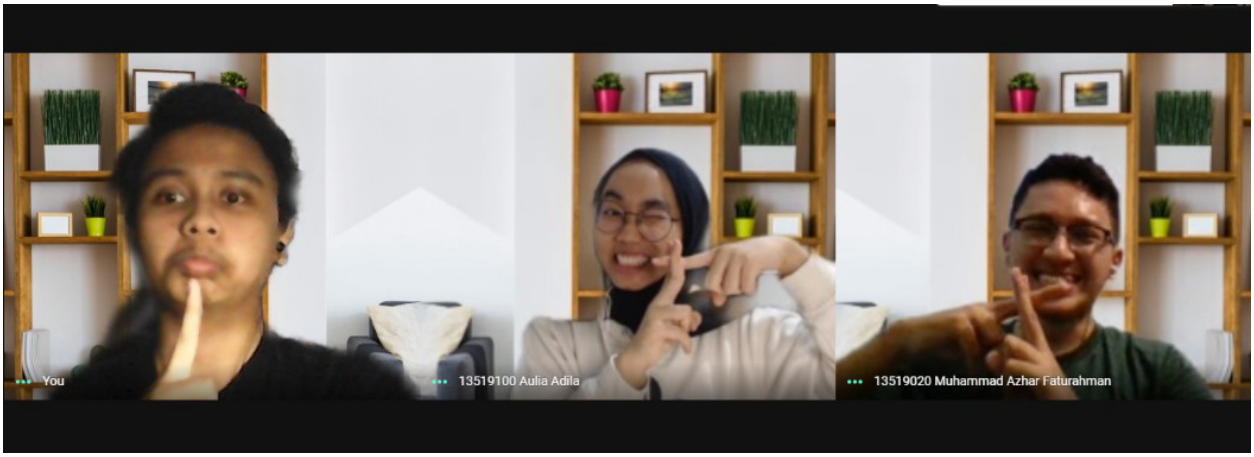
Laporan Tugas Besar 2 IF2211 Strategi Algoritma

Semester II Tahun 2020/2021

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



Disusun Oleh:

Muhammad Azhar Faturahman - 13519020

Aulia Adila - 13519100

Muhammad Dehan Al Kautsar - 13519200

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2020/2021

DAFTAR ISI

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.

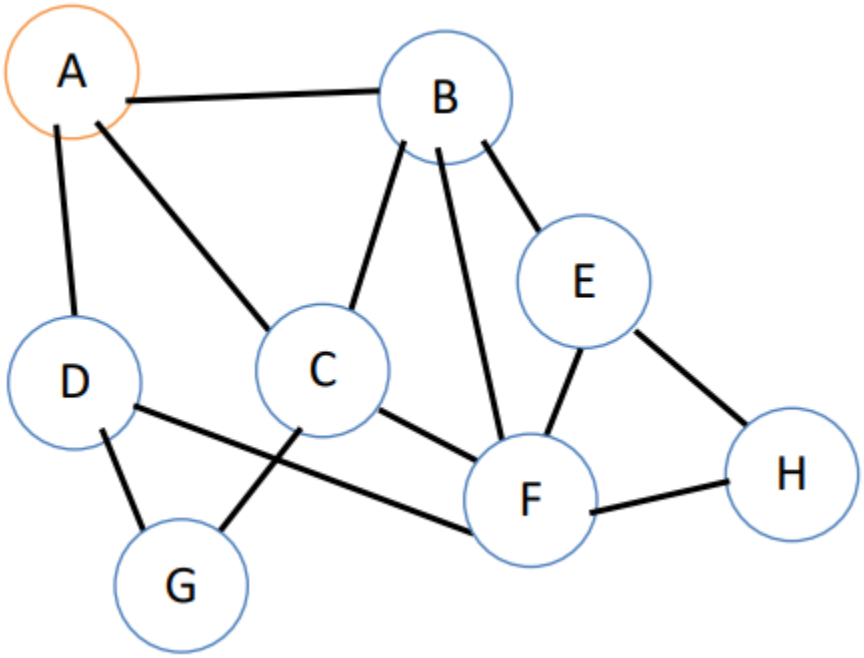
Contoh Input dan Output Program

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 1.1 Contoh Input Berkas File Eksternal

Visualisasi graf pertemanan yang dihasilkan dari file eksternal:



Gambar 1.2 Contoh Visualisasi Graf Pertemanan dari File Eksternal

Untuk fitur friend recommendation, misalnya pengguna ingin mengetahui daftar rekomendasi teman untuk akun A. Maka output yang diharapkan sebagai berikut:

```
Daftar rekomendasi teman untuk akun A:
Nama akun: F
3 mutual friends:
B
C
D

Nama akun: G
2 mutual friends:
C
D

Nama akun: E
1 mutual friend:
B
```

Gambar 1.3 Hasil output yang diharapkan untuk rekomendasi akun A

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

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

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

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

BAB II
LANDASAN TEORI

A. Dasar Teori

Graf traversal merupakan proses mengunjungi setiap vertex/node . Graf traversal digunakan untuk mencari jalur dalam suatu graph dari titik asal ke titik tujuan, mencari jalur terpendek antara dua node/vertex, menemukan semua jalur yang bisa dilalui dari titik asal ke titik tujuan.

Dalam melakukan penjelajahan sebuah graf, diperlukan metode-metode seperti yang akan dijelaskan berikutnya adalah *Breadth First Search* (BFS) dan *Depth First Search* (DFS).

Depth First Search (DFS) adalah algoritma untuk melintasi grafik terbatas. DFS mengunjungi simpul anak sebelum mengunjungi simpul disampingnya. Maka, ia melintasi kedalaman jalur tertentu sebelum menjelajahi lebarnya. Rekursi sangat umum digunakan saat mengimplementasikan algoritma DFS ini.

Algoritma dimulai dengan titik "root" yang dipilih. Kemudian simpul tersebut secara berulang-ulang bertransisi dari simpul saat ini ke simpul yang berdekatan dan belum dikunjungi, hingga ia tidak dapat lagi menemukan simpul yang belum dijelajahi untuk bertransisi ke dari lokasinya saat ini atau node yang dijelajahi sudah berupa node target yang dituju.

Berbeda dengan DFS, *Breadth First Search* (BFS) adalah algoritma untuk melintasi grafik terbatas yang mengunjungi simpul tetangga terlebih dahulu sebelum menjelajahi simpul anaknya. Oleh karena itu, ia melintasi jalur secara menyamping terlebih dahulu baru menuju kebawah. Pada umumnya dalam mengimplementasikan algoritma BFS kita tidak perlu menggunakan kode yang rekursif, namun tidak dibatasi pula bahwa algoritma ini harus tidak rekursif.

B. Penjelasan Singkat Mengenai C# Desktop Application Development

C# Desktop Application Development adalah sebuah kakas yang yang berbasis desktop yang mampu beroperasi tanpa menggunakan internet. Aplikasi berbasis desktop biasanya dibuat menggunakan 3 bahasa pemrograman, yaitu C++, C#, dan Java. C# sendiri dapat dibantu dengan framework .Net yang memudahkan programmer dalam melakukan pengkodingan karena bentuk visual aplikasinya yang sudah langsung terlihat oleh *programmer*-nya sehingga proses *debugging* menjadi lebih mudah.

Keunggulan desktop application yang menggunakan framework .Net dan bahasa pemrograman C# adalah developer dimungkinkan untuk membuat aplikasi windows base

yang di-*launch* melalui build di dalam Visual Studio. Keunggulan lainnya adalah dapat berjalan independen, tanpa perlu menggunakan *browser*, tidak perlu koneksi internet, karena

BAB III
ANALISIS PEMECAHAN MASALAH

A. Langkah-Langkah Pemecahan Masalah

Langkah yang kami ambil untuk memecahkan permasalah yang diberikan pada persoalan ini adalah dengan diawali menganalisis permasalahan dan mencoba untuk memetakan permasalahan tersebut menjadi sebuah program.

Pada kasus ini, permasalahan sangat bertumpu pada graf, oleh karena itu kami membutuhkan suatu Class untuk menyimpan dan merepresentasikan sebuah graf. Sehingga dibuatlah Class yang bernama Graph. Class Graph ini menyimpan 3 buah atribut, yaitu adjacentMatrix, count_node, dan node_dictionary. Atribut adjacentMatrix berguna untuk menyimpan matrix ketetanggaan antar setiap node, atribut count_node berguna untuk menyimpan jumlah simpul yang ada pada graf, sedangkan node_dictionary berfungsi untuk menyimpan nama simpul dan index simpul yang bersesuaian pada atribut adjacentMatrix. Setiap nama simpul haruslah unik.

Setelah membuat atribut Class Graph, kami kemudian membuat method yang akan diperlukan oleh Class Graph ini. Method yang kami buat diantaranya getNumberOfNode, getAllNodes, indexToName, initAdjacentMatrix, addToDictionary, foundIndex, foundAdj, addAdj, DFS, doDFS, BFS, doBFS, mutualFriend, allMutual, ExploreFriend, dan printRute. Method yang kami buat ini kebanyakan berfungsi untuk menyelesaikan permasalahan yang diberikan dan juga berfungsi untuk membangun graf yang dibutuhkan pada persoalan.

Kemudian setelah Class Graph berhasil dibuat dan diuji, kami mencoba untuk membuat GUI sesuai yang diminta pada spesifikasi. Proses pembuatan GUI ini dimulai dari membuat form utama yang berfungsi untuk menu utama saat program dibuka. Menu pada form utama ini terdiri dari memilih file graf yang diinginkan, memilih metode DFS atau BFS, menampilkan visualisasi graf, memilih akun untuk dicarikan teman, dan memilih akun untuk mengexplore teman.

Karena keterbatasan luas dari form utama, kami memutuskan untuk membuat form baru yang berfungsi untuk menampilkan hasil pencarian explore friend dan mutual friend sesuai spesifikasi persoalan. Setelah semua itu dibuat, maka permasalahan berhasil terpecahkan.

B. Proses Mapping Persoalan Menjadi Elemen-Elemen Algoritma BFS dan DFS

Algoritma BFS (*Breadth First Search*) yang kelompok kami gunakan mengambil konsep pencarian solusi dari graf statis. Oleh karena itu diperlukan sebuah struktur data/*class* yang di-*generate* sebelum dilakukannya proses BFS. Struktur data / *class* yang di-*generate* memiliki bentuk berupa graf. Setelah *class* graf terbentuk, tepat sebelum proses BFS dimulai kelompok kami menginisialisasi dua buah array, yang pertama berupa array of boolean yang menyatakan simpul mana saja yang sudah disinggahi, dan yang kedua adalah *array of string* yang berisi langkah-langkah dimana proses BFS menyinggahi sebuah simpul. Ketika sebuah simpul disinggahi, *array of string* tersebut akan mencatat simpul tersebut. Kemudian setelah semua persiapan telah dilakukan, kita dapat mulai mencari rute singgah dengan skema BFS, yaitu mengunjungi simpul yang bertetangga terlebih dahulu dibandingkan simpul anaknya. Selama proses ini, dibuat sebuah queue list yang berisi antrian sebuah simpul untuk dikeluarkan dari queue dan dimasukkan ke dalam *array of string*. Pada awalnya, sebuah queue yang kosong dimasukkan initial node dan proses BFS berlangsung hingga queue tersebut kosong atau simpul yang dituju sudah ditemukan. Pada akhirnya *array of string* akan berisi urutan simpul mana saja yang dikunjungi sistem. Namun untuk mencari jalur solusi dari algoritma ini diperlukan algoritma lainnya untuk mengubah jalur langkah menjadi jalur solusi. Setelah berhasil diubah, maka array of string terbaru itu lah yang menjadi solusi dalam algoritma ini.

Algoritma DFS (*Depth First Search*) yang kami gunakan mengambil konsep pencarian solusi dari graf statis. Oleh karena itu, diperlukan terlebih dahulu struktur data/class untuk merepresentasikan sebuah graf. Setelah graf tersedia, maka pencarian solusi dengan algoritma DFS dapat dibuat. Pada permasalahan ini, diberikan sebuah graf tak berarah dengan n buah simpul dan ada sisi yang menghubungkan antar simpul. Setiap simpul merepresentasikan akun *media social* seseorang dan setiap sisi merepresentasikan hubungan pertemanan antara 2 buah akun (simpul). Kemudian akan ditentukan rute/jalur pertemanan dari suatu akun ke akun lain. Pada kasus ini, pencarian rute pertemanan dilakukan menggunakan algoritma DFS. Sebelum algoritma DFS dapat dilakukan, diperlukan terlebih dahulu suatu list yang menyimpan rute yang sudah dipilih dan suatu list sepanjang n (banyak node) untuk menyimpan status apakah suatu node ke i sudah dikunjungi. Mula-mula list rute dan list status dikunjungi diisi kosong. Rute pencarian dimulai dari akun awal, masukkan akun ini ke list rute, lalu isi status simpul sebagai telah dikunjungi. Kemudian mulai mencari rute solusi dengan skema DFS, yaitu mengunjungi simpul tetangga secara rekursif dan pemilihan tetangga yang akan dikunjungi berdasarkan urutan *alphabetical*. Jika pada suatu simpul sudah tidak memiliki tetangga yang belum dikunjungi, maka akan *backtrack* ke simpul sebelumnya. Jika berhasil sampai ke simpul tujuan, maka pencarian dihentikan dan DFS akan mengembalikan rute hasil pencarian, jika tidak berhasil mencapai simpul tujuan dan semua simpul yang

mungkin dilewati sudah dicek, maka DFS dihentikan dan dikembalikan nilai gagal berupa rute kosong.

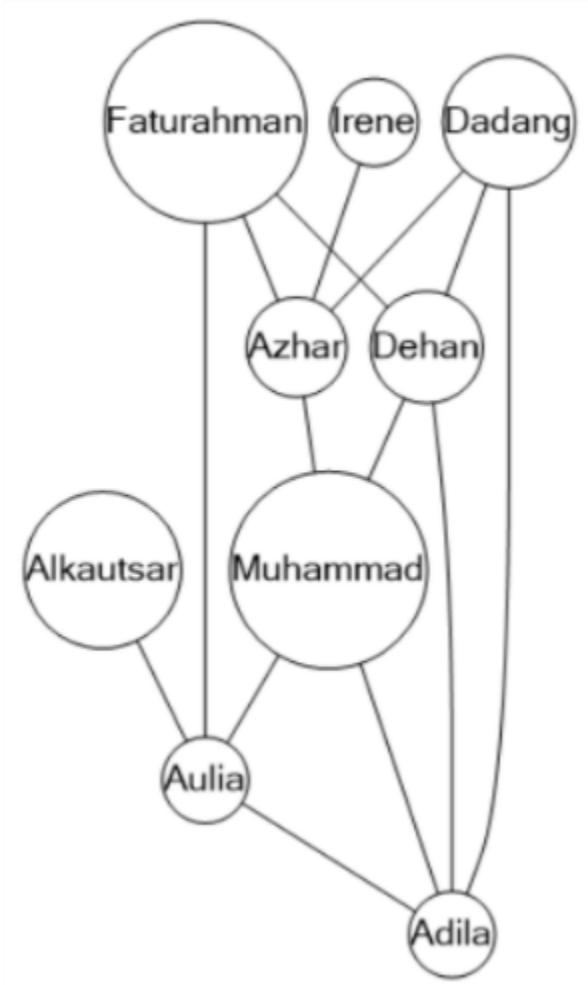
C. Contoh Ilustrasi Kasus Lain yang Berbeda dengan Contoh pada Spesifikasi Tugas

Berikut adalah contoh lain yang kami buat pula *text file* nya untuk dapat dijalankan di program kami. Di bawah adalah isi dari file eksternal yang kelompok kami buat.

```
14
Adila Aulia
Adila Muhammad
Muhammad Dehan
Muhammad Azhar
Dehan Adila
Dehan Faturahman
Alkautsar Aulia
Azhar Faturahman
Azhar Dadang
Aulia Faturahman
Dadang Dehan
Dadang Adila
Irene Azhar
Aulia Muhammad
```

Gambar 3.1 Contoh Input Berkas File Eksternal Lainnya

Berikut adalah hasil graf dari file eksternal diatas:



Gambar 3.2 Hasil Graf yang Terbentuk dari File Eksternal

Ketika file eksternal dan graf diatas terbentuk, kita dapat memilih simpul teman manakah yang ingin dicari menggunakan algoritma BFS dan DFS. Misalkan sebagai contoh akan dicari siapa saja friend recommendation dari “Adila” dan bagaimana explore friendnya dengan “Irene”. Jika menggunakan algoritma BFS, maka hasil explore friend yang akan dihasilkan adalah sebuah rute yaitu “Adila → Dadang → Azhar → Irene”. Sedangkan apabila menggunakan algoritma DFS, maka hasil explore friend yang akan dihasilkan adalah sebuah rute yaitu “Adila → Aulia → Faturahman → Azhar → Irene”. Selain itu muncul pula rekomendasi teman

BAB IV

IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Program

1. Fitur Friend Recommendation

```
function mutualFriend(username1, username2, graph)→ list of
string

DEKLARASI
friend1, friend2, mutual : list of string
constant nNode : integer = jumlah node dari graph
constant idx1 : boolean = index username1
constant idx2 : boolean = index username2

ALGORITMA
i traversal[0..nNode]
  if (isFoundAdjacent(idx1, i)) then
     friend1 ← indexToName(i)

i traversal[0..nNode]
  if (isFoundAdjacent(idx2, i)) then
     friend2 ← indexToName(i)

for item in friend1
   if (item in friend2) then
       mutual ← item

→ mutual
```

```
function allMutual(username1) → dictionary (Key: string, Value:
list of string)

DEKLARASI
friend, mutualT : list of string
constant nodeDictionary : dictionary (Key: string, Value: list of
string){dictionary of nodes in this graph}
mutual : dictionary (Key: string, Value: list of string)
constant thisGraph: graph {current graph}

ALGORITMA
for node in nodeDictionary
    if (isFoundAdjacent(username1, Key(node)) then
        friend ← Key(node)

for node in nodeDictionary
    mutualT ← mutualFriend(username1, Key(node), thisGraph)
    if (length(mutualT)≠0 and
        Key(node)≠username1 and
        Key(node) not in friend) then
        mutual[Key(node)] ← mutualT
→ mutual
```

```
procedure AddComponentFriendR()

DEKLARASI
constant dict : dictionary (Key: string, Value: list of string)
countManual : integer

ALGORITMA
output("Friend Recommendation for: ",initialNode)
dict ← allMutual(initialNode)

for node in dict.OrderByDescending(length(Value(node)))
   output("Nama akun: ", Key(node))
   countMutual ← length(Value(node))
   output(countMutual," Mutual Friend:\n")

   for item in Value(node)
      output(item)
```

2. Fitur Explore Friend

```
function ExploreFriend(username1, username2, pilihan : string) →
array of string

DEKLARASI
rute : array of string

ALGORITMA
if (pilihan = "DFS") then
     rute ←  DFS(username1, username2)
else if (pilihan = "BFS") then
     rute ←  BFS(username1, username2)

→ rute
```

```
procedure pathExploreEditor()

DEKLARASI
text : string
arrayOfPath : array of string
index : integer
nth : integer

ALGORITMA
text ← ""
arrayOfPath ← ExploreFriend(username1, username2, pilihan)

if (length(rute) = 0) then
     output("Tidak ada jalur koneksi yang tersedia\n")
     output("Anda harus memulai koneksi baru itu sendiri\n")
else
     index ← 0
     text ← text + arrayOfPath[index]
     while (index < length(arrayOfPath)-1) do
          text ← text + " -> " + arrayOfPath[index+1]
```

```
            index ← index + 1
      text ← text + "\n"
      nth ← index - 1
      depend on (nth)
            nth = 1 : text ← text + "1st Degree"
            nth = 2 : text ← text + "2nd Degree"
            nth = 3 : text ← text + "3rd Degree"
            else : text ← text + nth + "th Degree"

output(rute)
```

B. Penjelasan Struktur Data yang Digunakan dalam Program dan Spesifikasi Program

1. Class Graph

Class Graph merupakan implementasi dari Struktur Data Graph. Class Graph merepresentasikan graf pada dunia nyata, dimana akan terdapat *node* (simpul) dan *edge* (sisi). Pada Class Graph ini, digunakan implementasi berupa *Adjacent Matrix* untuk merepresentasikan sisi yang menghubungan antar simpul. Sehingga pada implementasinya, dibuat 3 buah atribut pada Class Graph yaitu :

```
adjacentMatrix  : matrix of boolean
count_node      : integer
node_dictionary : map<string,int> (sorted)
```

Atribut adjacentMatrix berguna untuk menyimpan matrix ketetanggaan antar setiap node, atribut count_node berguna untuk menyimpan jumlah simpul yang ada pada graf, sedangkan node_dictionary berfungsi untuk menyimpan nama simpul dan index simpul yang bersesuaian pada atribut adjacentMatrix. Setiap nama simpul haruslah unik.

Terdapat beberapa Method yang kami buat dan dapat digunakan untuk Class Graph ini, diantaranya getNumberOfNode, getAllNodes, indexToName, initAdjacentMatrix, addToDictionary, foundIndex, foundAdj, addAdj, DFS, doDFS, BFS, doBFS, mutualFriend, allMutual, ExploreFriend, dan printRute.

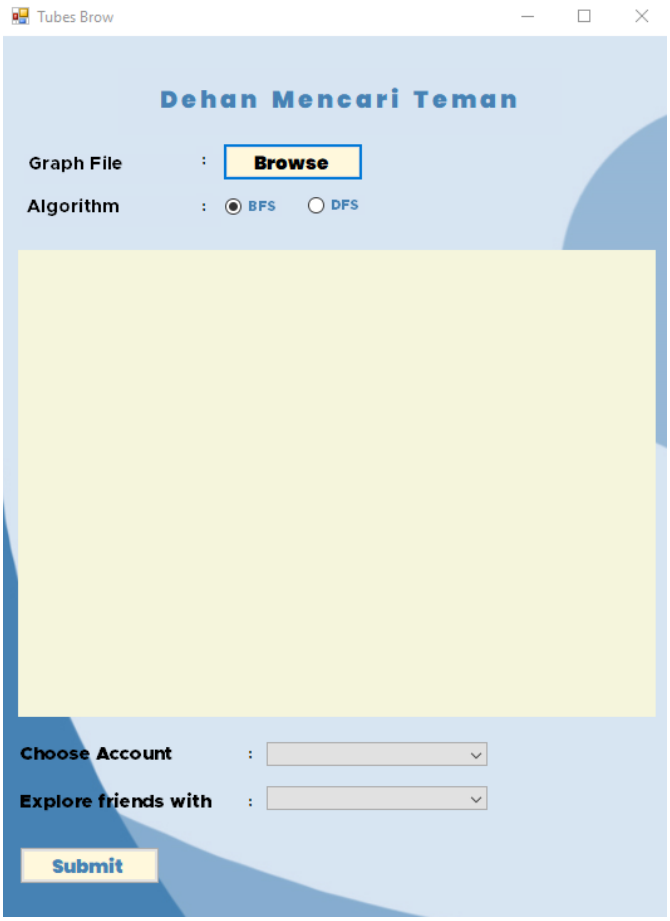
Method getNumberOfNode berfungsi untuk memberikan jumlah simpul yang terdapat pada graf. Method getAllNode akan mengembalikan semua simpul berupa list yang terdapat pada graf. Method indexToName berfungsi untuk mengubah nomor simpul menjadi nama simpul, initAdjacentMatrix berfungsi untuk menginisialisasi matrix ketetanggaan, addToDictionary berfungsi untuk menambah data simpul pada kamus dictionary, foundIndex untuk mengubah nama simpul menjadi nomor simpul, foundAdj untuk mengembalikan status ketetanggaan dua buah simpul, addAdj berfungsi untuk menambah keterangan status ketetanggaan antar dua simpul pada matriks ketetangaan.

Method DFS berfungsi untuk mencari rute/jalur antar dua buah simpul menggunakan metode DFS, fungsi DFS akan menginisialisasi nilai awal untuk

selanjutnya memanggil doDFS untuk melakukan pencarian rute. Method BFS berfungsi untuk mencari rute/jalur antar dua buah simpul menggunakan metode BFS, fungsi BFS akan menginisialisasi nilai awal untuk selanjutnya memanggil doBFS untuk melakukan pencarian rute. Method mutualFriend dan allMutual berguna untuk mencari mutualFriend dari suatu simpul. Method ExploreFriend berfungsi untuk melakukan eksplorasi antar 2 buah simpul, sedangkan method printRute berguna untuk menampilkan sebuah rute yang dimasukkan ke dalam parameternya.

C. Penjelasan Tata Cara Penggunaan Program

Untuk menggunakan program yang kelompok kami bangun, pengguna cukup menjalankan program yang bernama "Tubes2Stima-ckck.exe" dan program akan berjalan layaknya program normal. Kemudian program memiliki berbagai fitur yang dapat dilihat dibawah ini.



Gambar 4.1 Tampilan Awal Program

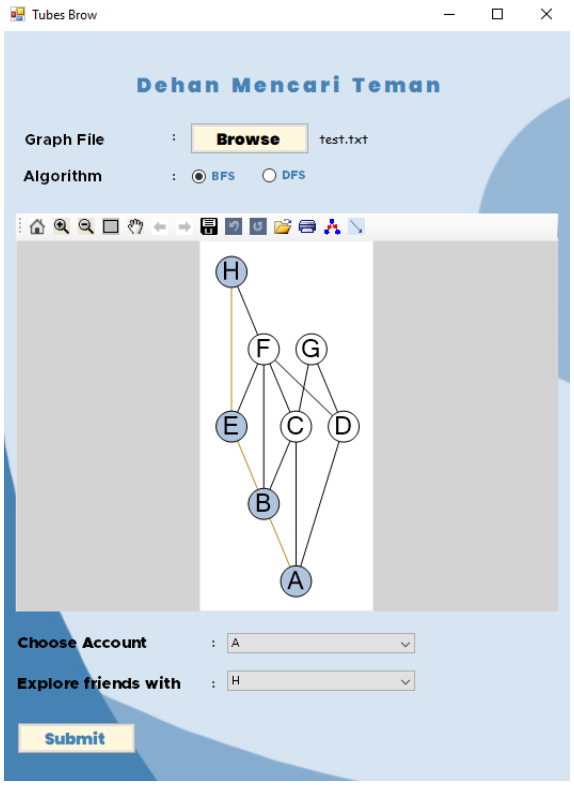
Dapat dilihat bahwa program ini akan memberi fitur kepada pengguna untuk melakukan load file yang sudah terdapat pada *local computer* pengguna. Setelah file dalam

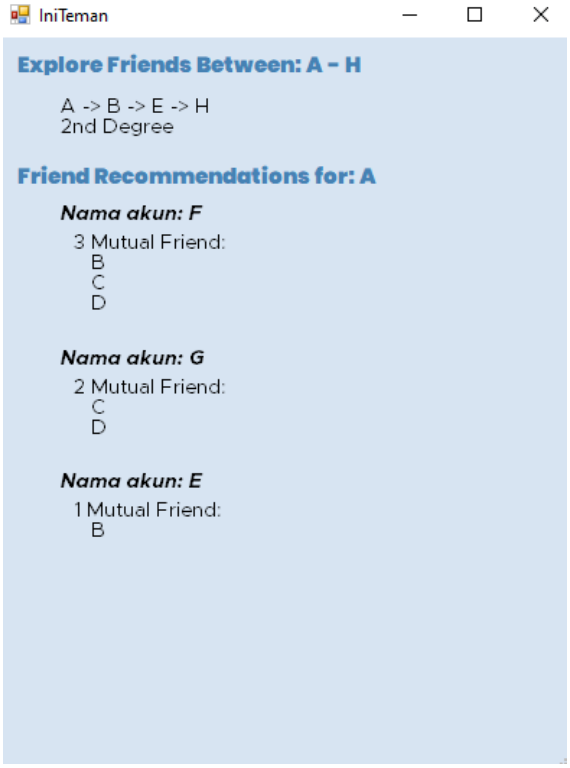
ekstensi .txt dibuka, graf akan divisualisasikan pada box yang kosong dibawah pilihan metode algoritma, dan ketika fitur lainnya yang berupa pemilihan metode pencarian dipilih, serta pemilihan akun yang ingin digunakan dan akun yang dituju telah dipilih oleh pengguna (dengan seluruh pilihan ditampilkan dalam *dropdown button*, atau dapat diketik pada kolom tersedia tanpa perlu memperhatikan apakah kapital atau tidak), graf akan divisualisasi berupa simpul dan sisi yang berwarna yang menunjukkan jalur solusi dari tiap masing-masing solusi. Ketika fitur submit ditekan, maka akan muncul form baru berupa hasil dari explore friends antara akun yang dipilih dan akun yang dituju, serta friend recommendation dari akun yang dipilih.

D. Hasil Pengujian

Data Uji 1 - BFS

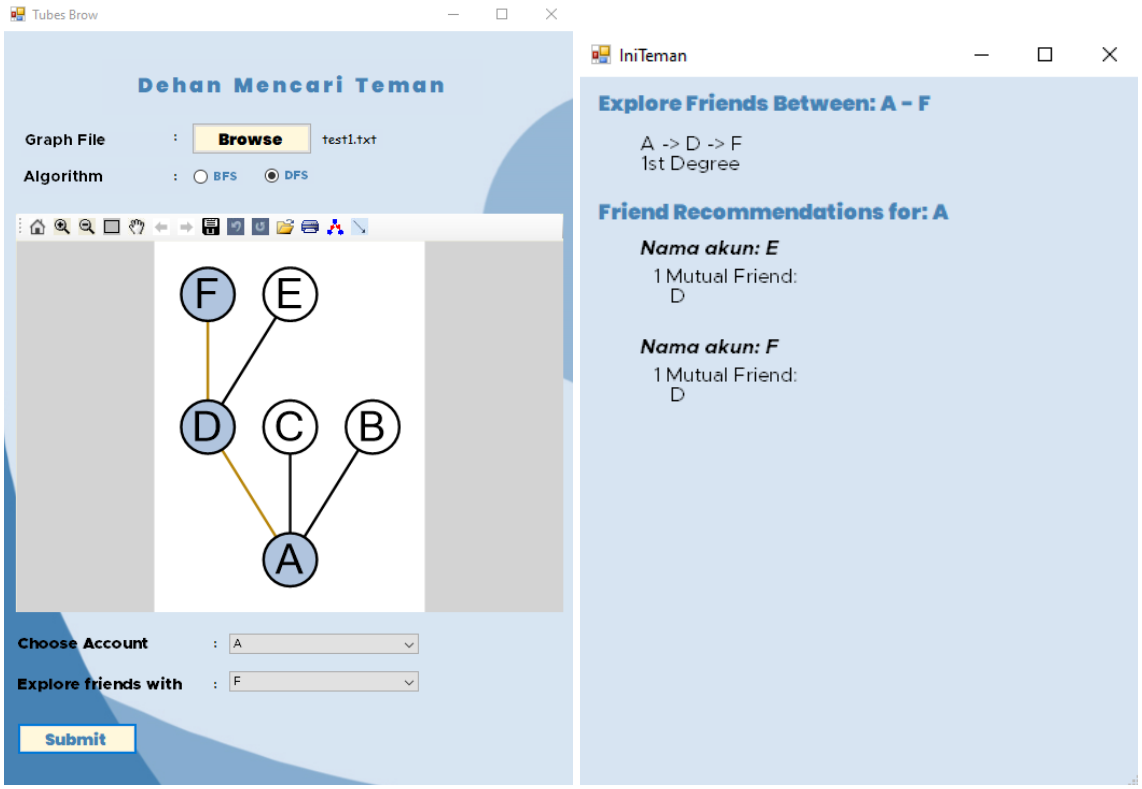
Data uji yang pertama diambil dari file eksternal test.txt dengan visualisasi seperti tampilan di bawah. Simpul dan garis yang diwarnai merupakan solusi untuk pencarian dengan algoritma BFS. Pada *page* selanjutnya, ditampilkan hasil *explore friends* dan *friend recommendation* yang diurutkan dari jumlah mutual friend terbanyak.





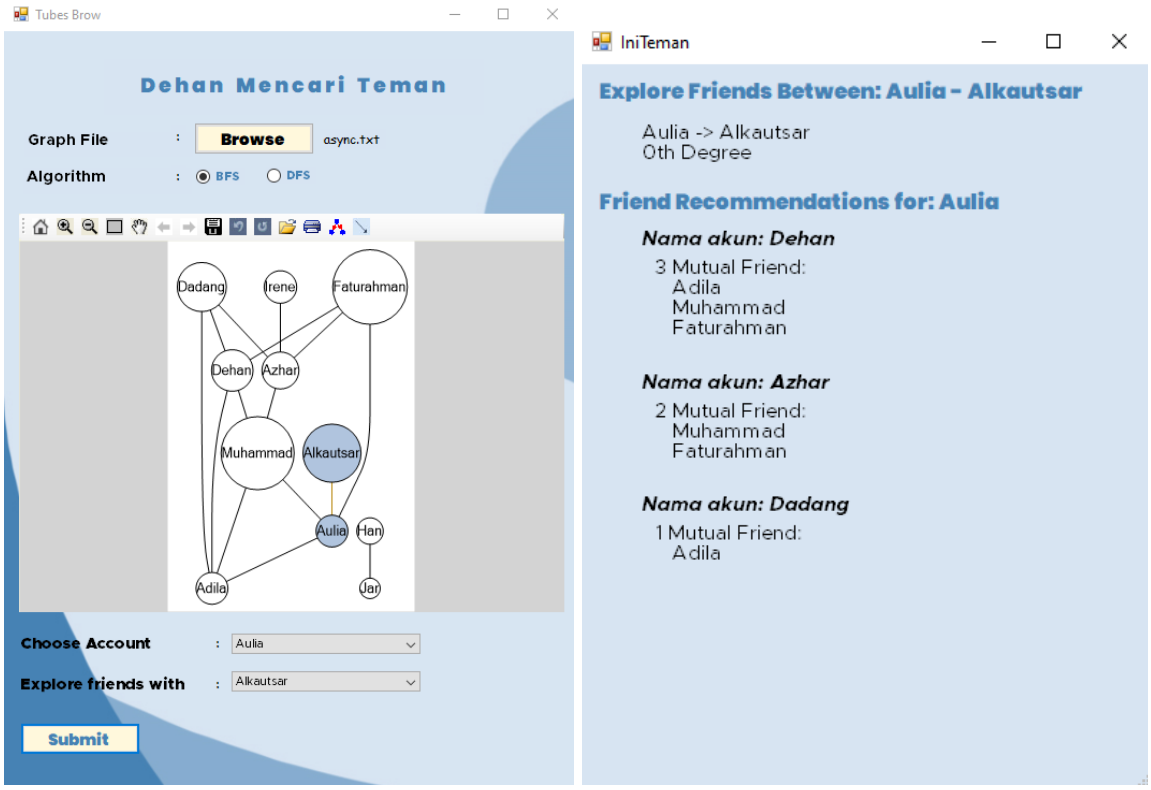
Data Uji 2 - DFS

Data uji yang kedua diambil dari file eksternal test1.txt dengan visualisasi seperti tampilan di bawah. Simpul dan garis yang diwarnai merupakan solusi untuk pencarian dengan algoritma DFS. Pada *page* selanjutnya, ditampilkan hasil *explore friends* dan *friend recommendation* yang diurutkan dari jumlah mutual friend terbanyak. Namun, karena akun E dan F memiliki jumlah mutual friend yang sama, maka output yang ditampilkan berurut berdasarkan abjad.



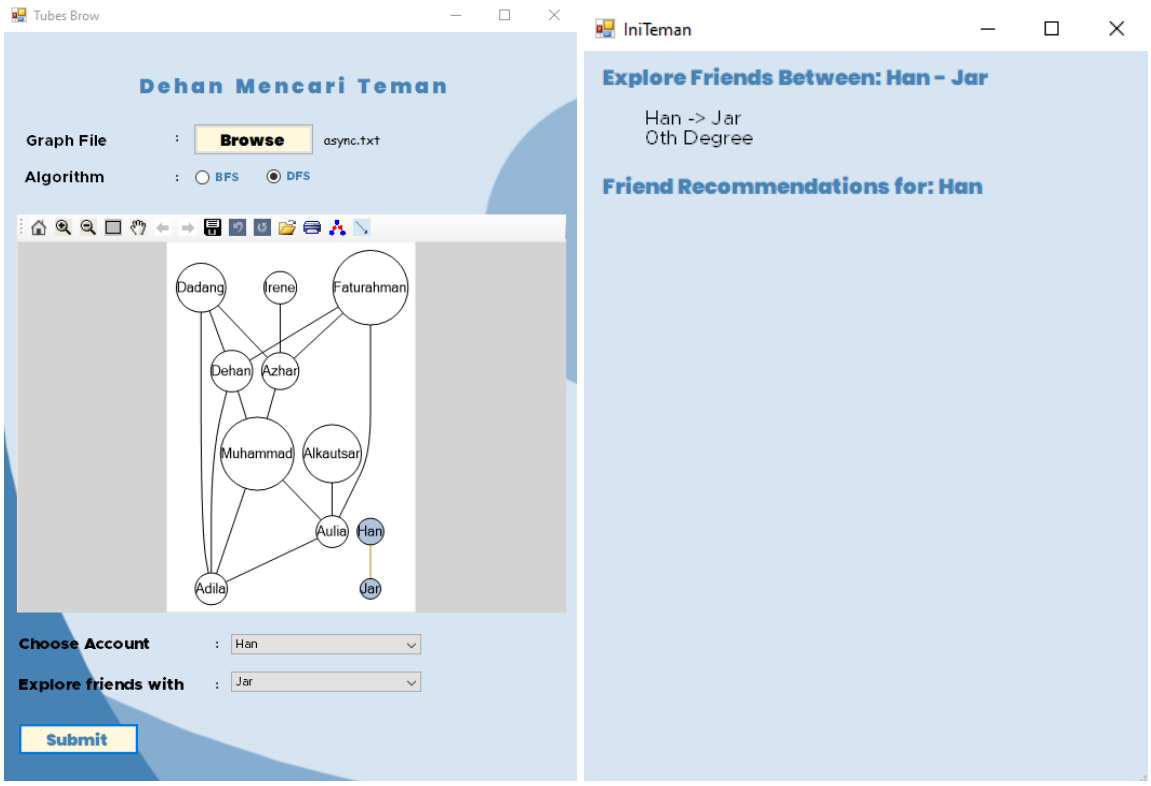
Data Uji 3 - BFS

Data uji yang ketiga diambil dari file eksternal async.txt dengan visualisasi seperti tampilan di bawah. Simpul dan garis yang diwarnai merupakan solusi untuk pencarian dengan algoritma BFS. Pada *page* selanjutnya, ditampilkan hasil *explore friends* dan *friend recommendation* yang diurutkan dari jumlah mutual friend terbanyak. Aulia dan Alkautsar sudah terhubung secara langsung sebagai teman, sehingga Explore Friends berada pada 0th degree.



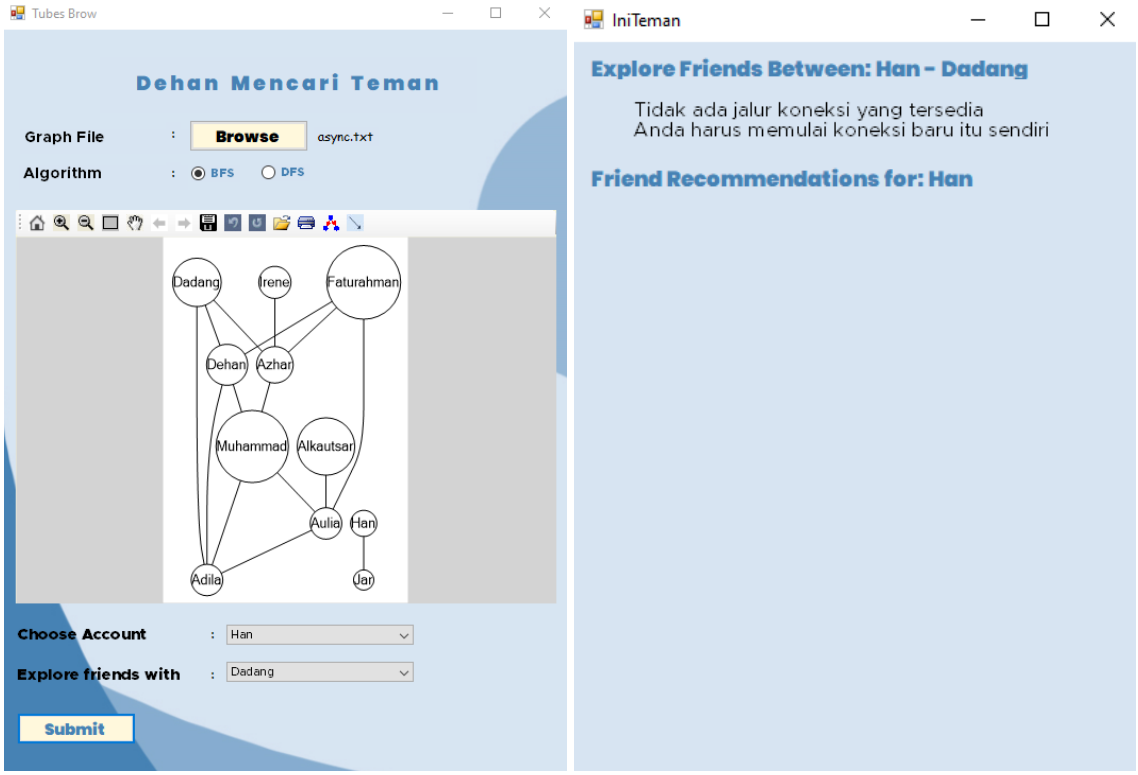
Data Uji 3 - DFS

Data uji yang ketiga diambil dari file eksternal async.txt dengan visualisasi seperti tampilan di bawah. Simpul dan garis yang diwarnai merupakan solusi untuk pencarian dengan algoritma BFS. Pada *page* selanjutnya, ditampilkan hasil *explore friends* dan *friend recommendation*. Han dan Jar langsung terhubung sebagai teman, sehingga Explore Friends berada pada 0th degree. Simpul Han tidak terhubung dengan simpul lain selain dengan Jar, sehingga tidak ada Friend Recommendation untuk simpul Han.



Data Uji 3 - Kasus Khusus

Data uji yang ketiga diambil dari file eksternal async.txt dengan visualisasi seperti tampilan di bawah. Simpul dan garis yang diwarnai merupakan solusi untuk pencarian dengan algoritma BFS. Pada *page* selanjutnya, ditampilkan hasil *explore friends* dan *friend recommendation*. Han dan Dadang tidak terhubung dalam graf (tidak ditemukan jalur koneksi sama sekali antar kedua akun), sehingga pada Explore Friends akan ditampilkan informasi bahwa kedua akun tidak dapat terhubung. Simpul Han tidak terhubung dengan simpul lain selain dengan Jar, sehingga Han tidak rekomendasi teman untuk Han (Friend Recommendation kosong)



E. Analisis dari Desain Solusi Algoritma BFS dan DFS yang Diimplementasikan

Berdasarkan desain solusi algoritma BFS dan DFS yang diimplementasikan, dapat dilihat bahwa hampir sepanjang waktu algoritma BFS memberikan hasil derajat connection yang lebih kecil dan baik dibandingkan algoritma DFS, namun implementasinya cukup panjang. Beberapa kasus yang menyebabkan DFS dapat lebih baik adalah bagaimana caranya graf direpresentasikan sedemikian rupa sehingga proses penjelajahan simpul oleh DFS dapat dilakukan dengan lebih cepat dibandingkan BFS dan tepat mirip seperti BFS. Pemilihan simpul mana yang ingin dituju terlebih dahulu sangat mempengaruhi kecepatan algoritma DFS.

BAB V
KESIMPULAN DAN SARAN

A. Kesimpulan

Berikut adalah beberapa kesimpulan yang kami dapatkan dari pengerjaan tugas besar mata kuliah IF2211 Strategi Algoritma

1. BFS dan DFS adalah salah satu metode untuk melakukan penjelajahan simpul dari representasi graf. Dimana BFS melakukan penjelajahan secara menyamping (breadth), sedangkan DFS melakukan penjelajahan secara mendalam (depth).
2. Jejaring pertemanan dari media sosial seperti Facebook, Twitter, dan lain-lain dapat ditelusuri menggunakan algoritma BFS dan DFS.
3. Kelompok kami berhasil dalam membuat aplikasi berbasis desktop untuk mencari penjelajahan pertemanan yang direpresentasikan ke dalam graf.
4. *Interface* aplikasi kelompok kami dibuat dengan kakas framework .Net dengan menggunakan bahasa pemrograman C# sehingga pengerjaan yang kelompok kami lakukan dapat mudah ketika membuat aplikasi berbasis desktop.
5. Program yang kelompok kami buat berhasil untuk menemukan solusi-solusi yang menggunakan algoritma BFS dan DFS dengan baik dan tepat.

B. Saran

Untuk kedepannya, diharapkan bagi kami untuk mempelajari terlebih dahulu bagaimana menggunakan visualisasi graf MSAGL dan cara membuat aplikasi desktop yang baik sehingga waktu pengerjaan dalam membuat aplikasi berbasis desktop ini dapat dipangkas.

C. Refleksi dan Komentar

Melalui tugas besar 2 IF2211 Strategi Algoritma mengenai pengaplikasian BFS dan DFS dalam penjelajahan jaringan pertemanan dalam media sosial ini, kami mendapat banyak sekali pengetahuan yang akan sangat berguna bagi kami secara pribadi dan kelompok di masa yang akan mendatang. Kami mendapat ilmu mengenai aplikasi bahasa pemrograman C# beserta kakas nya berupa framework .Net dan Visual Studio. Kami mendapat ilmu mengenai pembuatan aplikasi berbasis desktop, serta perancangan *class* yang lebih baik lagi karena bahasa pemrograman C# berbasis *object oriented*.

Melalui tugas besar ini pula kami dapat melakukan silaturahmi dan meningkatkan keterbukaan pikiran serta berdiskusi antar sesama baik itu secara akademik maupun secara kehidupan.

DAFTAR PUSTAKA

https://docs.microsoft.com/en-us/dotnet/csharp/, terakhir diakses pada tanggal 24 Maret 2021, pukul 09.02 WIB

http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf , terakhir diakses pada tanggal 24 Maret 2021, pukul 12.20 WIB