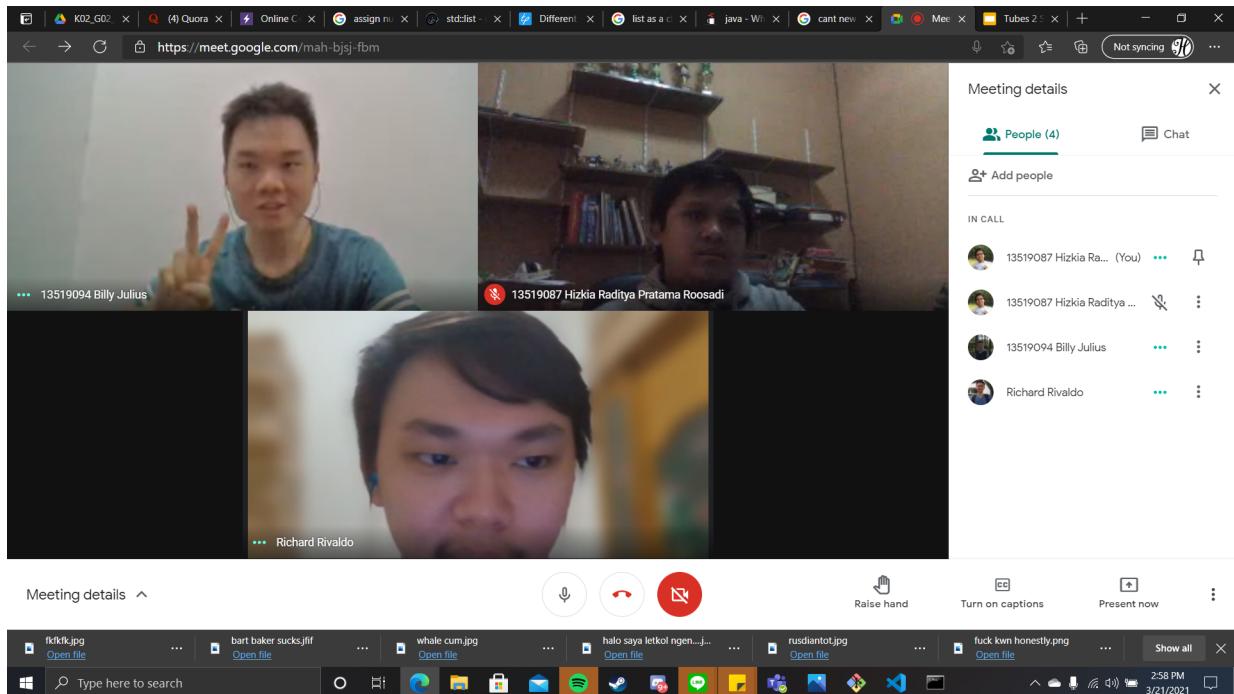


LAPORAN HASIL TUGAS BESAR 2 IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2020/2021

PENGAPLIKASIAN ALGORITMA BFS DAN DFS DALAM FITUR

PEOPLE YOU MAY KNOW JEJARING SOSIAL FACEBOOK



Disusun oleh :

Hizkia Raditya Pratama Roosadi 13519087

Billy Julius 13519094

Richard Rivaldo 13519185

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2021

DAFTAR ISI

BAB I DESKRIPSI TUGAS	3
BAB II LANDASAN TEORI	6
2.1 Dasar Teori Graph Traversal, BFS, dan DFS	6
2.2 C# Desktop Application Development	7
BAB III ANALISIS PEMECAHAN MASALAH	11
3.1 Langkah-Langkah Pemecahan Masalah	11
3.2 Mapping Elemen-Elemen BFS dan DFS	11
3.3 Contoh Ilustrasi Kasus Lain	12
BAB IV IMPLEMENTASI DAN PENGUJIAN	15
4.1 Implementasi Program	15
4.2 Struktur Data yang Digunakan	19
4.3 Cara Penggunaan Program	20
4.4 Hasil Pengujian	21
4.5 Analisis Hasil Pengujian	26
BAB V PENUTUP	28
5.1 Kesimpulan	28
5.2 Saran	28
5.3 Refleksi	28
5.4 Komentar	29
BAB VI DAFTAR PUSTAKA	30

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. Untuk fitur friend recommendation, misalnya pengguna ingin mengetahui daftar rekomendasi teman untuk 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.

Aplikasi yang akan dibangun dibuat berbasis GUI, dengan spesifikasi :

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 IF2211 Strategi Algoritma - Tugas Besar 2 7 (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/1XhFSpHU028Gaf7YxkmdbluLkQgVI3MY6gt1tPL30LA/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 antara 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.

BAB II LANDASAN TEORI

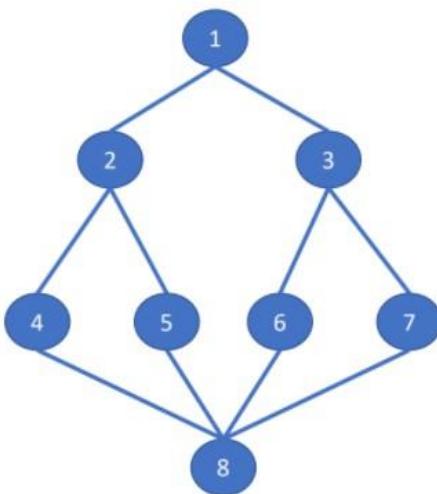
2.1 Dasar Teori Graph Traversal, BFS, dan DFS

Graph traversal merupakan salah satu algoritma yang sering digunakan, biasanya untuk melakukan proses ke graf, dengan mengunjungi tiap *vertex/node*. Ada beberapa jenis algoritma *graph traversal*, dua di antaranya adalah *Breadth First Search (BFS)* dan *Depth First Search (DFS)*.

a. Breadth First Search (BFS)

Algoritma *Breadth First Search* mengunjungi *vertex-vertex* di graf yang bertetanggaan dengan *vertex* yang dipilih. Setelah semua *vertex* tetangga sudah dikunjungi, penelusuran dilanjutkan ke tahap berikutnya, yaitu merujuk ke tiap *vertex* tetangga sebelumnya, untuk ditelusuri *vertex-vertex* tetangganya. Hal ini dilakukan terus sampai tujuan penelusuran tercapai atau semua *vertex* sudah dikunjungi. Untuk lebih jelasnya, perhatikan gambar berikut.

BFS: Ilustrasi



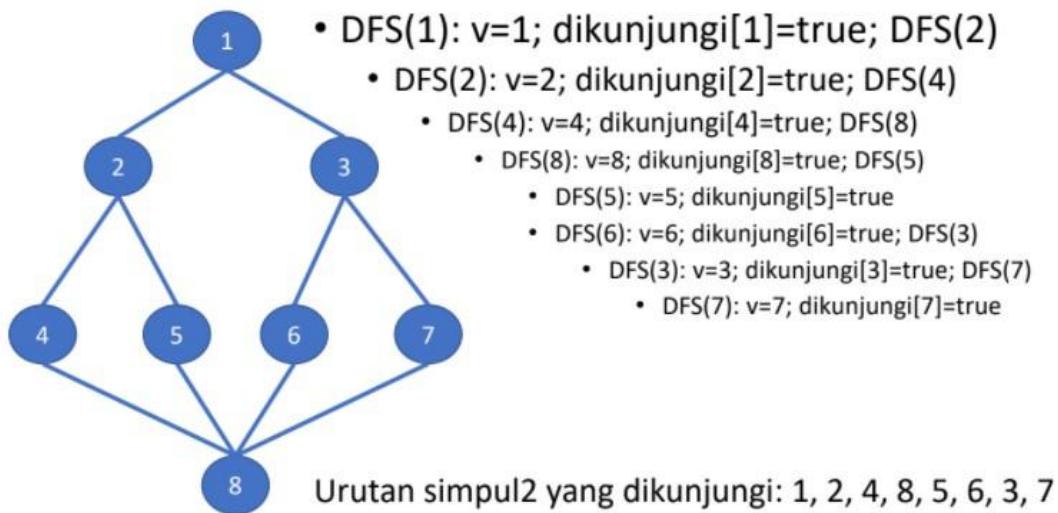
Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi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

b. Depth First Search (DFS)

Algoritma *Depth First Search* mengunjungi satu *vertex* yang bertetanggaan dengan *vertex* yang dipilih, kemudian penelusuran dilanjutkan ke *vertex* yang bertetanggaan dengan *vertex* sebelumnya. Ketika *vertex* yang dirujuk sudah tidak memiliki *vertex* yang bertetanggaan yang belum dikunjungi, penelusuran di-*backtrack* ke *vertex* sebelumnya, dan dilakukan penelusuran ke *vertex* tetangga yang lain. Hal ini dilakukan terus sampai tujuan penelusuran tercapai atau semua *vertex* sudah dikunjungi. Untuk lebih jelasnya, perhatikan gambar berikut.

DFS: Ilustrasi 1



2.2 C# Desktop Application Development

Sebelum masuk lebih jauh ke dalam pengembangan perangkat lunak *desktop*, akan dibahas mengenai .NET terlebih dahulu. Secara formal, .NET adalah sebuah *open-source platform* pengembangan yang gratis dan dikembangkan oleh Microsoft untuk membantu pengembang dalam membangun berbagai macam perangkat lunak. Dengan menggunakan .NET, pengembang bisa mengombinasikan berbagai macam kakas, bahasa pemrograman, dan pustaka untuk membangun perangkat lunak *web*, *mobile*, *desktop*, bahkan hingga *games*, IOT, dan *machine learning*.

Hingga saat ini, .NET sudah sampai pada versi 5.0, dan versi ini merupakan suksesor gabungan dari .NET Core dan .Net Framework sebelumnya. .NET Framework adalah versi ‘asli’ dari .NET, yang biasanya digunakan untuk mengembangkan aplikasi yang hanya dijalankan di Windows. .NET Core, lain halnya dengan .NET Framework, bisa digunakan untuk pengembangan *cross-platform*, baik *Windows* maupun *Linux* atau *macOS*, dan bersifat lebih fleksibel dan baru dibandingkan dengan .NET Framework. Keduanya saling beririsan dan berbagi beberapa API yang digunakan masing-masing .NET.

Microsoft sebagai pengembang .NET juga telah menyediakan .NET Standard yang merupakan basis dari semua .NET yang ada. .NET Standard merupakan implementasi dasar yang nantinya digunakan untuk mengimplementasikan .NET Framework dan .NET Core. Adapun Microsoft juga menyediakan sebuah *package manager* yang membantu pengembang dalam menggunakan pustaka-pustaka, dan kakas ini dikenal dengan NuGet.

Untuk menggunakan .NET, maka pengembang hanya perlu mengunduh *Visual Studio*. Visual Studio dapat dikatakan sebagai sebuah kakas yang sangat baik dan lengkap untuk menggunakan .NET. Dengan mengunduh Visual Studio, maka .NET akan otomatis terunduh dan langsung siap digunakan melalui Visual Studio. Selain itu, IDE ini juga telah memasukkan NuGet Package Manager di dalamnya sehingga pengembang dapat dengan mudah menambah pustaka baru, misalnya Microsoft Automatic Graph Layout atau MSAGL.

Dalam mengembangkan aplikasi dengan .NET, pengembang bisa menggunakan beberapa bahasa pemrograman, seperti C# (baca: C Sharp), F# (F Sharp), Visual C++, atau Visual Basic. Namun, dari bahasa-bahasa ini, C# cenderung lebih sering digunakan dengan .NET. Hal ini dikarenakan C# memiliki kompatibilitas terbaik dengan .NET dibandingkan bahasa-bahasa lain tersebut. C# juga dapat dikatakan sebagai bahasa pemrograman yang cukup cepat dalam memproses program yang ada.

C# merupakan sebuah bahasa pemrograman berorientasi objek berbasis komponen yang juga dikembangkan oleh Microsoft. C# dapat dikatakan merupakan bahasa yang mirip sekali dengan kombinasi Java dan C++. C# memiliki kecepatan yang cukup seperti C++, dan C# juga memiliki banyak sekali pustaka standar yang mirip dengan Java. Hal tersebut tentunya sangat memudahkan pengembang dalam menggunakan berbagai struktur data dan fungsionalitasnya.

Secara umum, di dalam sebuah berkas C#, harus terdapat paling tidak sebuah kelas, dan kelas ini harus ditampung di dalam sebuah *container* yang disebut dengan *namespace*. Dengan menggunakan sebuah *namespace* yang sama, maka beberapa kelas dapat berinteraksi dan digunakan dalam sebuah program yang sama, meskipun dirancang di dua file berbeda. Tidak seperti C++, C# tidak membutuhkan *header* dan setiap fungsionalitasnya dibangun seperti halnya di dalam Java.

Dalam mengembangkan sebuah *Graphical User Interface* (GUI), ada dua buah *framework* tampilan layar yang biasanya digunakan, yaitu *Windows Form* dan *Windows Presentation Foundation*. Meskipun pada saat ini WPF lebih sering digunakan karena versatilitasnya, Winforms dapat dikatakan sebagai sebuah *framework* yang sangat sederhana dan bisa dirancang dengan mudah oleh pengembang. Namun, keduanya memiliki *trade-off* kelebihan dan kelemahannya masing-masing. Di dalam tugas ini, pengembang akan menggunakan Winform sebagai *framework* dasarnya.

Sebuah aplikasi yang dibangun dengan Winforms akan didesain pada sebuah *form*, dan tentunya pengembang bisa menggunakan lebih dari satu *form* saja. Dengan menggunakan Visual Studio, maka pengembang bisa mengembangkan aplikasinya dengan dua metode, yaitu metode grafis dan metode programatik. Metode grafis yang dimaksud adalah dengan melakukan *drag-and-drop* objek-objek yang diinginkan ke dalam *form* tersebut, dan Visual Studio secara otomatis akan melakukan *generate* kode program sesuai dengan yang dilakukan pengembang terhadap objek tersebut.

Beberapa objek yang sering digunakan dalam Winforms tentunya mencakup *Button*, yaitu tombol biasa, *List Box*, yang merupakan sebuah *list* untuk menampung beberapa *items*, ataupun *Combo Box* yang sifatnya mirip dengan *Dropdown List* dan *List Box*. Selain itu, juga ada *Label* yang bisa digunakan untuk menandakan sebuah bagian pada aplikasi, serta *Text Box* dan *Rich Text Box* yang bisa menerima input masukkan pengguna. Tidak hanya itu, ada juga *Picture Box* yang juga bisa digunakan untuk menampung gambar secara khusus. Objek panel juga biasanya digunakan sebagai *dock* dari perangkat lunak.

Setiap objek yang ada pada Winform memiliki definisi atribut dan *method* masing-masing (kembali lagi pada konsep OOP). Pada Visual Studio, lagi-lagi pengembang dapat memodifikasi atribut ini secara grafis maupun programatik, namun modifikasi *method* harus dilakukan secara

programmatik. Hal lain yang perlu diperhatikan adalah bahwa karena perbedaan atribut tersebut, setiap objek memiliki batasan desain yang berbeda-beda. Misalnya, ada sebuah objek yang *background color* nya bisa dibuat transparan dan ada juga yang tidak.

Adapun metode programmatik yang dilakukan berkaitan dengan menulis secara langsung dan manual kode program yang diinginkan. Hal ini meliputi pembuatan objek secara manual, modifikasi objek yang tidak bisa dilakukan dengan metode grafis, dan tentunya termasuk *backend* atau cara kerja fungsionalitas dari program yang diinginkan. Nantinya, komponen-komponen ini akan dikompilasi dan berhubungan dengan beberapa kode Assembly yang ada dan dimasukkan menjadi sebuah berkas *Solution*. Di dalam sebuah *project*, minimal terdapat satu buah *Solution* dan berkas ini bisa diklasifikasikan berdasarkan fungsionalitasnya.

Meskipun terlihat sulit dan kompleks, pengembangan aplikasi berbasis GUI dengan Visual Studio dan C# ini dapat dilakukan dengan cukup mudah. Dengan bantuan pustaka yang ada dan fitur-fitur pendukung yang disediakan kakas ini, maka pengembang dapat dengan mudah mengembangkan sebuah aplikasi, jika tujuannya hanya untuk melakukan interaksi sederhana dengan penggunanya.

BAB III ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Secara garis besar, mulai dari pembacaan file masukan, ada beberapa tahap algoritma yang digunakan sampai mampu untuk memberikan fitur yang diinginkan :

1. Pembacaan file eksternal dan menerjemahkannya menjadi sebuah graf. Karena isi file eksternal berbentuk hubungan ketetanggan antara dua *vertex* yang dipisahkan antarbaris, proses translasi dilakukan dengan membaca tiap hubungan ketetanggan tersebut (per baris), memasukkan *vertex* dengan ke dalam graf (jika belum ada), dan mengisi hubungan ketetanggaannya yang direpresentasikan di atribut yang ada pada tiap *vertex/node*.
2. Menerima pilihan pengguna berupa jenis algoritma (*BFS/DFS*), akun yang dipilih, dan akun tujuan yang akan dijadikan tujuan untuk fitur *Explore Friend*.
3. Berdasarkan pilihan pengguna, akan ditampilkan hasil fitur *Explore Friend*. Apabila pilihannya adalah *BFS*, maka pencarian akan disesuaikan dengan algoritma *graph traversal BFS*, yaitu pencarian yang mendahulukan *vertex-vertex* yang merupakan tetangga dari *vertex* yang dirujuk. Sebaliknya, jika algoritma yang dipilih adalah *DFS*, pencarian akan dilakukan dengan mendahulukan pencarian sampai ke tetangga *vertex* yang paling bawah/dasar, kemudian *backtrack* ke *vertex* sebelumnya. Setelah *vertex* tujuan telah berhasil ditemukan, maka akan ditampilkan jalur dan hubungan pertemanan yang dimaksud (*degree connection*).
4. Untuk fitur *Friend Recommendation*, akan diisi terlebih dahulu atribut *mutuals* pada tiap *vertex* yang menyatakan daftar teman yang sama dengan *user* yang dirujuk. Setelah itu, akan dilakukan penampilan (*print*) *vertex-vertex* pada graf mulai dari *mutuals* yang paling banyak, diikuti nama teman yang sama tersebut.

3.2 Mapping Elemen-Elemen BFS dan DFS

Pada permasalahan tugas besar ini, akun-akun akan direpresentasikan sebagai *vertex/node* dan hubungan pertemanan lah yang menjadi sisinya. Sehingga, terbentuk graf tak berarah yang menggambarkan hubungan pertemanan antara semua akun-akun yang dimasukkan. Algoritma

BFS dan *DFS* diimplementasi pada fitur *Friend Explore*, yang dapat dilihat penjelasan lebih detailnya sebagai berikut.

a. Algoritma *BFS*

Pencarian jalur pertemanan dengan akun tujuan akan dilakukan dengan memeriksa teman dari akun yang dipilih sebagai *user*. Setelah semua teman dari *user* diperiksa, jika tidak ditemukan, pencarian akan dilanjutkan dengan mencari ke teman-teman dari teman *user*, begitu seterusnya sampai ditemukan akun tujuan atau ternyata tidak ada jalur pertemanan yang mungkin dengan akun tersebut.

b. Algoritma *DFS*

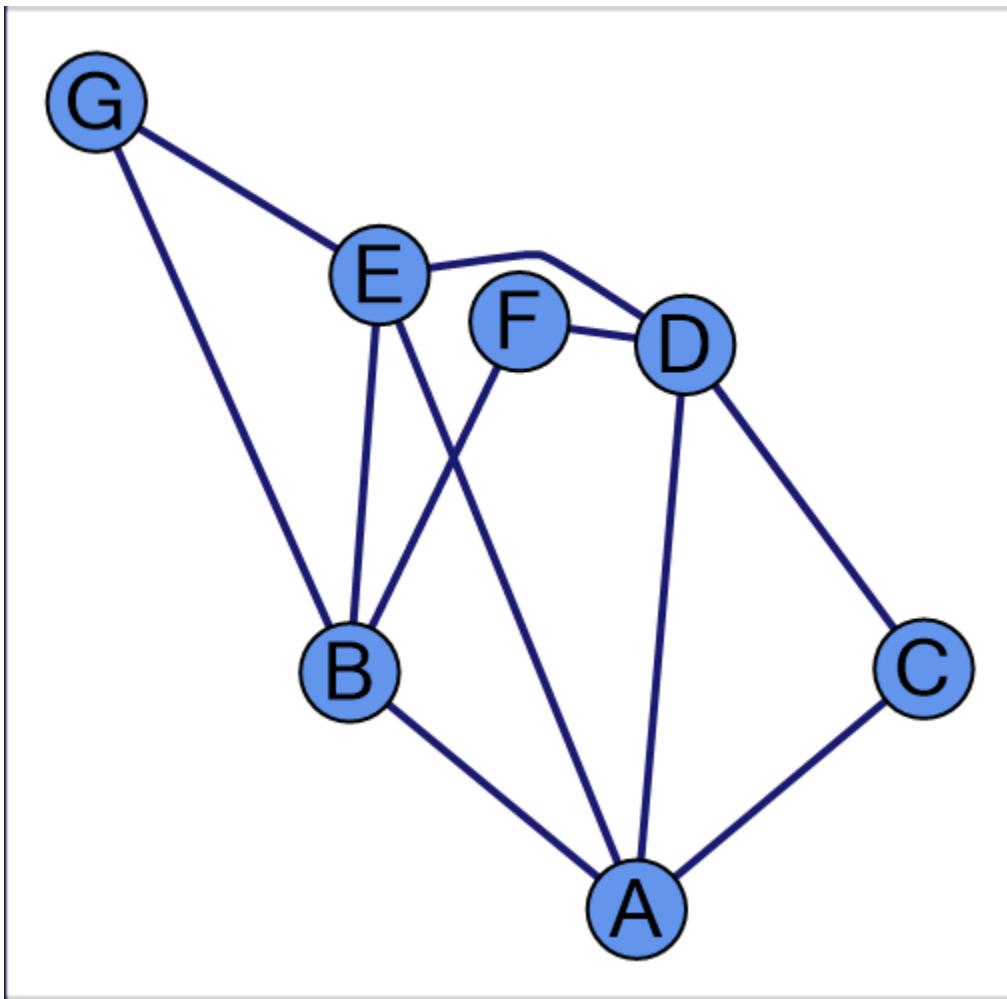
Dengan awal yang sama, pencarian jalur pertemanan dengan akun tujuan akan dilakukan dengan memeriksa teman dari akun yang dipilih sebagai *user*. Pencarian akan dimulai dari satu teman *user*, jika belum ditemukan, kemudian dilanjutkan dengan teman dari teman yang dirujuk sebelumnya. Hal ini diteruskan hingga pencarian telah sampai pada akun yang tidak memiliki teman yang belum diperiksa. Jika masih belum ditemukan, pencarian akan dimundurkan ke teman sebelumnya dan memeriksa teman dari teman tersebut, namun berbeda dari yang sudah diperiksa sebelumnya. Proses ini akan diulang sampai ditemukan akun tujuan atau ternyata tidak ada jalur pertemanan yang mungkin dengan akun tersebut.

3.3 Contoh Ilustrasi Kasus Lain

Untuk ilustrasi kasus lain, akan digunakan dua masukan file eksternal sebagai berikut.

```
11
A B
A C
A D
A E
B E
B F
B G
C D
D F
D E
E G
```

Berikut adalah implementasi graf dari file di atas.



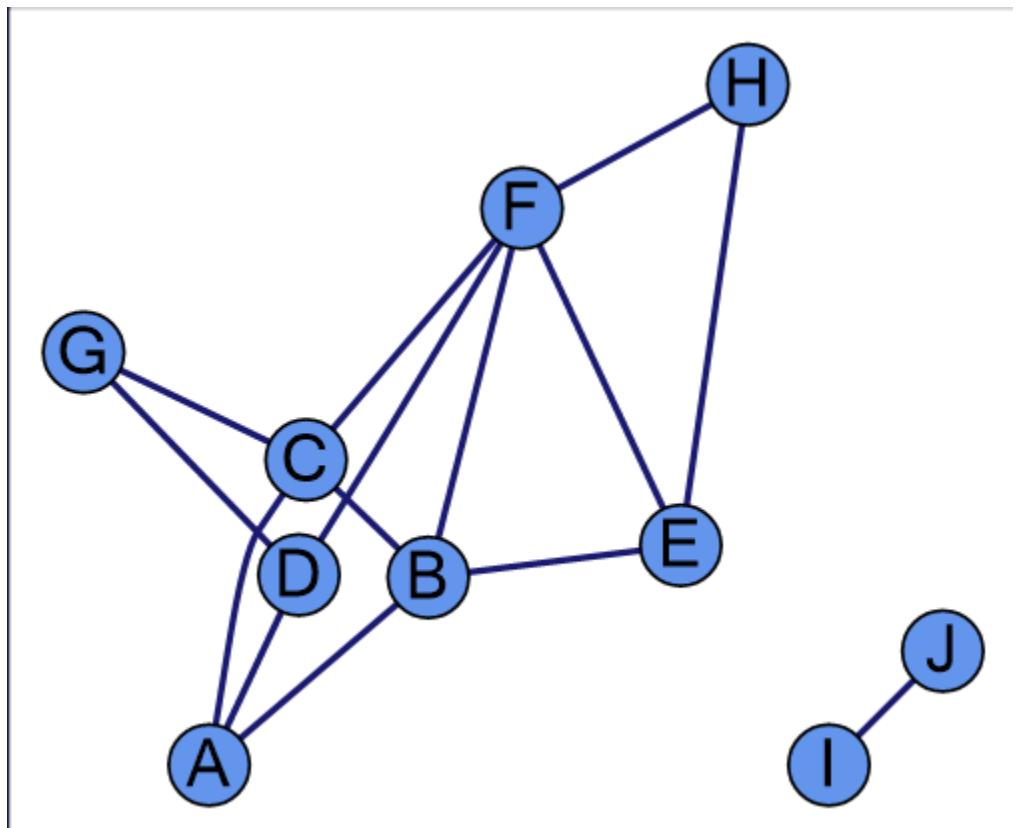
Gambar 3.1. Graf 1

Berikut adalah contoh kasus kedua

```
14  
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  
I J
```

Berikut adalah implementasi graf dari file di atas.



Gambar 3.2. Graf 2

BAB IV IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

Pada subbab ini, akan ditampilkan beberapa pseudocode dari algoritma yang penting dalam pembuatan fitur pada aplikasi ini.

a. *Recommend Friend*

```
function FriendRec(Graph G, string nameSelected) -> (string, List<string>)
KAMUS LOKAL
idxUser, max, currCount, idxMax : int
user, n : Node
un, nn, nameMax : string
currMutuals : List<string>
dict : (string, List<string>)

ALGORITMA
idxUser <- GetIndexFromNodeName(G, nameSelected)
node <- G.NodeList[idxUser]

foreach n in G.NodeList :
    if (not(Contains(user.neighbors, n.name)) then
        foreach un in user.neighbors :
            foreach nn in n.neighbors :
                if (un = nn and not(Contains(n.mutuals, un))) then
                    Add(n.mutuals, un)

SetVisitedTrue(G.NodeList[idxUser])
while (not(isVisitedAll(G))) do
    nameMax <- ""
    max <- 99
    foreach n in G.NodeList :
        currCount <- Count(n.mutuals)
        if (currCount > max and not(isVisited(n))) then
            nameMax <- n.name
            max <- currCount

    idxMax <- GetIndexFromNodeName(G, nameMax)
    SetVisitedTrue(G.NodeList[idxMax])

    if (max != 0) then
```

```

foreach n in G.NodeList[idxMax].mutuals :
    Add(currMutuals, n)
    Add(dict, nameMax, currMutuals)

UnvisitAll(G)
-> dict

```

b. Friend Exploration BFS

```

function FriendExploreBFS(Graph G, Node start, Node finish)-> (List<string>,int)

KAMUS LOKAL
solution,degreeCount: List<String>
simpulhidup: List<Node>
i,degree,countToIncDeg,countNeighs: integer
curNode: Node

ALGORITMA
{tambahkan start ke simpul hidup dan isi isVisited di start sebagai true}
simpulhidup.add(start)
start.setVisitedTrue()
degree<-0

if (start.neighbors.contains(finish.name)) then
{start dan finish sudah merupakan neighbors}
    solution.add(start.name)
    solution.add(finish.name)
else {start dan finish bukan tetangga}
    i<-0
    curNode<-simpulHidup[i]
    while (curNode.name!=finish.name and i<simpulhidup.count) do
        foreach(string nameNode in curNode.neighbors): {this visits each of the nodes in
neighbours hence the BFS part}
            secNode <- G.NodeList[G.GetIndexFromNodeName(nameNode)];
            if (!secNode.isVisited) then
                {get the total neighbors of one level}
                foreach (string n in secNode.neighbors){
                    thirdNode <- G.NodeList[G.GetIndexFromNodeName(n)];
                    if (!thirdNode.isVisited and (not degreeCount.Contains(thirdNode.name))) then

```

```

degreeCount.Add(thirdNode.name);
countNeighs++;

{ if it is time to increment, inc the degree, reset the counts}
if (--countToIncDeg = 0) then
    degree++;
    countToIncDeg <- countNeighs;
    countNeighs <- 0;

foreach(Node n in G.NodeList):
    if (nameNode=n.name and not n.isVisited) then {add neighbor nodes to last idx}
        simpulhidup.Add(n);
        n.SetVisitedTrue();

    i++
    if(i>=simpulhidup.count) then break; {if node is not found, break the loop}
    if(not solution.Contains(curNode.name)) then
        solution.Add(curNode.name); {add curNode to solution}

    curNode<-simpulhidup[i];

    {endwhile}
if (curNode.name=finish.name) then
    solution.Add(curNode.name) {check if last node is the finish node}

degree--; {count system always -1}

if (not solution.Contains(finish.name)) then
    solution.Clear() {clear the solution if it doesn't contain the finish node}

Set_All_Visited_False(G); {set all nodes.isVisited in G to false}

-> (solution,degree)

```

c. *Friend Exploration DFS*

```
function FriendExploreDFS(Graph G,Node start, Node finish)->List<Node>

KAMUS LOKAL

ALGORITMA
history,visited: List<Node>
i: integer
isAllNeighborsVisited: boolean
curNode: Node

ALGORITMA

history.Add(start);
start.SetVisitedTrue();

i<-0
curNode<-history[i]

while(not G.isVisitedAll() and not history.Contains(finish)) do
    foreach(Node n in G.NodeList):
        if (n.neighbors.Contains(curNode.name) and not n.isVisited) then
            history.Add(n);
            n.SetVisitedTrue();
            i++;
            break;

        curNode<-history[i] {curNode is last added Node}
        if(history.Contains(finish)) then break; {in case it already contains finish, then break}
        foreach(string n2 in curNode.neighbors):
            foreach(Node n3 in G.NodeList):
                if (n3.name=n2 and not n3.isVisited) then
                    isAllNeighborsVisited<-false;

                if (isAllNeighborsVisited) then
                    i--; {essentially backtracks}
                    if (i<0) then break; {just in case node is not found}
                    history.Remove(curNode)
                    curNode<-history[i] {set curNode as the previous node}
{endwhile}
```

```

if (i<0) then history.Clear {clear history}
Set_All_Visited_False(G)

->history

```

4.2 Struktur Data yang Digunakan

```

{struktur data node}

class Node:
    {attribute}
    name: string
    neighbors: List of string
    isVisited: boolean
    mutuals: List of string

    {method}
    Node() {ctor}
    AddNeighbor(input: string newneighbor) {prosedur add new neighbor}
    RemoveNeighbor(input: string neighbor} {prosedur remove neighbor}
    SetVisitedTrue() {method untuk me-set isVisited node ini sebagai true}

{struktur data graph}

class Graph:
    {attribute}
    NodeList: List of Node

    {method}
    Graph() {ctor}
    AddToList(input: Node newnode) {menambahkan newnode kedalam Graph.NodeList jika tidak ada sebelumnya}
    GetIndexFromNodeName(string name)->int {mengembalikan index node tersebut di graph jika ada, dari string nama}
    UnvisitAll() {me-set seluruh node.isVisited didalam NodeList sebagai false}
    isVisitedAll()->boolean {mengembalikan true jika seluruh node.isVisited didalam NodeList=true}
    makeGraphFromText(string filename)->Graph {mengembalikan graph dari file .txt}

```

4.3 Cara Penggunaan Program

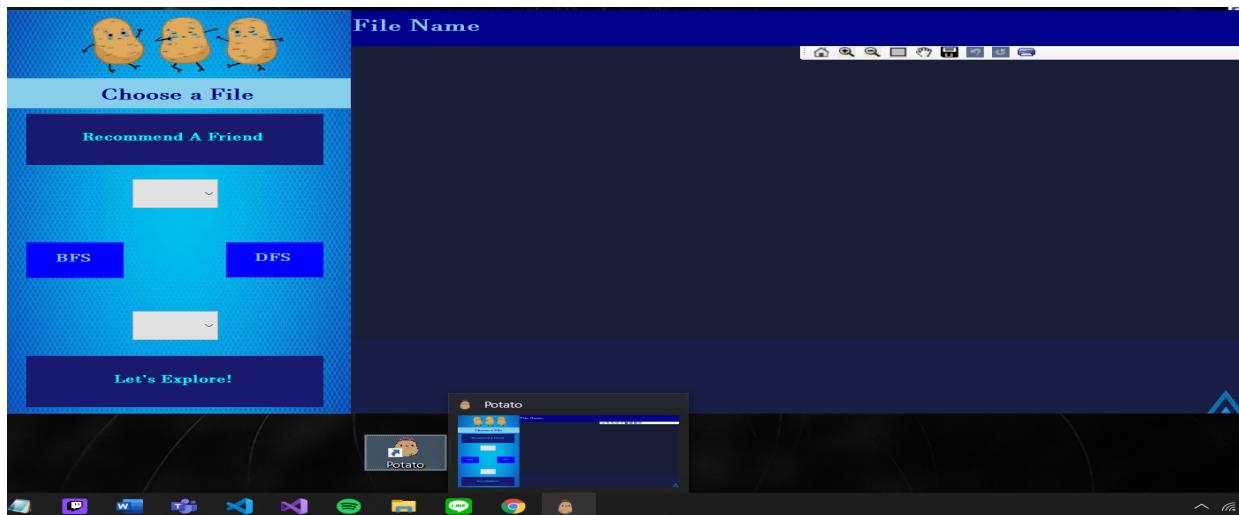
Langkah-langkah yang bisa digunakan oleh pengguna dalam menggunakan aplikasi ‘Potato’ adalah sebagai berikut.

1. Lakukan instalasi program dengan menjalankan Setup ‘Potato’. Ikuti instruksi yang diberikan (hanya memilih direktori tempat penyimpanan aplikasi ‘Potato’).
2. Setelah berhasil diinstal, aplikasi ‘Potato’ bisa diakses melalui *shortcut icon* yang ada di *Desktop*. Aplikasi ini bisa dijalankan dengan cara biasa (*double click shortcut* tersebut).
3. Tampilan awal ‘Potato’ akan muncul. Untuk memulai eksekusi program, tekan *button Choose a File* dan pilih berkas graf yang ingin dimasukkan.
4. Perhatikan bahwa nama berkas yang dimasukkan bisa dicek pada bagian atas aplikasi. Program juga akan menampilkan visualisasi graf secara langsung. Graf ini bisa diubah-ubah dan dikembalikan lagi seperti semula.
5. Untuk menggunakan fitur-fitur aplikasi, pastikan telah memasukkan input yang dibutuhkan oleh program untuk mengeksekusi fungsionalitas tersebut. Jika tidak, pengguna akan diarahkan oleh aplikasi untuk memasukkan input yang benar.
6. Untuk *button* yang tidak memiliki keterangan, bisa dilakukan *hovering* di *button* tersebut dan akan muncul sebuah *tooltip* untuk menjelaskan kegunaan tombol tersebut.
7. Untuk fitur Rekomendasi Teman, hanya ada satu *field* yang perlu diberikan, yaitu target rekomendasi. Jika *field* ini sudah diisi, maka fitur ini bisa langsung dieksekusi dan aplikasi akan memberikan hasil sesuai dengan kondisi graf masukan.
8. Untuk fitur Eksplorasi Teman, pengguna perlu memilih simpul asal (dalam hal ini, adalah target rekomendasi), simpul tujuan, dan pilihan metode traversal. Jika sudah diberikan dengan benar, maka program akan menghasilkan keluaran yang sesuai dengan kondisi graf masukan.

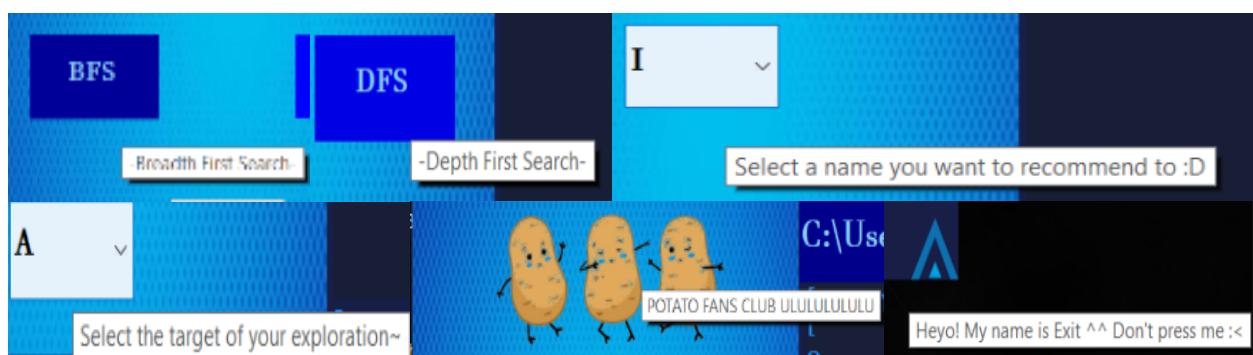
9. Program juga akan menghasilkan visualisasi graf dengan melakukan *highlighting* jalur eksplorasi teman yang dihasilkan. Jalur ini ditandai dengan warna yang berbeda dengan warna graf pada kondisi awal.
10. Aplikasi tidak memiliki *border* seperti pada aplikasi windows pada umumnya sehingga aplikasi tidak bisa di-resize maupun di-minimize. Untuk keluar dari aplikasi, telah disediakan tombol yang bisa melakukan fungsionalitas tersebut pada sudut aplikasi.

4.4 Hasil Pengujian

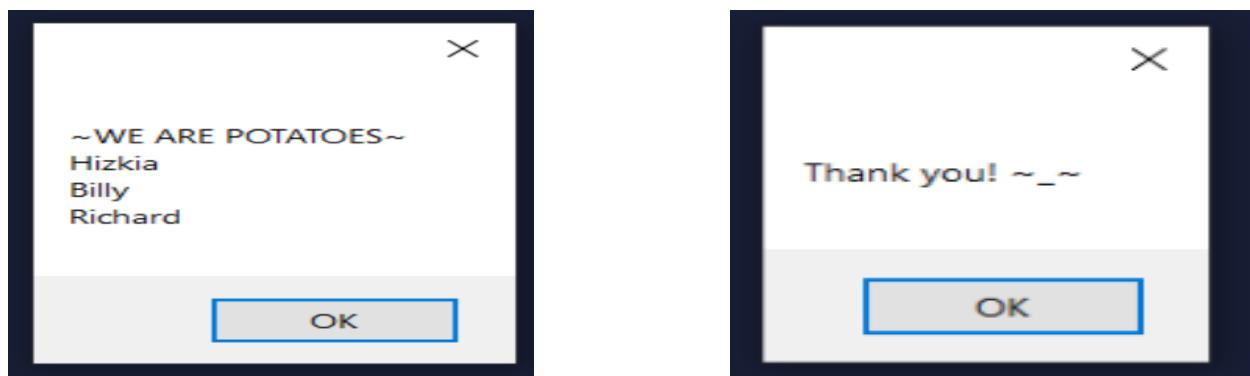
Berikut merupakan gambar tangkapan layar dari aplikasi yang kami buat dengan nama ‘Potato’.



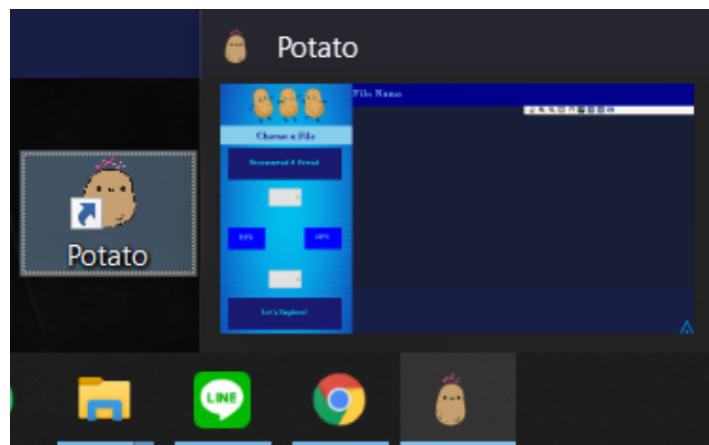
Gambar 1. Tampilan Awal Aplikasi Potato



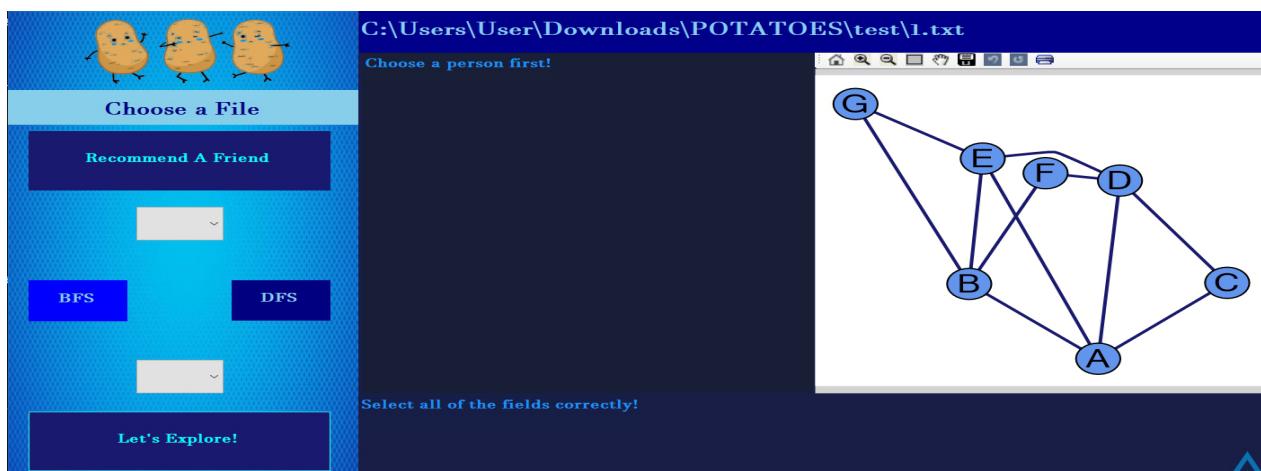
Gambar 2. Beberapa *Tooltips* dari Objek Aplikasi Saat *Mouse Hovering*



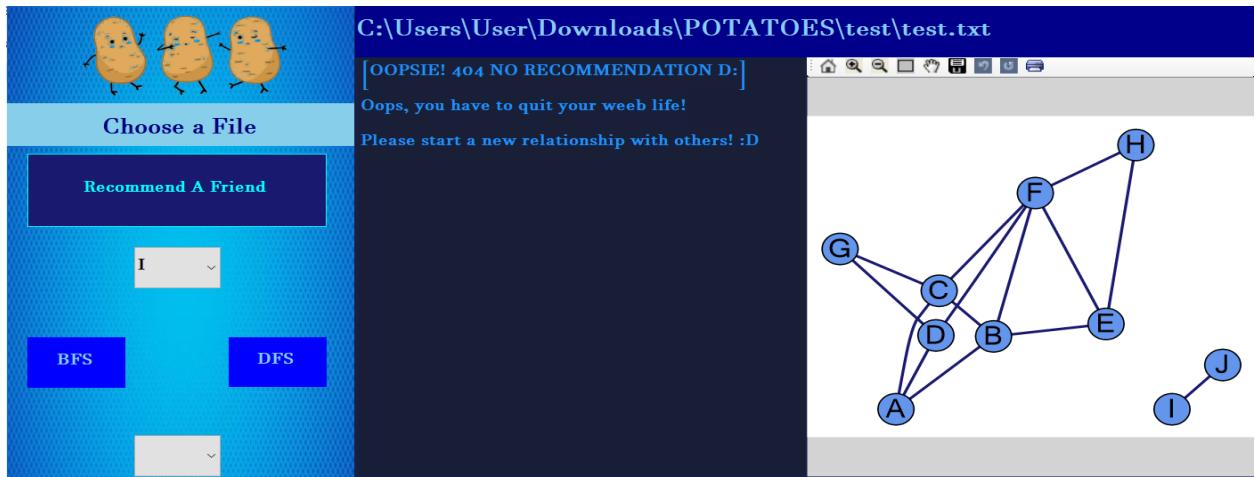
Gambar 3. *Message Box* saat Menekan Logo Aplikasi dan Tombol *Exit*



Gambar 4. Tampilan *Icon* Aplikasi di *Taskbar* dan *Desktop*



Gambar 5. Tampilan Aplikasi saat Input *Node* Belum Lengkap



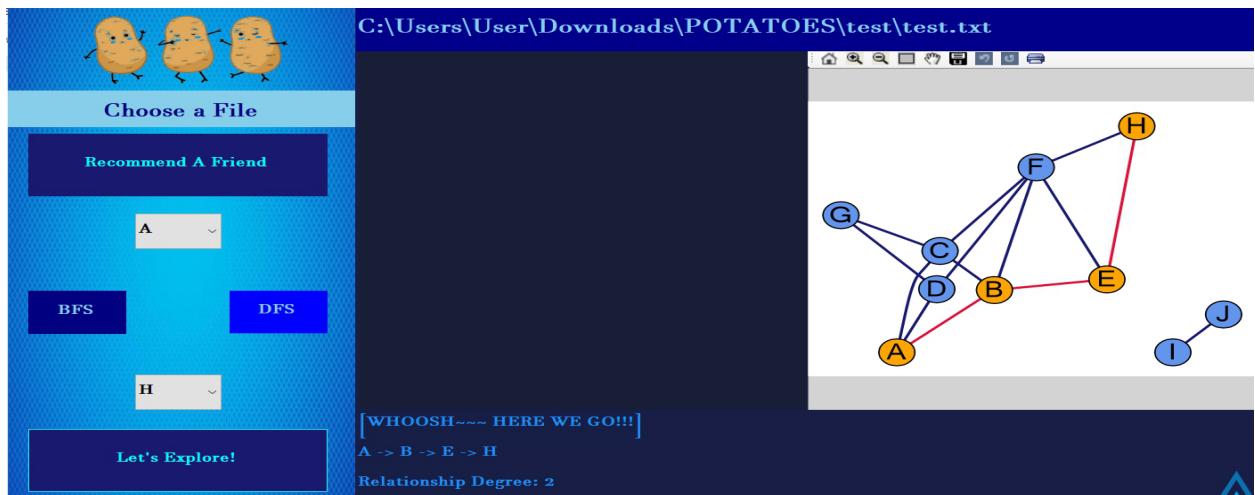
Gambar 6. Tampilan Aplikasi saat Tidak Ada Rekomendasi Teman



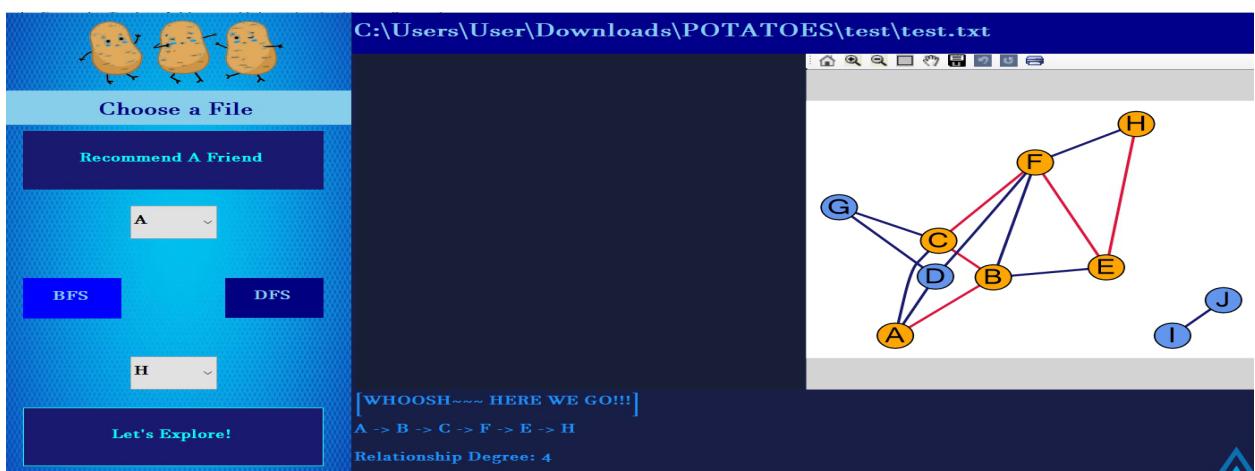
Gambar 7. Tampilan Aplikasi saat Metode Traversal Belum Dipilih



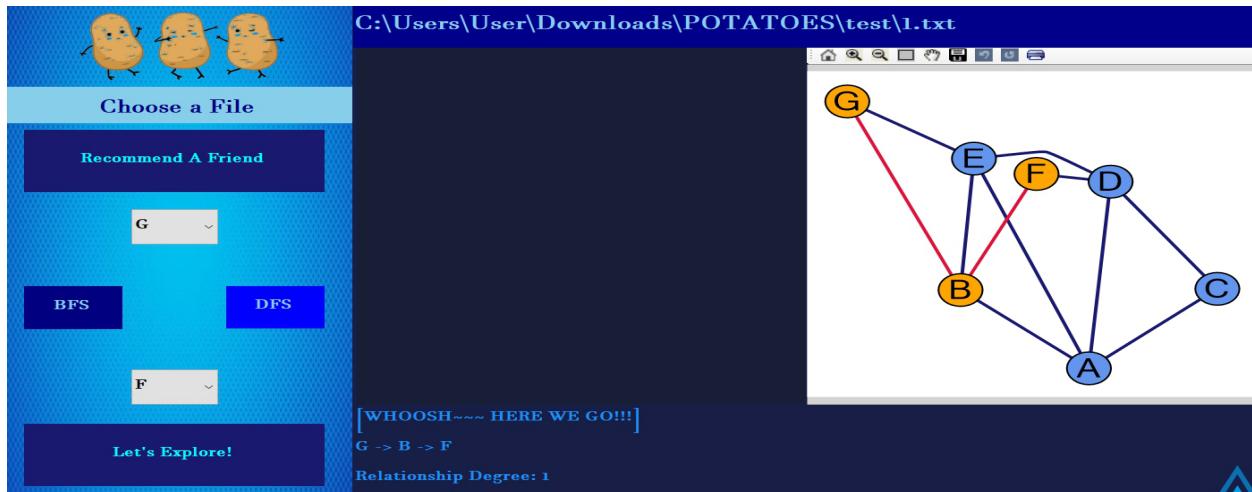
Gambar 8. Tampilan Aplikasi saat Memilih *Node* yang Sama



Gambar 9. Tampilan Aplikasi saat Eksplorasi Teman dengan BFS



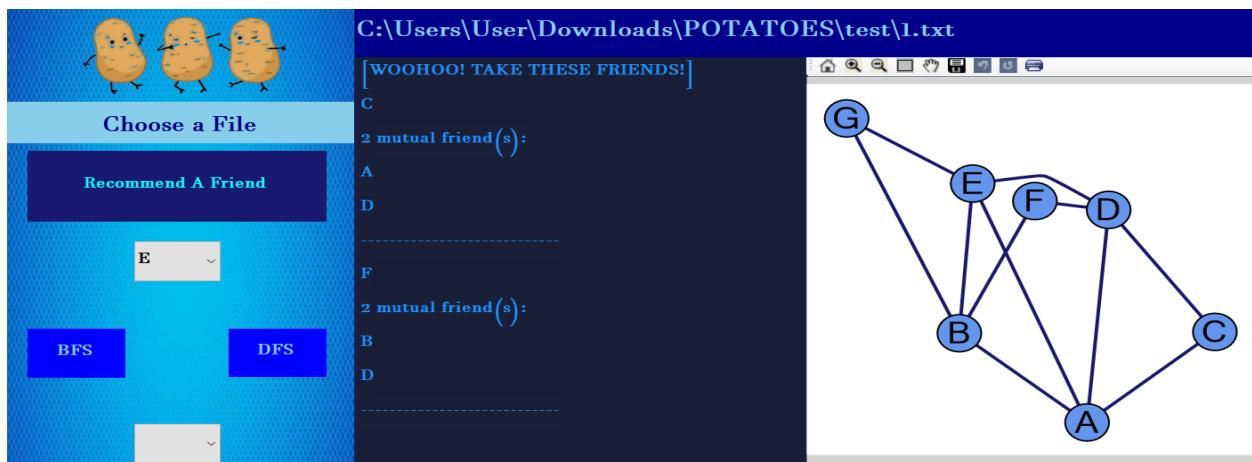
Gambar 10. Tampilan Aplikasi saat Eksplorasi Teman dengan DFS



Gambar 11. Tampilan Graf dan Eksplorasi Teman dengan BFS untuk *Testcase Lain*



Gambar 12. Tampilan Graf dan Eksplorasi Teman dengan DFS untuk *Testcase Lain*



Gambar 13. Tampilan Aplikasi untuk Fitur Rekomendasi Teman

4.5 Analisis Hasil Pengujian

Folder aplikasi sudah dilengkapi dengan sebuah *installer* dengan nama Potato.msi yang bisa digunakan untuk memasang aplikasi pada sistem operasi Windows. Dengan mengikuti prosedur instalasi yang ada, maka pengguna bisa dengan mudah memilih direktori instalasi dan memasang aplikasi Potato. Setelah itu, aplikasi Potato akan langsung hadir di layar *desktop* pengguna.

Ketika ingin memasukkan berkas eksternal melalui tombol *Choose a File*, pengguna akan diarahkan kepada *File Explorer*, dan tampilan ini akan melakukan *filtering* pada file yang berformat .txt saja. Pengguna bisa memasukkan berkas ini dengan cara yang biasa dilakukan ketika memasukkan berkas, misalnya dengan *double click* berkas yang diinginkan. Ketika sudah dipilih, maka aplikasi akan secara otomatis memvisualisasikan graf dari berkas tersebut.

Dari gambar-gambar di atas, dapat terlihat bahwa semua kasus sudah berhasil ditangani oleh aplikasi Potato. Kasus-kasus lain selain kasus normal untuk fitur Rekomendasi Teman adalah ketika belum ada input yang dipilih, atau ketika tidak ada rekomendasi yang bisa diberikan oleh sistem. Untuk fitur Eksplorasi Teman, sudah ditangani kasus lain berupa belum dipilihnya metode traversal, input yang tidak lengkap, dan tidak ditemukannya *path* atau jalur dari simpul asal menuju simpul tujuan.

Selain itu, sebagai bentuk kreativitas supaya aplikasi terlihat lebih menarik, aplikasi ini sudah dilengkapi dengan beberapa tampilan, misalnya pada *icon* aplikasi pada *desktop* dan *taskbar* ketika aplikasi dijalankan. Adapun di dalam aplikasi juga dimasukkan gambar yang bisa merepresentasikan pengembang. Selain itu, juga telah diberikan keterangan-keterangan untuk setiap objek yang ada di dalam aplikasi dalam bentuk *tooltips*.

Secara umum, aplikasi dibangun dengan menggabungkan program biasa berupa fitur-fitur pada aplikasi, yaitu Rekomendasi dan Eksplorasi Teman (termasuk mengubah *file* menjadi graf), dengan objek-objek aplikasi pada *Windows Form App* dengan .NET Framework. Beberapa objek yang digunakan di dalam aplikasi adalah *List Box*, *Button*, *Picture Box*, dan *Rich Text Box*. Selain itu, juga digunakan sebuah panel yang telah di-*dock* di kiri layar. Untuk visualisasi graf digunakan pustaka MSAGL dengan menggunakan daftar graf pada graf awal (graf dari berkas eksternal) dan graf jalur eksplorasi dalam melakukan *highlight nodes* serta *edges* untuk fitur Eksplorasi Teman.

Dari contoh kasus digunakan, beberapa perbedaan solusi dapat terlihat tergantung dari metode mana yang dipilih untuk menelusuri graf. Sebagai contoh, pada kasus menelusuri graf dengan metode DFS dan BFS untuk contoh kasus *node* A dan H menghasilkan solusi yang berbeda untuk kedua metode tersebut. Berdasar kasus ini, metode BFS akan lebih baik untuk dipakai jika *node* yang dituju berhubungan dekat dengan *node* asal. Sebaliknya, metode DFS akan lebih baik jika digunakan untuk mencari hubungan antara 2 *node* yang terhubung relatif jauh di dalam graf.

BAB V PENUTUP

5.1 Kesimpulan

Dari tugas besar kedua dari mata kuliah Strategi Algoritma ini, dapat disimpulkan bahwa implementasi *graph traversal* sangatlah luas, yang mungkin tidak disadari, seperti fitur *Friend Recommendation* dan *Explore Friend* pada beberapa media sosial yang terkenal. Algoritma *BFS* dan *DFS* memiliki pendekatan yang berbeda dan memberikan hasil yang berbeda pula, bergantung pada kasus yang sedang diselidiki. Seperti yang dapat dilihat pada laporan ini, algoritma yang dirancang berhasil mengimplementasikan algoritma *BFS* dan *DFS* untuk memberikan fitur *Friend Recommendation* dan *Explore Friend* dengan baik. Luaran yang dihasilkan juga sesuai dengan ekspektasi dan memenuhi teori algoritma *BFS* dan *DFS*.

5.2 Saran

Baik algoritma maupun program yang dikembangkan pada kesempatan kali ini masih jauh dari kata sempurna. Saran yang dapat diberikan adalah pengembangan algoritma dan *GUI* dari program agar lebih efisien dan menarik untuk pengguna program. Algoritma dapat diperbaiki dengan mengurangi memori yang dipakai, seperti penyederhanaan struktur data yang dipakai dan meminimalisir pembuatan variabel maupun struktur data baru. Untuk *GUI* dapat dibuat menjadi lebih menarik dengan memberikan tampilan yang lebih menarik, seperti melakukan desain *background image* maupun penggunaan warna.

5.3 Refleksi

Dalam proses pembuatan program ini, kendala paling besar yang menghambat adalah kesibukan ketiga pengembang program ini. Dengan banyaknya tugas ataupun beban kerja lain, pengembang program tidak dapat memaksimalkan program yang dihasilkan, walaupun luaran yang diharapkan telah sesuai spesifikasi yang diberikan. Ditambah dengan kurangnya pengalaman dengan bahasa pemrograman yang baru, pengembangan program menjadi memakan waktu yang lebih lama dari seharusnya. Akan tetapi, para pengembang sudah memberikan usaha yang maksimal agar program dapat dijalankan dan digunakan dengan baik oleh pengguna dan efisien secara algoritma.

5.4 Komentar

Tugas besar kedua dari mata kuliah ini memberikan praktik latihan yang sangat bagus, baik dari penggunaan dan implementasi algoritma yang baru, sampai pencarian dan pembelajaran bahasa pemrograman yang relatif baru. Dengan ini, para pengembang program mengucapkan terima kasih kepada dosen-dosen mata kuliah Strategi Algoritma yang telah merancang dan memberikan tugas-tugas yang telah ada dan akan ada di mata kuliah ini. Walaupun sedikit sukar dan menantang, para penulis dan pengembang program menikmati waktu yang dihabiskan dalam proses pembuatan program ini.

BAB VI DAFTAR PUSTAKA

<https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet>, diakses pada 21 Maret 2021 pukul 00.13 WIB.

<https://drive.google.com/file/d/1HrTJ92aTjCHTZDbdOtlVzz7dGoEU9WK9/view>, diakses pada 18 Maret 2021 pukul 17:45 WIB.

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>, diakses pada 18 Maret 2021 pukul 17:20 WIB.

<https://www.codecademy.com/articles/what-is-net>, diakses pada 21 Maret 2021 pukul 00.00 WIB.

<https://www.tutorialspoint.com/graphs-and-its-traversal-algorithms>, diakses pada 18 Maret 2021 pukul 17:46 WIB.

<https://www.w3schools.com/cs/default.asp>, diakses 21 Maret 2021 pukul 01.10 WIB.