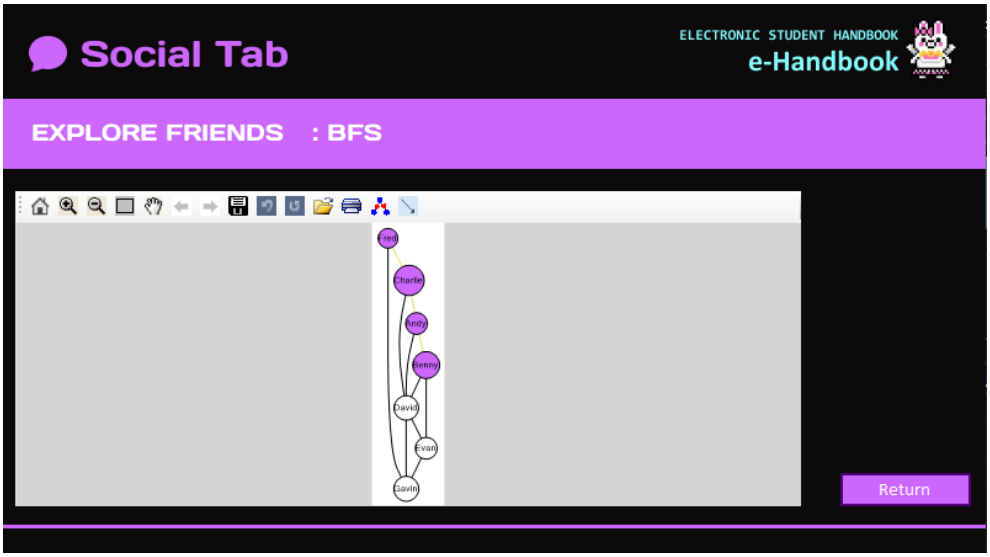
Gambar 4.4.6.1 Halaman Utama Aplikasi

b. *Friend recommendation* untuk Fred



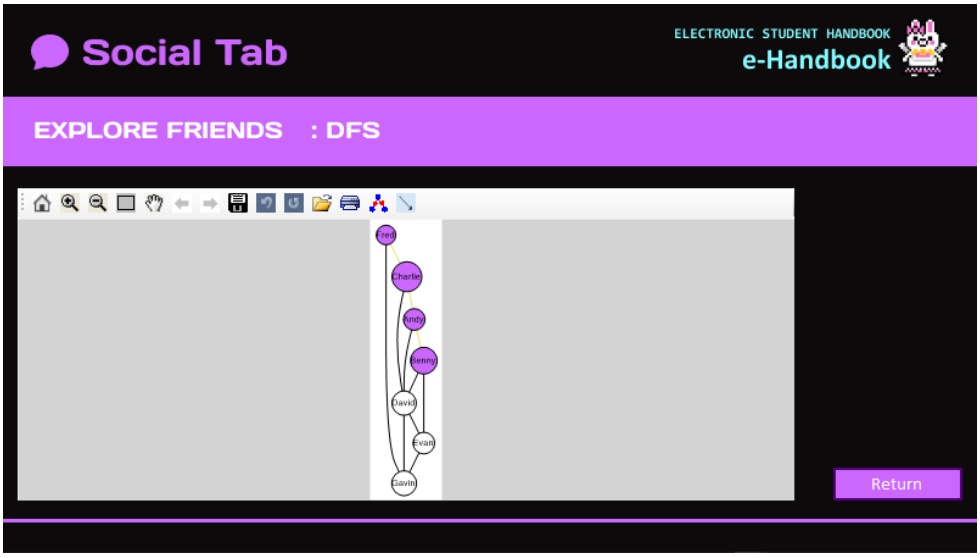
Gambar 4.4.6.2 Tampilan hasil *friend recommendation*

c. Jalur Fred ke Benny secara BFS



Gambar 4.4.6.3 Tampilan hasil *explore friends* dari Fred ke Benny dengan BFS

d. Jalur Fred ke Benny secara DFS



Gambar 4.4.6.4 Tampilan hasil *explore friends* dari Fred ke Benny dengan DFS

E. Analisis Efisiensi

Berdasarkan uji coba yang dilakukan, dapat dilihat bahwa BFS secara konsisten memberikan hasil yang lebih baik, yaitu nth degree connection yang lebih kecil dibandingkan dengan DFS. Namun, pada implementasinya, BFS lebih banyak mengalokasikan memori untuk path setiap item di queue dibandingkan dengan DFS yang hanya menyimpan satu path.

DFS merupakan cara yang lebih mudah digunakan, tapi hanya efektif untuk kasus dimana jalur yang dicari menyerupai urutan pembangkitan vertice pada graf. BFS merupakan cara yang konsisten memberikan hasil terbaik, yaitu jalur terpendek antara kedua akun, namun implementasinya membutuhkan penyimpanan path tiap vertice sehingga relatif lebih banyak menggunakan memori.

BAB V
KESIMPULAN DAN SARAN

A. Kesimpulan

Fitur friend recommendation dan explore friends yang seringkali ditemukan di platform media sosial dapat diimplementasikan dengan algoritma BFS dan DFS. Fitur friend recommendation diimplementasikan dengan BFS yang memiliki depth = 2. Sementara itu, fitur explore friends dapat dilakukan dengan kedua metode tersebut, dimana algoritma BFS diimplementasikan dengan queue, dan algoritma DFS dengan stack. Hasil yang diperoleh berupa List<string> dengan setiap string berisi nama vertice dari vertice awal dan vertice yang ingin dicari jalurnya. Pada program juga tersedia GUI dan visualisasi graf.

B. Saran

Dari hasil aplikasi yang telah dibuat masih banyak hal yang dapat diimplementasikan untuk membuat aplikasi berjalan lebih lancar baik secara tampilan maupun algoritmanya. Beberapa diantaranya adalah dalam tampilan dapat diberikan informasi mengenai *nth degree connection* pada fitur *friend recommendation* dan menampilkan urutan pada hasil fitur *explore friends.* Selain itu juga, dapat dilakukan optimasi program untuk menampilkan tampilan serta realisasi dari algoritma BFS dan DFS itu sendiri.

C. Refleksi

Setelah mengevaluasi pembagian tugas, implementasi program, dan pelaksanaan tugas secara menyeluruh, penulis menemukan beberapa hal yang dapat diperbaiki kedepannya. Pertama, eksplorasi dan pemanfaatan fitur yang ditawarkan C# lebih jauh lagi agar dapat mempersingkat, bahkan menyederhanakan algoritma yang digunakan. Kedua, eksplorasi cara-cara implementasi algoritma BFS dan DFS lainnya, seperti penggunaan fungsi rekursif dan lain-lain. Ketiga, untuk lebih memanfaatkan berbagai *tool* yang disediakan untuk pembuatan GUI.

D. Komentar

Tugas besar ini menarik karena memaksa penulis melakukan eksplorasi terhadap bahasa pemrograman dan *framework* yang belum dipelajari sebelumnya, serta berbagai cara mengimplementasikan algoritma *traversal* graf, yaitu BFS dan DFS. Selain itu, tugas besar ini juga melatih penggunaan fungsi-fungsi yang disediakan bahasa pemrograman untuk memproses berbagai struktur data seperti *List, HashMap, Dictionary, Tuple*, dan sebagainya, karena merupakan bagian penting dari pembuatan algoritma-algoritma yang menjadi topik tugas besar ini, mulai dari representasi graf sampai penyimpanan solusi yang ditemukan. Ditambah dengan mungkin tugas ini adalah pertama kalinya dalam sebuah tugas, penulis dapat membuat aplikasi fisik yang tidak hanya jalan melalui *command line*.

DAFTAR PUSTAKA

Tim Mata Kuliah IF2211 Strategi Algoritma. (2021). *Spesifikasi Tugas Besar*, http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Tugas-Besar-2-IF2211-Strategi-Algoritma-2021.pdf, diakses 25 Maret 2021.

Munir, Rinaldi. (2021). *Breadth/Depth First Search (Bagian 1).* http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf, diakses 25 Maret 2021.

Munir, Rinaldi. (2021). *Breadth/Depth First Search (Bagian 2).* http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf, diakses 25 Maret 2021.

Tutorials Point. *Data Structures: Breadth First Traversal.* https://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal.htm, diakses 26 Maret 2021.

Tutorials Point. *Data Structures: Depth First Traversal.* https://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal.htm, diakses 26 Maret 2021.

Microsoft. (2021). *A Tour of the C# Language.* https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/, diakses 26 Maret 2021.