

Pengaplikasian Algoritma BFS dan DFS dalam Fitur *People You May Know* Jejaring Sosial Facebook

LAPORAN TUGAS BESAR

Diajukan Untuk Memenuhi Tugas Besar 2 IF2211 Strategi Algoritma
Semester II Tahun 2020/2021



You've Got a Friend in Me

Syarifah Aisha | Gde Anantha | Kinantan Arya

Disusun Oleh

Gde Anantha Priharsena	(13519026)
Kinantan Arya Bagaspati	(13519043)
Syarifah Aisha Geubrina Yasmin	(13519089)

**TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG**

2020

BAB I

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.

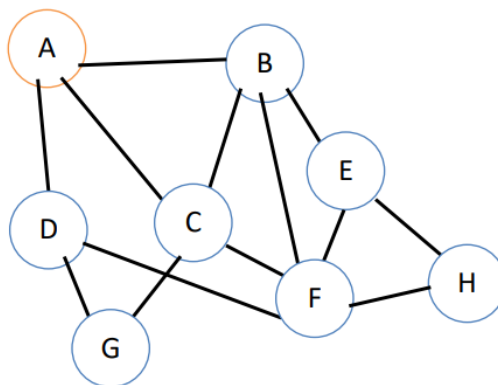
Contoh Input dan Output Program

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. 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

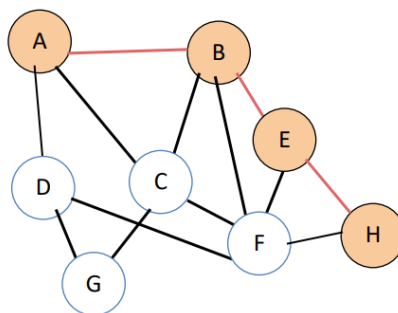
Untuk **fitur explore friends**, misalnya pengguna ingin mengetahui seberapa jauh jarak antara akun A dan H serta bagaimana jalur agar kedua akun bisa terhubung. Berikut output graf dengan penelusuran BFS yang dihasilkan.

Berikut output yang diharapkan untuk penelusuran menggunakan BFS.

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

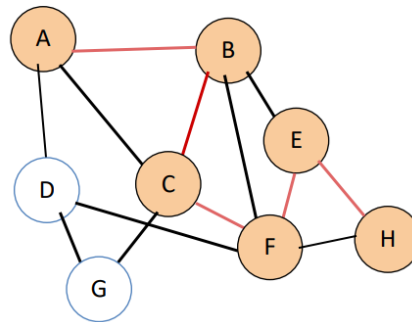
Gambar 4. Hasil output akun Nth degree connection akun A dan H menggunakan BFS

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



Gambar 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

BAB II

LANDASAN TEORI

2.1 Algoritma Traversal Graf

Algoritma traversal graf adalah algoritma untuk mengunjungi simpul dengan cara yang sistematis. Persoalan pada algoritma traversal graf direpresentasikan dalam bentuk graf (diasumsikan graf terhubung). Algoritma traversal graf dibagi menjadi dua yaitu pencarian melebar (breadth first search/BFS) dan pencarian mendalam (depth first search/DFS). Terdapat dua pendekatan representasi graf dalam proses pencarian yaitu graf statis dan graf dinamis. Graf statis adalah graf yang sudah terbentuk sebelum proses pencarian dilakukan, sedangkan graf dinamis adalah graf yang terbentuk saat proses pencarian dilakukan.

2.2 Pencarian Melebar (BFS)

Pada algoritma pencarian melebar (BFS), semua simpul pada level ke- n akan dikunjungi lebih dahulu sebelum mengunjungi level $n+1$. Biasanya, BFS dimulai dari simpul dengan “id” terkecil (biasanya 1- n atau A-Z) atau dari simpul paling kiri ke kanan. Proses ini diulangi hingga ditemukan solusi. Adapun algoritma BFS sebagai berikut:

1. Kunjungi simpul v
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul -simpul yang tadi dikunjungi, demikian seterusnya.

Berikut adalah algoritma BFS yang direpresentasikan dalam pseudocode:

```
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: antrian
procedure BuatAntrian(input/output q: antrian)
{membuat antrian kosong, kepala(q) diisi 0}
procedure MasukAntrian(input/output q: antrian, input v: integer)
{memasukkan v ke dalam antrian q pada posisi belakang}
```

```

procedure HapusAntrian(input/output q: antrian, input v: integer)
{menghapus v dari kepala antrian q}
function AntrianKosong(input q: antrian) → Boolean
{true jika antrian q kosong, false jika sebaliknya}

```

Algoritma

```

BuatAntrian(q)           {buat antrian kosong}
write(v)                 {cetak simpul awal yang dikunjungi}
dikunjungi(v)←true       {simpul v telah dikunjungi, tandai dengan true}
MasukAntrian(q, v)       {masukkan simpul awal kunjungan ke dalam antrian}

{kunjungi semua simpul graf selama antrian belum kosong}
while not AntrianKosong(q) do
    HapusAntrian(q,v)     {simpul v telah dikunjungi, hapus dari antrian}
    for tiap simpul w yang bertetangga dengan simpul v do
        if not dikunjungi(w) then
            write(w)       {cetak simpul yang dikunjungi}
            MasukAntrian(q, w)
            dikunjungi[w] ← true
        endif
    endfor
endwhile
{AntrianKosong(q) }

```

2.3 Pencarian Mendalam (DFS)

Pada algoritma pencarian mendalam (DFS), proses pencarian akan dilakukan pada semua level selanjutnya dari sebuah simpul sebelum dilanjutkan ke simpul-simpul pada level yang sama. Biasanya, BFS dimulai dari simpul dengan “id” terkecil (biasanya 1-n atau A-Z) atau dari simpul paling kiri ke kanan. Proses ini diulangi hingga ditemukan solusi. Adapun algoritma DFS sebagai berikut:

1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v.
3. Ulangi DFS mulai dari simpul w.
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.

5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Berikut adalah algoritma DFS yang direpresentasikan dalam pseudocode:

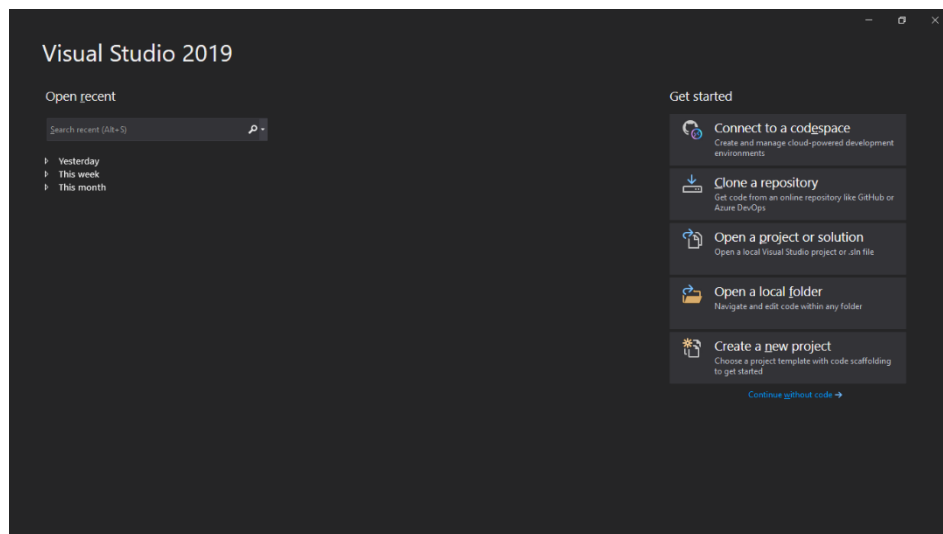
```
procedure DFS (input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS
  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke layar}
Deklarasi
  w: integer
Algoritma
  write(v)
  dikunjungi(v) ← true
  for w ← 1 to n do
    if A[v,w]=1 then {simpul v dan simpul w bertetangga}
      if not dikunjungi[w] then
        DFS(w)
      endif
    endif
  endfor
```

2.4 C# Desktop Application Development

C# merupakan salah satu bahasa pemrograman berorientasi objek yang biasanya digunakan untuk pemrograman *server-side website*, membangun aplikasi *desktop* ataupun *mobile*, pemrograman dan sebagainya. Sama seperti pemrograman berorientasi objek lainnya, C# juga menerapkan konsep-konsep objek seperti *inheritance*, *class*, *polymorphism*, *encapsulation*, dan lain-lain. Seperti bahasa-bahasa pemrograman pada umumnya, C# juga membutuhkan IDE dan *Framework*. Adapun IDE dan *Framework* yang digunakan adalah Visual Studio dengan *Framework .NET*.

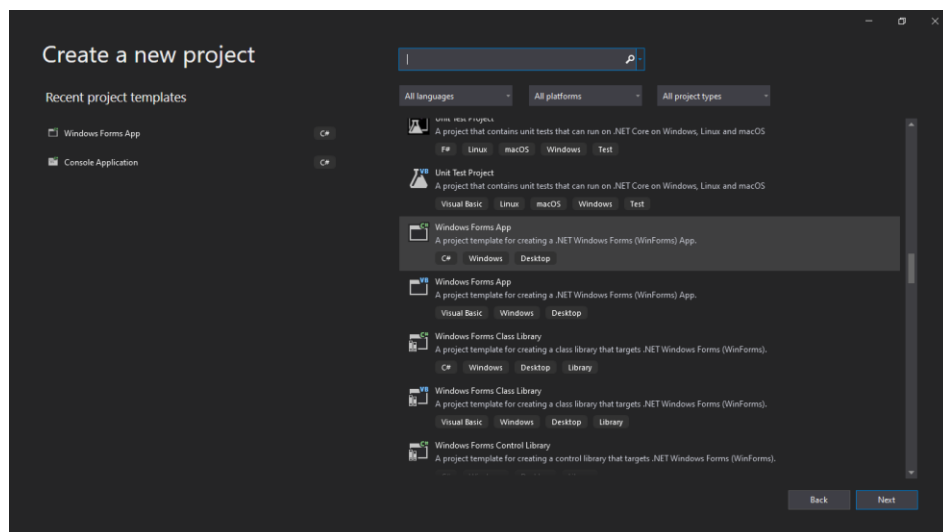
Untuk pengembangan aplikasi *desktop*, terdapat berbagai jenis aplikasi yang disediakan oleh Visual Studio antara lain Console Application, Windows Forms Application, WPF Application, dan sebagainya. Setiap jenis aplikasi memiliki kekurangan dan kelebihan masing-masing yang penggunaannya disesuaikan dengan kebutuhan pengguna. Pada Tugas Besar kali ini digunakan Windows Forms Application untuk merealisasikan GUI (*Graphical User Interface*) dari program yang dibuat. Adapun langkah-langkah untuk membuat Windows Forms Application dengan Visual Studio sebagai berikut:

1. Unduh Visual Studio dari laman berikut <https://visualstudio.microsoft.com/downloads/>
2. *Install* dan jalankan Visual Studio yang sudah diunduh sebelumnya.
3. Pada jendela yang terbuka, pilih “Create a new project”.



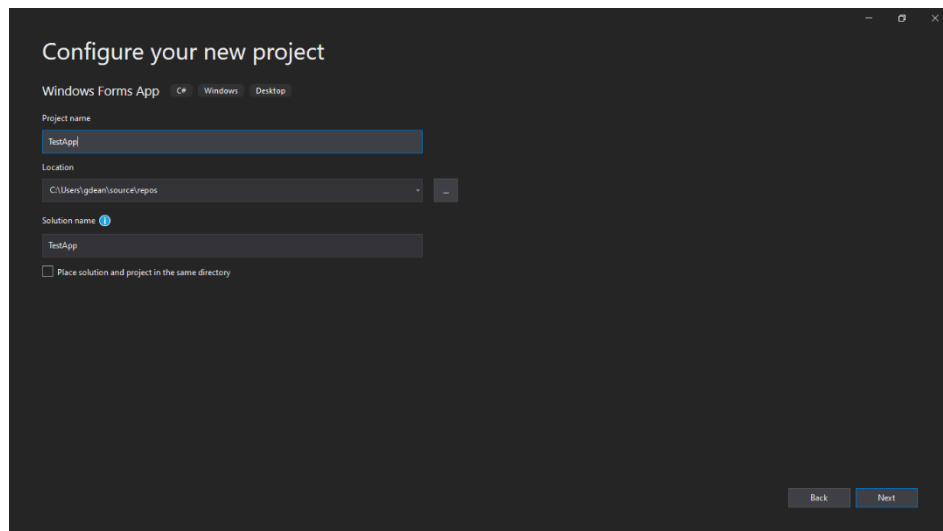
Gambar 8. Jendela Awal Visual Studio 19

4. Pilih Windows Forms App sebagai template dari aplikasi yang ingin dibuat.



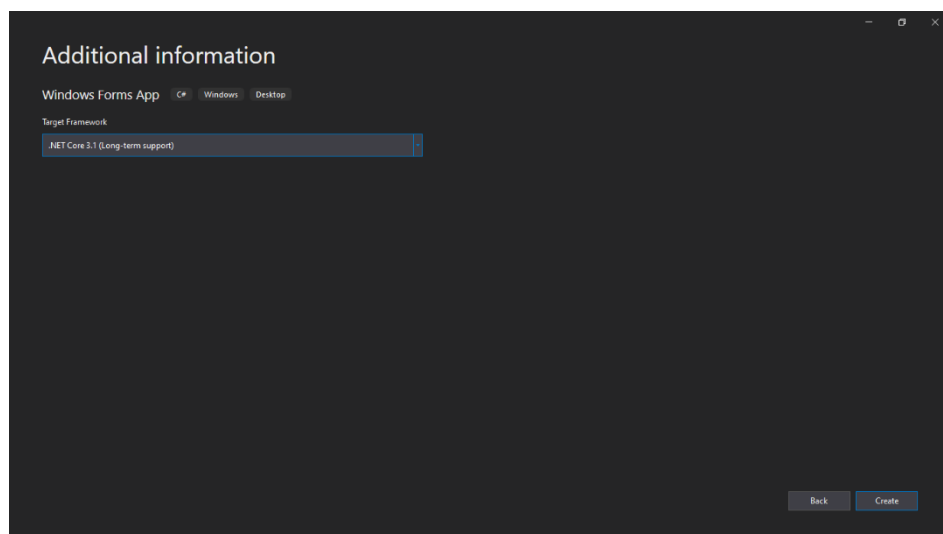
Gambar 9. Jendela “Create a new project”

5. Pada jendela “Configure your new project”, tuliskan nama dari aplikasi yang Anda buat dan tentukan lokasi penyimpanan dari aplikasi yang Anda buat.



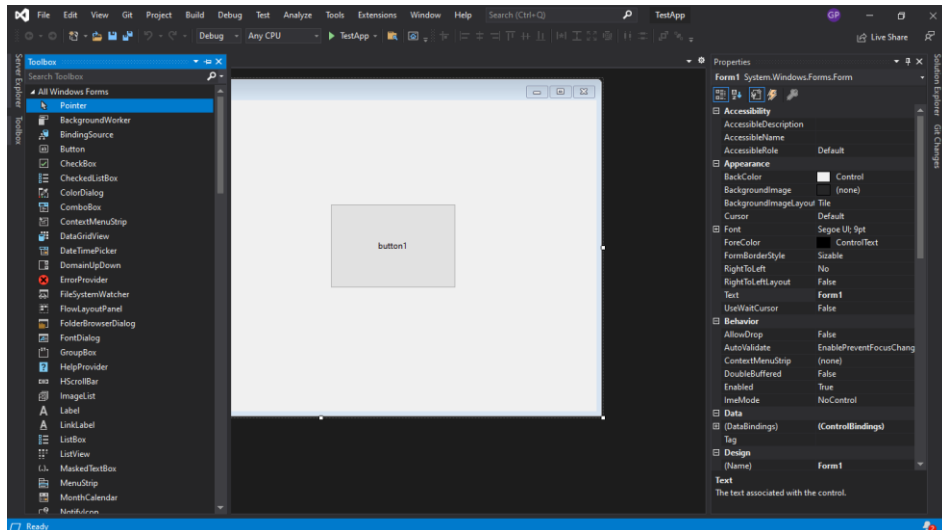
Gambar 10. Jendela “Configure your new project”

6. Pilih jenis Framework yang ingin digunakan. Disarankan untuk memilih Framework terbaru supaya bisa menggunakan banyak fitur yang disediakan oleh C# dan Visual Studio. Kemudian klik “Create”.



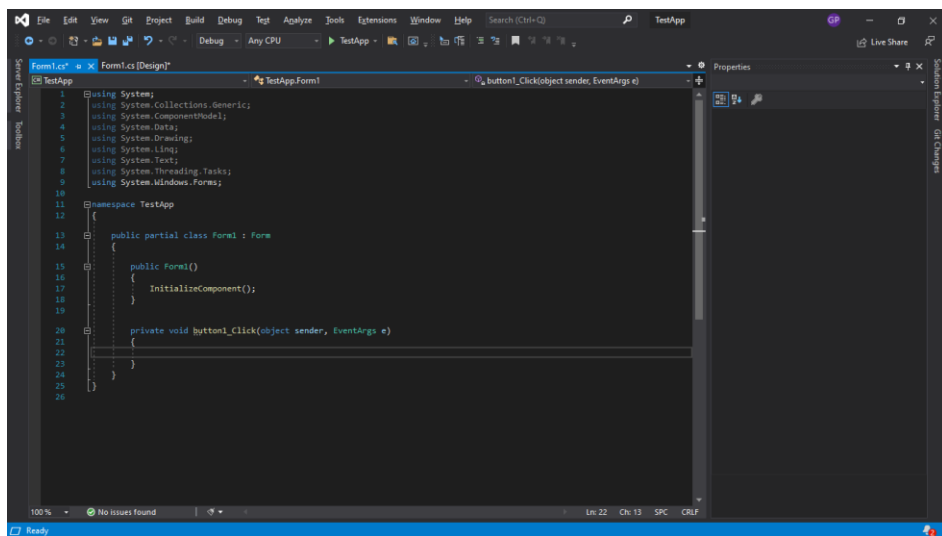
Gambar 11. Jendela “Additional information”

7. Akan ditampilkan form *default* dari Windows Form. Untuk menambahkan komponen pada Form, klik bar Toolbox di panel sebelah kiri. Tarik komponen yang ingin digunakan ke Form. Anda dapat berkreasi se bebas mungkin dengan komponen yang disediakan.



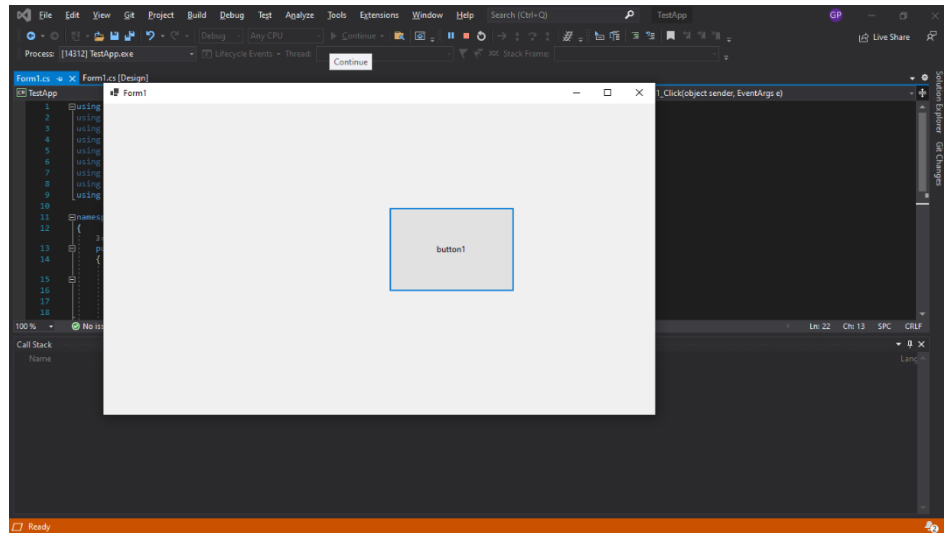
Gambar 12. Jendela utama proyek yang sedang dibuka

8. Untuk mengikat (*binding*) fungsi atau metode ke sebuah komponen, Anda dapat klik dua kali pada komponen yang ingin Anda gunakan. Lalu akan secara otomatis diarahkan ke sebuah *method* pada Form1.cs



Gambar 13. File cs yang dibuka secara otomatis saat melakukan *double click* terhadap komponen Forms.

9. Untuk menjalankan aplikasi, klik pada tombol “play” berwarna hijau diatas. Visual Studio secara otomatis akan melakukan *build* terhadap aplikasi yang Anda buat. Jika *build* berhasil, aplikasi (.exe) Anda akan dijalankan secara otomatis.



Gambar 13. Contoh tampilan aplikasi yang berhasil di-build

BAB III

Analisis Pemecahan Masalah

3.1 Langkah-Langkah Pemecahan Masalah

Dari permasalahan yang disebutkan pada Bab 1 Deskripsi Tugas, telah dilakukan langkah-langkah pemecahan masalah tersebut antara lain:

1. Memahami permasalahan yang dijelaskan pada Bab 1 Deskripsi Tugas.
2. Memahami konsep algoritma BFS dan DFS dalam traversal graf.
3. Memetakan permasalahan menjadi elemen-elemen BFS dan DFS.
4. Menyelesaikan beberapa contoh kasus yang merepresentasikan permasalahan tersebut dengan algoritma BFS dan DFS.
5. Mempelajari bahasa pemrograman C# dan meng-*install* Visual Studio sebagai IDE yang digunakan untuk melakukan pemrograman dengan bahasa C#.
6. Mengimplementasikan algoritma BFS dan DFS untuk setiap permasalahan dalam bahasa C# dalam bentuk Console Application.
7. Membuat GUI dalam Windows Forms Application yang dapat menerima masukan pengguna berupa file graf, pilihan fitur (*Friends Recommendation* / *Explore Friends*), pilihan algoritma (BFS/DFS), dan pilihan akun yang ditelusuri.
8. Membuat visualisasi graf pada GUI berdasarkan masukan file graf dengan *library* MSAGL.
9. Memindahkan algoritma yang dibuat pada Console Application ke GUI.
10. Melakukan *testing* terhadap GUI yang telah dibuat.

3.2 Mapping Elemen Algoritma

1. Permasalahan *Friends Recommendation*

Misalkan dilakukan pencarian *Friends Recommendation* untuk akun X, dengan akun adalah simpul dan relasi adalah sisi.

- a) Problem State: Mencari akun pada graf yang memiliki minimal 1 *mutual friends* dengan akun X
- b) Initial State: akun X
- c) Solution State: Semua akun yang memiliki minimal 1 *mutual friends* dengan akun X
- d) State Space: Himpunan Semua akun yang ada pada graf
- e) State Space Tree: Pohon dari semua akun pada State Space

- f) Solution Space: Himpunan semua akun yang tidak terhubung langsung dengan akun X.
- g) Solution: Himpunan semua akun yang memiliki minimal 1 *mutual friends* dengan akun X.

2. Permasalahan Explore Friends

Misalkan dilakukan pencarian Explore Friends dari akun X menuju akun Y, dengan akun adalah simpul dan relasi adalah sisi.

- a) Problem State: Mencari jalur/relasi yang menghubungkan akun X dan akun Y.
- b) Initial State: akun X dan akun Y
- c) Solution State: himpunan kosong atau himpunan akun-akun yang dilalui dari akun X untuk menuju Y
- d) State Space: Himpunan Semua akun yang ada pada graf
- e) State Space Tree: Pohon dari semua akun pada State Space
- f) Solution Space: Himpunan semua kombinasi akun yang saling terhubung yang dimulai dengan akun X.
- g) Solution : Himpunan semua kombinasi akun yang saling terhubung yang dimulai dengan akun X dan berakhir di Y

3.3 Contoh Ilustrasi

Contoh berkas file eksternal:

```

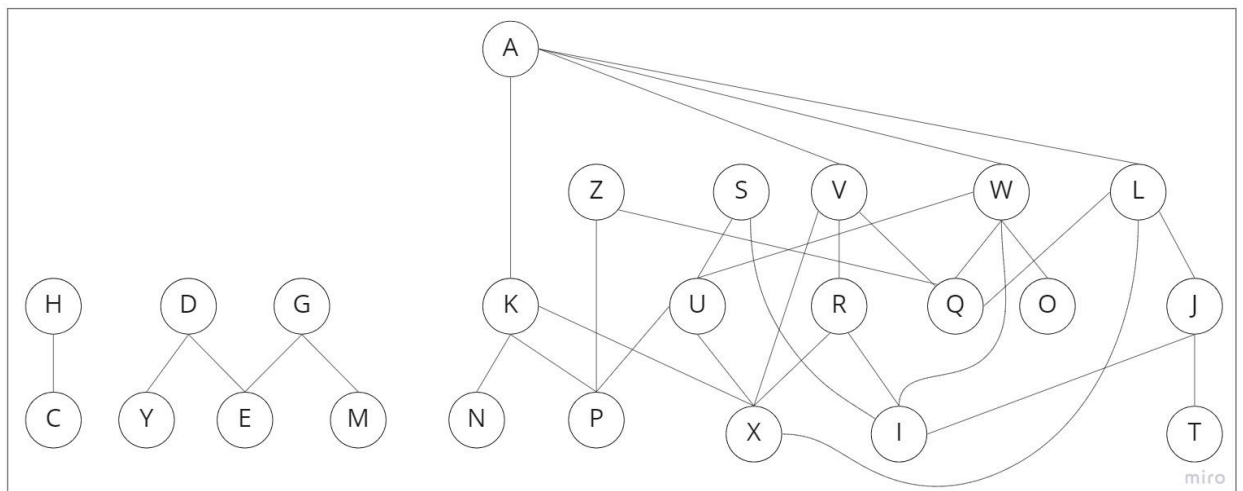
A L
A W
A K
A V
L J
L Q
L X
W O
W U
W Q
W I
H C
S U
S I
D Y
D E
Z Q
Z P
K N
K X
K P
V R
V Q

```

V X
V I
G M
G E
R X
R I
J I
J T
U X
U P

Gambar 14. Contoh input berkas file eksternal

Visualisasi graf pertemanan yang dihasilkan dari file eksternal:



Gambar 15. 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: Q

3 mutual friends:

L
V
W

Nama akun: X

3 mutual friends:

K
L
V

Nama akun: I

2 mutual friends:

V
W

Nama akun: J

1 mutual friends:

L

Nama akun: N
1 mutual friends:
K

Nama akun: O
1 mutual friends:
W

Nama akun: P
1 mutual friends:
K

Nama akun: R
1 mutual friends:
V

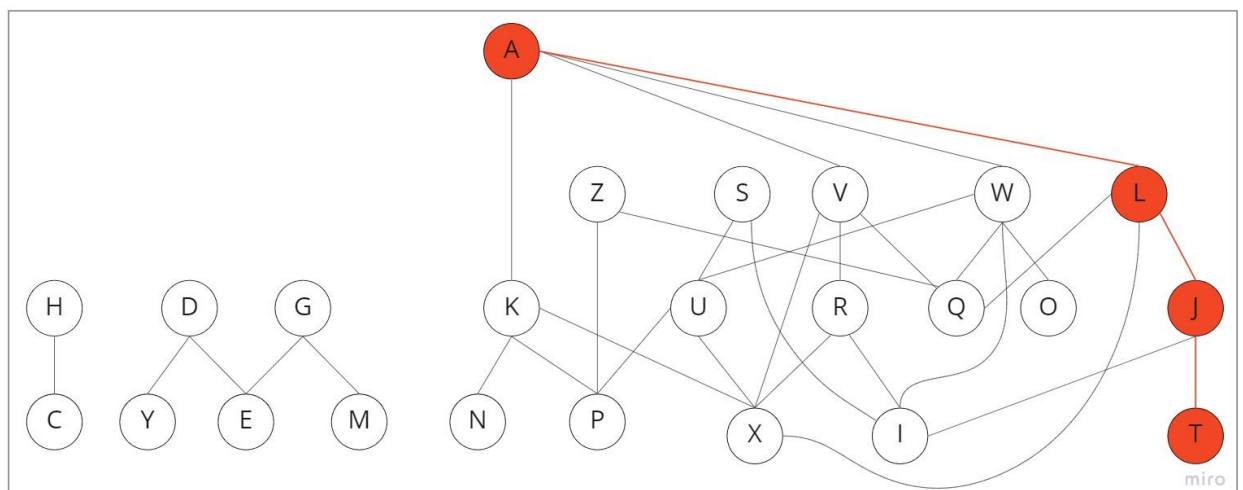
Nama akun: U
1 mutual friends:
W

Gambar 16. Hasil output yang diharapkan untuk rekomendasi akun A

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

Nama akun: A dan T
2nd-degree-connection
 $A \rightarrow L \rightarrow J \rightarrow T$

Gambar 17. Hasil output akun Nth degree connection akun A dan H menggunakan BFS

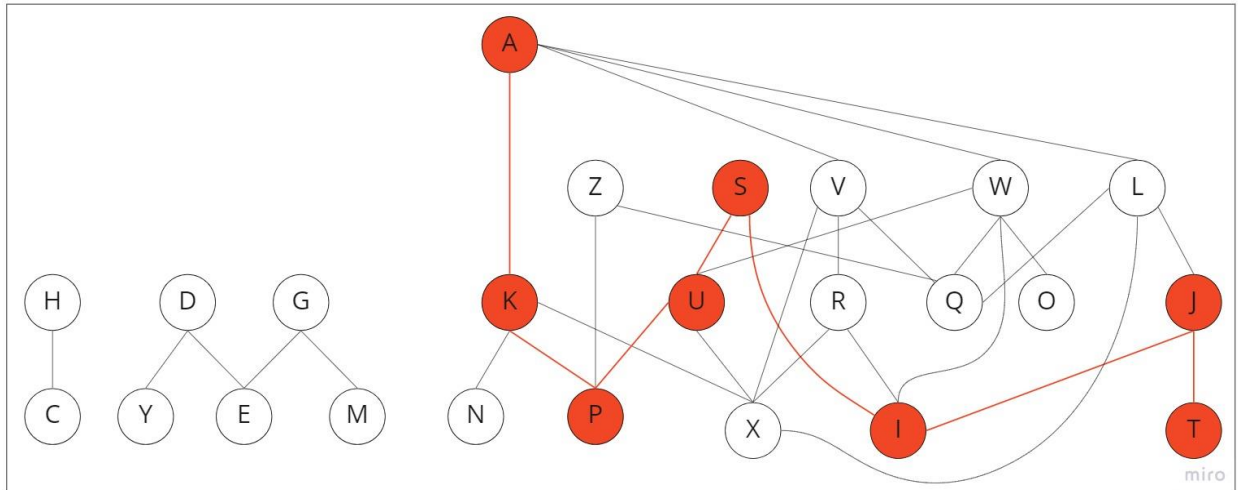


Gambar 18. Hasil visualisasi jalur koneksi menggunakan BFS

Sementara untuk *output* jalur dengan penelusuran DFS yang dihasilkan beserta graf hasil penelusurannya.

Nama akun: A dan T
6th-degree-connection
 $A \rightarrow K \rightarrow P \rightarrow U \rightarrow S \rightarrow I \rightarrow J \rightarrow T$

Gambar 19. Hasil output akun Nth degree connection akun A dan H menggunakan BFS



Gambar 20. 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 jalur koneksi yang dibentuk dari A ke H adalah

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

Gambar 21. Hasil output tidak ada jalur koneksi antara A dan H

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Program

Implementasi program utama kami terbagi menjadi 5 fitur, yaitu fitur *Explore Friend* dengan metode pencarian DFS dan BFS, fitur *Friend Recommendations* dengan metode pencarian DFS dan BFS, serta *main program*.

1. Fitur *Explore Friend* dengan metode BFS

```
Function BFS(input from: int, to: int) → int[]
{Pencarian jalur dari from sampai to pada graf
  Masukan: from adalah akun/node asal dan to adalah akun/node tujuan
  Keluaran : array of int berisi kumpulan node yang merepresentasikan jalur
yang dilalui dari node asal sampai tujuan dengan metode BFS}

Kamus Lokal
head, degree, visitedNode, i: int
backtrack, depth, path: int[]
queue: Queue<int>

procedure makeQueue(input/output q: Queue<int>)
{membuat Queue<int> kosong pada q}

procedure makeArray(input/output arr: int[], input size)
{membuat array of integer kosong sebesar size bernama arr}

function getFriendNodes(input i: int) → List<int>
{mengembalikan list of integer yang berisi semua akun yang bersisian dengan
akun i}

Algoritma
{backtrack merupakan array yang menyimpan ID node sebelum mencapai node ini,
diinisialisasi -1 semua pada awal program, menandakan node belum dikunjungi}
{depth merupakan kedalaman suatu node}
i traversal [0..(nodes-1)]
  backtrack[i] ← -1
makeQueue(q)
queue.enqueue(from)
depth[from] ← 0

while backtrack[to]<0 and !queue.isEmpty() do
  queue.pop(head)
  degree ← getFriendNodes(head)
  i traversal [0..(degree-1)]
    visitedNode ← getFriendNodes(head)[i]
    if backtrack[visitedNode]<0 then {belum pernah dikunjungi}
      queue.enqueue(visitedNode)
      depth[visitedNode] ← depth[head]+1
      backtrack[visitedNode] ← head
```

```

{selanjutnya path pengunjungan akan direkonstruksi dari belakang memanfaatkan array backtrack}
if backtrack[to] != -1 then
    makeArray(path, (depth[to]+1))
    visitedNode ← to
    i traversal [depth[to]..0]
        path[i] ← visitedNode
        visitedNode ← backtrack[visitedNode]
    → path
else {to tidak dikunjungi, maka tidak ada jalur ke dari to ke from}
    makeArray(path, 0)
    → path

```

2. Fitur *Explore Friend* dengan metode DFS

```

function ExploreFriendDFS(input from: string, to: string) → string[]
{Mencari jalur yang dilewati dari node from sampai node to
Masukan : from: node/akun asal, to: node/akun tujuan}

Kamus
visitedNodes : bool[] {terinisialisasi false dari awal}
i, nodes: int {jumlah nodes graf (terdefinisi)}
AdjacencyList: List<List<int>> {list tetangga tiap nodes (terdefinisi)}
Path : int[]
Path_string : string[]
stack: Stack<int>

procedure Push(input stack: Stack<int>, var: int)
{menambahkan elemen variabel ke atas stack of integer}

function Pop(input stack: Stack<int>) → int
{Mengembalikan elemen paling atas stack}

function getFriendNodes(input q: int) → list of integer
{mengembalikan list of integer yang berisi semua akun yang bersisian
dengan akun q}

function getIdxAccount(input q: string) → integer
{mengembalikan index dari akun q}

function getAccuntByIdx(input q: integer) → string
{mengembalikan akung dengan indeks q}

function Count(input l: list of integer) → integer
{mengembalikan banyaknya elemen dari l}

function stackCount(input l: stack of integer) → integer

```

```

{mengembalikan banyaknya elemen dari l}

function StackToArray(input stack: Stack<int>) → int[]
{mengembalikan array of int yang di copy dari stack, terurut dari yang
terbawah}

procedure DFS(input current: int, to: int, visitedNodes: bool[], input/output
stack: Stack<int>)
{Pencarian node to dengan metode rekursif dan menyimpan node yang dilewati
dengan stack}
If (not(visitedNodes[current])) then
    VisitedNodes ← True
    Push(stack, current)
    count_adj ← count(getFriendNodes(current))
    i traversal [0..count_adj]
        If (not(visitedNodes[to])) then
            Temp ← getFriendNodes(current)[i]
            DFS(Temp, to, visitedNodes, stack)
    If (not(visitedNodes[to])) then
        Pop(stack, Temp)

Algoritma
DFS(getIdxAccount(from), getIdxAccount(to), visitedNodes, stack)
if (stackCount(stack) > 0) then
    Path = StackToArray(stack)
    i traversal [0..Count(path)]
        Path_string[i] = getAccuntByIdx(path[i])
→ Path_string

```

3. Fitur *Friend Recommendation* dengan metode BFS

```

function FriendRecommendationBFS (input v: string) → string
{Traversal graf dengan algoritma pencarian BFS.
Masukan: v adalah akun yang ingin dicari Friend Recommendation-nya
Keluaran: Jika ditemukan minimal 1 akun yang memiliki minimal 1 mutual
friends, maka tampilkan nama akun beserta mutual friends-nya. Jika tidak
ditemukan, tampilkan pesan "There is no friends recommendation for v"}

Kamus Lokal
currentNode, countFriends: integer
prinResult: string
recommended: list of string
accessed, queue, tempFriend, masterFriend: list of integer
result, sortedResult: Dictionary<string, List<string>>

procedure makeListOfString(input/output v: list of string)

```

{membuat list of string kosong pada v}

procedure makeListOfInteger(input/output v: list of string)

{membuat list of integer kosong pada v}

procedure makeDictionary(input/output v: Dictionary<string,List<string>>)

{membuat Dictionary<string,List<string>> kosong pada v}

function getFriendNodes(input q: int) → list of integer

{mengembalikan list of integer yang berisi semua akun yang bersisian dengan akun q}

function getIdxAccount(input q: string) → integer

{mengembalikan index dari akun q}

function getAccuntByIdx(input q: integer) → string

{mengembalikan akung dengan indeks q}

function Contains(input l: list of integer, input n: integer) → Boolean

{mengembalikan True jika n terdapat di l, mengembalikan False jika sebaliknya}

function isEmpty(input l: list of integer) → Boolean

{mengembalikan True jika l kosong, mengembalikan False jika sebaliknya}

function Count(input l: list of integer) → integer

{mengembalikan banyaknya elemen dari l}

procedure AddInt(input/output l: list of integer, input n: integer)

{memasukkan n ke dalam l pada posisi belakang}

procedure AddString(input/output l: list of string, input n: string)

{memasukkan n ke dalam l pada posisi belakang}

procedure AddPair(input/output v: Dictionary<string,List<string>>, k:

string, l: list of string)

{memasukkan elemen pair dengan key = k dan value = l ke dictionary v}

procedure RemoveInt(input/output l: list of string, input n: string)

{menghapus n dari l}

```

procedure Sort(input/output v: Dictionary<string,List<string>>)
{mengembalikan dictionary v yang terurut berdasarkan banyaknya elemen pada
value}

function getResult(input v: Dictionary<string,List<string>>,
input a:string)→ string
{Jika v tidak kosong, maka konversi dictionary v menjadi string sesuai
keluaran friends recommendation. Jika kosong kembalikan "There is no friends
recommendation for a"}

```

Algoritma

```

makeDictionary(result)
makeDictionary(sortedResult)
makeListOfInteger(accessed)
makeListOfInteger(tempFriend)
makeListOfInteger(masterFriend)
makeListOfInteger(queue)

masterFriend ← getFriendNodes(getIdxAccount(v))
queue ← masterFriend

while (not(isEmpty(queue))) do
    currentNode ← queue[0]
    tempFriend ← getFriendNodes(currentNode)
    if(Contains(masterFriend, currentNode)) then
        i traversal [0..Count(tempFriend)-1]
            if (not(Contains(accessed, tempFriend[i])) and not(Contains(queue,
tempFriend[i])) and (tempFriend[i]≠getIdxAccount(v))) then
                AddInt(queue, tempFriend[i])
    else
        makeListOfString(recommended)
        i traversal [0..Count(tempFriend)-1]
            if(Contains(masterFriend, tempFriend[i])) then
                AddString(recommended, getAccountByIdx(tempFriend[i]))
            if (not(Contains(accessed, tempFriend[i])) and not(Contains(queue,
tempFriend[i])) and (tempFriend[i]≠getIdxAccount(v))) then
                AddInt(queue, tempFriend[i])
        AddPair(result, getAccountByIdx(currentNode), recommended)

```

```

AddInt(accessed, currentNode)
RemoveInt(queue, currentNode)
{endwhile}

sortedResult ← sort(result)
printResult ← getResult(sortedResult)
→ result

```

4. Fitur *Friend Recommendation* dengan metode DFS

```

function FriendRecommendationDFS (input v: string) → string
{Pencarian rekomendasi teman pada graf dengan algoritma pencarian DFS.
Masukan: v adalah akun yang ingin dicari Friend Recommendation-nya
Keluaran: Jika ditemukan minimal 1 akun yang memiliki minimal 1 mutual
friends, maka tampilkan nama akun beserta mutual friends-nya. Jika tidak
ditemukan, tampilkan pesan "There is no friends recommendation for v"}

Kamus Lokal
node, i, j: int
toReturn_string: string
parents: List<List<int>>
toSort: List<Tuple<int, int>>
toReturn: List<Tuple<int, List<int>>>

function getFriendNodes(input i: int) → List<int>
{mengembalikan list of integer yang berisi semua akun yang bersisian dengan
akun i}

procedure RecursiveFriendRecommendation(input from: int, input depth: int,
input/output parents List<List<int>>)
{menambahkan node mana saja yang dapat mengakses tiap node dalam kedalaman
depth dari from}
    degree ← adjacencyList[from].Count
    if degree > 1 then
        i traversal [0..(degree-1)]
            RecursiveFriendRecommendation(adjacencyList[from][i], depth - 1,
parents);
    Else {telah mencahai leaf dari pohon rekursi}
        i traversal [0..(degree-1)]
            parents[adjacencyList[from][i]].Add(from);

Algoritma
{inisialisasi list berisi (nodes-1) buah list<int> kosong berisi parents,
kemudian diisi dengan metode rekursi yang telah dideskripsikan (depth=2)}
RecursiveFriendRecommendation(nodeIdx[from], 2, parents)

{Inisialisasi lagi list of tuple<int, int> toSort yang berturut turut berisi
banyak mutual friend (didapat dari banyak parent) dan -nodeIdx. Diisi
demikian agar saat diurutkan menurut aturan tuple, akan terurut menaik
berdasar banyak parent, kemudian menurun berdasar nodeIdx, agar setelah
dibuat string menjadi urut abjad}
i traversal [0..(nodes-1)]

```

```

    If i != nodeIdx[from] then
        toSort.Add({parents[i].Count, -i})
toSort.Sort()

{Kemudian inialisasi List of Tuple<int, List<int>> toReturn yang berturut-
turut merupakan nodeIdx, dan daftar mutual friend. ToReturn sudah terurut
sesuai spesifikasi karena diambil dari toSort yang dibalik}
toReturn_string ← ""
i traversal [(nodes-2)..0]
    node ← -toSort[i].Item2
    if not adjacencyList[from].Contains(nodes) then
        toReturn_string ← toReturn_string + "Mutual friends with " +
nodeNames[node] + ":\n"
        i traversal [0..(parents[i].Count - 1)]
            toReturn_string ← toReturn_string + nodeNames[node] + "\n"
if toReturn_string = "" then
    toReturn_string ← "There is no friends recommendation for " +
nodeNames[from]

```

5. Main Program

```

procedure Main(g: Graph)
{program akan menjalankan aplikasi sampai pengguna meng-klik tombol X untuk
keluar dari aplikasi.}

```

Kamus Lokal

features, algorithm, acc1, acc2: string

Algoritma

```

while (1) do
    input(features)
    input(algorithm)
    input(acc1)
    input(acc2)

    if(features = "Friend Recommendation") then
        if(acc1 ≠ "")then
            if(algorithm = "Depth First Search (DFS)")then
                output(SortedFriendRecommendation(g, acc1))
            else
                output(FriendRecommendationBFS(g, acc1))
        else
            if(acc1 ≠ "" and acc2 ≠ "")then
                if(algorithm = "Depth First Search (DFS)")then
                    output(ExploreFriendDFS(g, acc1, acc2))

```

```

        else
            output (ExploreFriendsBFS(g, acc1, acc2))
    {endwhile}

```

4.2. Struktur Data

Class Graph

```

- nodes: int
- adjacencyList: List<List<int>>
- nodeNames: List<string>
- nodeIdx: Dictionary<string, int>
- edges: List<Tuple<string, string>>

+ Graph(string)
+ getFriendNodes(int): List<int>
+ getNodeNames() : List<string>
+ getEdges() : List<Tuple<string, string>>
+ BFS(int, int) : List<int>
+ DFS(int, int, ref List<boolean>, Stack<int>)
+ ExploreFriendBFS(string, string) : List<string>
+ ExploreFriendDFS(string, string) : List<string>
+ FriendRecommendationDFS(int, int) : List<string>
+ SortedFriendRecommendation(string, boolean) : string
+ FriendRecommendationBFS(string): string

```

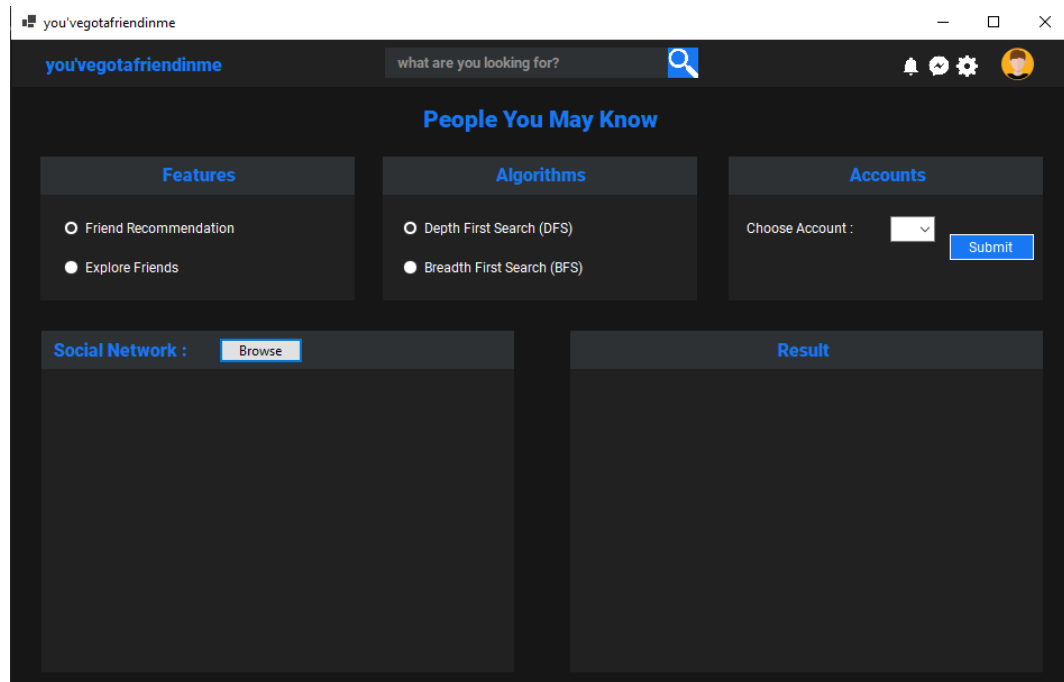
4.3. Tata Cara Penggunaan Program

1. Jalankan Aplikasi Tubes-2-Stima-youvegotafrindinme.exe yang ada di folder bin.

Name	Date modified	Type	Size
AutomaticGraphLayout.dll	12/08/2020 14:07	Application exten...	1.565 KB
AutomaticGraphLayout.Drawing.dll	16/11/2020 0:51	Application exten...	148 KB
Microsoft.Msagl.GraphViewerGdi.dll	16/11/2020 1:28	Application exten...	153 KB
Microsoft.Msagl.WpfGraphControl.dll	16/11/2020 0:52	Application exten...	66 KB
Tubes-2-Stima-youvegotafrindinme.deps	24/03/2021 23:32	JSON File	4 KB
Tubes-2-Stima-youvegotafrindinme.dll	24/03/2021 23:32	Application exten...	137 KB
Tubes-2-Stima-youvegotafrindinme	24/03/2021 23:32	Application	171 KB
Tubes-2-Stima-youvegotafrindinme.pdb	24/03/2021 23:32	Program Debug D...	22 KB
Tubes-2-Stima-youvegotafrindinme.run...	24/03/2021 23:32	JSON File	1 KB
Tubes-2-Stima-youvegotafrindinme.run...	24/03/2021 23:32	JSON File	1 KB

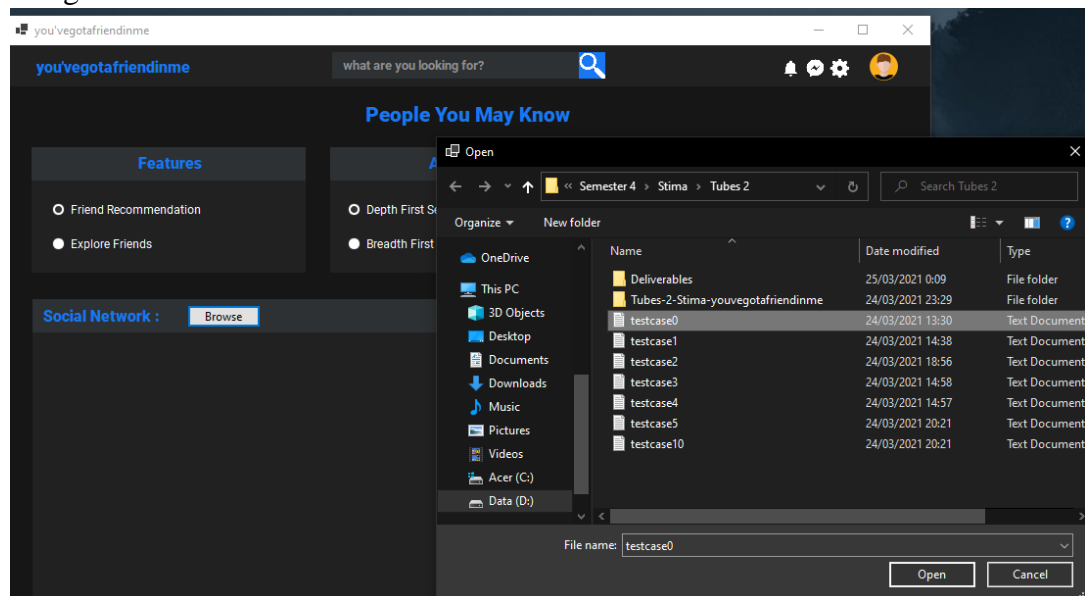
Gambar 22. Isi dari folder bin

2. Akan ditampilkan menu utama dari aplikasi you'vegotafriendinme.



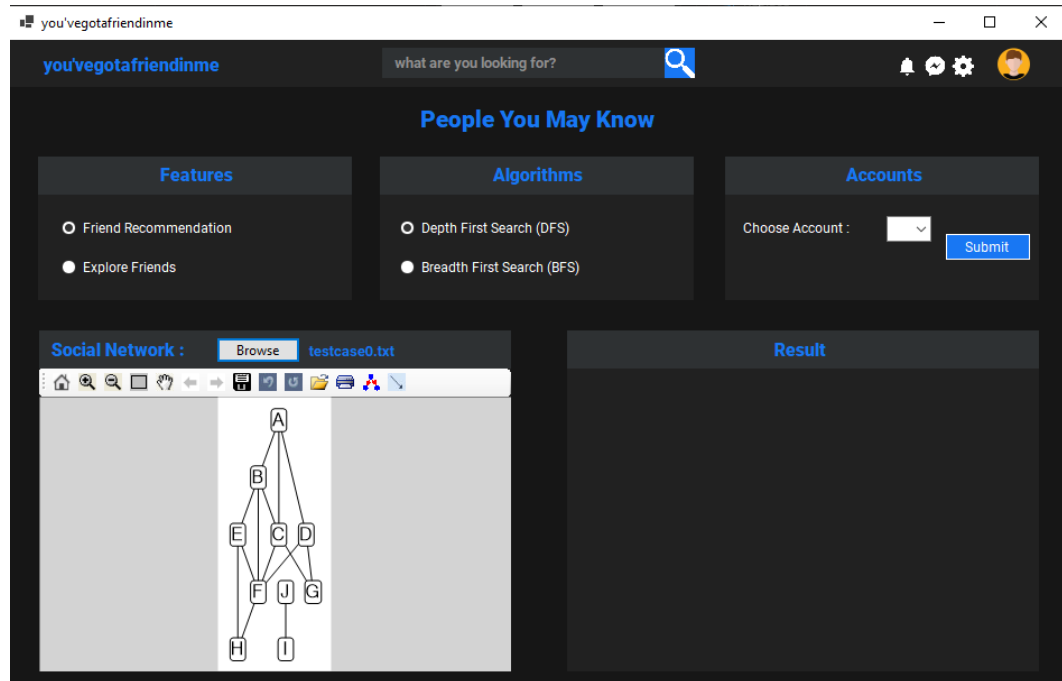
Gambar 23. Tampilan Menu Utama Aplikasi you'vegotafriendinme

3. Klik tombol “Browse” untuk memilih file graf yang ingin digunakan. Pastikan format isi file graf sesuai.



Gambar 24. Tampilan Desktop saat klik tombol “Browse”

4. Jika format file sesuai, akan ditampilkan visualisasi graf dari file graf yang dimasukkan kedalam aplikasi.



Gambar 25. Tampilan Menu Utama setelah memilih file graf dengan format yang sesuai

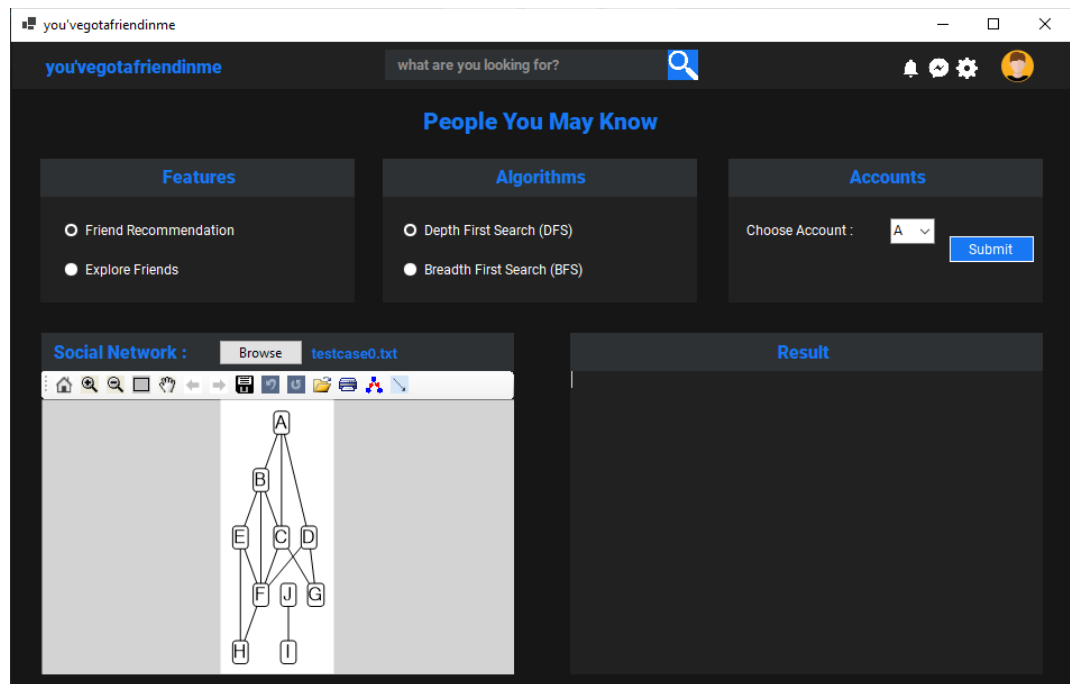
```

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

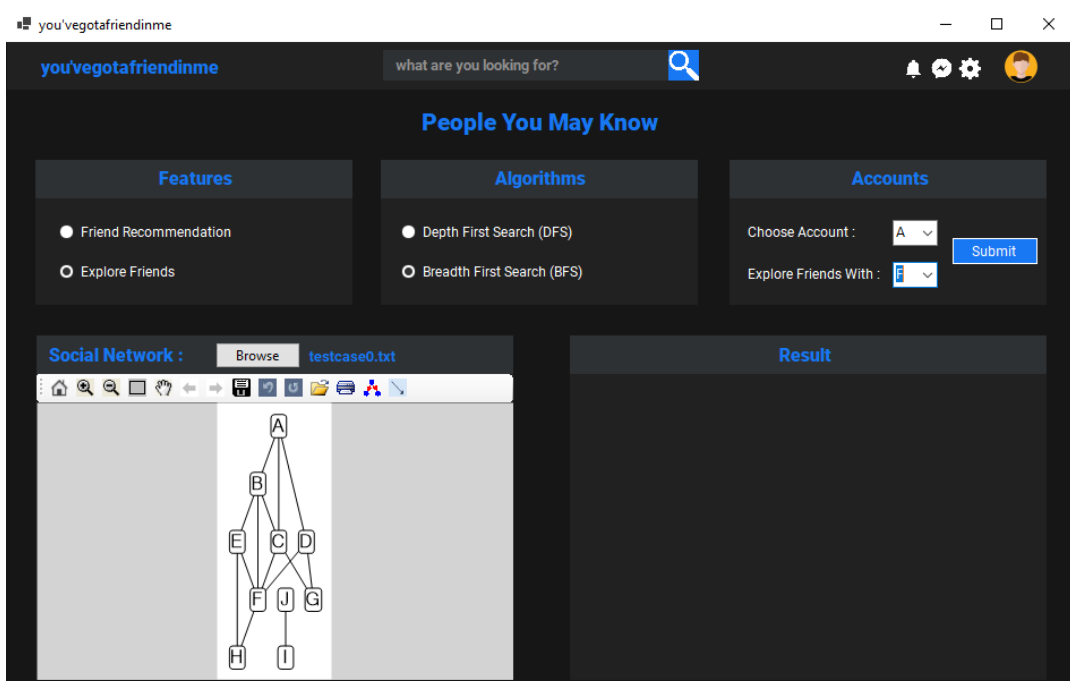
```

Gambar 26. Contoh format file graf yang sesuai

5. Pilih fitur dan algoritma yang ingin Anda gunakan. Kemudian pilih akun-akun yang dibutuhkan untuk menggunakan fitur tersebut. Fitur Friend Recommendation hanya membutuhkan 1 akun yang ada pada graf, sedangkan fitur Explore Friends membutuhkan dua akun yang ada pada graf.

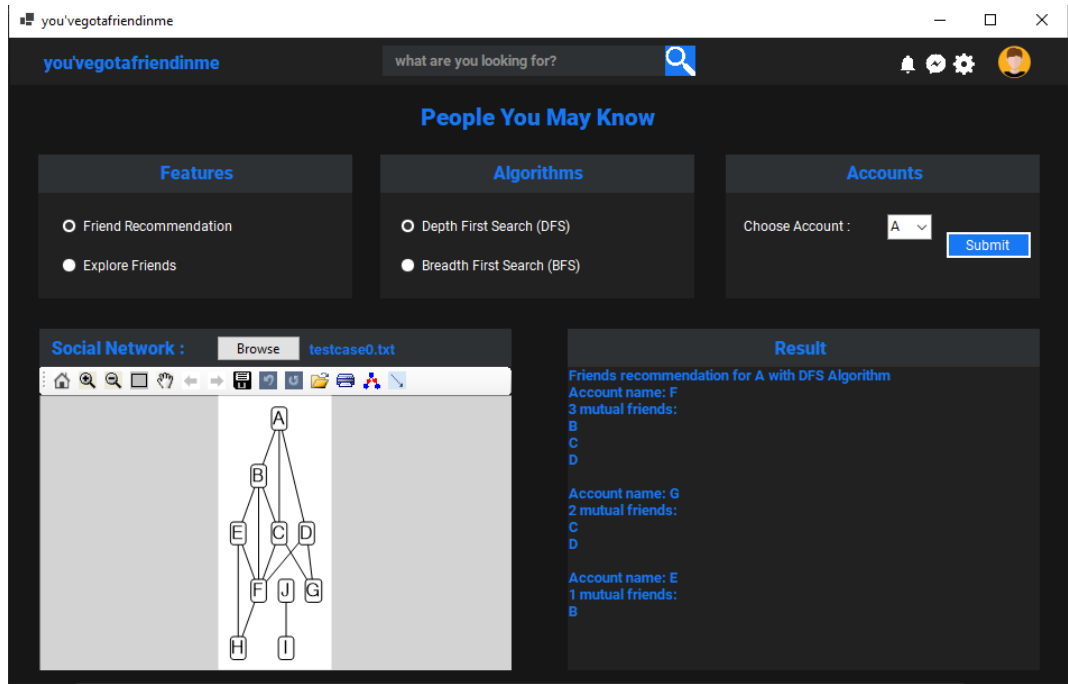


Gambar 27. Contoh pemilihan fitur Friend Recommendation, algoritma, dan akun yang benar.

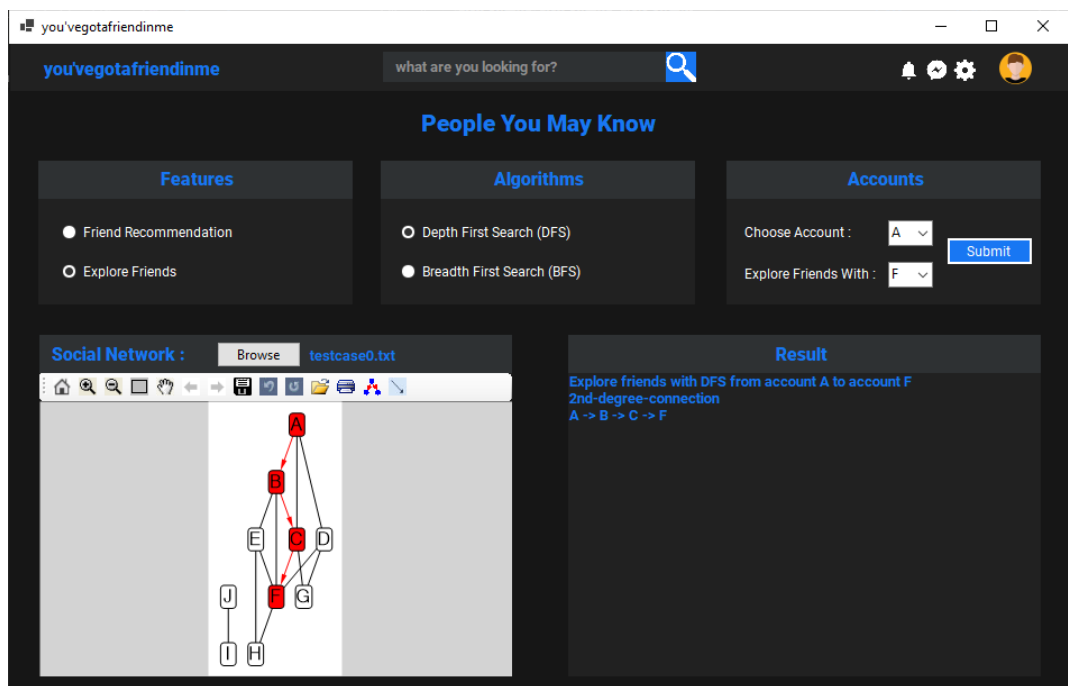


Gambar 28. Contoh pemilihan fitur Explore Friends, algoritma, dan akun yang benar

6. Untuk menampilkan hasilnya, klik tombol “Submit”. Hasil akan ditampilkan di kolom “Result”.



Gambar 29. Contoh hasil dari fitur Friend Recommendation berdasarkan Gambar 27



Gambar 30. Contoh hasil dari fitur Explore Friends berdasarkan Gambar 28

7. Jika Anda sudah selesai menggunakan aplikasi, Anda dapat keluar dari aplikasi dengan klik tomo “X” di pojok kanan atas aplikasi.

4.4. Hasil Pengujian

File yang digunakan untuk pengujian, yaitu:

1. test1.txt

```
A L
A W
A K
A V
L J
L Q
L X
W O
W U
W Q
W I
H C
S U
S I
D Y
D E
Z Q
Z P
K N
K X
K P
V R
V Q
V X
V I
G M
G E
R X
R I
```

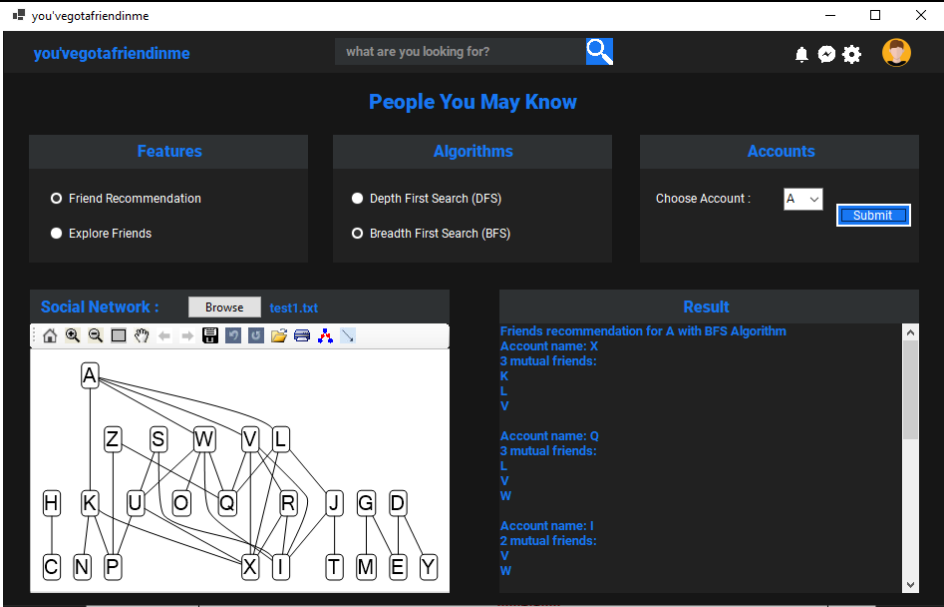
J I
J T
U X
U P

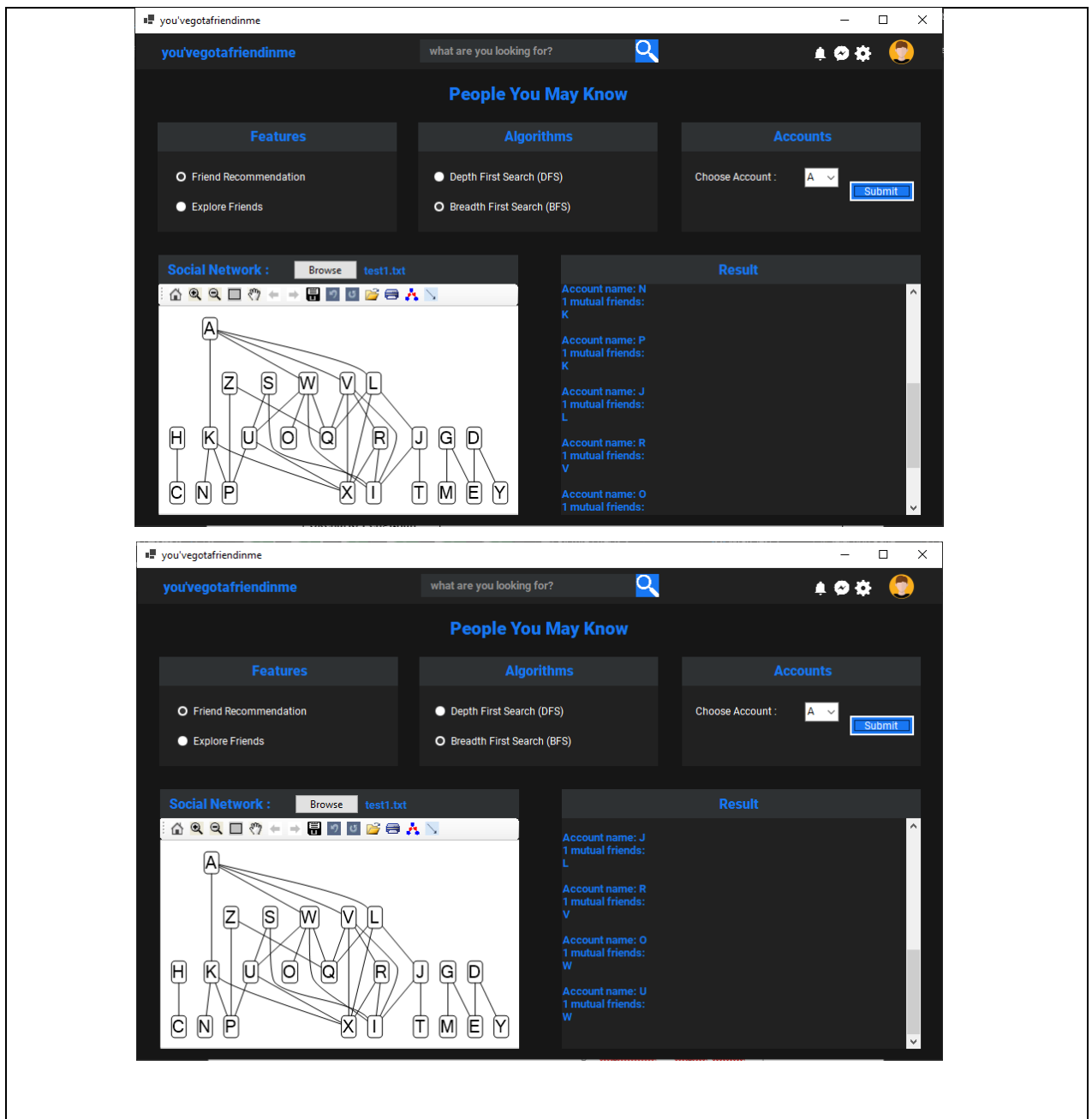
2. test2.txt

Amari Hudson
Amari Jamari
Amari Kai
Hudson Valen
Hudson Lennon
Hudson Pax
Hudson Finley
Ode Shay
Ode Nour
Ode Billie
Valen Cahaya
Valen Jamari
Valen Zorion
Valen Kai
Valen Taj
Cahaya Jamari
Cahaya Zorion
Cahaya Kai
Jamari Emery
Jamari Lennon
Qadan Gael
Qadan Nour
Qadan Izumi
Xiao Gael
Xiao Nour
Xiao Uri
Xiao Izumi

Xiao Riley
Emery Finley
Shay Gael
Shay Uri
Shay Izumi
Shay Devin
Zorion Pax
Zorion Kai
Zorion Yildiz
Zorion Finley
Gael Nour
Gael Uri
Gael Izumi
Gael Devin
Gael Marley
Nour Billie
Nour Izumi
Nour Devin
Nour Riley
Nour Marley
Uri Billie
Uri Izumi
Billie Devin
Billie Riley
Izumi Riley
Izumi Marley
Pax Kai
Pax Finley
Devin Riley
Kai Finley
Kai Taj
Yildiz Finley

Berikut adalah hasil pengujian dengan beberapa scenario pengujian:

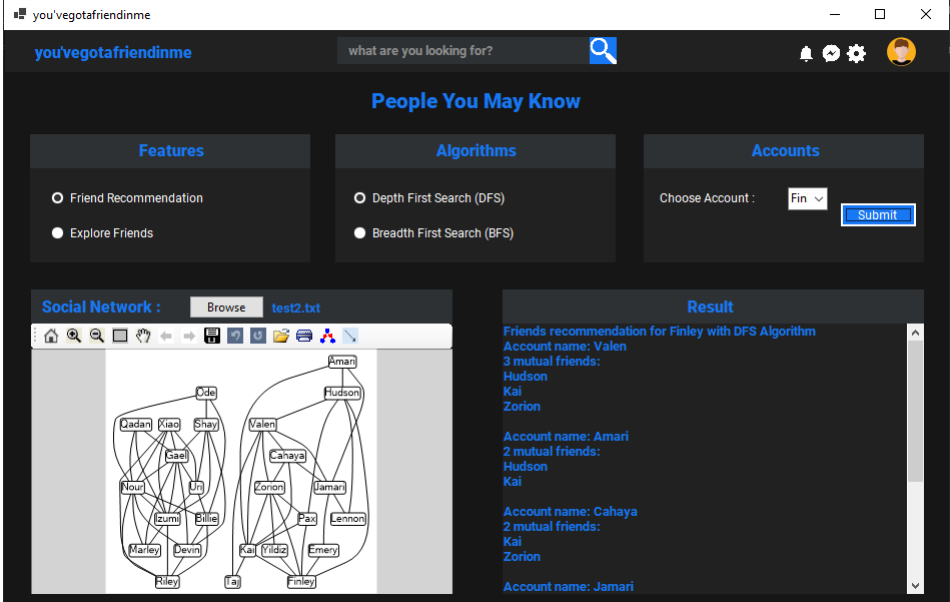
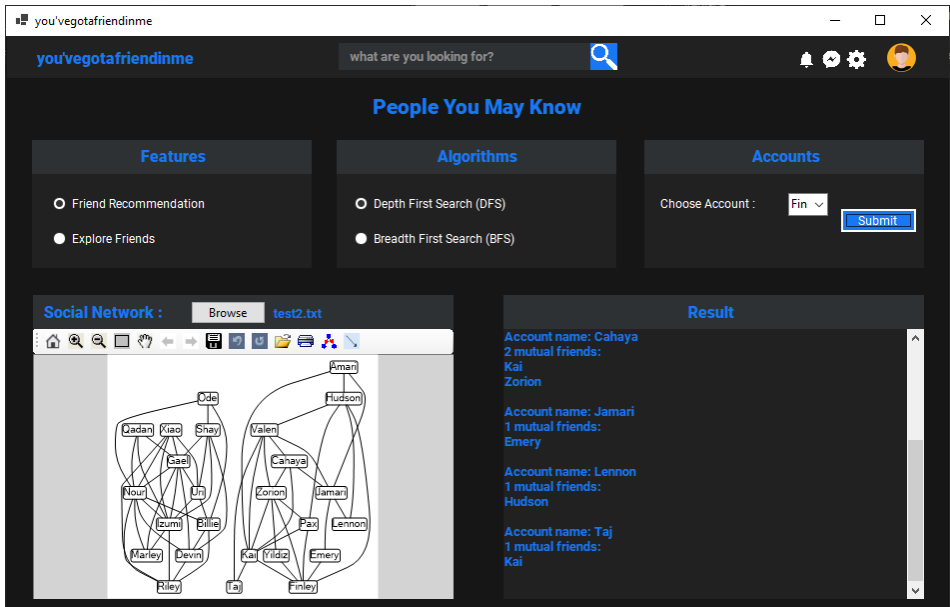
Testing 1	
File Graf	test1.txt
Skenario Pengujian	<ul style="list-style-type: none">• Menggunakan file graf dengan nama akun yang hanya terdiri dari satu karakter• Memilih fitur Friend Recommendation• Algoritma yang dipilih BFS atau DFS (hasilnya akan sama, tetapi jika ada lebih dari 1 akun yang memiliki jumlah mutual friends yang sama ada kemungkinan urutannya tidak sama karena disesuaikan dengan akun mana dulu yang dikunjungi)• Akun yang dipilih adalah akun A• Hasil: Ditemukan rekomendasi teman untuk akun A
Hasil Pengujian	
 The screenshot shows a web application titled 'you'vegotafriendinme'. At the top, there's a search bar with the placeholder 'what are you looking for?'. Below this, the main heading is 'People You May Know'. There are three main sections: 'Features' with radio buttons for 'Friend Recommendation' (selected) and 'Explore Friends'; 'Algorithms' with radio buttons for 'Depth First Search (DFS)' and 'Breadth First Search (BFS)' (selected); and 'Accounts' with a 'Choose Account:' dropdown set to 'A' and a 'Submit' button. Below these, there's a 'Social Network' section with a 'Browse' button and a file input showing 'test1.txt'. The network is visualized as a graph with nodes labeled with letters (A, Z, S, W, V, L, H, K, U, O, Q, R, J, G, D, C, N, P, X, I, T, M, E, Y) and edges representing connections. To the right of the graph is a 'Result' section titled 'Friends recommendation for A with BFS Algorithm'. It lists 'Account name: X' with '3 mutual friends: K, L, V', 'Account name: Q' with '3 mutual friends: L, V, W', and 'Account name: I' with '2 mutual friends: V, W'.	



Testing 2	
File Graf	test2.txt
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan file graf dengan nama akun terdiri dari lebih dari satu karakter Memilih fitur Friend Recommendation Algoritma yang dipilih BFS atau DFS (hasilnya akan sama, tetapi jika ada lebih dari 1 akun yang memiliki jumlah mutual friends

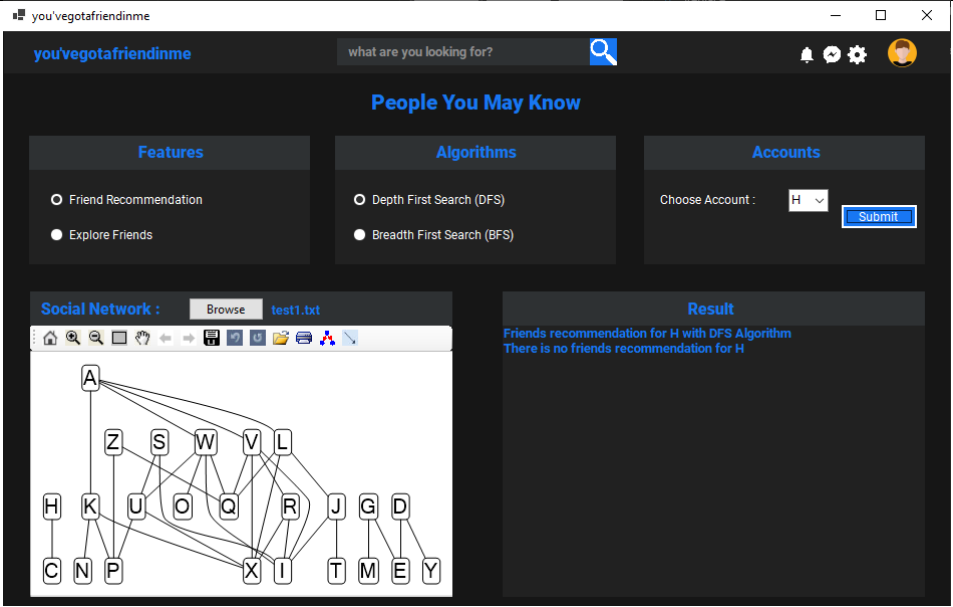
	<p>yang sama ada kemungkinan urutannya tidak sama karena disesuaikan dengan akun mana dulu yang dikunjungi)</p> <ul style="list-style-type: none"> • Akun yang dipilih adalah akun Finley • Hasil: Ditemukan rekomendasi teman untuk akun Finley
--	--

Hasil Pengujian

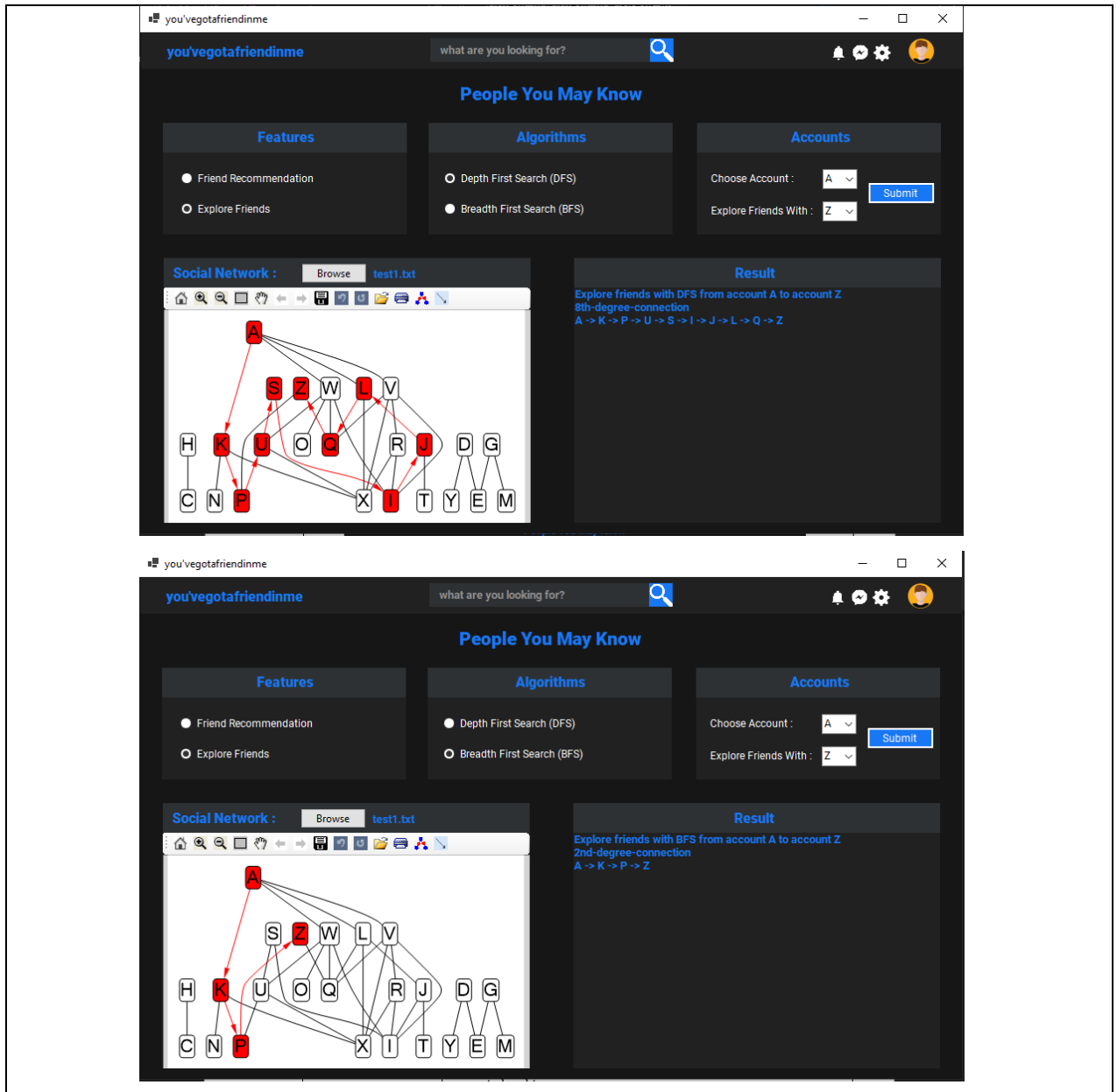
	
	

Testing 3

File Graf	test1.txt
Skenario Pengujian	<ul style="list-style-type: none"> • Menggunakan file graf dengan nama akun terdiri dari lebih dari satu karakter

	<ul style="list-style-type: none"> • Memilih fitur Friend Recommendation • Algoritma yang dipilih BFS atau DFS (hasilnya akan sama) • Akun yang dipilih adalah akun H • Tidak ditemukan rekomendasi teman untuk akun H
Hasil Pengujian	
	

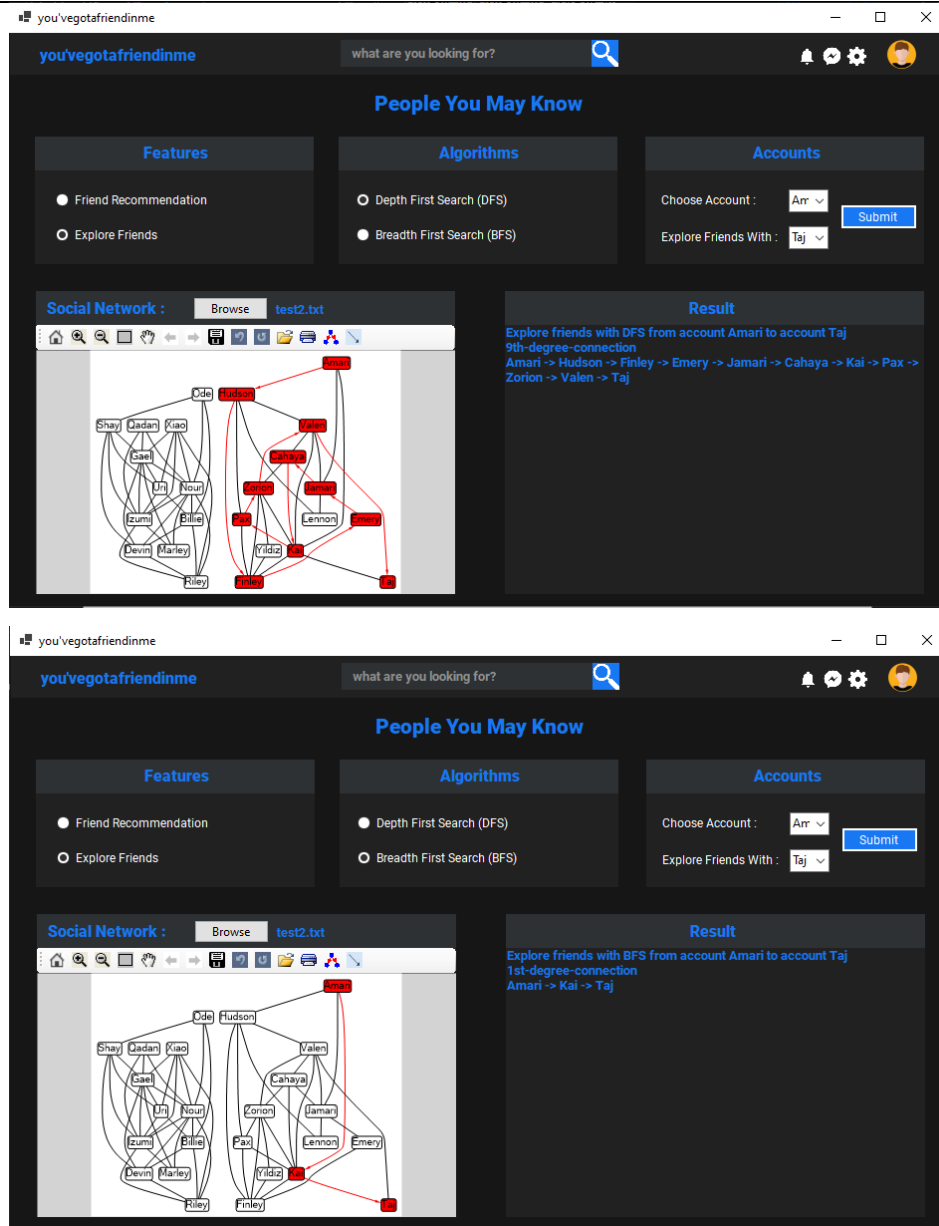
Testing 4	
File Graf	test1.txt
Skenario Pengujian	<ul style="list-style-type: none"> • Menggunakan file graf dengan nama akun yang hanya terdiri dari satu karakter • Memilih fitur Explore Friends • Algoritma yang dipilih BFS atau DFS (akan dibandingkan hasilnya) • Akun pertama yang dipilih adalah akun A dan Akun kedua yang dipilih adalah akun Z • Hasil: Ditemukan jalur koneksi yang menghubungkan A dan Z
Hasil Pengujian	



Testing 6	
File Graf	test2.txt
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan file graf dengan nama akun terdiri dari lebih dari satu karakter Memilih fitur Explore Friends Algoritma yang dipilih BFS atau DFS (akan dibandingkan hasilnya) Akun pertama yang dipilih adalah akun Amari dan Akun kedua yang dipilih adalah akun Taj

	<ul style="list-style-type: none"> Hasil: Ditemukan jalur koneksi yang menghubungkan Amari dan Taj
--	---

Hasil Pengujian



Testing 7	
File Graf	test1.txt
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan file graf dengan nama akun yang hanya terdiri dari satu karakter Memilih fitur Explore Friends

	<ul style="list-style-type: none"> • Algoritma yang dipilih BFS atau DFS (akan dibandingkan hasilnya) • Akun pertama yang dipilih adalah akun A dan Akun kedua yang dipilih adalah akun C • Hasil: Tidak ditemukan jalur koneksi yang menghubungkan A dan C
--	--

Hasil Pengujian

you'vegotafriendinme

what are you looking for?

People You May Know

Features

● Friend Recommendation

○ Explore Friends

Algorithms

○ Depth First Search (DFS)

● Breadth First Search (BFS)

Accounts

Choose Account : A

Explore Friends With : C

Submit

Social Network :

Browse test1.txt

Result

Explore friends with DFS from account A to account C

There isn't any connection available

You have to start the new connection itself

you'vegotafriendinme

what are you looking for?

People You May Know

Features

● Friend Recommendation

○ Explore Friends

Algorithms

● Depth First Search (DFS)

○ Breadth First Search (BFS)

Accounts

Choose Account : A

Explore Friends With : C

Submit

Social Network :

Browse test1.txt

Result

Explore friends with BFS from account A to account C

There isn't any connection available

You have to start the new connection itself

Testing 8

File Graf	test1.txt
-----------	-----------

Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan file graf dengan nama akun yang hanya terdiri dari satu karakter Memilih fitur Explore Friends Algoritma yang dipilih BFS atau DFS (akan dibandingkan hasilnya) Akun pertama yang dipilih adalah akun A dan Akun kedua yang dipilih adalah akun W Hasil: A dan W sudah berteman
--------------------	---

Hasil Pengujian

The image displays two screenshots of a web application interface for a social network. The application is titled "you'vegotafriendinme" and features a search bar at the top with the placeholder text "what are you looking for?". Below the search bar, there are three main sections: "Features", "Algorithms", and "Accounts".

Features: Includes "Friend Recommendation" and "Explore Friends".

Algorithms: Includes "Depth First Search (DFS)" and "Breadth First Search (BFS)".

Accounts: Includes a "Choose Account" dropdown menu (set to "A") and an "Explore Friends With" dropdown menu (set to "W"). A "Submit" button is located next to these dropdowns.

Social Network: A graph visualization showing a network of nodes (letters) and edges (connections). The nodes are arranged in a hierarchical structure, with 'A' at the top, connected to 'Z', 'S', 'W', 'V', and 'L'. 'Z' is connected to 'H', 'K', and 'U'. 'S' is connected to 'U' and 'O'. 'W' is connected to 'O' and 'Q'. 'V' is connected to 'Q' and 'R'. 'L' is connected to 'R' and 'J'. 'H' is connected to 'C' and 'N'. 'K' is connected to 'N' and 'P'. 'U' is connected to 'P' and 'X'. 'O' is connected to 'X' and 'I'. 'Q' is connected to 'X' and 'I'. 'R' is connected to 'I' and 'T'. 'J' is connected to 'T' and 'M'. 'G' is connected to 'M' and 'E'. 'D' is connected to 'E' and 'Y'. 'X' is connected to 'I' and 'T'. 'I' is connected to 'T' and 'M'. 'T' is connected to 'M' and 'E'. 'M' is connected to 'E' and 'Y'. 'E' is connected to 'Y'.

Result: The result section displays the message: "Explore friends with DFS from account A to account W. Both of them are already friends." This message is repeated in both screenshots, indicating that the result is the same for both DFS and BFS algorithms in this specific case.

4.5. Analisis Desain Solusi Algoritma

Perbedaan yang paling dasar dari BFS dan DFS ialah sesuai namanya, DFS mencari ke kedalaman node yang masih bisa ditelusuri, sementara BFS melakukan iterasi setiap lapisan node baru yang dikunjungi. Struktur data paling umum yang dipakai DFS ialah Stack karena First In Last Out, sementara BFS ialah Queue karena First In First Out. Kasus-kasus dimana DFS dan BFS mengungguli atau diungguli yang lain dapat kita bedah lebih lanjut dengan penjabaran keuntungan masing-masing algoritma berikut.

Keuntungan BFS dibandingkan DFS ialah bahwa BFS melakukan iterasi per lapisan, sehingga dalam kasus mono-weighted graph, BFS dijamin menghasilkan jalur terpendek. Fakta uniknya, penelusuran BFS pada mono-weighted graph sama persis dengan Algoritma Dijkstra. Jika dilanjutkan hingga Queue kosong, BFS dapat menghitung jarak minimal tiap node pada graph dengan node awal. Lebih lanjutnya lagi, dapat dilakukan multisource BFS yakni BFS dengan node awal lebih dari satu untuk menyelesaikan permasalahan tertentu. Kompleksitas waktu dari BFS ialah jumlah edge dari connected-component tempat node asal berada, sedangkan kompleksitas memorinya ialah jumlah node dalam connected-component node asal. Connected component ialah node pada graph yang saling terhubung.

Selanjutnya DFS juga terkadang lebih menguntungkan dari BFS untuk sedikit kasus misalnya saat graph berbentuk tree, misalkan node awal merupakan akarnya, jumlah subpohon dari akar cukup banyak, dan node yang dituju berada pada subpohon-subpohon awal. Maka BFS akan membuang waktu mencari pada lapisan demi lapisan yang jumlahnya dapat bertambah banyak secara eksponensial, sementara DFS akan mencari satu subpohon dulu secara menyeluruh. Kerugian DFS yakni jalur yang dihasilkan tidak selalu minimal tentunya berpengaruh pada pengecekan node awal dan yang dituju sudah terhubung dengan edge atau belum. Namun disamping itu, DFS dapat diimplementasikan dengan metode rekursi yang sangat pendek dan mudah dalam proses pengembangan kode.

Meskipun BFS jauh lebih disarankan dalam pathfinding, DFS dapat melakukan hal yang tidak dapat dilakukan BFS, misalnya pada kasus Dynamic Programming (DP) on Tree, konstruksi Bridge Tree, pencarian Articulation Point, Biconnected Component/Strongly Connected Component (BC/SCC), dan lainnya. DP on Tree ialah struktur data yang menyimpan banyaknya node pada subtree dari masing-masing node. BC/SCC ialah connected component yang bila dihapus satu edge manapun akan tetap terhubung. Articulation Point ialah node yang jika dihapus akan membuat graph menjadi tidak terhubung. Terakhir Bridge Tree ialah pohon yang node nya berupa BC/SCC pada graph, dan node nya adalah tiap edge pada graph yang jika dihapus akan membuat graph tidak terhubung.

BAB V

KESIMPULAN DAN SARAN

1. Kesimpulan

Berdasarkan yang sudah dipaparkan sebelumnya, dapat diambil kesimpulan bahwa struktur data graf dapat merepresentasikan *social network* dengan baik karena memudahkan dalam proses traversal dari atribut-atributnya. Proses traversal yang dimaksud adalah metode pencarian DFS dan BFS.

Metode pencarian DFS dan BFS pada kasus mencari *friend recommendation* sebuah akun dan *explore friend* dari dua buah akun dapat menemukan solusi yang baik dan tidak jauh berbeda. Namun, karena terdapat perbedaan cara penelusuran *node*, maka jalur yang dilalui saat proses traversal berbeda dan metode BFS cenderung menghasilkan solusi jalur yang lebih pendek. Dalam hal ini, program kami sudah dapat menerapkan metode pencarian BFS dan DFS dalam fitur *People You Know* di jejaring sosial Facebook dengan tepat.

2. Saran

Saran-saran yang dapat kami berikan untuk tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021 adalah:

- a. Algoritma yang digunakan pada Tugas Besar ini masih memiliki banyak kekurangan sehingga sangat memungkinkan untuk dilakukan efisiensi, misalnya dengan tidak menggunakan fungsi yang sama berulang-ulang. Oleh karena itu, dalam pengembangan program ini, masih bisa dilakukan efisiensi kinerja.
- b. Memperjelas spesifikasi dan batasan-batasan setiap program pada *file* tugas besar untuk mencegah adanya multitafsir dan kesalahpahaman pada proses pembuatan program.
- c. Penulisan *pseudocode* tampak kurang perlu dikarenakan program yang lumayan panjang dan membaca program lebih mudah daripada membaca *pseudocode* dengan asumsi program sudah *well commented*.

3. Refleksi dan Komentar

Pada pengerjaan Tugas Besar 2 Strategi Algoritma ini, kami sadar akan pentingnya menentukan prioritas dan pembagian kerja yang baik saat bekerja dalam sebuah tim karena pada dunia nyata setiap anggota tim memiliki kesibukan atau tuntutan lain diluar tugas besar ini. Faktor penting lainnya adalah komunikasi dan transparansi antar anggota. Hal ini akan meminimalisir kemungkinan terjadinya miskomunikasi dan memudahkan untuk *keep in track* progress yang sudah dikerjakan oleh masing-masing anggota sehingga dapat mengatasi masalah lebih cepat dan taktis. Diluar dari hal yang telah disebutkan, yang terpenting adalah untuk selalu melatih keterampilan diri sendiri secara rutin dan berkala serta mengeksplor pengetahuan-pengetahuan yang tidak diajarkan dalam kelas sehingga dapat lebih adaptif saat dihadapkan pada suatu permasalahan.

DAFTAR PUSTAKA

- [1] 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 22 Maret 2021.
- [2] 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 22 Maret 2021.
- [3] Behance. (n.d.). Facebook new look - refine. Diakses online dari https://www.behance.net/gallery/85610473/Facebook-new-look-Refine?tracking_source=search pada 22 Maret 2021.