

TUGAS BESAR 2
IF2211 STRATEGI ALGORITMA
Pengaplikasian Algoritma BFS dan DFS dalam Fitur People You May Know
Jejaring Sosial Facebook

IF2211 Strategi Algoritma
Semester II Tahun Ajaran 2020/2021



Disusun oleh:

Thomas Ferdinand Martin	13519099
Muhammad Akram Al Bari	13519142
Azmi Muhammad Syazwana	13519151

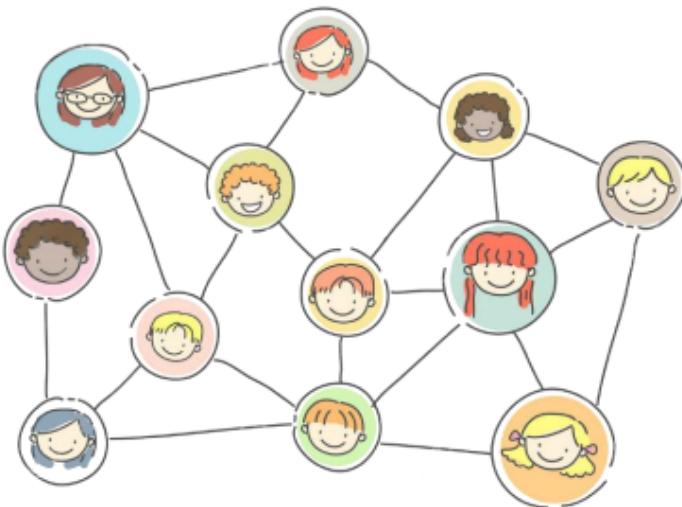
TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2021

BAB 1

Deskripsi Tugas

Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan beberapa fitur dari People You May Know dalam jejaring sosial media (Social Network). Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), Anda dapat menelusuri social network pada akun facebook untuk mendapatkan rekomendasi teman seperti pada fitur People You May Know. Selain untuk mendapatkan rekomendasi teman, Anda juga diminta untuk mengembangkan fitur lain agar dua akun yang belum berteman dan tidak memiliki mutual friends sama sekali bisa berkenalan melalui jalur tertentu.



Gambar 1. Ilustrasi graf pertemanan pada social network Facebook

(Sumber: <https://www.freecodecamp.org/news/deep-dive-into-graph-traversals-227a90c6a261/>)

Facebook sebagai salah satu pelopor jejaring sosial sejak tahun 2004 kini telah menembus angka 2,6 miliar pengguna. Fitur friend recommendation menjadi sangat penting karena banyaknya jumlah pengguna Facebook. Fitur bernama "People You May Know" ini dapat memberikan pengguna rekomendasi teman yang sebaiknya di-add, misalnya teman sekolah, teman kuliah, mantan pacar, atau orang yang kita kenal lewat suatu kegiatan tertentu. Faktor utama saran pertemanan melalui fitur tersebut adalah berdasarkan mutual friend yang dimiliki oleh kedua akun pengguna. Misalnya pengguna A dan C belum berteman di facebook, tetapi keduanya berteman dengan pengguna B, berarti A dan C memiliki mutual friend yang sama, yaitu B. Semakin banyak mutual friend yang dimiliki antar kedua akun, maka semakin tinggi rekomendasi akun tersebut untuk di-add.

Selain itu, di setiap social media, termasuk Facebook, pengguna dapat mengeksplorasi akun pengguna lainnya yang tidak memiliki mutual friends sama sekali. Akan tetapi, dengan menelusuri graf pertemanan antar akun sehingga kita dapat mengetahui 'jarak' antar akun agar bisa saling terhubung dan berteman.

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.

Aplikasi yang akan dibangun dibuat berbasis GUI. Berikut ini adalah contoh tampilan dari aplikasi GUI yang akan 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/1XhFSpHU028Gaf7YxkmdbluLkQgVI3MY6gt1tPL30LA/edit?usp=sharing>
- 4) Terdapat dua fitur utama, yaitu:
 - A. Fitur friend recommendation

- a. 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.
 - b. 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
- a. 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).
 - b. Program menerima pilihan dua akun yang belum berteman.
 - c. Program akan menampilkan nilai N-th degree connection antar kedua akun dan memberikan jalur melalui akun mana saja sampai kedua aku bisa terhubung.
 - d. 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 2

LANDASAN TEORI

2.1 Dasar Teori

2.1.1 Pengertian Graf Traversal

Graf traversal berarti memproses (mengunjungi) simpul dari suatu graf secara sistematis, dalam beberapa urutan tertentu (berdasarkan topologi graf). Algoritma penjelajahan graf biasanya dimulai dengan simpul awal dan dari simpul awal dilanjutkan dengan simpul yang terhubung dengannya. Masalah dengan traversal grafik adalah tidak semua simpul pada graf dapat dijangkau dari simpul awal dan grafik mungkin berisi siklus yang dapat menyebabkan putaran tak terbatas Kedua masalah ini dapat diatasi dengan menandai node yang sudah dikunjungi dan menguji daftar penanda setiap kali node akan dikunjungi. Jika setiap simpul dalam graf akan dilintasi oleh algoritma berbasis pohon (seperti DFS atau BFS), algoritma tersebut harus dipanggil setidaknya satu kali untuk setiap simpul graf yang terhubung. Ini mudah dilakukan dengan melakukan iterasi melalui semua simpul pada graf. Dikenal dua buah algoritma traversal, kedua traversal ini algoritma adalah **Depth-first Search** (DFS) dan **Breadth-first Search** (BFS). Algoritma ini telah terbukti sangat berguna untuk banyak aplikasi yang melibatkan graf kecerdasan buatan dan penelitian operasi. Selain itu, mereka sangat diperlukan untuk penyelidikan yang efisien atas properti dasar grafik seperti konektivitas dan keberadaan siklus.

2.1.2 Pengertian Breadth-first Search

Breadth-first search (BFS) adalah algoritma traversal pada graf. Alur pencarian dimulai di suatu simpul dan mengeksplorasi semua simpul tetangga pada kedalaman saat ini sebelum pindah ke simpul berikutnya tingkat kedalaman. Tiap simpul hidup dimasukan ke dalam sebuah queue untuk mempermudah pencarian. Misal Traversal dimulai dari simpul v.

Algoritma:

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.

Struktur data:

1. Matriks ketetanggaan $A = [a_{ij}]$ yang berukuran $n \times n$,
 $a_{ij} = 1$, jika simpul i dan simpul j bertetangga,
 $a_{ij} = 0$, jika simpul i dan simpul j tidak bertetangga.
2. Antrian q untuk menyimpan simpul yang telah dikunjungi.
3. Tabel Boolean, diberi nama “dikunjungi”

dikunjungi : array[1..n] of boolean

$dikunjungi[i] = \text{true}$ jika simpul i sudah dikunjungi

$dikunjungi[i] = \text{false}$ jika simpul i belum dikunjungi

2.1.3 Pengertian Depth-first Search

Depth-first search adalah salah satu algoritma traversal pada graf. Algoritma dimulai pada node root dan mencari sejauh mungkin di setiap simpul sebelum melakukan backtracking. Misal traversal dimulai dari simpul v.

Algoritma:

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.

2.2 C# Desktop Application Development

C# atau yang dibaca C sharp adalah bahasa pemrograman dapat digunakan untuk berbagai fungsi misalnya untuk pemrograman server-side pada website, membangun aplikasi desktop ataupun mobile, pemrograman game dan sebagainya. Selain itu C# juga bahasa pemrograman yang berorientasi objek, jadi C# juga mengusung konsep objek seperti inheritance, class, polymorphism dan encapsulation.

C# sangat bergantung dengan framework yang disebut .NET Framework, framework inilah yang nanti digunakan untuk mengcompile dan menjalankan kode C#. C# dikembangkan oleh Microsoft dengan merekrut Anders Helsing. Tujuan dibangunnya C# adalah sebagai bahasa pemrograman utama dalam lingkungan .NET Framework (lihat C#). Banyak pihak juga yang menganggap bahwa Java dengan C# saling bersaing, bahkan ada juga yang menyatakan jika pernah belajar Java maka belajar C# akan sangat mudah dan begitu juga sebaliknya.

Dalam pengembangan aplikasi kali ini, kami memanfaatkan Windows Form App. Windows Form App adalah framework UI untuk membangun aplikasi desktop Windows. WFA menyediakan salah satu cara paling produktif untuk membuat aplikasi desktop berdasarkan perancangan visual yang disediakan di Visual Studio. Fungsionalitas seperti penempatan drag-and-drop kontrol visual memudahkan pembuatan aplikasi desktop. Dengan WFA, dapat dikembangkan aplikasi berbasis UI yang mudah digunakan, diperbarui, dan berfungsi saat offline atau saat tersambung ke internet. WFA dapat mengakses perangkat keras lokal dan sistem file komputer tempat aplikasi berjalan.

BAB 3

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah pemecahan masalah

Pada contoh kasus tugas besar kali ini, diberikan permasalahan berupa graph yang hendak ditelusuri hubungan-hubungan tiap node-nya. Kasus pada tugas kali ini merupakan representasi dari kasus yang terdapat pada dunia nyata, yakni pada sebuah situs jejaring sosial. Pada jejaring sosial, setiap user memiliki “teman”, dan setiap user yang memiliki “teman” dapat direpresentasikan sebagai graph. Persoalan yang diminta untuk diselesaikan, adalah bagaimana caranya supaya kita bisa memberikan rekomendasi “teman baru” kepada user tertentu, serta mencari “jalur pertemuan” yang mungkin dibuat untuk menghubungkan user satu dengan lainnya yang statusnya belum “berteman”.

Untuk menyelesaikan persoalan yang diminta, digunakan algoritma *Breadth First Search (BFS)* dan algoritma *Depth First Search (DFS)*. Di mana, kedua algoritma ini adalah algoritma yang digunakan untuk melakukan penelusuran terhadap suatu graph. Untuk permasalahan rekomendasi, dengan menggunakan DFS ataupun BFS, mula-mula kita memilih suatu node asal (user yang ingin dicari rekomendasi temannya) dan dengan menggunakan algoritma penelusuran graph, kita menelusuri satu per satu node “tetangga” dari user tersebut. Hal ini dilakukan terus menerus sampai kita mendapatkan suatu node yang *depthnya* bernilai 2 dihitung dari node asal. Semua node (user) yang memiliki *depth* 2 inilah

yang merupakan rekomendasi teman bagi user. Dalam menyelesaikan persoalan “jalur pertemanan”, mula-mula ditetapkan terlebih dahulu node asal (user1) dan node tujuan (user2), kemudian graph akan ditelusuri seluruh hubungan nodenya dengan menggunakan algoritma DFS maupun BFS sampai node tujuan berhasil ditemukan atau sampai seluruh node pada graph selesai ditelusuri.

3.2 Mapping Persoalan Menjadi Algoritma BFS dan DFS.

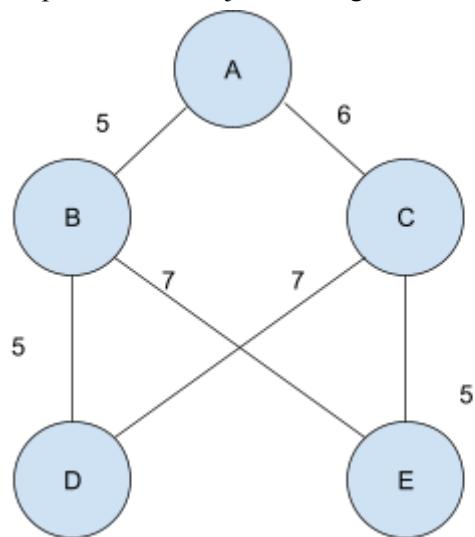
Pada persoalan kali ini, mapping yang dilakukan adalah dengan merepresentasikan setiap *user* yang ada menjadi sebuah node yang direpresentasikan dengan karakter alfabet. Kemudian, disimpan pula *user-user* yang saling berhubungan sebagai suatu pasangan node. Sekumpulan pasangan node inilah yang nantinya akan membentuk graph untuk diselesaikan dengan algoritma BFS dan DFS.

Mapping Persoalan:

- Node: User
- Pasangan Node: Teman user
- Graph: Gabungan seluruh pasangan node

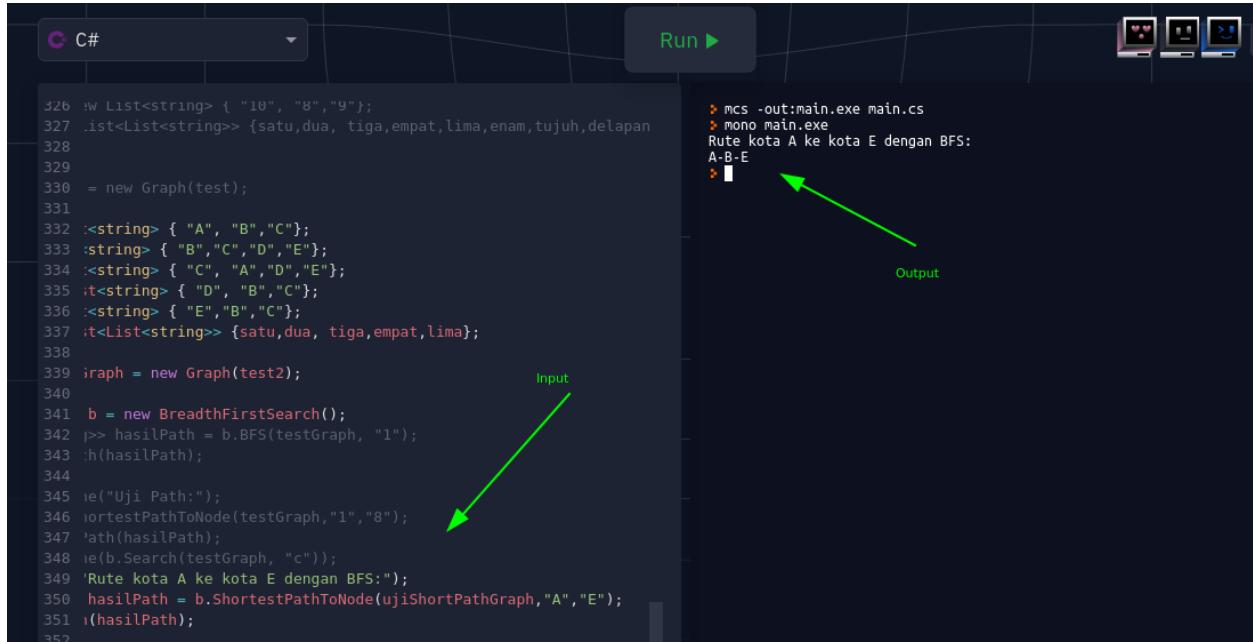
3.3 Ilustrasi Kasus Lain

Pada dasarnya, algoritma BFS dan DFS adalah algoritma penelusuran. Sebagai ilustrasi kasus lain, kelompok kami mengambil ilustrasi sederhana yakni kasus pemilihan rute dari suatu kota ke kota lain. Misalnya saja, terdapat 5 kota yakni: A, B, C , D, dan E. Persoalannya adalah, bagaimana rute untuk sampai ke kota E jika berangkat dari kota A. Sebagai ilustrasi, diketahui hubungan jalur setiap kota:



Dengan menggunakan algoritma BFS, akan didapat rute: A-B-E dengan nilai total: $5+7 = 12$. Sedangkan algoritma DFS akan memberikan rute A-B-D-C-E dengan nilai total: $5+5+7+5 = 22$. Seperti yang dapat dilihat, pada program yang kami buat, terdapat perbedaan rute yang dipilih oleh setiap algoritma. Hal ini disebabkan karena pendekatan yang dilakukan oleh DFS dan BFS sedikit berbeda. Di mana, DFS terus berusaha mencari node yang lebih dalam pada setiap penelusurannya, sedangkan BFS menelusuri setiap node pada suatu *depth* tertentu sampai selesai. Apabila kasus yang diminta adalah untuk mendapatkan

rute terpendek menuju, maka kedua algoritma ini sama-sama tidak bisa digunakan untuk memecahkan persoalan. Terdapat Algoritma *Djikstra* yang lebih cocok untuk menyelesaikan persoalan yang seperti itu.



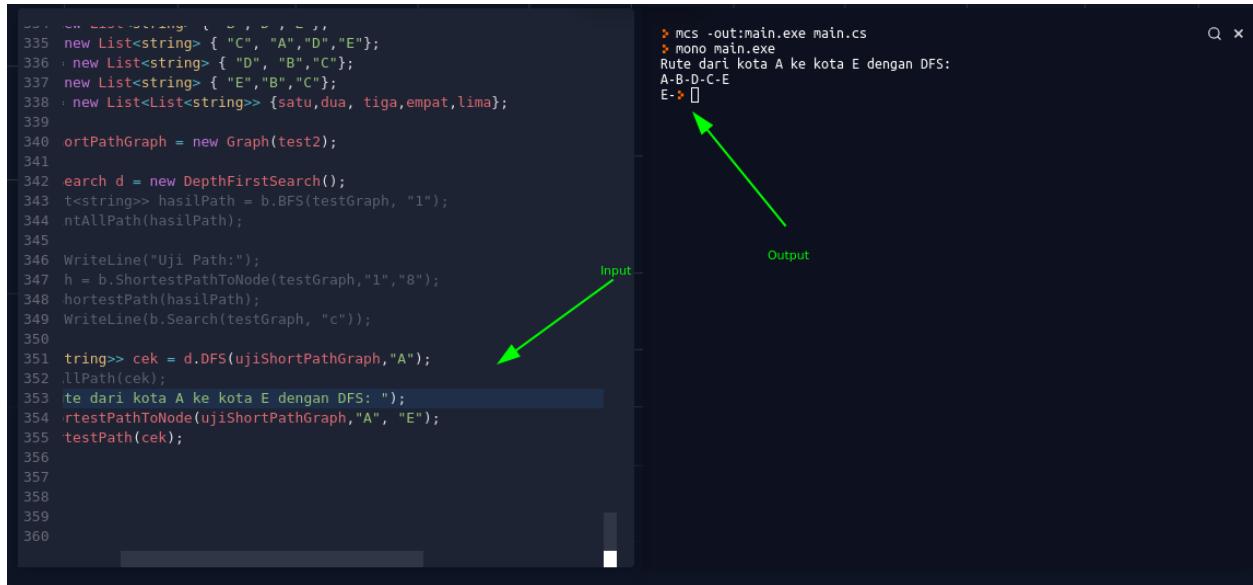
The screenshot shows a C# development environment. On the left, the code editor displays a portion of a C# program. A green arrow points from the text "Input" to the line of code "342 t>> hasilPath = b.BFS(testGraph, "1");". Another green arrow points from the text "Output" to the terminal window on the right, which shows the command "mono main.exe" followed by the output "Rute kota A ke kota E dengan BFS: A-B-E".

```

326     List<string> { "10", "8","9"};
327     .list<List<string>> {satu,dua, tiga,empat,lima,enam,tujuh,delapan
328
329
330     = new Graph(test);
331
332     <string> { "A", "B","C"};
333     <string> { "B","C","D","E"};
334     <string> { "C", "A","D","E"};
335     <string> { "D", "B","C"};
336     <string> { "E","B","C"};
337     <List<string>> {satu,dua, tiga,empat,lima};
338
339     graph = new Graph(test2);
340
341     b = new BreadthFirstSearch();
342     t>> hasilPath = b.BFS(testGraph, "1");
343     h(hasilPath);
344
345     ie("Uji Path:");
346     iortestPathToNode(testGraph,"1","8");
347     ath(hasilPath);
348     ie(b.Search(testGraph, "c"));
349     'Rute kota A ke kota E dengan BFS:';
350     hasilPath = b.ShortestPathToNode(ujiShortPathGraph,"A","E");
351     i(hasilPath);
352

```

rute yang didapat dengan algoritma BFS



The screenshot shows a C# development environment. On the left, the code editor displays a portion of a C# program. A green arrow points from the text "Input" to the line of code "342 t>> hasilPath = b.BFS(testGraph, "1");". Another green arrow points from the text "Output" to the terminal window on the right, which shows the command "mono main.exe" followed by the output "Rute dari kota A ke kota E dengan DFS: A-B-D-C-E E->[]".

```

335     new List<string> { "C", "A","D","E"};
336     new List<string> { "D", "B","C"};
337     new List<string> { "E","B","C"};
338     new List<List<string>> {satu,dua, tiga,empat,lima};
339
340     ortPathGraph = new Graph(test2);
341
342     search d = new DepthFirstSearch();
343     t>> hasilPath = b.BFS(testGraph, "1");
344     ntAllPath(hasilPath);
345
346     WriteLine("Uji Path:");
347     h = b.ShortestPathToNode(testGraph,"1","8");
348     hortestPath(hasilPath);
349     WriteLine(b.Search(testGraph, "c"));
350
351     tring>> cek = d.DFS(ujiShortPathGraph,"A");
352     llPath(cek);
353     te dari kota A ke kota E dengan DFS: ";
354     rtestPathToNode(ujiShortPathGraph,"A", "E");
355     testPath(cek);
356
357
358
359
360

```

rute yang didapat dengan algoritma DFS

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

Graph.cs

Procedure readFromFile (input text: string, output adj: ListOfAdj, output node: ListOfNode)

```
i ← 0
for (character in text)
    if (i = 0)
        nNodes ← character
    else
        addNode(character[0])
        addNode(character[1])
        addAdj(character[0], character[1])
        addListOfEdge(character[0], character[1])
    i++
```

function addNode (input node : string) → boolean

```
if (not isExist(node))
    nodes.Add(node)
    → true
→ false
```

Procedure addAdj (input node : string, input adjNode : string, output adj : ListOfAdj)

```
isNodeExist ← addNode(node)
isadjNodeExist ← addNode(adjNode)

if (not isNodeExist)
```

```
newNodes ← Empty ListOfNode
```

```
adj.Add(newNodes)
```

```
if (not isadjNodeExist)
```

```
newNodes ← Empty ListOfNode
```

```
adj.Add(newNodes)
```

```
i traversal[0..nodes.Count]
```

```
if (nodes[i] = node)
```

```
adj[i].Add(adjNode)
```

```
adj[i].Sort
```

```
if (nodes[i] = adjNode)
```

```
adj[i].Add(node);
```

```
adj[i].Sort
```

```
function isExist(input node: string) → boolean
```

```
if (node is in ListOfNodes)
```

```
→ true
```

```
→ false
```

```
function friendsOfAccount(input acc : string) → ListOfString
```

```
foreach (node in nodes)
```

```
if (node == acc)
```

```
foreach (string friend in adj[i])
```

```
if (friend != acc)
```

```
friends.Add(friend);
```

```
→ friends
```

```

function sortedMutual(input acc: string) → ListOfString
    listAdj(node)
    foreach(adj in listAdj)
        if (node in adj and notadj(node, acc))
            mutuals.Add(node)
    mutuals.Sort based on count mutual
    → mutuals

function notAdj(input node1: string, input node2: string) → boolean
    if (node1 adj with node2)
        → true

function BFS(input g: graph, input node: string) → ListOfString
    dikunjungi(node) ← true
    MasukAntrian(q,node)
    Hasil.Add(node)
    while not AntrianKosong(q) do
        HapusAntrian(q,v)
        for tiap simpul w yang bertetangga dengan simpul v do
            if not dikunjungi (w) then
                Hasil.Add(w)
                MasukAntrian(q,w)
                dikunjungi(w) ← true
            endif
        endfor
    endwhile
    → Hasil

```

```

function DFS(input g: graph, input node: string) → ListOfString

dikunjungi(node) ← true

Hasil.Add(node)

for w ← 1 to n do

    if A(v,w)=1 then

        if not dikunjungi (w) then

            Hasil.Add(w)

            DFS(w)

        endif

    endif

endfor

→ Hasil

```

4.2 Penjelasan Struktur Data

Struktur data Graph:

A. Atribut

1. private List<List<string>> adj;// adj[i] merupakan kumpulan simpul tetangga nodes[i]
2. private List<string> nodes; // nodes berisi kumpulan simpul pada graf
3. public List<List<string>> adj2 { get; set; } // adj versi 2 berisi simpul dan sisi2nya, mempermudah DFS dan BFS
4. private int nNodes;
5. private List<(string, string)> listOfEdge; // list dari sisi pada graf untuk mempermudah visualisasi

B. Method

1. Graph() // konstruktor
2. public void readFromFile(string[] text) // prosedur pembacaan file
3. public bool addNode(string node) // prosedur menambahkan node
4. public List<(string, string)> getListOfEdge() // Getter listOfEdge
5. public void addAdj(string node, string adjNode) // Menambahkan sisi ke dalam graf disertai sorting untuk mempermudah algoritma BFS dan DFS
6. bool isExist(string node) // Mengembalikan true apabila ada Node node dalam graf
7. public List<string> getNode() //Mengembalikan daftar simpul pada graf
8. public List<List<string>> getAdjNode() //Mengembalikan daftar sisi pada graf

9. public List<string> friendsOfAccount(string acc) //Mengembalikan list simpul tetangga dari node a
10. public List<List<string>> sortedMutual(string acc) //Menghitung mutual friend, jalur diperoleh dari BFS versi singkat
11. public bool notAdj(string node1, string node2) //Cek apakah node1 bertetangga dengan node2 based on list node1

C. Method DFS

1. public string Search(Graph graphInput, string nameToSearchFor) // Method yang akan mengembalikan node apabila suatu node ditemukan pada graph
2. public int SearchNodeIndex(Graph graphInput, string nodeToSearch) // Method yang mengembalikan letak index node pada graph
3. public List<List<string>> DFS(Graph graphInput, string startingNode) //Method utama untuk menelusuri graph dengan algoritma DFS
4. public List<List<string>> ShortestPathToNode(Graph graphInput, string startingNode, string endNode) //Algoritma untuk menemukan rute dari suatu node ke node lainnya dengan DFS
5. public bool cekAdaBug(List<List<string>> container, List<string> bug) //Method antara yang digunakan untuk mengecek apakah terdapat bug pada hasil penelusuran DFS
6. public void PrintAllPath(List<List<string>> AllPath) //Method untuk mengeluarkan hasil penelusuran DFS ke layar; digunakan untuk testing
7. public void PrintShortestPath(List<List<string>> ShortestPath) //Method untuk mengeluarkan hasil penelusuran DFS ke layar; digunakan untuk testing

D. Method BFS

1. public string Search(Graph graphInput, string nameToSearchFor) //Method yang menerima node berupa string dan juga graph. Akan dicari apakah node ada di dalam graph.
2. public int SearchNodeIndex(Graph graphInput, string nodeToSearch) //Method yang menerima graph dan juga sebuah node bertipe string. Mengembalikan index node pada graph
3. public List<List<string>> BFS(Graph graphInput, string startingNode) //Algoritma utama untuk melakukan penelusuran graph dengan BFS
4. public List<List<string>> ShortestPathToNode(Graph graphInput, string startingNode, string endNode) //Method untuk menemukan jarak dari suatu node asal ke node tujuan dengan metode BFS
5. public void PrintAllPath(List<List<string>> AllPath) //Method untuk melakukan print hasil BFS ke console; digunakan hanya untuk keperluan testing
6. public void PrintShortestPath(List<List<string>> ShortestPath) //Method untuk melakukan print hasil BFS ke console; digunakan hanya untuk keperluan testing

Struktur data Form:

A. Atribut

1. Graph undirectedGraph // menyimpan graph yang akan digunakan
2. Bool isDFS // true apabila saat itu DFS
3. Bool isBFS // true apabila saat itu BFS

4. String account // menyimpan string akun utama
5. String friends_with // menyimpan string akun lain
6. Microsoft.Msagl.Drawing.Graph visualGraph // visualizer

B. Method

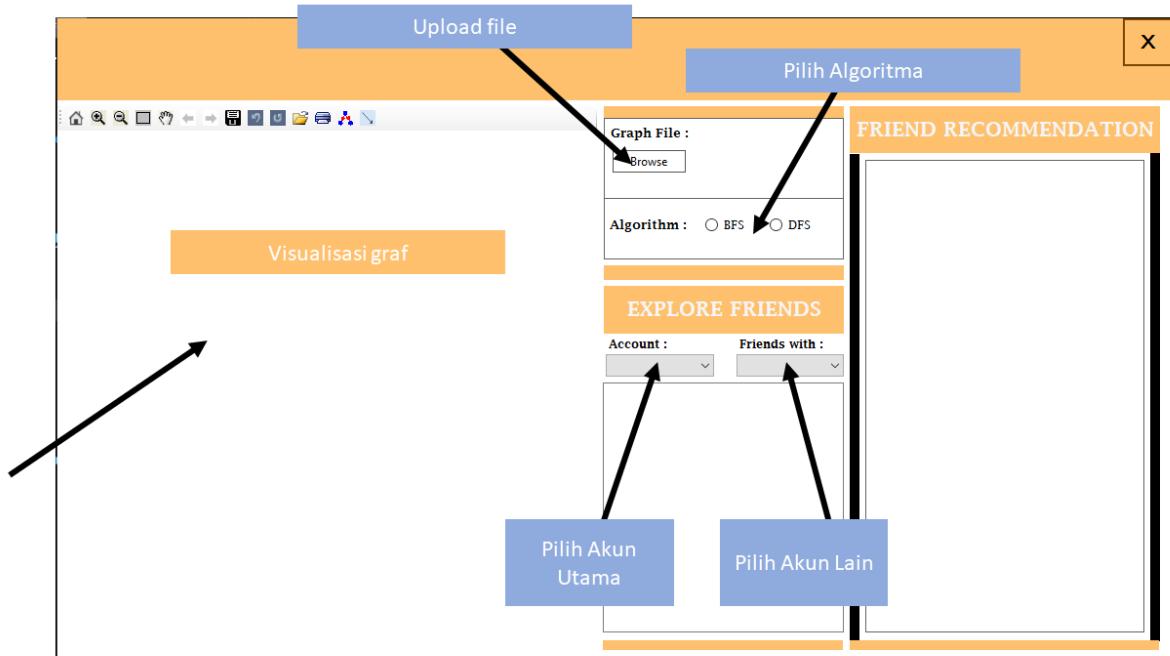
1. Event handler Windows Form Application
2. public Void getFriendRecommendation() // menampilkan rekomendasi teman
3. public Void exploreFriends() // menampilkan hasil explore friends

4.3 Tata Cara Penggunaan Program

Program merupakan aplikasi berbasis windows yang menerapkan simulasi People You May Know. Program pertama menerima file masukan bertipe .txt, dengan isi berupa baris pertama adalah jumlah pengguna (simpul) dan baris-baris selanjutnya adalah hubungan pertemanan antar pengguna (sisi). Program memiliki 2 fitur yaitu *Explore Friends* dan *Friend Recommendation*.

Explore Friends digunakan untuk menampilkan keterhubungan pengguna dengan pengguna lain. Jalur keterhubungan pengguna dimunculkan dalam visualisasi graf. Program juga menampilkan derajat keterhubungan antar pengguna. Dalam mencari keterhubungan, program menyediakan dua metode yaitu secara BFS dan DFS.

Friend Recommendation digunakan untuk menampilkan pengguna lain dengan derajat keterhubungan sebesar 1 atau dalam konteks aplikasi Facebook, rekomendasi pengguna muncul dari temannya teman pengguna. Tampilan rekomendasi teman diurutkan berdasarkan mutual friends terbanyak.

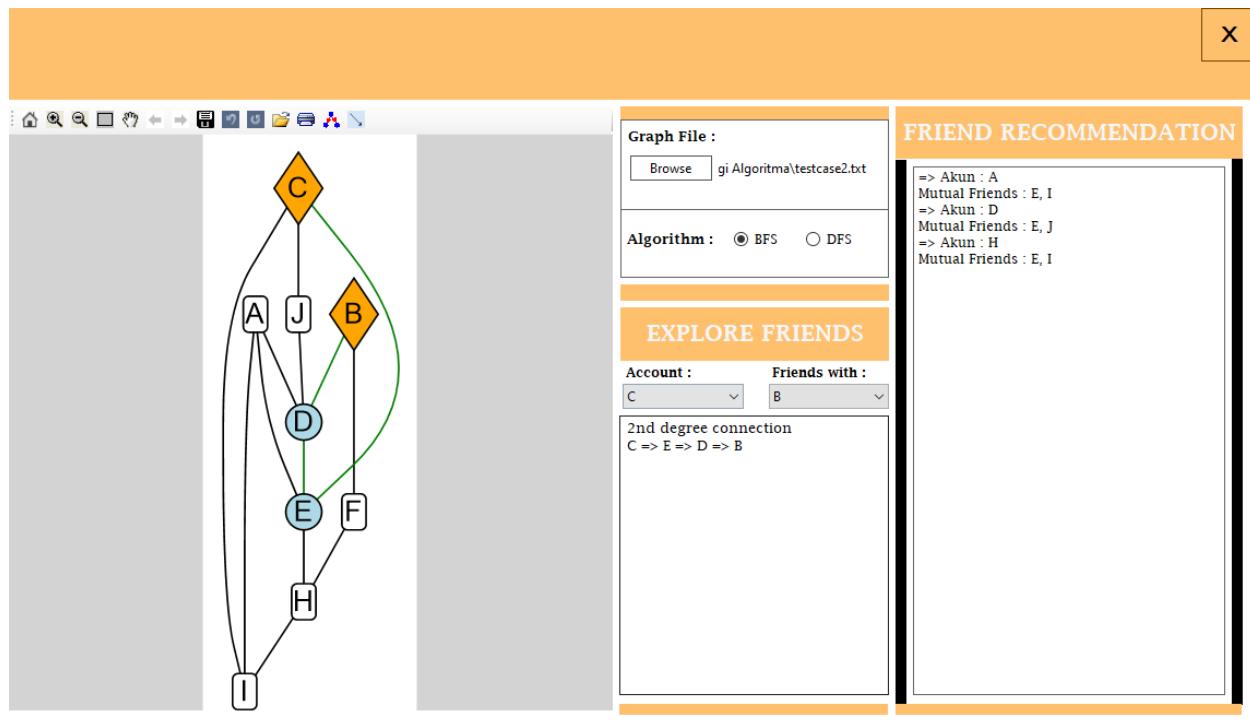


Gambar 4.1 Tampilan dan Cara Penggunaan

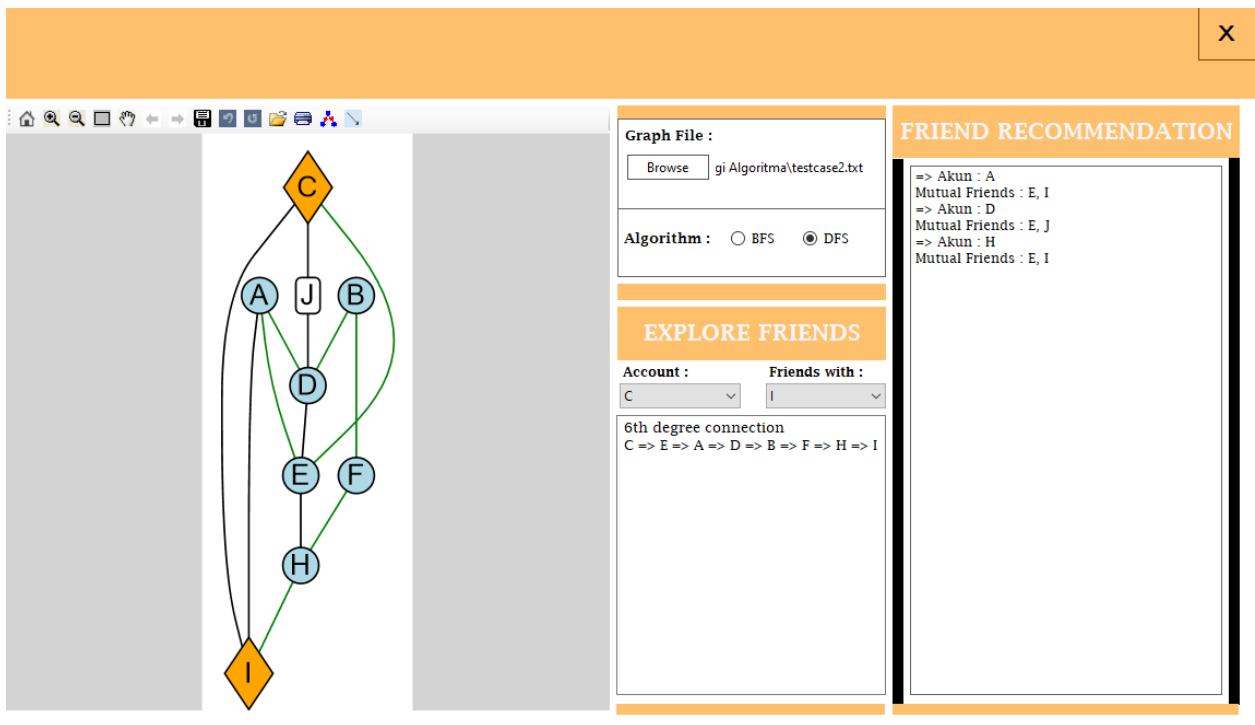
Langkah penggunaan program :

1. Jalankan file NasiPadang.exe
2. Upload file graf keterhubungan melalui tombol “Browse” dan pastikan file bertipe txt
3. Pilih algoritma yang diinginkan (**BFS/DFS**)
4. Terlebih dahulu pilih account “utama” yang ingin dicari rekomendasi teman dan keterhubungannya dengan pengguna lain pada pilihan “Account”
5. Program akan menampilkan rekomendasi teman untuk Account berupa teks pada panel **Friend Recommendation**
6. Pilih pengguna kedua yang ingin dicari derajat keterhubungannya dengan pengguna pertama pada pilihan “**Friends With**”. Jalur keterhubungan akan muncul pada visualisasi graf dengan sisi warna hijau menandakan sisi yang dilewati dan simpul berwarna merah menandakan simpul awal dan tujuan.
7. Derajat keterhubungan akan ditampilkan pada panel **Explore Friends**

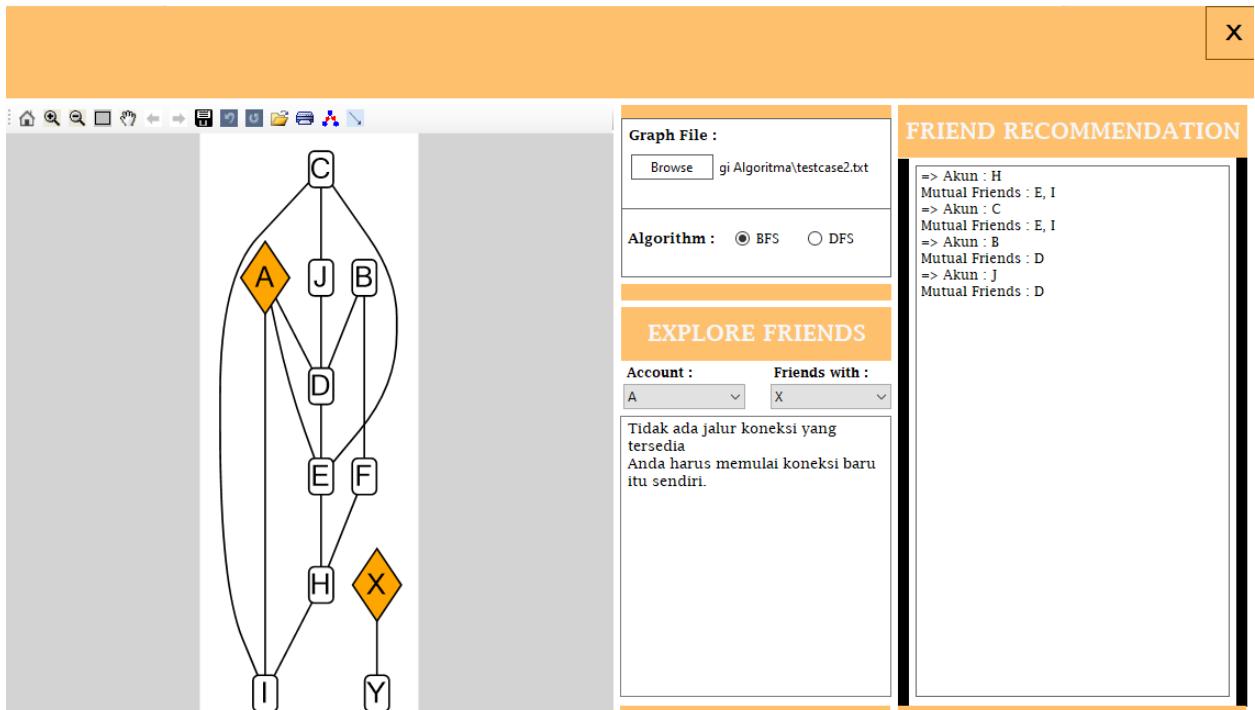
4.4 Hasil Pengujian



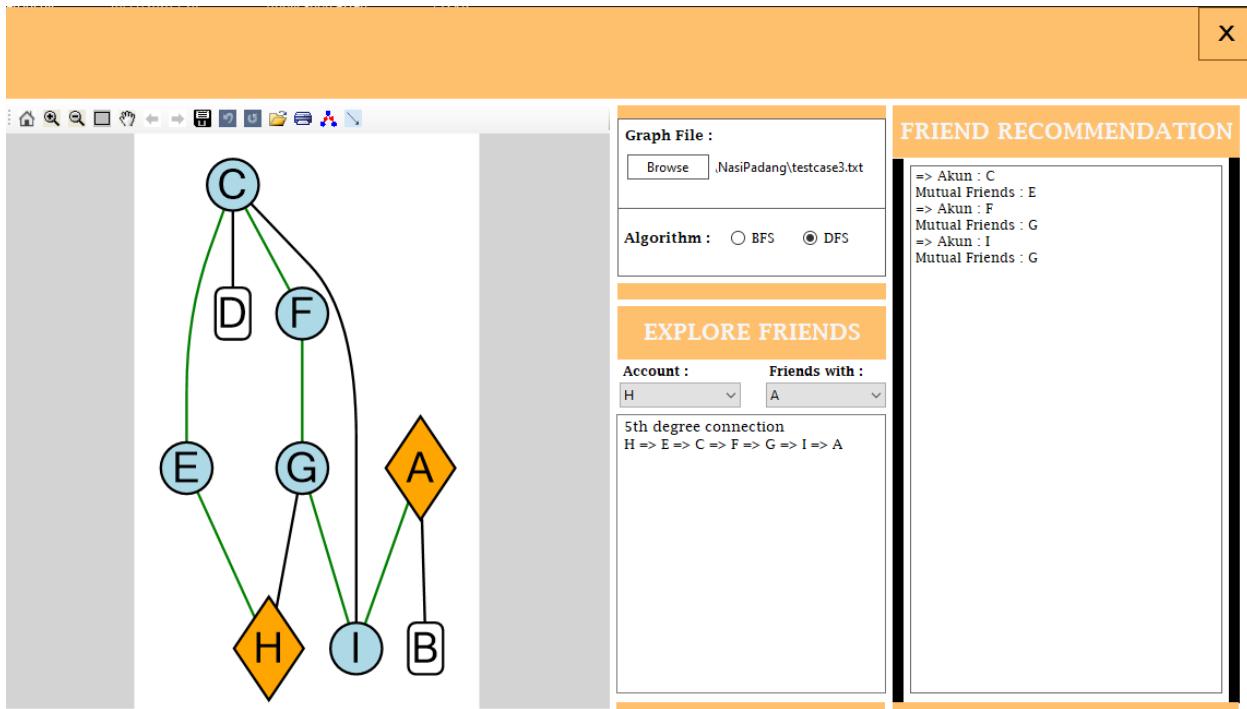
Gambar 4.2 Skenario BFS dari C ke B dan Hasil Rekomendasi Teman untuk C File Test Case 2



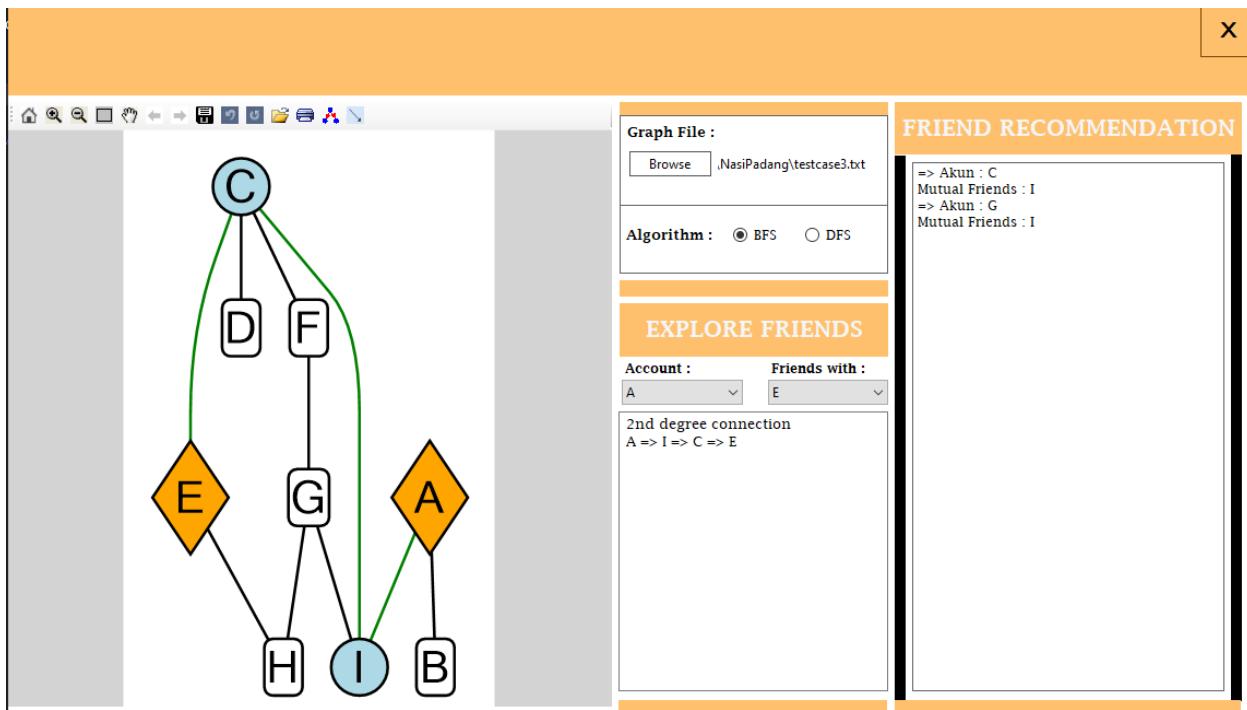
Gambar 4.3 Skenario DFS dari C ke I dan Rekomendasi Teman untuk C File Test Case 2



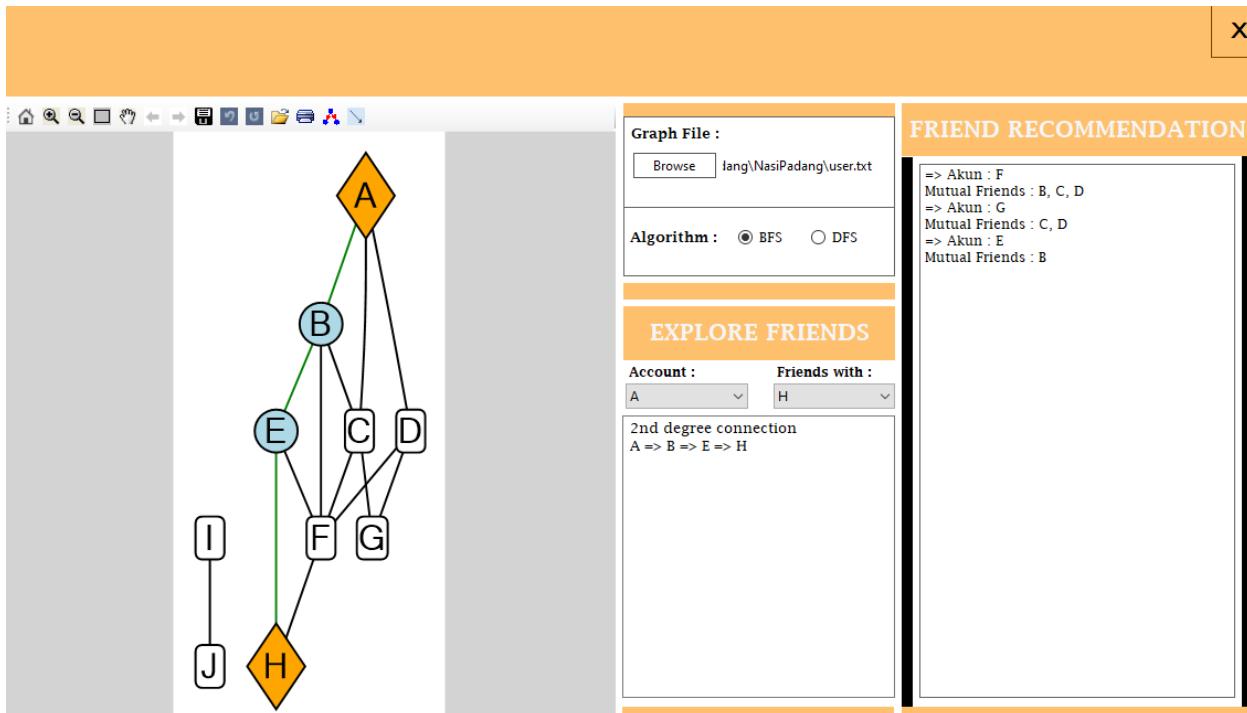
Gambar 4.4 Skenario Tidak Ditemukan Jalur Keterhubungan Antara A dan X File Test Case 2



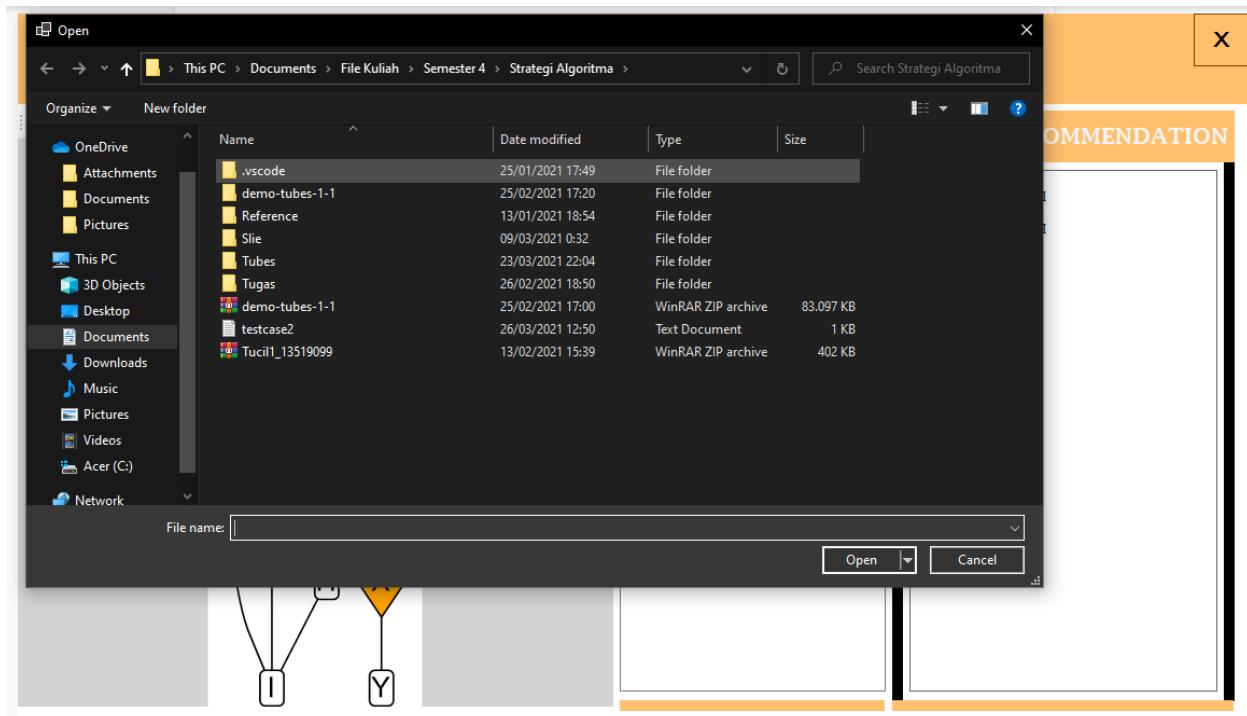
Gambar 4.5 Skenario DFS untuk Test Case 3



Gambar 4.6 Skenario BFS untuk Test Case 3



Gambar 4.7 Skenario BFS untuk Test Case 1



Gambar 4.8 Proses Upload File

4.5 Analisis

Untuk mengukur seberapa baik teknik pencarian DFS dan BFS, kita bisa menggunakan 4 aspek sebagai berikut: *Completeness*, *Optimality*, *Time Complexity*, dan *Space Complexity*. *Completeness* artinya apakah teknik yang digunakan bisa menjamin ditemukan solusi jika memang ada. *Optimality* yang artinya apakah bisa dijamin bahwa jalur untuk sampai kepada solusi adalah jalur paling minimal, dan *Time Complexity* serta *Space Complexity* yang berkaitan dengan kerumitan algoritma.

Pada kasus pencarian teman, baik DFS maupun BFS menjamin bahwa pasti akan ditemukan solusi jika ada. Artinya, baik DFS maupun BFS memenuhi aspek *Completeness*. Pada realisasinya, jalur yang ditemukan oleh DFS pasti akan selalu lebih panjang atau sama dengan jalur yang ditemukan oleh BFS. Sehingga, pada aspek *Optimality*, BFS lebih baik dibandingkan DFS. Untuk menghitung *Time Complexity* dan *Space Complexity*, kita bisa menggunakan istilah sebagai berikut:

- b: (*branching factor*) maksimum percabangan yang mungkin dari suatu simpul
- d: (*depth*) kedalaman dari solusi terbaik (*cost terendah*)
- m: maksimum kedalaman dari ruang status

Pada kasus BFS, baik *Time Complexity* maupun *Space Complexity* pada setiap kasus adalah $O(b^d)$, dikarenakan BFS pasti akan membangkitkan setiap simpul pada *depth* tertentu. Sedangkan DFS, memiliki *Time Complexity* $O(b^m)$ dan *Space Complexity* nya adalah $O(bm)$ dikarenakan pada DFS yang didahului adalah mencari sedalam-dalamnya node sampai tidak ada lagi yang bisa ditelusuri baru kemudian melakukan *backtrack*.

Berdasarkan analisis tersebut, dapat disimpulkan bahwa BFS lebih baik daripada DFS, di hampir setiap kasus pada persoalan ini. DFS hanya mampu menghasilkan rute yang pendeknya sama dengan BFS ketika node-node yang dikunjungi merupakan node yang memang langsung mengantarkan kepada tujuan akhir.

Kemudian, pada kasus pencarian rekomendasi teman, BFS jauh lebih baik dan lebih mudah untuk dimanfaatkan ketimbang DFS. Hal ini dikarenakan BFS membangkitkan semua simpul pada setiap kedalaman, sehingga pencarian teman yang merupakan rekomendasi (yang memiliki *depth* 2) dapat langsung dipetakan dengan mudah. Sedangkan DFS, secara dasarnya tidak mampu untuk memetakan semua kemungkinan rekomendasi teman tanpa melakukan modifikasi yang cukup signifikan pada algoritma DFS.

BAB 5

KESIMPULAN

5.1 Kesimpulan dan Saran

Graf traversal adalah algoritma pencarian simpul pada suatu graf. Terdapat dua contoh graf traversal yang digunakan pada Tugas Besar kali ini yaitu BFS dan DFS. BFS dan DFS adalah algoritma graf yang digunakan untuk mencari jalur keterhubungan antara 2 simpul. Salah satu penerapan BFS dan DFS adalah fitur *explore friends* dan *friend recommendation* pada media sosial seperti Facebook. Aplikasi yang dikembangkan kali ini adalah aplikasi berbasis windows. Dalam penggunaanya, BFS menghasilkan

rute yang lebih singkat ketimbang DFS. Dalam pembangunan aplikasi, digunakan framework .NET yang memanfaatkan bahasa C# sebagai bahasa pemrogramannya.

5.2 Refleksi dan Komentar

Komentar terhadap tugas ini adalah sangat membimbing karena kami dapat mempelajari hal baru yang dalam hal ini terkait dengan pengembangan aplikasi berbasis windows menggunakan framework .NET dan bahasa C# yang sebelumnya belum pernah diajarkan. Kami jadi terpacu untuk mengeksplorasi hal-hal baru diluar dari yang diajarkan di kuliah.

DAFTAR PUSTAKA

<http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Tugas-Besar-2-IF2211-Strategi-Algoritma-2021.pdf> diakses pada Jumat, 26 Maret 2021

<https://docs.microsoft.com/en-us/dotnet/csharp/> diakses pada Jumat, 26 Maret 2021

Anany V. Levitin. 2003. Introduction to the Design and Analysis of Algorithms. Addison-Wesley Longman Publishing Co., Inc., USA.