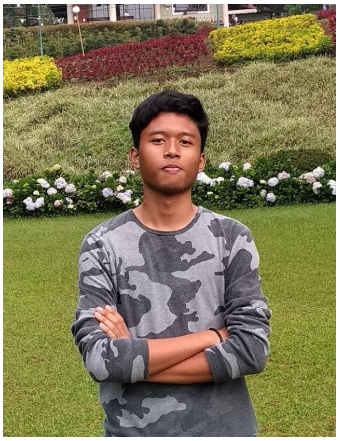
LAPORAN TUGAS BESAR II
IF2211 STRATEGI ALGORITMA
Semester II Tahun 2020/2021



Dipersiapkan oleh:
Kelompok 24 - UnlinkedIn

13519010	Rexy Gamaliel Rumahorbo
13519021	Arjuna Marcelino
13519159	Benidictus Galih Mahar Putra

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2021

Daftar Isi

I. Deskripsi Tugas

1.1 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.

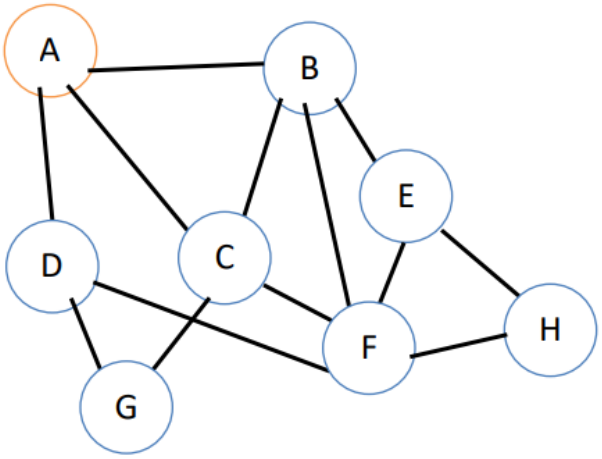
1.2 Contoh *Input* dan *Output* Program

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. Contoh input berkas file eksternal

Visualisasi graf pertemanan yang dihasilkan dari file eksternal:



Gambar 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 3. Hasil output yang diharapkan untuk rekomendasi akun A

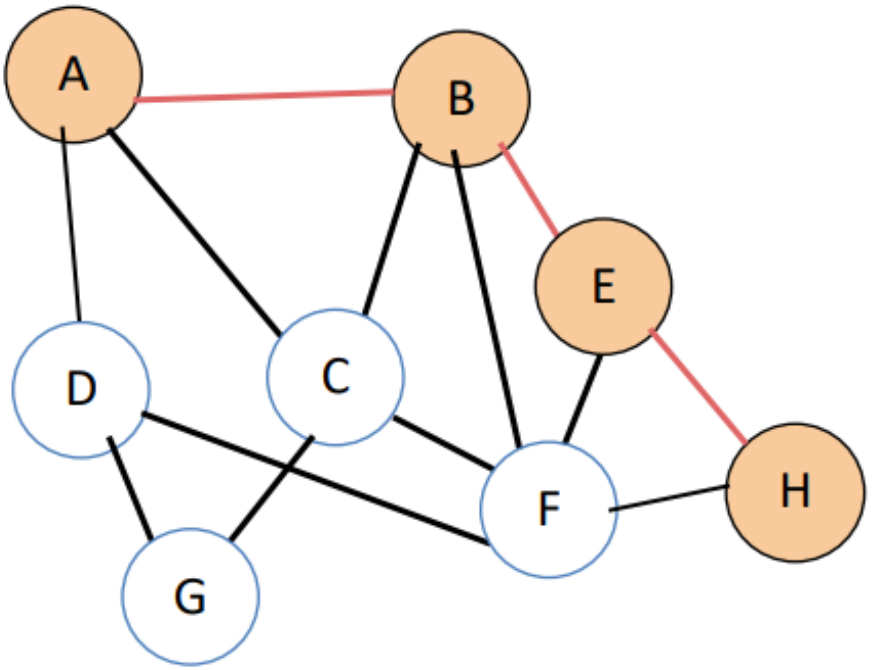
Untuk fitur explore friends, misalnya pengguna ingin mengetahui seberapa jauh jarak antara akun A dan H serta bagaimana jalur agar kedua akun bisa terhubung.

Berikut output yang diharapkan untuk penelusuran menggunakan BFS.

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

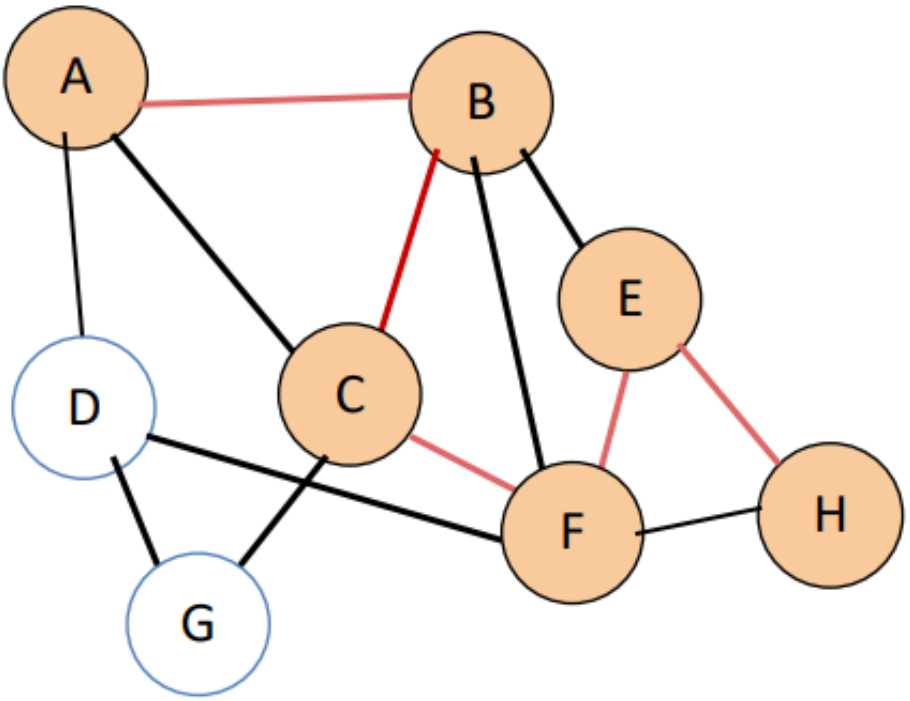
Gambar 4. Hasil output akun Nth degree connection akun A dan H menggunakan BFS

Perhatikan busur antara akun A dan H, terbentuk salah satu jalur koneksi sebagai berikut: A-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 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 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.

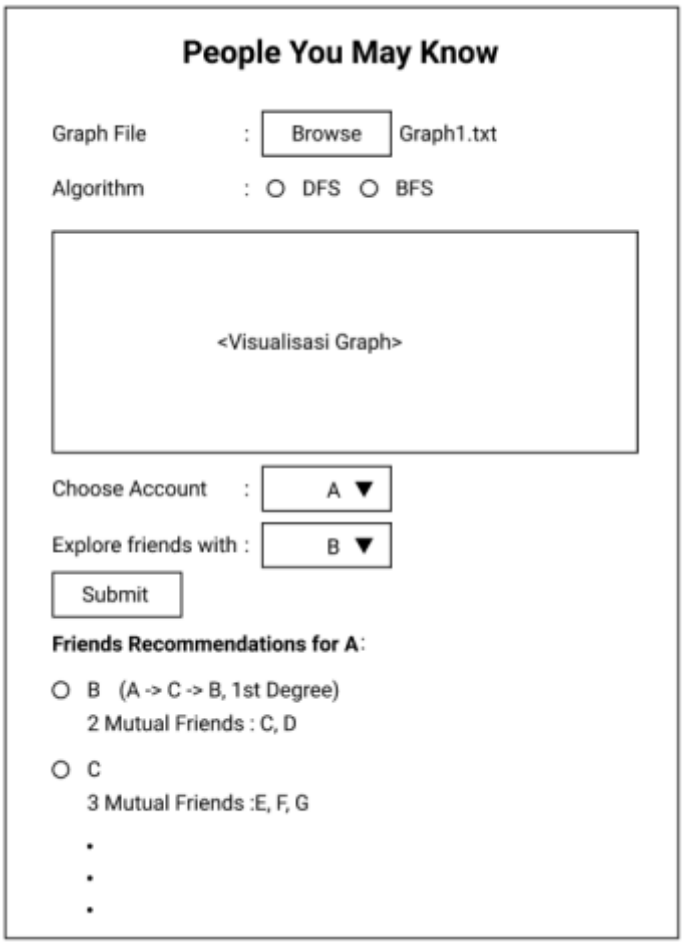
Misalnya terdapat dua orang baru, yaitu J dan I yang hanya terhubung antara J-I. Maka jalur koneksi yang dibentuk dari A ke J adalah.

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

Gambar 7. Hasil output tidak ada jalur koneksi antara A dan J

1.3 Spesifikasi Program

Aplikasi yang akan dibangun dibuat berbasis GUI. Berikut ini adalah contoh tampilan dari aplikasi GUI yang akan dibangun.



Gambar 8. Tampilan layout dari aplikasi desktop yang dibangun

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.

Program yang dibuat harus memenuhi spesifikasi wajib sebagai berikut:

1) Buatlah program dalam bahasa C# untuk melakukan penelusuran social network facebook sehingga diperoleh daftar rekomendasi teman yang sebaiknya di-add. Penelusuran harus memanfaatkan algoritma BFS dan DFS.

2) Awalnya program menerima sebuah berkas file eksternal yang berisi informasi pertemanan di facebook. 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 (lebih jelasnya akan diberikan contoh pada bagian 3).

3) Program kemudian 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.

   Proses visualisasi ini boleh memanfaatkan pustaka atau kakas yang tersedia. Sebagai referensi, salah satu kakas yang tersedia untuk melakukan visualisasi adalah MSAGL (https://github.com/microsoft/automatic-graph-layout) Berikut ini adalah panduan singkat terkait penggunaan MSAGL oleh tim asisten yang dapat diakses pada:
   https://docs.google.com/document/d/1XhFSpHU028Gaf7YxkmdbluLkQgVl3MY6gt1tPL30LA/edit?usp=sharing

4) Terdapat dua fitur utama, yaitu:

   a) Fitur Friend Recommendation

      i) Program menerima sebuah pilihan akun dari user yang hendak dicari rekomendasi temannya. Pemilihan nama akun akan diterima melalui GUI. Cara pemilihan dibebaskan, bisa input dari keyboard atau meng-klik langsung sebuah node dari graf.

ii) Program akan menampilkan daftar rekomendasi teman seperti pada fitur People You May Know facebook berupa nama akun tersebut secara terurut mulai dari mutual friend terbanyak antar kedua akun beserta daftar nama akun mutual friend.

b) Fitur Explore Friends

i) Dua akun yang tidak memiliki mutual friend, masih memiliki peluang untuk berteman jika kedua akun mempunyai common Nth degree connection, yaitu jalur yang menghubungkan kedua akun yang terpisah sejauh N akun (node pada graf).

ii) Program menerima pilihan dua akun yang belum berteman.

iii) Program akan menampilkan nilai N-th degree connection antar kedua akun dan memberikan jalur melalui akun mana saja sampai kedua aku bisa terhubung.

iv) Dari graph yang sudah dibentuk, aplikasi harus dapat menyusun jalur koneksi hasil explore friends antara akun satu dengan akun yang ingin dituju Aplikasi juga harus dapat menunjukkan langkah-langkah pencarian, baik dengan algoritma BFS maupun DFS.

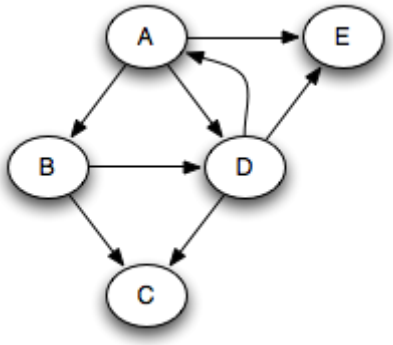
v) Jika tidak ditemukan jalur koneksi sama sekali antar kedua akun karena graf not fully connected, maka tampilkan informasi bahwa kedua akun tidak dapat terhubung.

5) Mahasiswa tidak diperkenankan untuk melihat atau menyalin library lain yang mungkin tersedia bebas terkait dengan pemanfaatan BFS dan DFS.

II. Landasan Teori

2.1 Traversal Graf (*Graph Traversal*)

Traversal graf atau *graph traversal* adalah sebuah proses mengunjungi setiap simpul dengan cara yang sistematik. Traversal graf dapat digunakan untuk mencari jalur dalam suatu graf dari titik asal ke titik tujuan, mencari jalur terpendek antara dua *node/vertex*/simpul, menemukan semua jalur yang bisa dilalui dari titik asal ke titik tujuan, dan menyortir graf secara topologis.



Gambar 9. Contoh suatu graf
(https://www.cs.cornell.edu/courses/cs2112/2012sp/lectures/lec24/lec24-12sp.html)

Secara umum, traversal graf adalah algoritma pencarian solusi yang tidak ada informasi tambahan (*uninformed/blind search*). Representasi graf dalam proses pencarian ini dapat dibedakan menjadi dua jenis pendekatan, yaitu graf statis dan graf dinamis. Graf statis adalah graf yang sudah terbentuk sebelum proses pencarian dilakukan (graf direpresentasikan sebagai struktur data). Sedangkan graf dinamis adalah graf yang terbentuk saat proses pencarian dilakukan (graf tidak tersedia sebelum pencarian, graf dibangun selama pencarian solusi).

Algoritma traversal graf terbagi menjadi dua, yaitu pencarian melebar (*breadth first search* atau BFS) dan pencarian mendalam (*depth first search* atau DFS). Jika setiap simpul dalam grafik akan dilintasi oleh algoritma berbasis pohon (seperti DFS atau BFS), maka algoritma harus dipanggil setidaknya satu kali untuk setiap komponen yang terhubung dari grafik. Ini mudah dilakukan dengan melakukan iterasi melalui semua simpul pada graf, melakukan algoritma pada setiap simpul yang masih belum dikunjungi saat diperiksa. Penjelasan lebih rinci mengenai kedua algoritma tersebut akan dibahas pada bagian berikutnya.

2.2 Pencarian Melebar (*Breadth First Search*/BFS)

BFS atau *Breadth First Search* merupakan pencarian graf yang dilakukan dengan mengunjungi graf secara melebar (mengunjungi simpul - simpul tetangganya dahulu) . Teknik pencarian BFS dimulai dengan mengunjungi sebuah simpul awal. Kemudian kunjungi simpul - simpul yang bertetangga dengan simpul awal tadi. Setelah itu, kunjungi lagi simpul - simpul yang bertetangga selanjutnya dan masih belum dikunjungi sebelumnya.

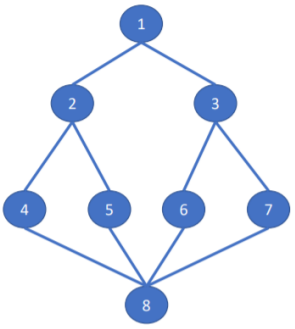
Pencarian solusi dibagi menjadi dua pendekatan berdasarkan ketersediaan graf sebelum pencarian.

```
procedure BFS(input v:integer)
{
  Traversal graf dengan algoritma pencarian BFS.
  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke
layar
}

Deklarasi
  w : integer
  q : antrean

  procedure BuatAntrean(input/output q : antrean)
  { membuat antrean kosong, kepala(q) diisi 0 }

  procedure MasukAntrean(input/output q:antrean,
        input v:integer)
  { memasukkan v ke dalam antrean q pada posisi
        belakang }

  procedure HapusAntrean(input/output q:antrean,
        output v:integer)
  { menghapus v dari kepala antrean q }

  function AntreanKosong(input q:antrean) -> boolean
  { true jika antrean q kosong, false jika sebaliknya }

Algoritma
  BuatAntrean(q) { buat antrean kosong }

  write(v) { cetak simpul awal yang dikunjungi }
  dikunjungi[v] <- true { simpul v telah dikunjungi,
        tandai dengan true }

  MasukAntrean(q,v) { masukkan simpul awal kunjungan ke
        dalam antrean }
        { kunjungi semua simpul graf selama antrean
        belum kosong }

  while not AntreanKosong(q) do
    HapusAntrean(q,v) { simpul v telah dikunjungi,
        hapus dari antrean }
    for tiap simpul w yang
        bertetangga dengan simpul v do
      if not dikunjungi[w] then
        write(w) {cetak simpul yang dikunjungi}
        MasukAntrean(q,w)
```

Gambar 10. *Pseudocode* dari algoritma BFS

2.2.1 Graf Statis

Graf statis merupakan graf yang sudah terbentuk sebelum pencarian dilakukan. Graf yang disediakan disimpan dalam struktur data program dapat berbentuk array, matriks, graf, dsb. Untuk memudahkan pencarian BFS, dibuat sebuah struktur data yang terdapat matriks ketetanggaan, antrian, dan tabel *boolean*. Matriks ketetanggan dibuat berukuran *n* x *n* dengan *n* merupakan banyaknya simpul yang terdapat dalam graf. Matriks A_{ij} diisi dengan ‘0’, jika simpul i dan simpul j tidak bertetangga, sedangkan Matriks A_{ij} diisi dengan ‘1’, jika simpul i dan simpul j bertetangga. Kemudian antrian *q* diisi dengan simpul - simpul yang telah dikunjungi sebelumnya. Untuk tabel *boolean* B_i diisi dengan true, jika simpul i telah dikunjungi sedangkan tabel *boolean* B_i diisi dengan false, jika simpul i belum dikunjungi.



Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

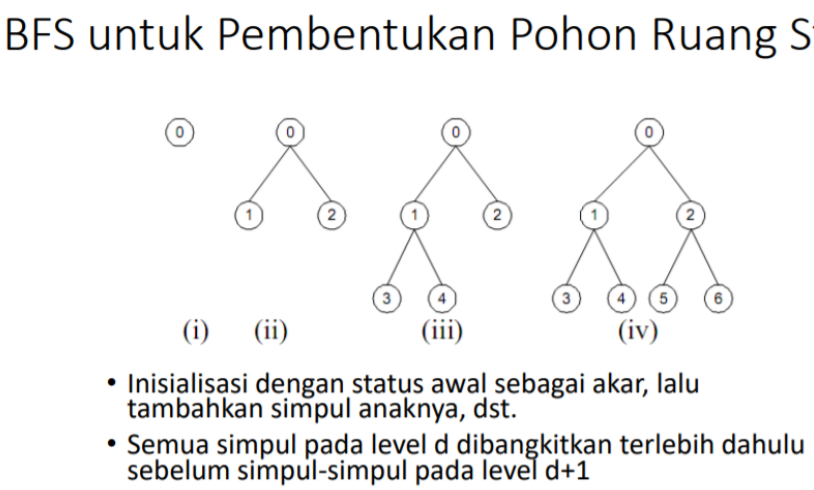
Urutan simpul2 yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

Gambar 11. Langkah Pencarian BFS dan Struktur Datanya

(informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf)

2.2.2 Graf Dinamis

Graf dinamis merupakan graf yang belum dibentuk sebelum pencarian dan akan dibentuk pada saat pencarian sedang berlangsung. Untuk graf dinamis struktur data dari BFS mirip dengan graf statis. Graf dinamis direpresentasikan dengan beberapa simpul dan cabang. Simpul - simpul graf didefinisikan sebagai *problem state*, simpul akar merupakan *initial state*, dan simpul daun merupakan *final state.* Cabang dari pohon dinamis merupakan operator legal menuju *goal*/*final state*. Kemudian terdapat ruang status yang merupakan himpunan semua simpul dan ruang solusi yang merupakan status solusi. Dalam pencariannya terdapat evaluasi yang merepresentasikan keefisienan sebuah teknik BFS ini. Aspek - aspek evaluasi terdapat aspek *completeness*, *optimality, time complexity,* dan *space complexity*. aspek *completeness* merupakan aspek yang berisi solusi terjamin yang jika dapat dibentuk. Kemudian *optimality* merupakan teknik penjaminan apakah terdapat *cost* terkecil yang dapat dibentuk sebuah solusi. *Time complexity* merupakan waktu yang diperlukan untuk mencapai solusi. Terakhir *space complexity* yang merupakan seberapa banyak memori yang diperlukan untuk melakukan pencarian. Kemudian kompleksitas waktu dan ruang diukur dengan *branching factor*, *depth*, dan *maximum depth*. Pertama *branching factor* merupakan maksimum pencabangan yang mungkin untuk suatu simpul. Kemudian *depth* merupakan kedalaman dari solusi yang memiliki *cost* terendah. Selain itu, *maximum depth* merupakan kedalaman maksimum yang dapat dibentuk dari ruang status. Nilai *maximum depth* dapat mencapai *infinite*(∞).



Gambar 12. Langkah Pencarian BFS dalam Graf Dinamis

(informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf)

2.3 Pencarian Mendalam (*Depth First Search*/DFS)

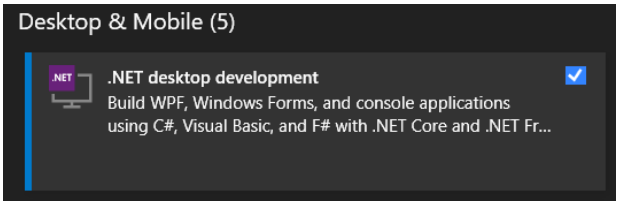
Pencarian Mendalam/*Depth First Search* merupakan algoritma penelusuran graf yang dilakukan dengan menelusuri sebuah node yang bertetanggaan dengan suatu node yang sedang dikunjungi hingga tidak ada lagi node yang dapat dikunjungi atau sampai menemukan solusi. Pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri dan dilanjutkan pada node sebelah kanan. Jika solusi ditemukan, maka tidak diperlukan proses *backtracking* yaitu penelusuran balik untuk mendapatkan jalur yang diinginkan. Jika belum ditemukan, maka dapat dilakukan proses runut balik/*backtracking*, yaitu mencari node lain yang dapat ditelusuri dari node terakhir yang dikunjungi. Pencarian pada algoritma ini akan berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi. Layaknya BFS, pemecahan masalah dengan BFS dapat dilakukan secara statis maupun dinamis.

```
procedure DFS(input v:integer)
{
  Mengunjungi seluruh simpul graf dengan algoritma
        pencarian DFS
  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang 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 13. *Pseudocode* dari algoritma BFS

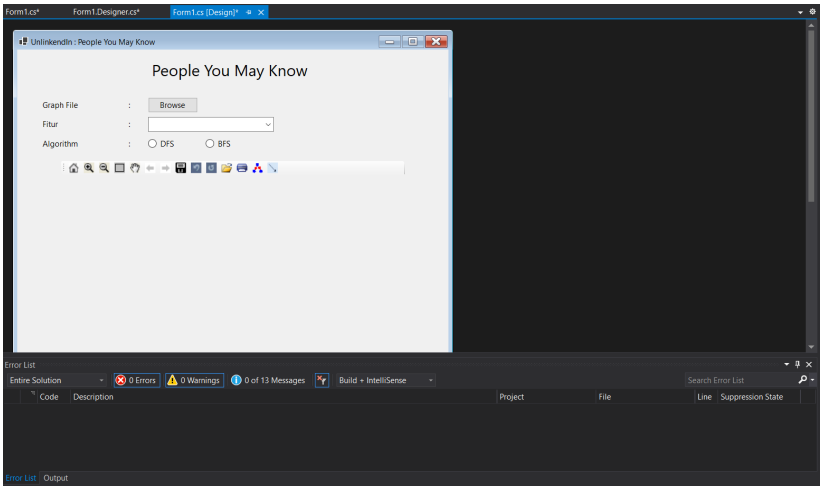
2.4 C# *Desktop Application Development*

Dalam pengembangan sebuah aplikasi, terdapat *tools* yang dapat digunakan untuk mem-*build* sebuah aplikasi salah satunya adalah Visual Studio. Dalam Visual Studio perkakas yang dapat digunakan salah satunya adalah .NET Core yang digunakan untuk mem-*build* aplikasi dari berbagai platform. Untuk mendapatkan .NET Core tsb dapat diunduh pada https://dotnet.microsoft.com/download. Setelah itu, untuk dapat mem-*build* aplikasi dapat mengunduh visual studio dengan perkakas .NET desktop development pada https://visualstudio.microsoft.com/downloads/



Gambar 14. .NET pada saat Install Visual Studio

Dalam Visual Studio diberikan *workspace project* yang ditampilkan pada *Solution explorer*. Untuk melihat folder *project* dapat dibuka dengan *view* > *Solution explorer* atau dengan Ctrl + Alt + L. Dalam *Solution explorer* terdapat dua file yang terbentuk di awal yaitu *Form1.cs* dan *Program.cs*. *Form1.cs* digunakan untuk membuat tampilan GUI dan implementasi setiap objek yang ditampilkan di GUI tsb. Untuk melihat GUI *project* dapat dibuka *Solution explorer* kemudian klik kanan pada file Form1.Designer.cs. Kemudian pilih *View designer* atau dengan Shift + F7. Untuk menambahkan fitur atau object pada GUI dapat ditambahkan dengan membuka *toolbox* di sebelah kiri GUI *Designer*. *Toolbox* ini dapat langsung diaplikasikan dengan menarik salah satu fitur yang tersedia di *toolbox* kemudian diletakan pada GUI yang sedang dibuka. Untuk mengisi kode yang ingin dijalankan oleh fitur ini, dapat dilakukan dengan *double klik* pada fitur tadi dan terbentuk sebuah *method* pada file *form1.cs*.



Gambar 15. Designer dari Form1.cs

C# juga bahasa yang berorientasi objek sehingga pemecahan setiap programnya menjadi sebuah objek yang memiliki peran masing - masing. Untuk syntax C# sendiri sebagian besar mirip dengan syntax java.

III. Analisis Pemecahan Masalah

3.1 Langkah-Langkah Pemecahan Masalah

Langkah-langkah yang kami terapkan dalam menyelesaikan permasalahan ini adalah sebagai berikut.

1. Membaca file eksternal dengan format file .txt
2. Mengubah input file eksternal ke dalam bentuk integer *numVar* yang menyatakan banyak variabel, *array of string* yang berisi nama-nama variabel, *dictionary* tuple <string, int> berisi pasangan <variabel, index> untuk memudahkan pencarian index, dan matriks boolean yang menyatakan keterhubungan antar simpul.
3. Fitur friend recommendation mencari isi / nama graph yang terhubung dengan jarak 2
4. Isi / nama graph yang disimpan diurut dari jumlah jalur terbanyak

[Nama]	[Jumlah Keterhubungan yang jaraknya 2]
A	2
B	3

Tabel 1. Isi dari Tabel Friend Recommendation

5. Fitur explore friend mencari semua jalur graf yang ada dari A (initial state) sampai B (goal state)
6. Kalau tidak ditemukan goal state, print tidak jalur yang menghubungkan

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

3.2.1 Fitur Friend Recommendation

Dalam fitur friend recommendation memakai algoritma BFS, dengan struktur data *string id* dan *integer id_main* sebagai input awal, *integer total_var* sebagai banyaknya variabel yang ada di program, *dictionary of tuple <string, int> varDictionary* sebagai variabel dan indeks, *Queue of string* antrian sebagai antrian node, *array of string list_var* sebagai matriks yang berisi variabel, *matriks of boolean graph_link* sebagai matriks yang merepresentasikan simpul i dan j yang saling terhubung, *array of boolean link_visited* sebagai *array* yang sudah dikunjungi, *array of integer total_linked* sebagai *array* yang jumlah keterhubungan dengan indeks yang mewakili variabel, dan *matriks of string mutual_friends* sebagai variabel yang terhubung dengan *initial* dan *goal* node.

3.2.2 Fitur Explore Friend

a. BFS

Pada fitur *explore friend* dengan algoritma BFS, diperlukan struktur data berikut untuk merepresentasikan graf: *adjacency matrix* berupa matriks *boolean* berukuran N x N di mana N adalah banyak variabel/node, *queue* of node yang digunakan untuk menyimpan deretan node yang akan ditelusuri, *array of boolean* untuk mencatat node yang telah dikunjungi, serta struktur data pelengkap lainnya seperti *dictionary* <string, int> dan *array of string* untuk 'menerjemahkan' nama node ke dalam index dan sebaliknya. Selain itu, disimpan juga nama node asal dan node tujuan.

Pada pemecahan masalah ini, BFS direpresentasikan di dalam kelas BFS dengan atribut-atribut seperti yang telah disebutkan di atas. BFS memiliki method utama ConstructPath() yang mengembalikan *stack of string* yang merupakan jalur yang hasil pencarian algoritma BFS. Selain itu, BFS juga memiliki method getName(int) dan getIdx(string) untuk 'menerjemahkan' nama node ke nomor indeks dan sebaliknya.

b. DFS

Pada fitur *explore friend* dengan algoritma DFS, diperlukan struktur data berikut untuk merepresentasikan graf: *adjacency matrix* berupa matriks *boolean* berukuran N x N di mana N adalah banyak variabel/node, *stack* of node yang digunakan untuk menyimpan deretan node yang akan ditelusuri, *array of boolean* untuk mencatat node yang telah dikunjungi, serta struktur data pelengkap lainnya seperti *dictionary* <string, int> dan *array of string* untuk 'menerjemahkan' nama node ke dalam index dan sebaliknya. Selain itu, disimpan juga nama node asal dan node tujuan.

Pada pemecahan masalah ini, DFS direpresentasikan di dalam kelas DFS dengan atribut-atribut seperti yang telah disebutkan di atas. DFS memiliki method utama ConstructPath() yang mengembalikan *stack of string* yang merupakan jalur yang dihasilkan dari pencarian algoritma DFS. Selain itu, DFS juga memiliki method getName(int) dan getIdx(string) untuk 'menerjemahkan' nama node ke nomor indeks dan sebaliknya.

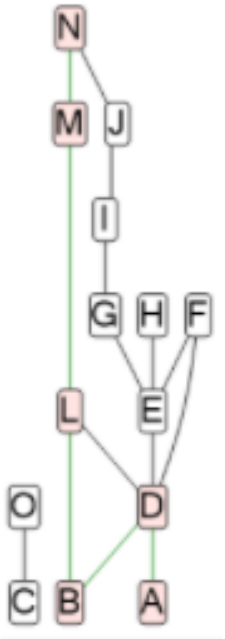
Sekilas, Algoritma yang digunakan dan proses mapping yang diterapkan pada algoritma BFS dan DFS memiliki kemiripan. Namun, ada perbedaan yang cukup mendasar yaitu dalam memproses node selanjutnya yang akan dikunjungi. Jika pada algoritma BFS, node yang akan dikunjungi disimpan di dalam queue yang artinya node yang baru ditambahkan akan dikunjungi paling akhir. Sedangkan, pada algoritma DFS, node yang yang akan dikunjungi disimpan dalam stack yang artinya node yang baru ditambahkan akan dikunjungi pertama kali.

3.3 Contoh Ilustrasi Kasus Lain

Contoh Kasus :

15
A D
B D
B L
C O
D L
D E
D F
E G
E F
E H
G I
I J
J N
N M
M L

Dengan visualisasi graf sebagai berikut :



Gambar 15. Visualisasi Graf dari Contoh Kasus

IV. Implementasi dan Pengujian

4.1 Implementasi Program

4.1.1 Friend Recommendation

```
/* Atribut */
id                  : string
id_main, total_var  : integer
varDictionary       : dictionary of tuple <string, int>
antrian             : Queue of string
list_var            : array of string
graph_link          : matriks of boolean
link_visited        : array of boolean
total_linked        : array of integer
mutual_friends      : matriks of string

/* inisialisasi simpul awal dengan nilai true */
link_visited[id_main] ← true

/* memasukan simpul awal ke dalam antrian */
antrian.Enqueue(list_var[id_main])

/* memasukan simpul - simpul tetangga ke dalam antrian
 * dan set nilai boolean kunjunginya menjadi true */
i traversal [0..total_var]
    if(graph_link[id_main][i]) then
        antrian.Enqueue(list_var[i])
        link_visited[i] ← true
{ end traversal loop }
/* mengeluarkan simpul awal dari antrian */
antrian.Dequeue()

/* Mengakses antrian satu per satu */
while (antrian.Count > 0) do
    /* graf level kedua untuk mencari final node dan mutual friends */
    i traversal [0..total_var]
        /* jika ada keterhubungan antara antrian
 * pertama dengan pencarian node */
        if (graph_link[getIdx(antrian.Peek())][i])
              /* jika node belum dikunjungi */
              if (not link_visited[i])
                    /* assign nilai string ke dalam
```

```
 * matriks mutual friends */
                    mutual_friends[i][total_linked[i]] ←
antrian.Peek()
                    /* nEff baris matriks bertambah */
                    total_linked[i] ← total_linked[i] + 1
   { end traversal loop }
    antrian.Dequeue()
{ end while loop }
```

4.1.2 Explorer Friend (DFS)

```
/* Atribut */
numVar              : integer
varList             : array of string
varDictionary       : Dictionary tuple <string, int>
varStack            : Stack of string
matriks             : matriks of boolean
awal                : string
akhir       : string
visited             : array of bool

Stack<string> path ← Stack<string>()

if (awal.Equals(akhir)) then

            path.Push(awal)
      → path

Dictionary<string, string> precNode ← Dictionary<string, string>()
// mencatat node sebelumnya dari sebuah node
int currentIdx ← getIdx(awal)
int dIdx ← getIdx(akhir)

visited[currentIdx] ← true
varStack.Push(awal)

while (varStack.Count > 0) do
      // indeks node yang sedang dievaluasi
      currentIdx ← getIdx(varStack.Pop())

      // jika node tujuan ditemukan
      if (currentIdx = dIdx) then
        break

      int j ← 0
```

```
      // mengevaluasi setiap node yang bertetanggan dengan
currentNode
      while (j < numVar) do
            int othersIdx ← 0
            bool found ← false
            // memprioritaskan berdasarkan abjad
            while (othersIdx < numVar and not found) do
                  if (matriks[currentIdx, othersIdx] and not
visited[othersIdx]) then
                        visited[othersIdx] ← true
                        varStack.Push(getName(othersIdx))
                        precNode.Add(getName(othersIdx),
getName(currentIdx))
                        currentIdx ← othersIdx
                        found ← true
            othersIdx ← othersIdx + 1
      j ← j + 1

      // menyimpan jalur ke dalam stack
      if (precNode.ContainsKey(akhir)) then
            string tempStr ← akhir
            path.Push(tempStr)
            while (tempStr <> awal)
                  tempStr ← precNode[tempStr]
                  path.Push(tempStr)
            → path
      else
            → path
```

4.1.3 Explorer Friend (BFS)

```
/* Atribut */
numVar      : integer
varList           : array of string
varDictionary     : Dictionary<string, int>
varQueue          : Queue<string>
matrix      : matrix of boolean
visited           : array of boolean
sNode, dNode: string
path              : Stack<string>
currentIdx        : integer
othersIdx         : integer
dIdx              : integer
precNode          : Dictionary<string, string>  { mencatat sebuah
node yang dikunjungi sebelum node tersebut }
```

```
{ Initialisasi semua atribut }
currentIdx ← getIdx(sNode)
dIdx ← getIdx(dNode)

{ cek jika kedua node sama }
if (currentIdx == dIdx) then
      { kembalikan path dengan 1 node: sNode }
      path.Push(sNdode)
      → path
endif

visited[currentIdx] ← True
varQueue.Enqueue(sNode)

while (varQueue.Count > 0) do { lakukan sepanjang-panjangnya hingga
varQueue habis }
      currentIdx ← getIdx(varQueue.Dequeue())
      { jika node ditemukan }
      if (currentIdx == dIDx) then
            break
      endif

      { mengecek setiap node yang terhubung }
      for ({setiap node}) do
            if (matrix[currentIdx][othersIdx] && not
visited[othersIdx]) then
                  visited[othersIdx] ← True
                  varQueue.Enqueue(getName(othersIdx))
                  precNode.Add(getName(othersIdx),
getName(currentIdx))
      endfor
endwhile

if ({dNode ada di precNode}) then { jalur ditemukan }
      { backtrack jalur dari dNode ke sNode }
      tempString ← dNode
      path.Push(dNode)
      while (tempString != sNode) do
            tempStr ← precNode[tempStr]
            path.Push(tempStr)
      endwhile
→ path
else
      → NULL
```

4.2 Penjelasan Struktur Data

4.2.1 Friend Recommendation

Friend recommendation menggunakan pencarian secara BFS dengan *queue of string* untuk menyimpan node bertetangga dengan initial node. *String id* menyimpan data node initial dalam bentuk string. *Integer id_main* merupakan node initial dalam bentuk *integer*. *Dictionary tuple* menyimpan data node dan indeksnya pada matriks. *Integer total_var* untuk menyimpan jumlah semua variabel unik yang ada di program. *Array of string list_var* untuk menyimpan *string* semua variabel. *Matriks of boolean graph_link* untuk menyimpan keterhubungan antara node dengan node lainnya. *Array of boolean link_visited* untuk menyimpan data apakah node sudah dikunjungi. *Array of integer total_linked* untuk menyimpan *nEff* baris setiap node. *Matriks of string mutual_friends* untuk menyimpan isi *string* variabel pada node level pertama dan saling terhubung antara node pertama dan node *goal state*.

Struktur Data:

string id

int id_main

dictionary tuple <string, int> varDictionary

Queue of string antrian

integer of total_var

array of string list_var

matriks of boolean graph_link

array of boolean link_visited

array of integer total_linked

matriks of string mutual_friends

4.2.2 Explore Friends

a. Pencarian BFS

Pada algoritma BFS diperlukan struktur data *matrix of boolean* untuk *adjacency matrix* untuk merepresentasikan graf*, queue of string* untuk menyimpan barisan node yang akan ditelusuri, serta *array of boolean* yang menandakan apakah suatu node sudah pernah dikunjungi atau belum. Pada kelas BFS disimpan juga *array of string* yang menyimpan nama-nama node, *dictionary* <string, int> yang menyimpan indeks setiap node, dan *int* numVar yang menyimpan banyak node.

Struktur Data:

int numVar;

string[] varList;

Dictionary<string, int> varDictionary;

Queue<string> varQueue;

bool[,] matrix;

string sNode;

string dNode;

bool[] visited;

b. Pencarian DFS

Pada algoritma DFS diperlukan struktur data *matrix of boolean* untuk *adjacency matrix* untuk merepresentasikan graf*, stack of string* untuk menyimpan barisan node yang akan ditelusuri, serta *array of boolean* yang menandakan apakah suatu node sudah pernah dikunjungi atau belum. Pada kelas BFS disimpan juga *array of string* yang menyimpan nama-nama node, *dictionary* <string, int> yang menyimpan indeks setiap node, dan *int* numVar yang menyimpan banyak node.

Struktur Data:

int numVar;

string[] varList;

Dictionary<string, int> varDictionary;

Stack<string> varStack;

bool[,] matrix;

string sNode;

string dNode;

bool[] visited;

4.3 Penjelasan Tata Cara Penggunaan Program

4.3.1 Persiapan program

1. Download C# sesuai OS masing-masing.

2. Salin link https://github.com/benidictusgalihmp/UnlinkendIn.

3. Pilih direktori yang ingin Anda jadikan sebagai tempat programnya dengan perintah `cd nama-direktori-anda`.

4. Klon link *git repository* dengan perintah berikut

```
git clone https://github.com/benidictusgalihmp/UnlinkendIn
```

5. Buka folder *repository* tersebut dan persiapkan file yang berisi soal yang ingin diselesaikan dan masukkan pada folder test.

6. Buka folder bin kemudian pilih file yang bernama **UnlinkendIn.exe** .

7. Program *People You May Know* dapat dijalankan.

4.3.2 Cara Menggunakan Program

1. Pastikan sudah tersedia file teks yang sudah mengikut format yang sudah ditentukan.

2. Lakukan tahap **Persiapan Program** seperti yang sudah dijelaskan di atas.

3. Jalankan program kemudian.

4. Masukkan file dengan cara menekan tombol **Browse**.

5. Pilih fitur apa yang ingin digunakan. Ada dua fitur yang disediakan, yaitu fitur *friend recommendation* dan *explorer friend*.

6. Untuk fitur *explorer friend*, Anda dapat memilih algoritma yang ingin digunakan (BFS atau DFS).

7. Untuk fitur *friend recommendation*, Anda tidak dapat memilih algoritma yang digunakan, dan sudah ditentukan *by default* oleh program.

8. Pilih Akun yang ingin Anda gunakan di daftar akun.

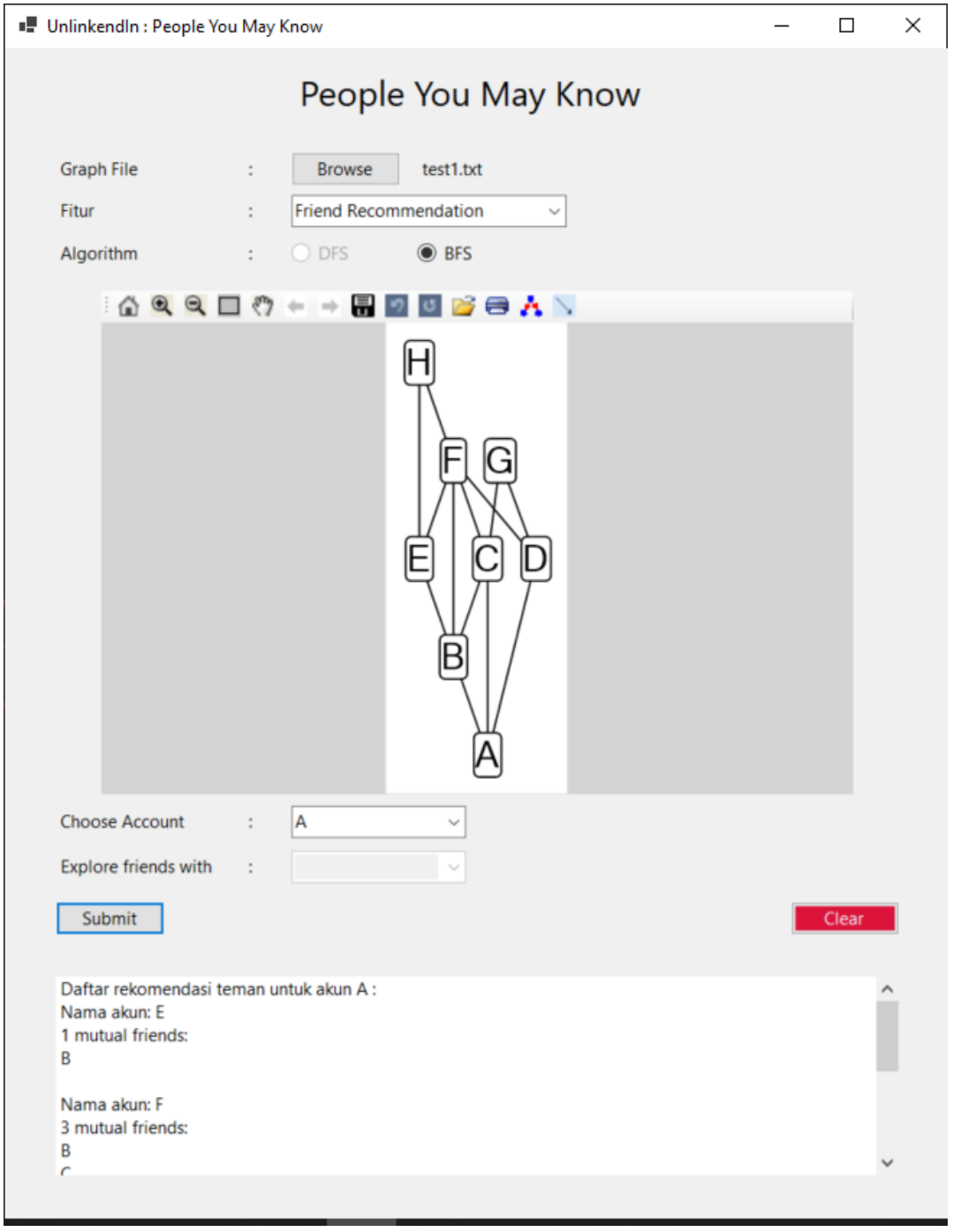
9. Pilih Akun teman atau orang lain yang ingin anda cari tahu keterhubungannya dengan akun pertama (untuk fitur *friend recommendation*, Anda tidak dapat memilih pada pilihan ini).

10. Tekan tombol **Submit** untuk menjalankan program dan program akan menampilkan hasil yang ditemukan beserta visualisasi grafnya.

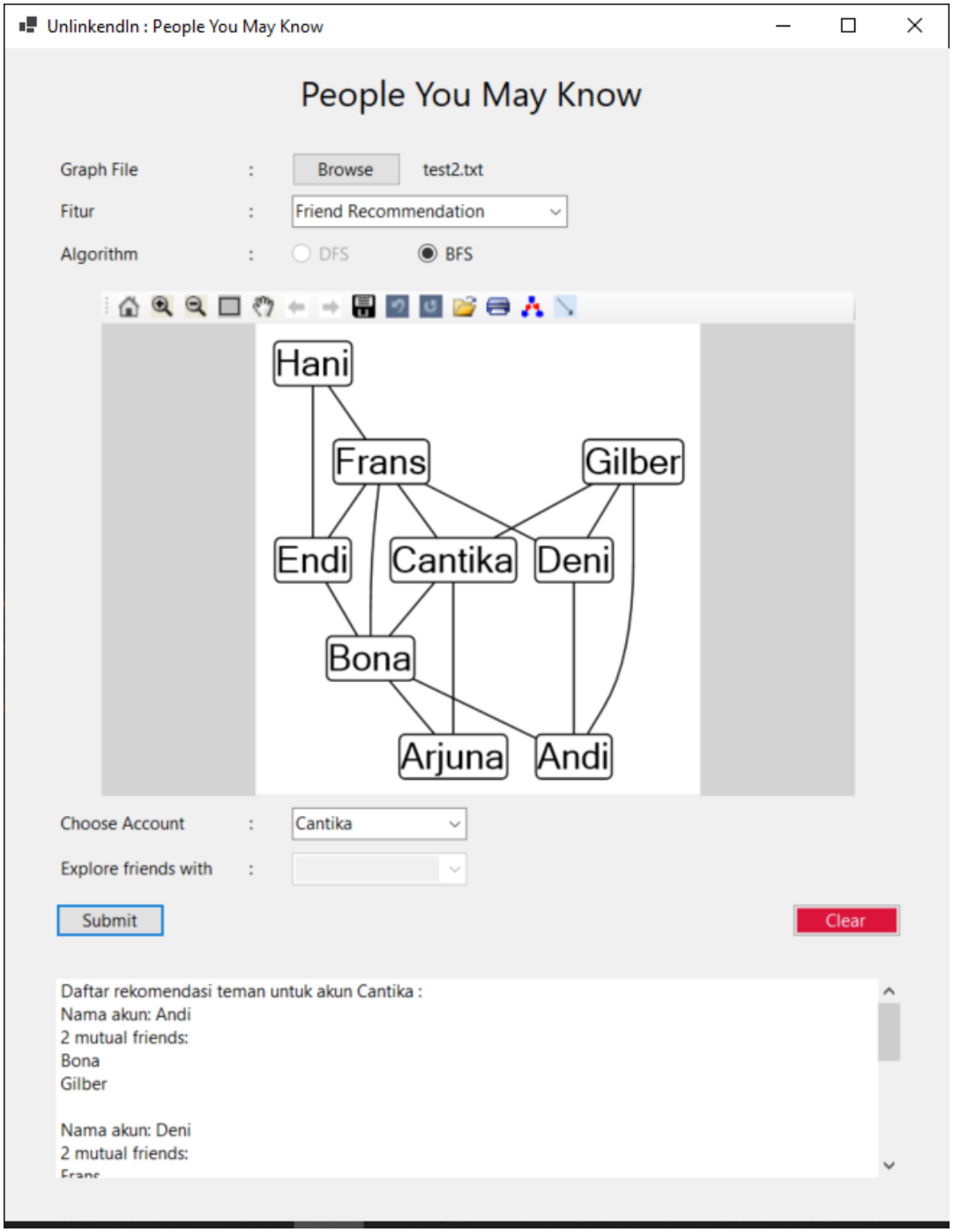
11. Tekan tombol **Clear** untuk menghapus seluruh data dan pilihan yang sudah Anda masukkan.

4.4 Hasil Pengujian

4.4.1 Friend Recommendation



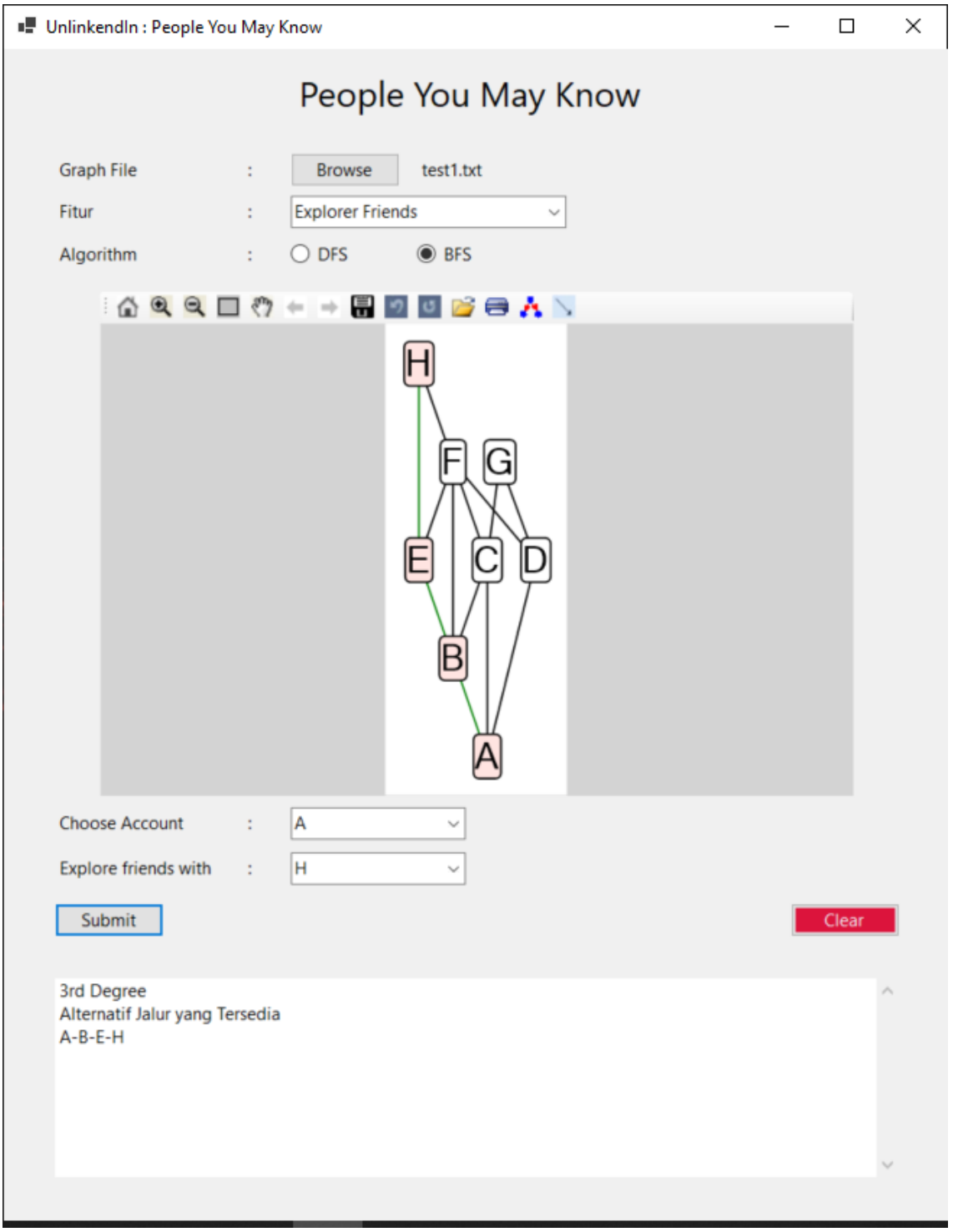
Gambar 16. Hasil Pengujian Fitur *Friend Recommendation*



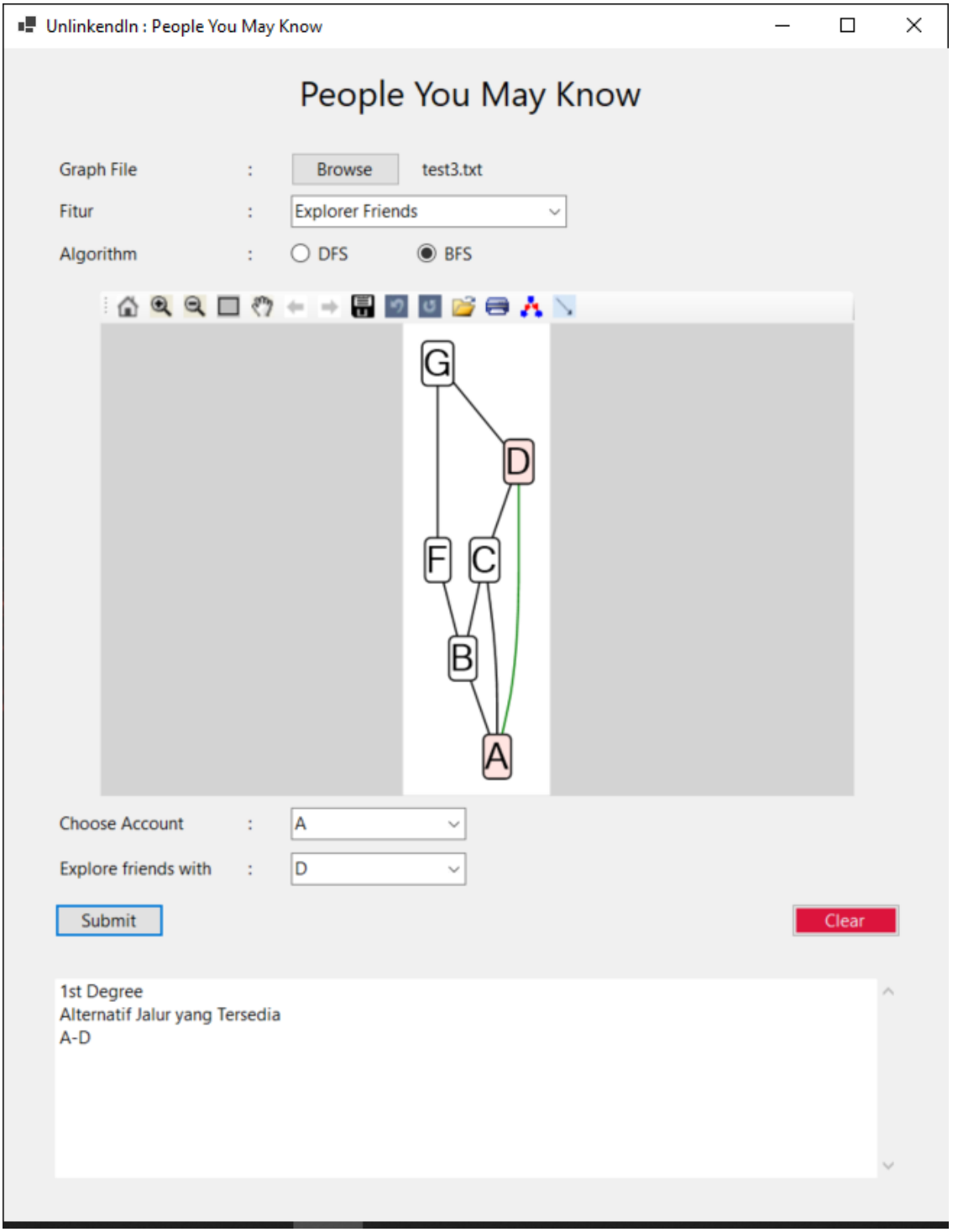
Gambar 17. Hasil Pengujian Fitur *Friend Recommendation 2*

4.4.2 Explore Friends

a. BFS

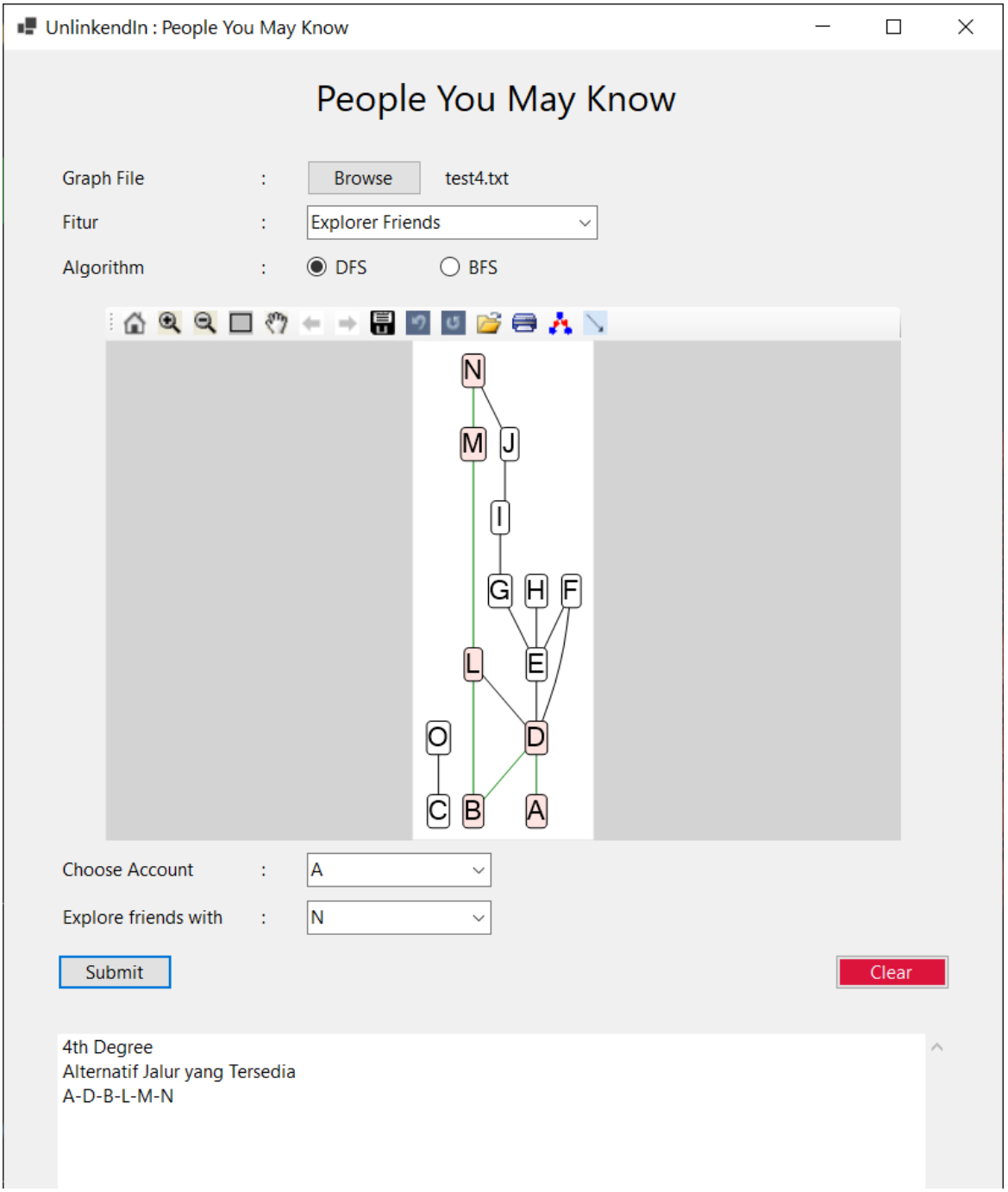


Gambar 18. Hasil Pengujian Fitur *Explore Friend* dengan BFS

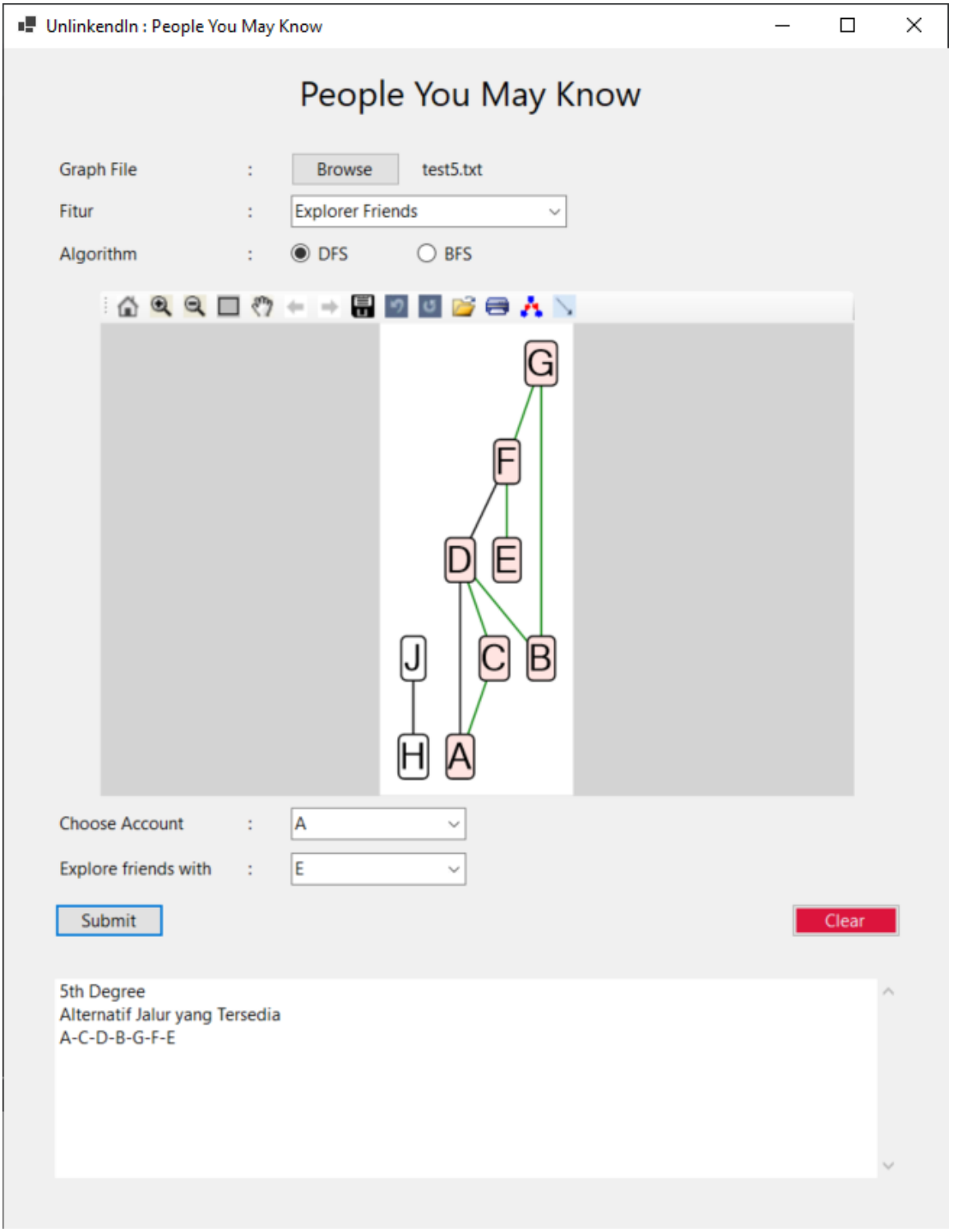


Gambar 19. Hasil Pengujian Fitur *Explore Friend* dengan BFS 2

b. DFS



Gambar 20. Hasil Pengujian Fitur *Explore Friend* dengan DFS



Gambar 21. Hasil Pengujian Fitur *Explore Friend* dengan DFS 2

4.5 Analisis dari Desain Solusi Algoritma BFS dan DFS yang Diimplementasikan

4.5.1 Friend Recommendation

Completeness: yes

Optimality: yes

Time Complexity: $O(V^2)$ dengan V merupakan banyak node

Space Complexity: $O(V^2)$ dengan V merupakan banyak node

Friend Recommendation dapat selalu memberikan solusi. Ketika diberikan sebuah kasus satu keterhubungan (A B), solusi tidak ditemukan tetapi diberikan solusi dengan "Tidak ditemukan friend recommendation". Kasus optimality selalu didapat karena setiap jarak node yang ditempuh memiliki jarak yang sama (jarak terjauh = 2). Time complexity didapat dari banyak loop dari `i traversal [0..total_var]` yang dilakukan dua kali dengan begitu pendekatannya mendekati V^2 . Untuk jumlah memori yang dipakai `matriks graph_link` dan `matriks mutual_friends` sehingga banyak memori yang terpakai terbesar menjadi V^2 .

4.5.2 Explore Friends

a. BFS

Completeness: yes

Optimality: yes

Time Complexity: O(V + E), dengan V: banyak node, E: banyak keterhubungan

Space Complexity: O(V), digunakan untuk mengkonstruksi *queue* dan *boolean array*

Explore Friend dapat selalu memberikan solusi. Kemudian optimality-nya juga selalu mendapatkan jalur terpendek karena pendekatan solusi dilakukan dengan mengunjungi node daun terlebih dahulu dan apakah node tsb merupakan solusi. Jangkauan node yang dicapai lebih jauh karena dilakukan periksa setiap depth bertambah satu. Time complexitynya didapat dari `for ({setiap node}) do`

dan `while (tempString != sNode) do`. Space complexitynya didapat dari `matrix` *boolean array dan* `Queue<string>.`

b. DFS

Completeness: yes

Optimality: no

Time Complexity: O(V*(V-1) + E), dengan V: banyak node, E: banyak keterhubungan

Space Complexity: O(V), digunakan untuk mengkonstruksi *stack* dan *boolean array*

Explore Friend dapat selalu memberikan solusi. Kasus optimality tidak selalu didapat karena setiap node yang ditempuh dicek secara mendalam terlebih dahulu dan kemudian mengecek node tetangga lainnya. Hal ini memungkinkan solusi yang dihasilkan lebih panjang dari pada yang seharusnya. Time complexity didapat dari banyak loop dari `while (j < numVar)` dan dimulai dari `j = 0` yang dilakukan dua kali dengan pengulangan satu lagi dimulai dari `i = j` dengan begitu pendekatannya mendekati $V^*(V-1)$. Untuk jumlah memori yang dipakai `stack varStack` dan `matriks boolean array` sehingga banyak memori yang terpakai terbesar menjadi V .

V. Kesimpulan dan Saran

5.1 Kesimpulan

Algoritma DFS dan BFS dapat diterapkan pada suatu permasalahan pencarian solusi, seperti mencari jalur dalam suatu graf dari titik asal ke titik tujuan, mencari jalur terpendek antara dua *node/vertex*/simpul, menemukan semua jalur yang bisa dilalui dari titik asal ke titik tujuan, dan menyortir graf secara topologis. Salah satu contoh penerapan yang dapat ditemukan dalam sehari-hari adalah hubungan akun pada sosial media. Salah satu fitur yang dapat digunakan dalam sosial media adalah fitur rekomendasi pertemanan dan jalur koneksi keterhubungan antar akun. Kedua permasalahan tersebut dapat diselesaikan dengan algoritma DFS dan BFS.

5.2 Refleksi

Dengan diberikan tugas besar ini, kami menjadi lebih mengerti akan materi tentang DFS dan BFS yang diberikan pada saat perkuliahan. Selain itu, proses development aplikasi dengan menggunakan C# pada Visual Studio membuat kami menjadi lebih paham cara untuk membangun sebuah aplikasi sederhana pada platform Windows khususnya.

Daftar Pustaka

Cornell CIS Computer Science (2012). *Graph Traversal*. CS 2112 Spring 2021 Lecture 24 : Graph Traversal. Diakses pada 25 03, 2021, dari https://www.cs.cornell.edu/courses/cs2112/2012sp/lectures/lec24/lec24-12sp.html

Munir, R., dan Nur Ulfa Maulidevi (2021). *Breadth/Depth First Search (BFS/DFS) (Bagian 1)*. Bahan Kuliah IF2211 Strategi Algoritma. Diakses pada 25 03, 2021, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf

Munir, R., dan Nur Ulfa Maulidevi (2021). *Breadth/Depth First Search (BFS/DFS) (Bagian 2)*. Bahan Kuliah IF2211 Strategi Algoritma. Diakses pada 25 03, 2021, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf

Program Studi Teknik Informatika (2021, 03 03). *Pengaplikasian Algoritma BFS dan DFS dalam Fitur People You May Know Jejaring Sosial Facebook*. Tugas Besar II IF2211 Strategi Algoritma Semester II Tahun 2020/2021. Diakses pada 25 03, 2021, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Tugas-Besar-2-IF2211-Strategi-Algoritma-2021.pdf

GeeksforGeeks (2020, 04 12). *Breadth First Search or BFS for a Graph*. Diakses pada 26 03, 2021, dari https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/