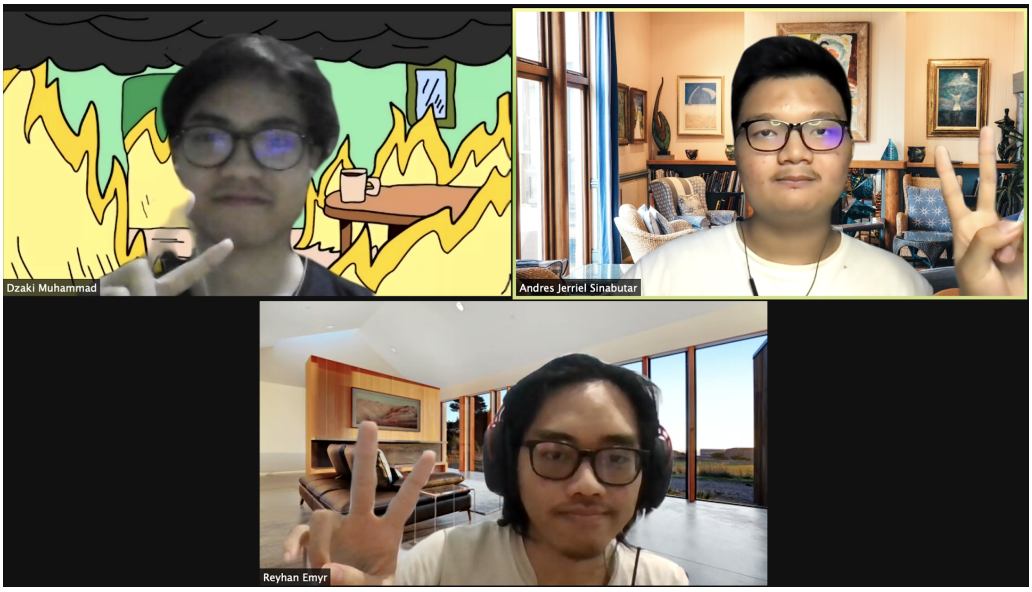
LAPORAN TUGAS BESAR 2 IF2211 STRATEGI ALGORITMA
SEMESTER II TAHUN 2020/2021

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



Anggota:

1. Dzaki Muhammad - 13519049 - K1
2. Reyhan Emyr Arrosyid - 13519167 - K4
3. Andres Jerriel Sinabutar - 13519218 - K4

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG

2021

BAB 1

DESKRIPSI TUGAS

Dalam tugas besar ini, Anda akan 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), Anda 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.

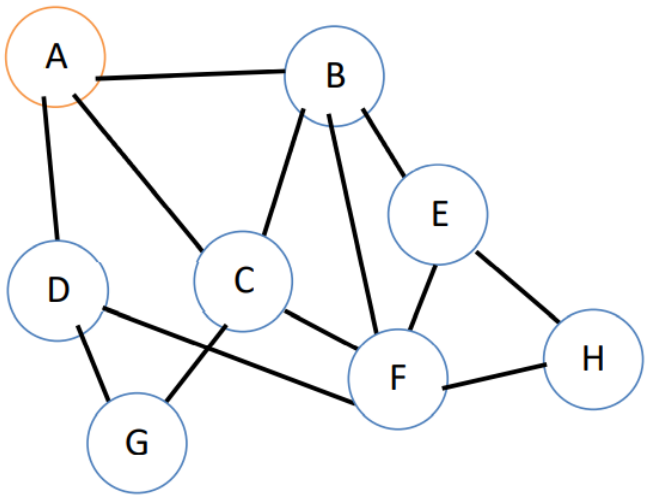
Contoh Input dan Output Program

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
```

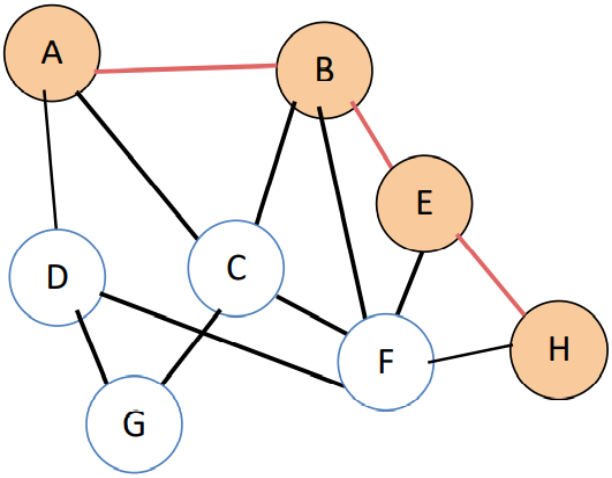
Gambar 1.3. Hasil output yang diharapkan untuk rekomendasi akun A

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.

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

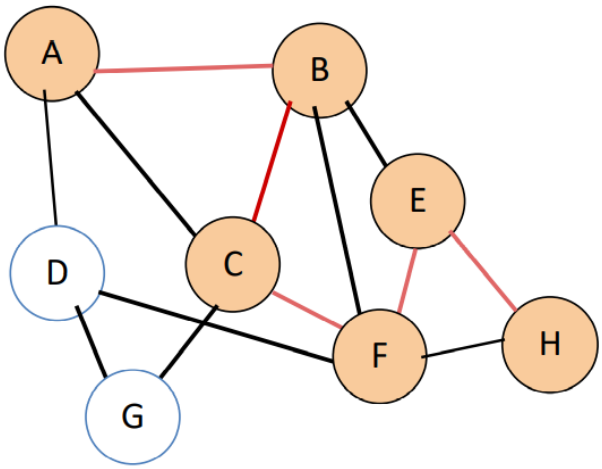
Gambar 1.4. Hasil output akun Nth degree connection akun A dan H menggunakan BFS

Perhatikan 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 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 1.5. 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 2

LANDASAN TEORI

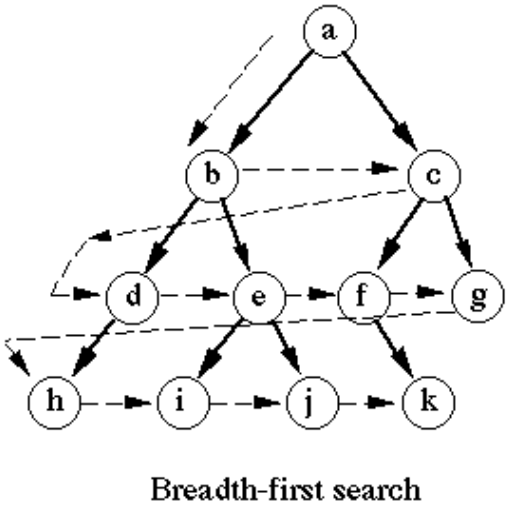
2.1 Traversal Graf

Traversal graf adalah adalah proses penelusuran simpul sebuah graf secara sistematis. Dalam algoritma traversal graf, setiap simpul graf dikunjungi secara terstruktur sesuai aturan tertentu. Terdapat dua algoritma traversal graf, yaitu *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS).

2.2 *Breadth-First Search* (BFS)

Breadth-First Search (BFS) adalah metode traversal graf dengan pendekatan pencarian simpul secara melebar. Pencarian dilakukan dimulai dari suatu simpul dan simpul tetangga diekspansi berdasarkan kedalaman. Misal traversal graf dimulai dari simpul v. Algoritma BFS :

1. Kunjungi simpul v
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu
3. Untuk setiap simpul yang dikunjungi tersebut kunjungi simpul yang belum dikunjungi dan bertetangga simpul tersebut
4. Ulangi langkah 3 hingga semua simpul dikunjungi

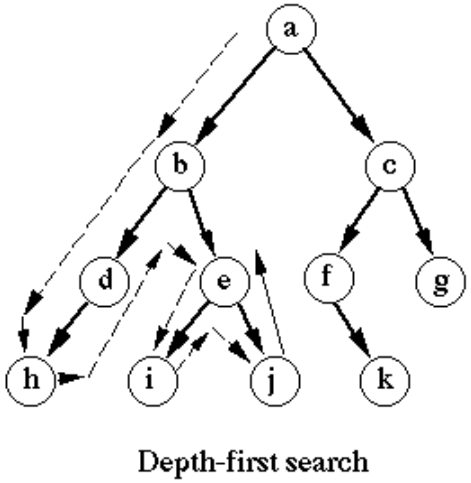


Gambar 2.2.1 Ilustrasi Algoritma BFS

2.3 *Depth-First Search* (DFS)

Depth-First Search (DFS) adalah metode traversal graf dengan pendekatan pencarian simpul secara mendalam. Penelusuran simpul dilakukan dengan mengunjungi simpul terdalam terlebih dahulu. Ketika tidak ada lagi simpul yang belum dikunjungi, algoritma "mundur" mencari simpul lain yang belum dikunjungi. Misal traversal graf dimulai dari simpul v. Algoritma DFS :

1. Kunjungi simpul v
2. Kunjungi simpul w yang merupakan simpul tetangga dari v
3. Ulangi algoritma DFS secara rekursif dengan simpul awal adalah simpul w
4. Apabila proses pencarian mencapai suatu simpul u sehingga tidak ada lagi simpul tetangga yang belum dikunjungi, dilakukan pencarian runut-balik ke simpul terakhir yang dikunjungi sebelum simpul u dan memiliki simpul tetangga yang belum dikunjungi.
5. Pencarian berakhir apabila semua simpul telah dikunjungi atau tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai.



Gambar 2.3.1 Ilustrasi Algoritma DFS

2.4 C# *Desktop Application Development*

Desktop Application adalah sebuah perangkat lunak yang berjalan secara lokal di komputer. *Desktop Application* dapat berupa *console* ataupun Graphical User Interface (GUI). Salah satu bahasa yang biasa digunakan untuk membuat *desktop application* adalah C#. Bahasa C# merupakan salah satu bahasa pemrograman berorientasi objek. Pengembangan *desktop application* dengan bahasa C# dilakukan menggunakan *framework* .NET. Untuk mempermudah proses pengembangan, digunakan IDE bernama Visual Studio.

BAB 3

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

Untuk menyelesaikan masalah yang sudah disebutkan di atas, diperlukan beberapa langkah-langkah sebagai berikut:

1. Memahami permasalahan yang diberikan
2. Memahami konsep algoritma BFS dan DFS dalam traversal graf
3. Mempelajari bahasa C#
4. Memetakan persoalan menjadi elemen-elemen BFS dan DFS
5. Menggambarkan input file txt menjadi simpul dan sisi yang membentuk graf
6. Mengimplementasikan algoritma BFS dan DFS dalam bahasa C# untuk menyelesaikan persoalan
7. Membuat visualisasi graf menggunakan MSAGL
8. Membuat GUI
9. Menghubungkan GUI dengan program utama

3.2 Pemetaan Persoalan

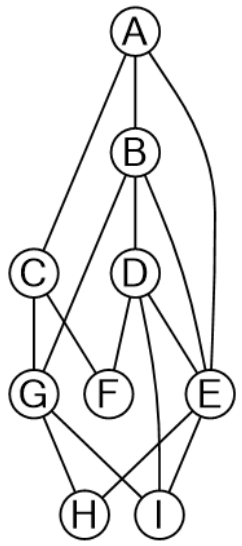
Seperti yang sudah dijelaskan pada Bab 3.1, diperlukan pemetaan persoalan menjadi elemen-elemen BFS dan DFS terlebih dahulu sebelum mengimplementasikan solusi pada program. Elemen-elemen yang terdapat dalam algoritma BFS antara lain simpul graf, matriks ketetanggaan, antrian, dan tabel boolean untuk mengindikasikan suatu simpul sudah dikunjungi atau belum. Sedangkan elemen-elemen yang terdapat dalam algoritma DFS antara lain simpul graf, matriks ketetanggaan, dan tabel boolean untuk mengindikasikan suatu simpul sudah dikunjungi atau belum. Maka pemetaan persoalan adalah sebagai berikut.

1. Simpul graf : Simpul graf yang dibentuk adalah setiap akun berbeda yang terdapat dalam file txt
2. Matriks ketetanggaan : Ketetanggaan direpresentasikan sebagai atribut sebuah simpul yang berisi daftar simpul-simpul yang bertetangga dengan simpul terkait

3. Tabel boolean : Tabel boolean direpresentasikan dalam bentuk dictionary dengan key menunjukkan simpul dan value menunjukkan indikasi simpul tersebut sudah dikunjungi atau belum
4. Antrian (khusus BFS) : Antrian dalam algoritma BFS direpresentasikan dengan menggunakan struktur data Queue yang terdapat dalam bahasa C#

3.3 Ilustrasi Pemecahan Masalah

Misalkan kasus permasalahan tergambarkan sebagai graf pada gambar 3.3.1. Dari graf tersebut akan dijalankan fitur *friend recommendation* dan fitur *explore friend*.



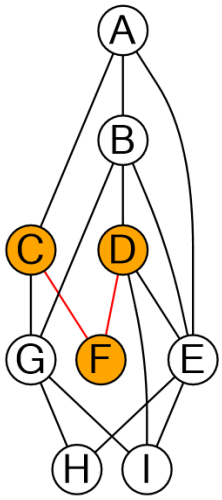
Gambar 3.3.1 Contoh graf permasalahan

Untuk fitur *friend recommendation*, misalkan ingin dicari rekomendasi teman dari simpul A. Pertama-tama akan dibentuk daftar simpul tetangga dari simpul yang bertetangga dengan simpul A dan bukan merupakan simpul tetangga dari simpul A, untuk kasus ini daftar simpul tersebut adalah {D, G, F, H, I}. Kemudian untuk setiap simpul dari daftar tersebut akan dicari semua simpul yang merupakan tetangga simpul A yang juga merupakan tetangga dari simpul dari daftar tersebut (*mutual friend*). Pada contoh ini, *mutual friend* dari A dan D adalah {B, E}, *mutual friend* dari A dan G adalah {B, C}, *mutual friend* dari A dan F adalah {C}, *mutual friend* dari A dan H adalah {E}, *mutual friend* dari A dan I adalah {E}. Lalu simpul-simpul pada daftar tersebut akan diurut terlebih dahulu berdasarkan banyaknya *mutual friend* sebelum dicetak sebagai luaran pada antarmuka program.

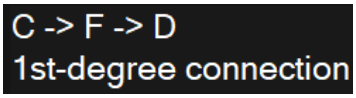
```
List of friend recommendation for A
Account name: D
2 mutual friend(s):
B
E

Account name: G
2 mutual friend(s):
B
C

Account name: F
1 mutual friend(s):
C

Account name: H
1 mutual friend(s):
E

Account name: I
1 mutual friend(s):
E
```

Gambar 3.3.2 Ilustrasi luaran program fitur *friend recommendation* simpul A

Untuk fitur *explore friend*, misalkan ingin dicari eksplorasi jalur pertemanan antara simpul C dengan simpul D. Jika menggunakan metode BFS maka simpul C akan mengunjungi semua simpul tetangganya terlebih dahulu dalam pencarian simpul D, jika simpul D tidak ditemukan maka pencarian dilanjutkan dengan mengunjungi simpul tetangga dari setiap simpul yang telah dikunjungi. Proses pencarian ini dilakukan dengan cara yang sama sesuai dengan algoritma BFS, sehingga jalur eksplorasi dari simpul C ke simpul D dengan metode BFS adalah C → F → D.

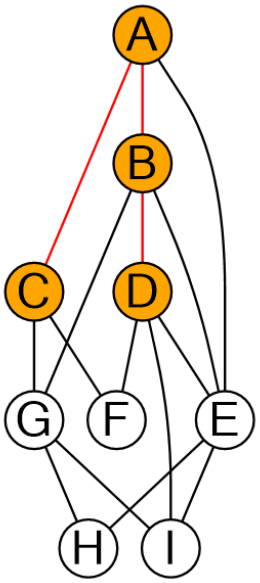


Gambar 3.3.3 Ilustrasi jalur dari simpul C ke simpul D dengan BFS

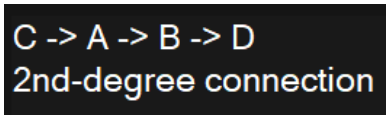


Gambar 3.3.4 Ilustrasi luaran fitur explore friend simpul C dan D dengan BFS

Sedangkan jika menggunakan metode DFS, simpul C akan mengunjungi salah satu simpul tetangganya terlebih dahulu kemudian apabila simpul D belum ditemukan, proses pencarian dilakukan secara rekursif sesuai dengan algoritma DFS dimulai dari simpul yang baru dikunjungi tersebut hingga simpul D ditemukan. Jalur eksplorasi dari simpul C ke simpul D dengan metode DFS adalah C → A → B → D.



Gambar 3.3.5 Ilustrasi jalur dari simpul C ke simpul D dengan DFS



Gambar 3.3.6 Ilustrasi luaran fitur explore friend simpul C dan D dengan DFS

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

4.1.1 DFS

```
function dfs(s, e: Vertex; dikunjungi: Dictionary<string, boolean>; prev:
Dictionary<string, string>) → List<string>

KAMUS
n : Vertex
next : Vertex
function findVertex(name : string) → Vertex
{fungsi untuk mencari simpul yang terdapat di dalam graf berdasarkan nama
simpul}
function path(s, e: Vertex; prev: Dictionary<string, string> →
List<string>
{fungsi untuk menghasilkan list berupa simpul-simpul jalur dari simpul s
ke simpul e berdasarkan Dictionary prev}

ALGORITMA
dikunjungi[s.Name] ← True
n ← findVertex(s.Name)
if (n.Name ≠ e.Name) then
  foreach(next in n.Edges)
    if(not dikunjungi[next.Name]) then
      prev[next.Name] ← n.Name
      dfs(next, e, dikunjungi, prev)
→ path(s,e,prev)
```

4.1.2 BFS

```
function bfs(s, e: Vertex; dikunjungi: Dictionary<string, boolean>; prev:
Dictionary<string, string>) → List<string>

KAMUS
q : Queue<Vertex>
node : Vertex
n : Vertex
next : Vertex
function findVertex(name : string) → Vertex
{fungsi untuk mencari simpul yang terdapat di dalam graf berdasarkan nama
simpul}
function path(s, e: Vertex; prev: Dictionary<string, string> →
List<string>
{fungsi untuk menghasilkan list berupa simpul-simpul jalur dari simpul s
ke simpul e berdasarkan Dictionary prev}

ALGORITMA
q ← new Queue<Vertex>
```

```
q.Enqueue(s)
dikunjungi[s.Name] ← True
while (q.Count > 0) do
  Node ← q.Dequeue()
  n ← findVertex(s.Name)
  if (n.Name = e.Name) then
    break
  foreach(next in n.Edges)
    if(not dikunjungi[next.Name]) then
      dikunjungi[next.Name] ← True
      prev[next.Name] ← n.Name
      q.Enqueue(next)
→ path(s,e,prev)
```

4.1.3 Explore Friend

```
function explore_friend(s, e: Vertex; metode: string) →
Tuple<List<string>, string>

KAMUS
dikunjungi : Dictionary<string, boolean>
prev : Dictionary<string, string>
path : List<string>
resultString : string
Node : Vertex
AdjacencyList : List<Vertex>
n : string
N, M : integer
function ToString(i: integer) → string {fungsi mengubah integer menjadi
string}

ALGORITMA
dikunjungi ← new Dictionary<string, boolean>
prev ← new Dictionary<string, string>
foreach(node in AdjacencyList)
  dikunjungi[node.Name] ← False
  prev[node.Name] ← null

{Menyesuaikan metode]
if(metode = "BFS") then
  path ← bfs(s, e, dikunjungi, prev)
else if(metode = "DFS") then
  path ← dfs(s, e, dikunjungi, prev)
else
  → new Tuple<List<string>, string>

resultString ← ""
{Kasus tidak ada jalur dari s ke e}
if(path.Count = 0) then
  resultString ← resultString + "Tidak ada jalur koneksi yang tersedia.
Anda harus memulai koneksi baru itu sendiri."

{Kasus ada jalur dari s ke e}
else
```

```
  foreach(n in path)
    resultString ← resultString + n
    if(n ≠ path[path.Count -1]) then
      resultString ← resultString + " -> "
  {Menulis Nth degree connection}
    N ← path.Count -2
    resultString ← resultString + ToString(N)
    if(N mod 10 > 3 or (N mod 100 >= 11 and N <= 13) or N mod 10 = 0) then
      resultString ← resultString + "th-degree connection"
    else
      M = N mod 10
      depend on (M)
        M = 1 :
          resultString ← resultString + "st-degree connection
          break
        M = 2 :
          resultString ← resultString + "nd-degree connection
          break
        M = 3 :
          resultString ← resultString + "rd-degree connection
          break
→(path, resultString)
```

4.1.4 Friend Recommendation

```
function FriendRecommendation(v: Vertex) → string

KAMUS
recommendation: Dictionary<string, List<string>>
friends: List<string>
friend: Vertex
friend: string
recomFriend: string
mutual: Vertex
sortedKeys: List<Tuple<string, int>>
key: string
result: string
tuple: Tuple<string, int>
mutual: string

ALGORITMA
foreach (friend in v.Edges)
  friends.Add(friend.Name)

foreach (friend in friends)
  foreach (s in findVertex(friend).Edges)
    if (!recommendation.ContainsKey(s.Name) && !friends.Contains(s.Name)
&& !String.Equals(v.Name, s.Name)) then
      recommendation.Add(s.Name, new List<string>())

foreach (recomFriend in recommendation.Keys)
  foreach (mutual in findVertex(recomFriend).Edges)
    if (friends.Contains(mutual.Name)) then
      recommendation[recomFriend].Add(mutual.Name)
```

```
foreach (key in recommendation.Keys)
  sortedKeys.Add(new Tuple<string, int>(key, recommendation[key].Count))

sortedKeys.Sort((a, b) => b.Item2.CompareTo(a.Item2));

result ← ""

if (recommendation.Count == 0) then
  result += "There is no friend recommendation for " + v.Name + "\n"
else
  result += "List of friend recommendation for " + v.Name + "\n"
  foreach (tuple in sortedKeys)
    result += "Account name: " + tuple.Item1 + "\n"
    result += "Mutual friends:\n"
    foreach (mutual in recommendation[tuple.Item1])
      result += mutual + "\n"

return result;
```

4.1.5 Program utama

```
KAMUS
graph : Graph
metode : string
pilihanFitur : string
akun : Vertex
friend : Vertex
result : string

ALGORITMA
input(graph)
input(metode)
input(pilihanFitur)
input(akun)
input(friend)

if (pilihanFitur = "Explore") then
  if (akun ada di graf and friend ada di graf) then
    result ← graph.explore_friend(akun, friend, metode).Item2
    output(result)
  else
    output("Please select the accounts")
  endif
else {pilihanFitur = "Recommendation"}
  if (akun ada di graf) then
    result ← graph.recommend(akun)
    output(result)
  else
    output("Please select the account")
  endif
endif
```

4.2 Struktur Data Program

4.2.1 Kelas Vertex

Kelas Vertex
Deskripsi Kelas : Representasi simpul yang berada pada graf
Atribut Kelas : 1. Name : Nama dari simpul 2. Edges : List semua simpul yang terhubung dengan simpul ini
Method Kelas : 1. Vertex : Konstruktor vertex dengan parameter nama simpul

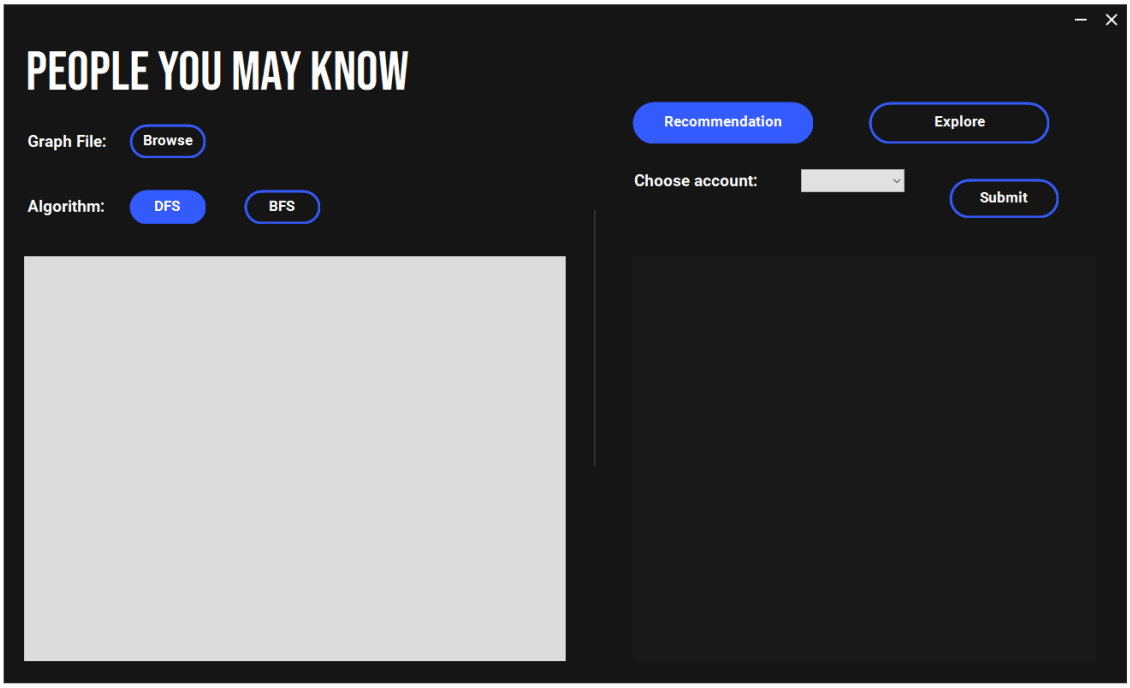
4.2.2 Kelas Graph

Kelas Graph
Deskripsi Kelas : Representasi Graf
Atribut Kelas : 1. AdjacencyList : List semua simpul yang terdapat pada graf ini
Method Kelas : 1. Graph : Konstruktor graf 2. GraphFromFile : Konstruksi graf dari sebuah file txt 3. AddVertex : Menambahkan simpul pada graf 4. findVertex : Mencari simpul pada graf dengan nama tertentu 5. AddEdge : Menambah simpul tetangga dari suatu simpul 6. RemoveEdge : Menghapus ketetanggaan dari dua simpul yang bertetangga 7. IsNewVertex : Memeriksa keberadaan suatu simpul pada graf 8. IsConnected : Memeriksa apakah suatu simpul v1 bertetangga dengan simpul v2 9. PrintToConsole : Mencetak semua simpul pada graf beserta tetangganya 10. bfs : Mengembalikan jalur dari simpul s ke simpul e pada graf dengan metode BFS 11. dfs : Mengembalikan jalur dari simpul s ke simpul e pada graf dengan metode DFS 12. path : Mengembalikan jalur dari simpul s ke simpul e pada graf 13. explore_friend : Fitur Explore Friend 14. FriendRecommendation : Fitur Friend Recommendation 15. getMSAGLGraph : Mengembalikan graf dalam bentuk graf MSAGL untuk divisualisasikan

4.3 Tata Cara Penggunaan Program

4.3.1 Antarmuka Program

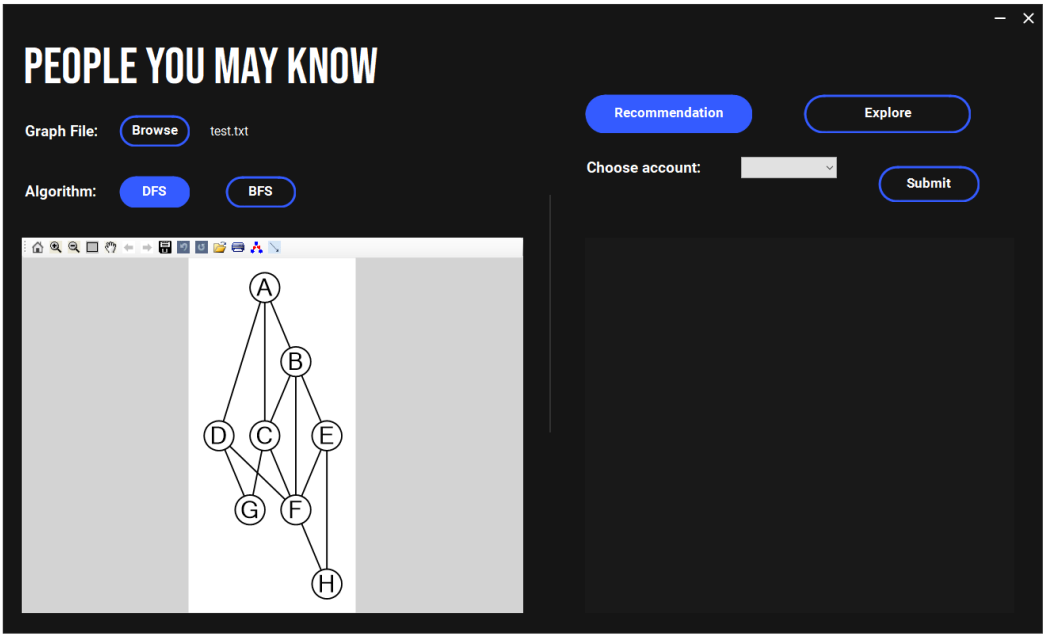
Untuk menjalankan program, jalankan file Tubes2.exe yang terdapat pada folder bin. Setelah program berhasil dijalankan, program akan menampilkan tampilan seperti berikut.



Gambar 4.3.1.1 Tampilan Awal Program

4.3.2 Proses Input File

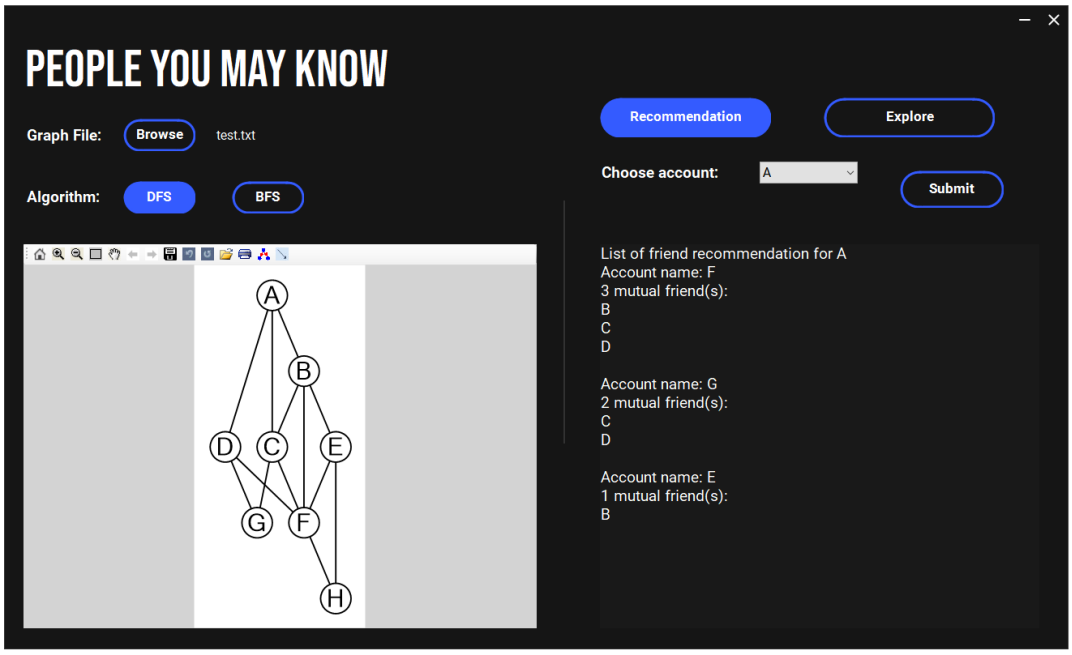
Untuk input file, tekan tombol “Browse” lalu pilih file yang akan diubah menjadi bentuk graf. File harus mengikuti format yang terdapat pada Bab 1. Setelah input file berhasil, program akan menampilkan visualisasi graf pada kotak yang berada di kiri bawah layar.



Gambar 4.3.2.1 Tampilan Program Setelah Input File

4.3.3 Penggunaan Fitur *Friend Recommendation*

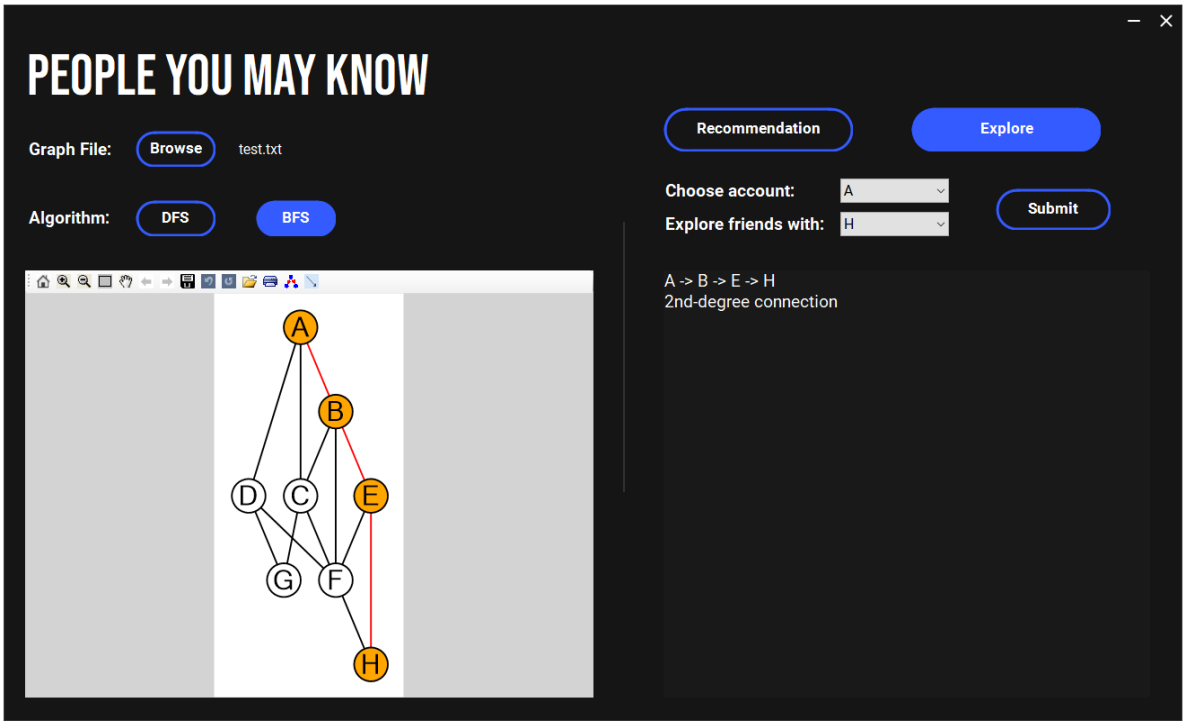
Untuk menggunakan fitur *friend recommendation*, pilih tombol "Recommendation". Setelah itu, pilih akun yang ingin dicari rekomendasinya pada *dropdown*. Setelah akun dipilih, tekan tombol "Submit" lalu hasil program akan ditampilkan pada kotak yang berada di kanan bawah.



Gambar 4.3.3.1 Tampilan Program Fitur *Recommendation*

4.3.4 Penggunaan Fitur *Explore Friend*

Untuk menggunakan fitur *friend recommendation*, pilih tombol "Explore". Setelah itu, pilih metode traversal graf yang ingin digunakan. Setelah metode sudah dipilih, pilihlah akun yang asal dan akun tujuan pada *dropdown*. Setelah akun dipilih, tekan tombol "Submit" lalu hasil program akan ditampilkan pada kotak yang berada di kanan bawah.

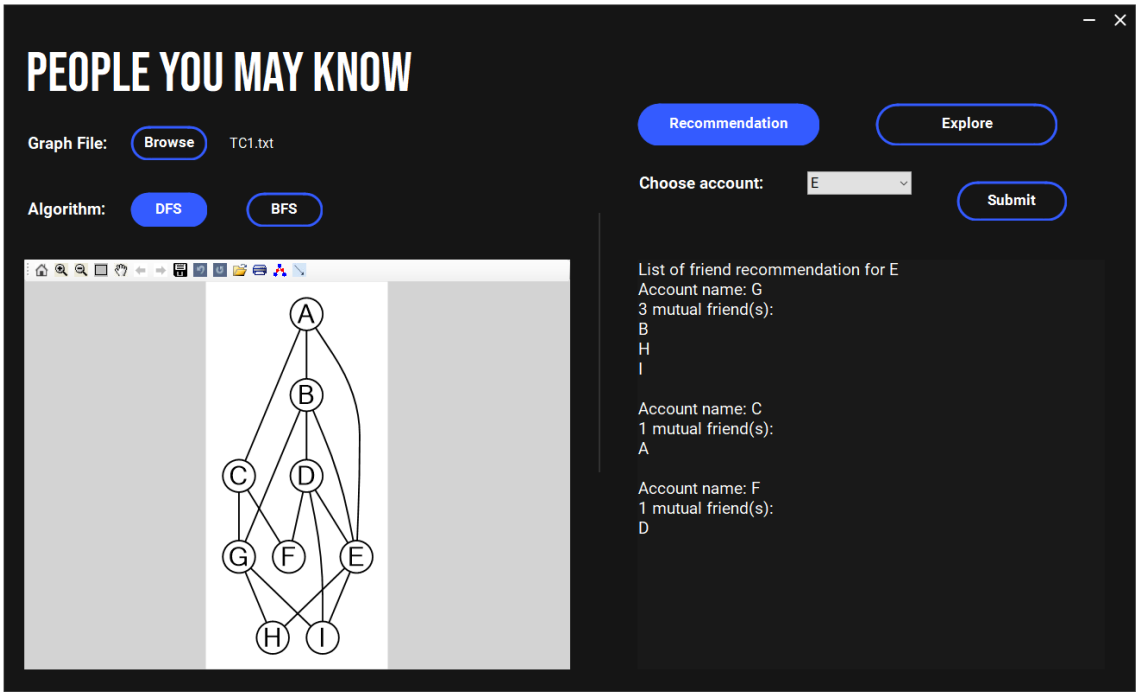


Gambar 4.3.4.1 Tampilan Program Fitur *Explore Friend*

4.4 Hasil Pengujian dan Analisis

4.4.1 Kasus Uji 1

Input
Fitur: *Friend Recommendation* File: TC1.txt Akun: E
Hasil:

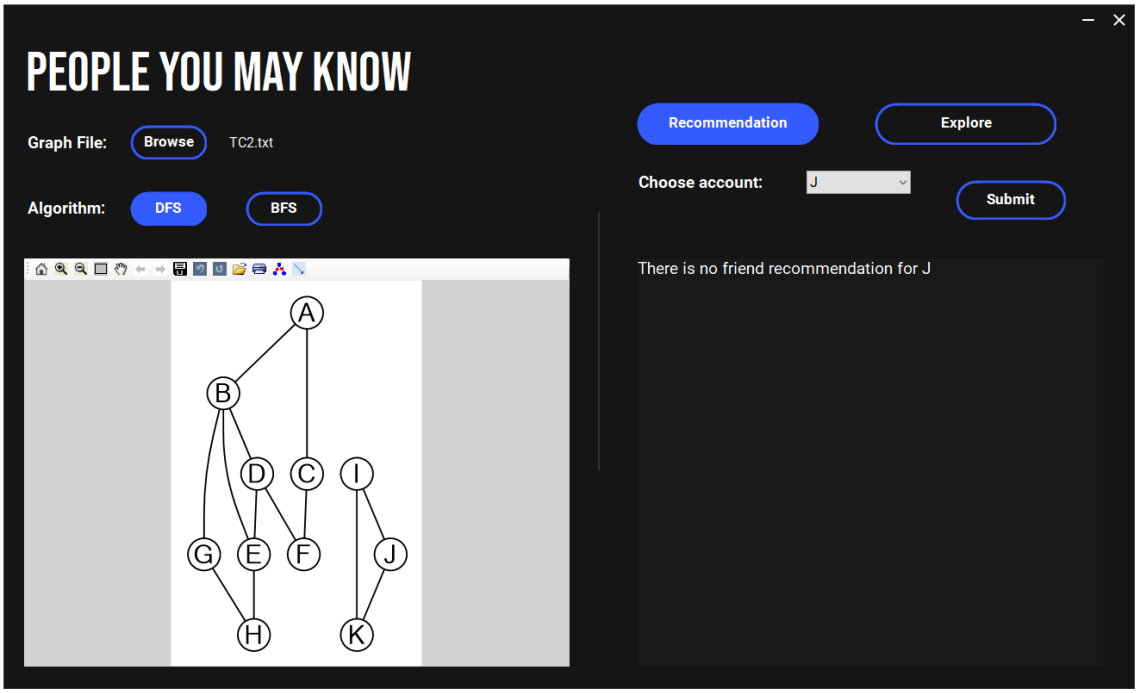


Analisis:

Dari visualisasi graf di atas, dapat dilihat bahwa simpul E memiliki sisi-sisi yang terhubung dengan simpul A, B, D, H, dan I. Kita akan mencari rekomendasi teman untuk akun E dengan mencari simpul-simpul yang terhubung dengan simpul yang bersisian dengan simpul E (*mutual friends*). Setelah itu, ditemukan bahwa terdapat tiga akun yang memiliki *mutual friends* dengan akun E, yaitu akun G, C, dan F sehingga rekomendasi teman bagi akun E adalah ketiga akun tersebut.

4.4.2 Kasus Uji 2

Input
Fitur: *Friend Recommendation* File: TC2.txt Akun: J
Hasil:

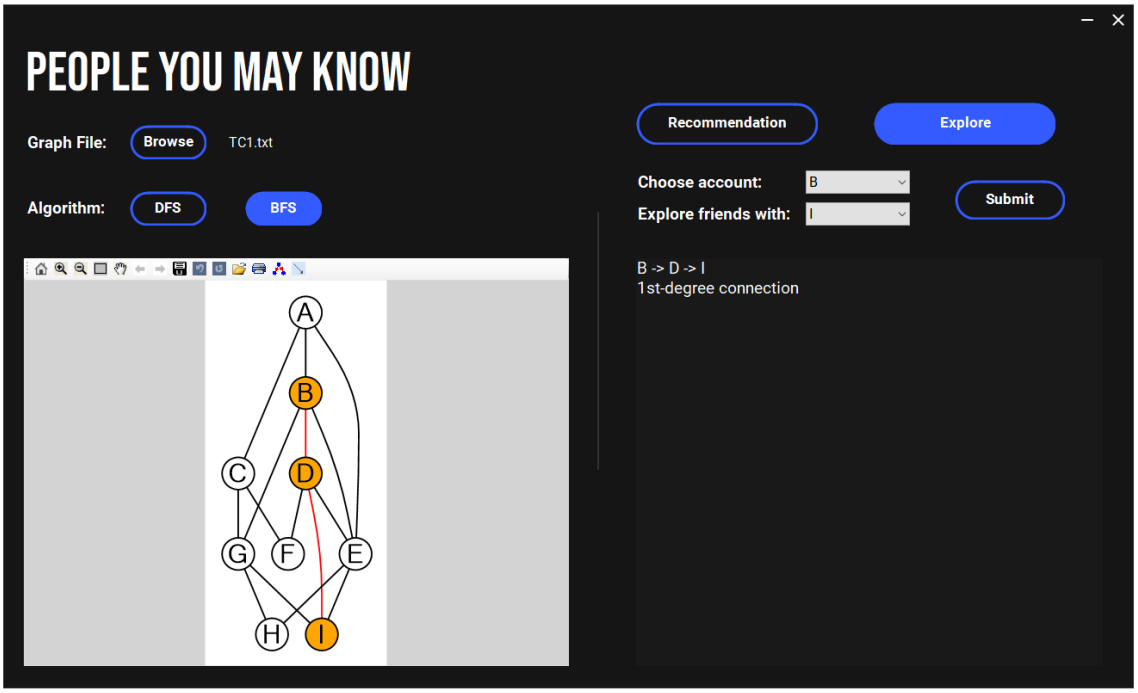


Analisis:

Dari visualisasi graf di atas, dapat dilihat bahwa simpul J yang berada di kanan bawah memiliki dua sisi yang terhubung dengan dua simpul lainya, yaitu I dan K. Akan tetapi, simpul I dan K tidak memiliki sisi yang terhubung dengan simpul lainnya sehingga tidak dapat ditemukan rekomendasi teman bagi J.

4.4.3 Kasus Uji 3

Input
Fitur: *Explore Friends* File: TC1.txt Akun asal: B Akun tujuan: I Metode: BFS
Hasil:



Analisis:

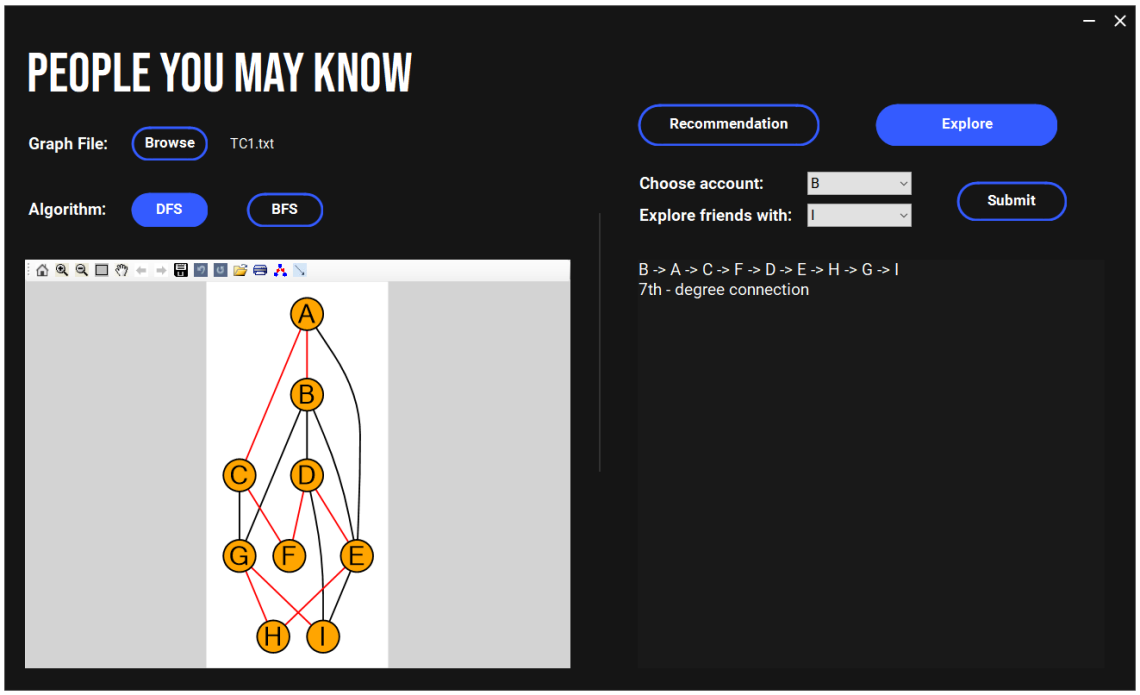
Dari hasil pengujian di atas, simpul B akan dicari koneksinya dengan simpul I. Karena menggunakan metode BFS, pencarian dilakukan pada simpul terdekat terlebih dahulu. Pertama-tama simpul yang bertetanggaan dengan B dikunjungi yaitu A dan D. Karena solusi belum ditemukan, pencarian dilanjutkan dan akhirnya ditemukan jalur yaitu B → D → I. Karena pencarian dilakukan berdasarkan jarak, dapat dipastikan solusi yang ditemukan merupakan solusi yang optimal jika dilihat dari jarak simpul awal ke simpul tujuan.

4.4.4 Kasus Uji 4

Input

Fitur: *Explore Friends*
File: TC1.txt
Akun asal: B
Akun tujuan: I
Metode: DFS

Hasil:



Analisis:

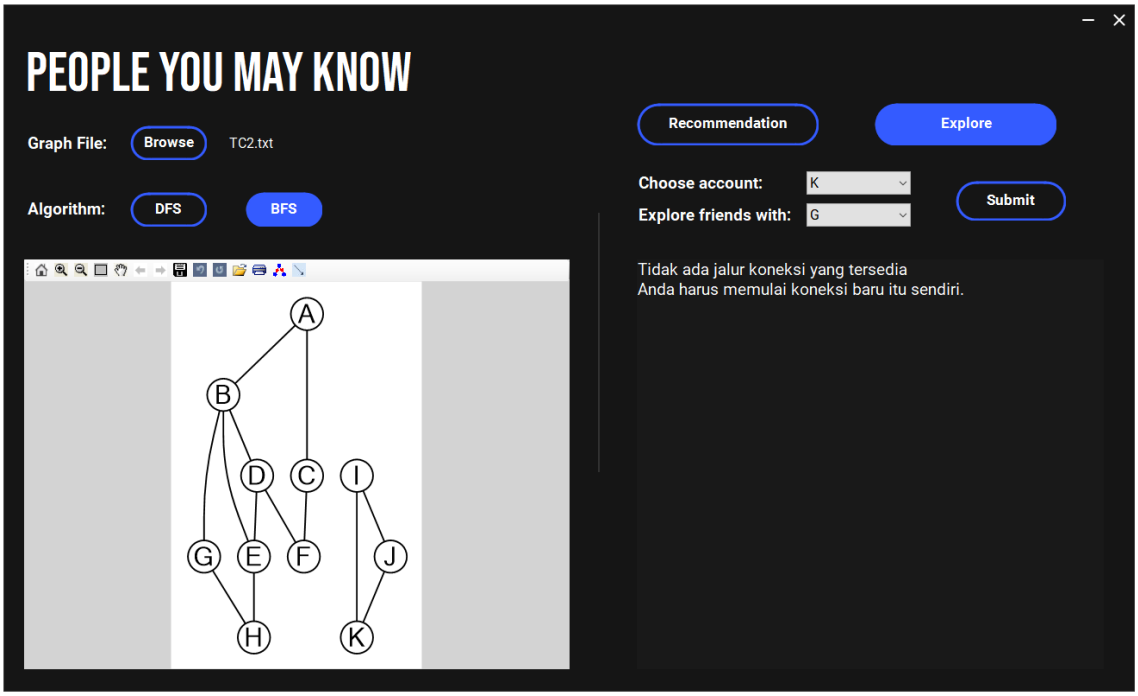
Dari hasil pengujian di atas, simpul B akan dicari koneksinya dengan simpul I. Karena menggunakan metode DFS, pencarian dilakukan secara mendalam terlebih dahulu dengan memprioritaskan urutan abjad. Program menelusuri dari B ke A lalu ke C hingga mencapai I. Karena pencarian dilakukan mendalam sesuai urutan abjad, dengan input yang sama pada kasus uji 3, program menghasilkan solusi yang jauh berbeda dan tidak optimal dihitung dari jarak simpul asal ke simpul tujuan. Hal ini dapat terjadi ketika simpul tetangga yang menuju solusi optimal berada pada urutan abjad yang lebih tinggi dibandingkan simpul tetangga lainnya.

4.4.5 Kasus Uji 5

Input

Fitur: *Explore Friends*
File: TC2.txt
Akun asal: K
Akun tujuan: G
Metode: BFS

Hasil:



Analisis:

Dari visualisasi graf di atas, dapat dilihat dan disimpulkan bahwa simpul K dan simpul G tidak memiliki jalur koneksi di antara keduanya (tidak bisa saling terhubung). Oleh karena itu, algoritma BFS maupun DFS tidak bisa menemukan solusi untuk kasus ini dan menampilkan pesan “Tidak ada jalur koneksi yang tersedia. Anda harus memulai koneksi baru itu sendiri.”

BAB 5

KESIMPULAN, SARAN, DAN REFLEKSI

5.1 Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma semester 2 2020/2021 berjudul "Pengaplikasian Algoritma BFS dan DFS dalam Fitur People You May Know Jejaring Sosial Facebook", kami berhasil membuat *Desktop Application* yang dapat melakukan fitur *explore friends* dan *friend recommendation* sederhana. Fitur *friend recommendation* digunakan untuk mencari teman yang diurutkan berdasarkan *mutual friend* sebuah akun. Fitur *explore friends* digunakan untuk mencari koneksi antara dua akun berbeda. Jalur koneksi ini dapat dicari menggunakan algoritma BFS dan DFS sesuai dengan masukan pengguna.

Dari pengerjaan tugas besar ini, kami mendapatkan beberapa kesimpulan, yaitu:

1. Algoritma BFS tidak akan menemukan jalan buntu. Jika ada satu solusi, BFS akan menemukannya. Jika ada lebih dari satu solusi, maka solusi minimum akan ditemukan (solusi terbaik).
2. Algoritma BFS membutuhkan memori cukup besar karena menyimpan semua *node* yang terdapat dalam satu pohon. Selain itu, BFS membutuhkan waktu yang cukup lama karena akan menguji tiap *n* level untuk menemukan atau mendapatkan solusi pada level yang ke *(n-1)*.
3. Berbeda dengan algoritma BFS, algoritma DFS memakai memori yang lebih sedikit. DFS hanya menyimpan sekitar *bd* dimana b adalah faktor percabangan dan d adalah kedalaman solusi, jauh berbeda dengan BFS yang harus menyimpan semua simpul yang pernah dibangkitkan.
4. Saat menggunakan algoritma DFS, jika pohon yang dibangkitkan mempunyai level yang dalam (tak terhingga), tidak ada jaminan untuk menemukan solusi, artinya algoritma DFS tidak *complete*.
5. Algoritma DFS juga tidak optimal. Hal ini dibuktikan jika terdapat lebih dari satu solusi yang sama tetapi berada pada level yang berbeda, DFS tidak menjamin dapat menemukan solusi terbaik.

5.2 Saran

Saran-saran yang dapat kami berikan untuk tugas besar IF2211 Strategi Algoritma semester 2 2020/2021 adalah:

1. Menggunakan *tools* dan *stack* yang dapat berjalan di semua sistem operasi yang umum digunakan. Hal ini penting karena pengguna komputer dengan sistem operasi selain Windows kesulitan dalam mengerjakan dan menjalankan tugas besar ini.
2. Konsep dari algoritma BFS dan DFS dalam program ini dapat dikembangkan lebih jauh dengan optimalisasi pada *source code* yang ada untuk menghemat waktu komputasi dan memori. Selain itu, fitur *friends recommendation* dan *explore friends* dapat diintegrasikan untuk membuat media sosial di masa mendatang.
3. Fitur *friends recommendation* dan *explore friends* yang ada pada program ini juga bisa dimanfaatkan bagi para mahasiswa di ITB untuk bisa membangun jejaring sosial satu dengan lainnya. Hal ini bisa dicapai dengan memanfaatkan LMS yang sebenarnya bisa dikembangkan juga menjadi media sosial.

5.3 Refleksi

Setelah menyelesaikan tugas besar IF2211 Strategi Algoritma semester 2 2020/2021, kami dapat merefleksikan beberapa hal, yaitu:

1. Perencanaan dan pembagian tugas sangat penting dalam mengerjakan tugas besar secara berkelompok.
2. Lebih merapikan *source code* agar program lebih mudah dipahami.
3. Pentingnya *comments* dalam bagian-bagian kode program sehingga rekan-rekan tim yang lain dapat memahami bagian kode yang kita tulis.
4. Dalam menyelesaikan sebuah masalah dengan pendekatan informatika, penting bagi kita untuk menyusun strategi algoritma yang sesuai sehingga penyelesaian masalah bisa dilakukan dengan efisien.

DAFTAR PUSTAKA

Munir, Rinaldi dan Maulidevi, Nur Ulfa. (2021), Breadth/Depth First Search (BFS/DFS) Bagian 1. Diakses online dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf pada 17 Maret 2021.

Munir, Rinaldi dan Maulidevi, Nur Ulfa. (2021), Breadth/Depth First Search (BFS/DFS) Bagian 2. Diakses online dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf pada 17 Maret 2021.

Vivadifferences, 8 Difference Between DFS (Depth First Search) And BFS (Breadth First Search) In Artificial Intelligence. Diakses online dari https://vivadifferences.com/difference-between-dfs-and-bfs-in-artificial-intelligence/ pada 23 Maret 2021.