

LAPORAN TUGAS BESAR 2

Mata Kuliah IF2211 Strategi Algoritma

Dosen Pengampu: Rinaldi Munir, Nur Ulfa Maulidevi

Dwi Hendratmo Widyatoro, Rila Mandala



Disusun Oleh:

Jeane Mikha Erwansyah 13519116

Cynthia Rusadi 13519118

Jeremia Axel 13519188

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

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.

BAB II

LANDASAN TEORI

2.1 Traversal Graf

Traversal graf merupakan proses mengunjungi setiap simpul dan digunakan untuk mencari jalur dalam suatu graf dari simpul asal ke simpul tujuan dengan mencari jalur terpendek antara kedua simpul tersebut dan menemukan semua jalur yang dapat dilalui dari simpul asal sampai simpul tujuan. Algoritma traversal graf, mengunjungi simpul dengan cara yang sistematik, terbagi menjadi *breadth first search* (pencarian melebar) dan *depth first search* (pencarian mendalam).

2.2 Breadth First Search (BFS)

Breadth First Search (BFS) mempunyai nama lain yaitu pencarian melebar. Algoritmanya adalah mengunjungi simpul v , lalu menyimpan semua simpul yang bertetangga dengan simpul v secara berurutan, dan kemudian mengunjungi simpul yang tadi disimpan satu persatu, demikian seterusnya.

2.3 Depth First Search (DFS)

Depth First Search (DFS) merupakan pencarian mendalam dengan traversal dimulai dari simpul v . Urutan algoritmanya adalah mengunjungi simpul v , mengunjungi simpul w yang bertetangga dengan simpul v , dan ulangi pencarian mendalam mulai dari simpul w . Ketika sudah mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, dilakukan pencarian runut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi. Pencarian akan berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

2.4 C# Desktop Application Development

Integrated Development Environment (IDE) adalah sebuah program yang dapat digunakan untuk berbagai aspek untuk pengembangan *software* dan *desktop application* adalah

aplikasi yang sudah terpasang secara lokal dan dapat di-*install* secara *online* atau dengan menggunakan CD *drive*. *Framework* yang dapat digunakan untuk C# *Desktop Application Development* adalah Visual Studio, ASP.NET Web Matrix, SharpDevelop, PrimalCode, Eclipse, dan lain-lainnya.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

3.1.1 Breadth First Search

1. Memasukkan node akar ke dalam *queue*
2. Mengambil node dari awal *queue* dan dicek apakah node tersebut merupakan solusinya atau bukan. Jika iya, pencarian selesai dan hasilnya dikembalikan.
3. Masukkan seluruh node anak ke dalam *Queue*. Jika *queue* sudah kosong dan sudah melakukan pengecekan terhadap seluruh node, pencarian selesai.
4. Kembali ke nomor 2

3.1.2 Depth First Search

1. Memasukkan node akar ke dalam *stack*
2. Mengambil node dari teratas *stack* dan dicek apakah node tersebut merupakan solusinya atau bukan. Jika iya, pencarian selesai dan hasilnya dikembalikan.
3. Masukkan seluruh node ke dalam *stack*. Jika *stack* sudah kosong dan sudah melakukan pengecekan terhadap seluruh node, pencarian selesai
4. Kembali ke nomor 2

3.1.3 Friend Recommendations

1. Menyimpan node-node tetangga dari node akar ke sebuah *list A*
2. Menyimpan node-node tetangga dari *list A* ke dalam *list B*
3. Melakukan pengecekan node-node tetangga yang beririsan antara node akar dengan *list B*

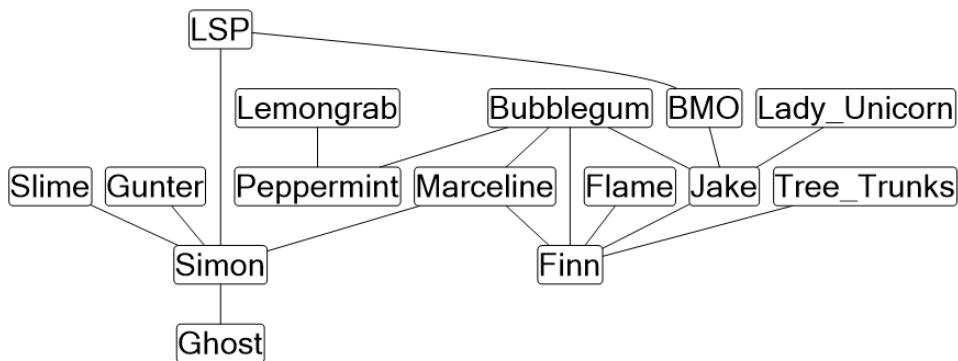
3.2 Proses Mapping

Program akan menerima sebuah file dengan *extension .txt* dan akan menghasilkan sebuah graf tidak berarah G . Untuk fitur *explore friends*, yang menggunakan algoritma *Breadth First Search* dan *Depth First Search*, akan menerima 2 *input* dari pengguna, yaitu akun awal dan akun akhir yang ingin dicari hubungannya. Kedua *input* ini awalnya merupakan sebuah *string* dan

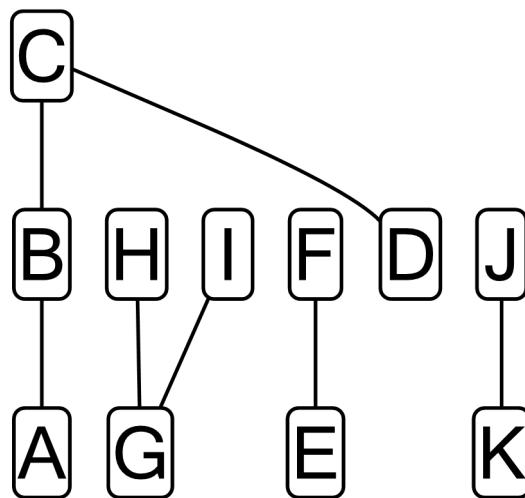
program akan mengubahnya menjadi sebuah node yang tersedia di dalam graf tidak berarah G dengan *output* dari algoritma tersebut adalah sebuah *list* dari *string*, yang merupakan jalur pertemanan di antara kedua akun tersebut. Selain fitur *explore friends*, terdapat fitur *friend recommendations* juga. *Input* yang diterima hanya 1, yaitu akun awal, dan bersifat *string*. Program akan mengubah *string* tersebut menjadi sebuah node yang tersedia di dalam graf tidak berarah G dan *output* dari algoritma tersebut adalah *list* dari friendRec (kelas yang terdiri dari nama teman yang direkomendasi, *list* dari *string* teman mutual, dan jumlah teman mutual).

3.3 Ilustrasi Kasus

- Kasus 1, graf tidak berarah:



- Kasus 2, hutan:



BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

4.1.1 Main Program

```
list_node <- []
src <- ... # masukan pengguna
dest <- ... # masukan pengguna

for item in list_node
    if item merupakan edge
        undirectedGraph <- edge item[0] dan item[1]
    else if item merupakan node
        undirectedGraph <- node item[0]
    endif
endfor

if dest ≠ null then
    if searching_algorithm = BFS
        undirectedGraph.BFS(src, dest)
    else if searching_algorithm = DFS
        undirectedGraph.DFS(src, dest)
    endif
endif

showGraph()
showFriendsRecommendation()
```

4.1.2 Algoritma Breadth First Search

```
BFS (start_node, end_node):
    queue_jalur = []
    jalur = []
    jalur.enqueue(start_node)
    queue_jalur.enqueue(jalur)
    while (queue_jalur > 0) :
        jalur_saat_ini = queue_jalur.dequeue()
        node_saat_ini = elemen terakhir dari jalur_saat_ini
        if node_saat_ini adalah end_node:
            return jalur_saat_ini
        endif
        for node_tetangga in node_saat_ini.tetangga :
            if node_tetangga not in jalur_saat_ini:
                jalur_baru = jalur_saat_ini
                queue_jalur.enqueue(jalur_baru)
```

```

        endif
    endfor
endwhile
```

4.1.3 Algoritma Depth First Search

```

DFS (start_node, end_node):
    stack_jalur = []
    jalur = []
    dikunjungi = []
    backtrack = false
    lastAdded = false
    node_saat_ini= start_node
    while (node_saat_ini != end_node):
        if not backtrack:
            stack.pop()
            for node_tetangga in node_saat_ini.tetangga:
                if node_tetangga not in dikunjungi and
                node_tetangga not in jalur:
                    stack.push(node_tetangga)
                endif
            endfor
        else:
            if node_saat_ini = path[idx_terakhir] and
            node_saat_ini terhubung dengan stack.top():
                node_saat_ini = stack.top()
                if node_saat_ini = end_node:
                    break
                endif
                stack.pop()
                for node_tetangga in node_saat_ini.tetangga:
                    if node_tetangga not in dikunjungi and
                    node_tetangga not in jalur:
                        stack.push(node_tetangga)
                    endif
                endfor
            endif
        endif

        if node_saat_ini terhubung dengan stack.top():
            backtrack = false
            jalur.enqueue(node_saat_ini)
            node_saat_ini = stack.top()
        else:
            backtrack = true
            dikunjungi.enqueue(node_saat_ini)
            jalur.dequeue()
            node_saat_ini = path[idx_terakhir]
    endif
```

```

        if path[idx_terakhir] = end_node:
            lastAdded = true
            break
        endif
    endwhile

    if not lastAdded:
        path.enqueue(node_saat_ini)
    endif

```

4.1.4 Algoritma Friend Recommendations

```

recFriends(start_node):
    teman = start_node.tetangga
    namaRekomen = []
    temanMutual= []

    for namaTeman in teman:
        for nama in namaTeman.tetangga:
            if nama not in namaRekomen and nama != start_node and
               nama tidak terhubung dengan teman:
                namaRekomen.enqueue(nama)
            endif
        endfor
    endfor

    i = 0
    for namaDirekomendasi in namaRekomen:
        for namaTeman in teman:
            if namaTeman terhubung dengan namaDirekomendasi:
                temanMutual[i].enqueue(namaTeman)
            i++
        endif
    endfor
endfor

return namaRekomen, temanMutual

```

4.2 Struktur Data

Untuk setiap algoritma, struktur datanya adalah sebagai berikut:

- BFS:
 1. Graf: menggunakan list ketetanggaan. Tiap simpul di dalam graf, memiliki list simpul yang terhubung dengannya. Struktur data ini dipilih karena lebih efisien secara spasial.

2. *Queue* dari Node *List* penyimpanan *_path*: untuk menyimpan node-node yang dapat dilalui/diperiksa. Menggunakan Queue karena BFS menerapkan algoritma *queue*.
 3. Node *List* *currentPath*: untuk menyimpan jalur yang akan dilalui oleh algoritma BFS dari node awal sampai node akhir
 4. Node *List* *new_path*: untuk menyimpan jalur yang telah dilalui oleh suatu node
- DFS:
 1. Graf: menggunakan list ketetanggaan. Tiap simpul di dalam graf, memiliki list simpul yang terhubung dengannya. Struktur data ini dipilih karena lebih efisien secara spasial.
 2. string *List* *path*: untuk menyimpan jalur yang akan dilalui oleh algoritma DFS dari node awal sampai node akhir
 3. string *Stack* *stack*: untuk menyimpan node-node yang dapat dilalui/diperiksa. Menggunakan Stack karena DFS menerapkan algoritma *stack*.
 4. string *List* *visited*: untuk menyimpan node-node apa saja yang tidak perlu dilalui/diperiksa kembali karena menjamin bahwa node tersebut tidak akan memiliki hasil. Digunakan untuk keperluan *backtracking*.
 - Friend Recommendation:
 1. Node *List* *temen*: untuk menyimpan nama teman-teman dari *input*
 2. friendRec *List* *recommended*: untuk menyimpan nama-nama dari teman yang direkomendasi, beserta nama teman mutual, dan jumlah teman mutual untuk setiap nama dari teman yang direkomendasi
 3. string *List* *RecFrens*: untuk menyimpan nama-nama dari teman yang direkomendasi. Digunakan agar nama teman-teman yang direkomendasi bersifat unik.

4.3 Tata Cara Penggunaan Program

1. Menekan tombol ‘Browse’ dan memilih sebuah *file* dengan *extension .txt*
2. Memilih algoritma BFS atau DFS (tidak dipilih keduanya sekaligus) untuk fitur *explore friends*
3. Memilih akun untuk ‘Choose Account’ dari *dropdown*-nya, sebagai akun awal untuk fitur *explore friends* dan mencari teman rekendasinya
4. Memilih akun untuk ‘Explore friends with’ dari *dropdown*-nya, sebagai akun akhir untuk fitur *explore friends*

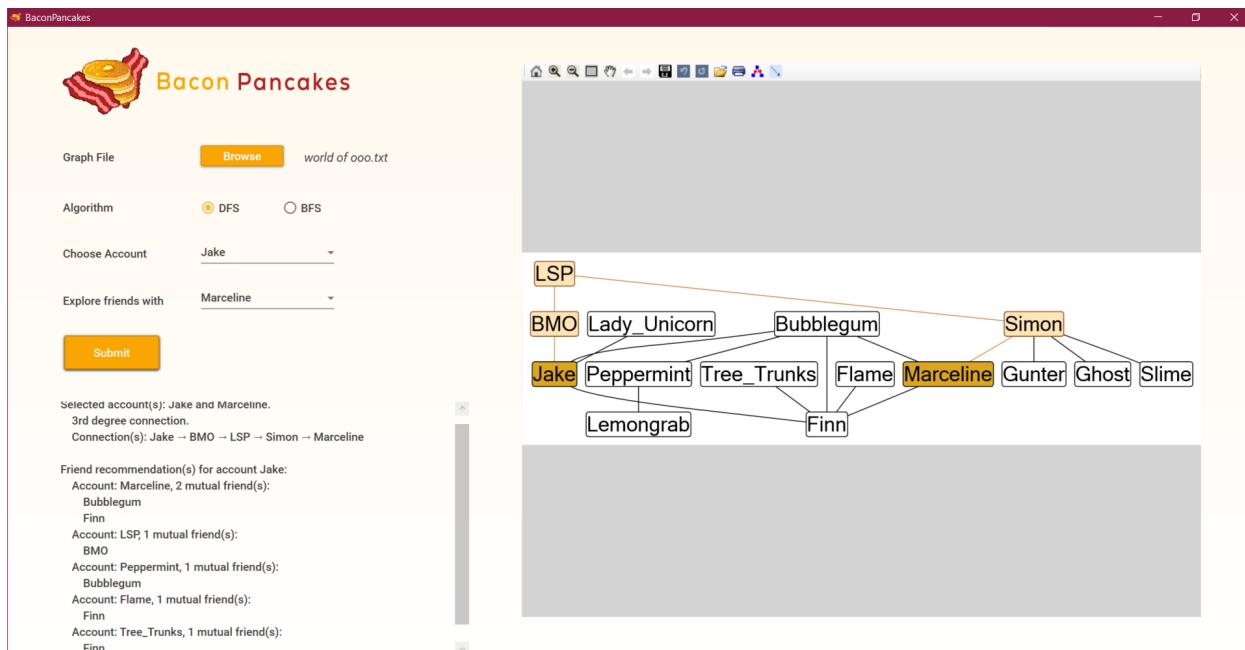
5. Menekan tombol ‘Submit’ untuk melihat jalur pertemanan dari akun awal dengan akun akhir dan melihat nama teman-teman yang direkomendasi untuk akun awal, beserta memperlihatkan nama teman-teman mutualnya
6. Node-node pada graf dapat dipindahkan sesuai keinginan pengguna dengan menggunakan *cursor*
7. Jika ingin mengganti graf, kembali ke nomor 1
8. Jika ingin mengganti algoritma, kembali ke nomor 2

4.4 Hasil Pengujian

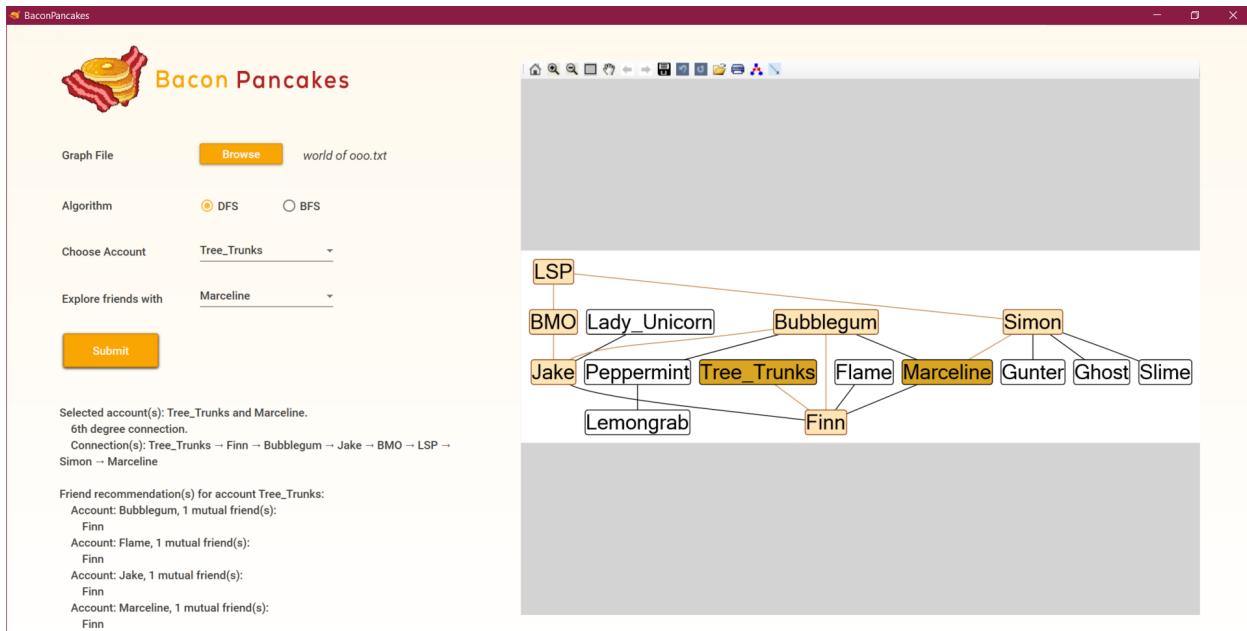
Screenshot di bawah merupakan hasil pengujian untuk setiap skenario dan setiap penggunaan algoritma BFS dan DFS untuk fitur *explore friends*. Fitur *friend recommendations* akan memberikan hasil yang sama untuk kedua algoritma.

4.4.1 Graf tidak berarah

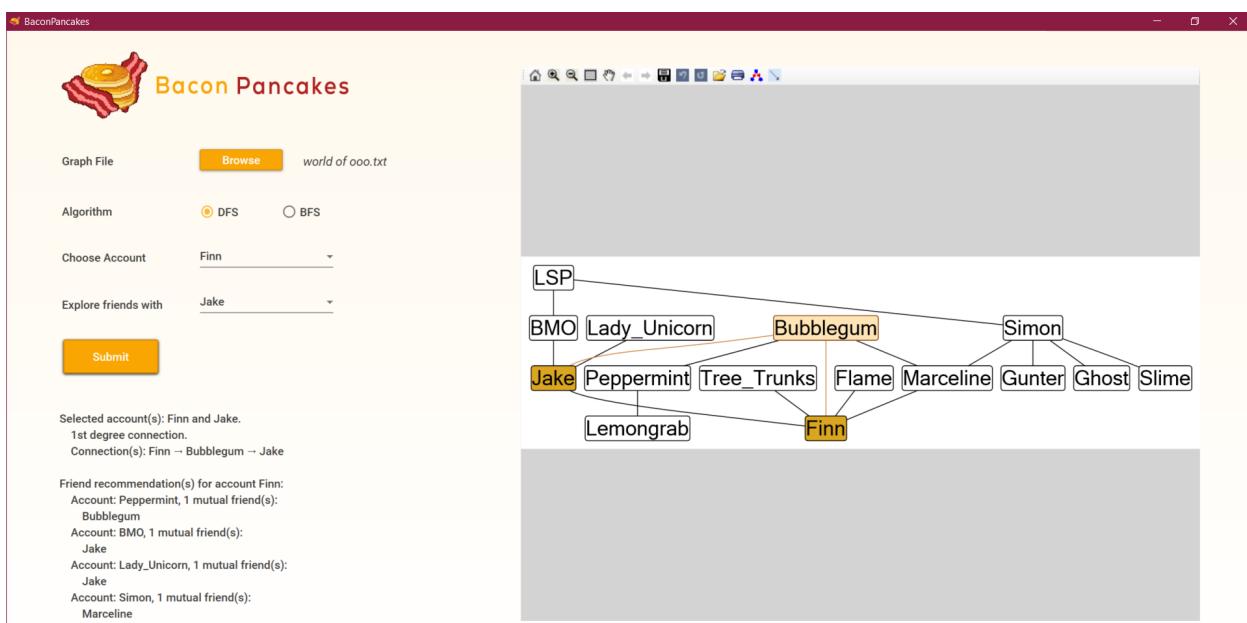
- DFS



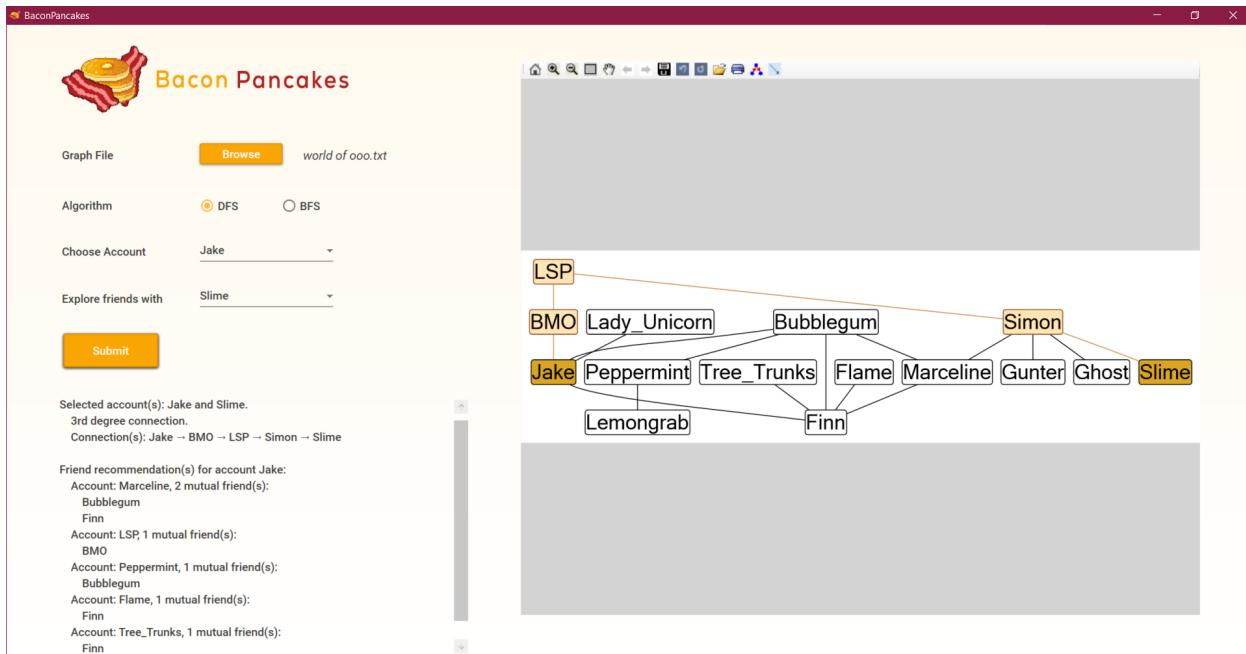
Gambar 4.4.1.1 DFS Graf Tidak Berarah (1)



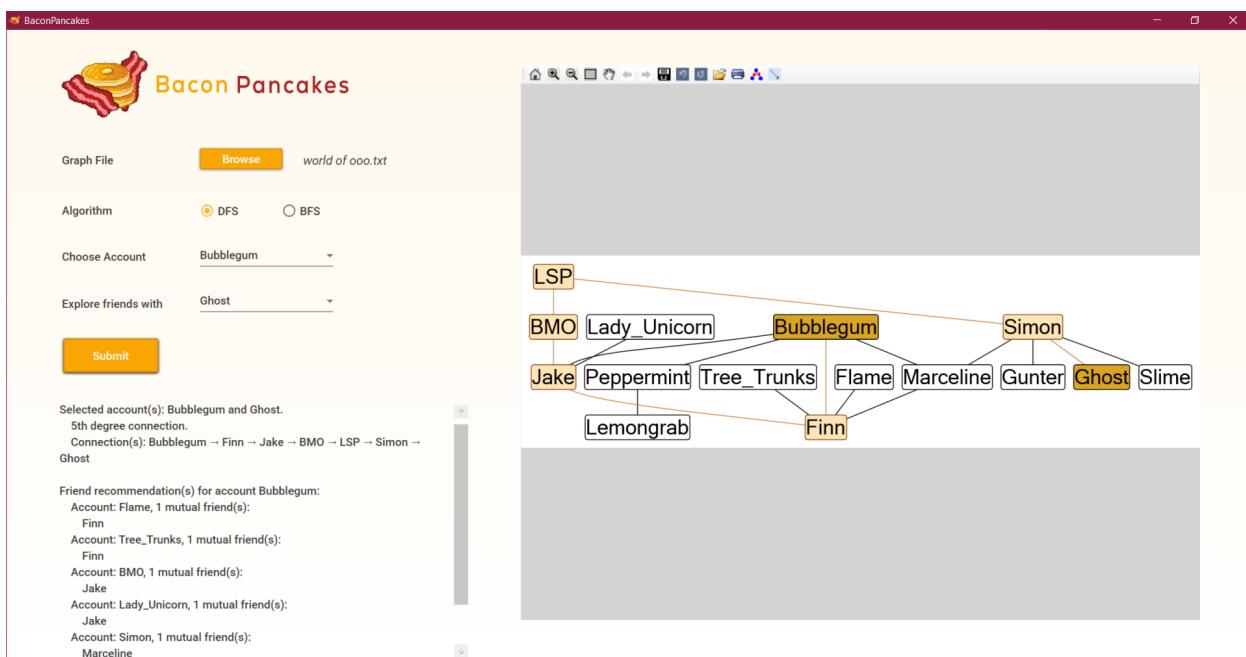
Gambar 4.4.1.2 DFS Graf Tidak Berarah (2)



Gambar 4.4.1.3 DFS Graf Tidak Berarah (3)

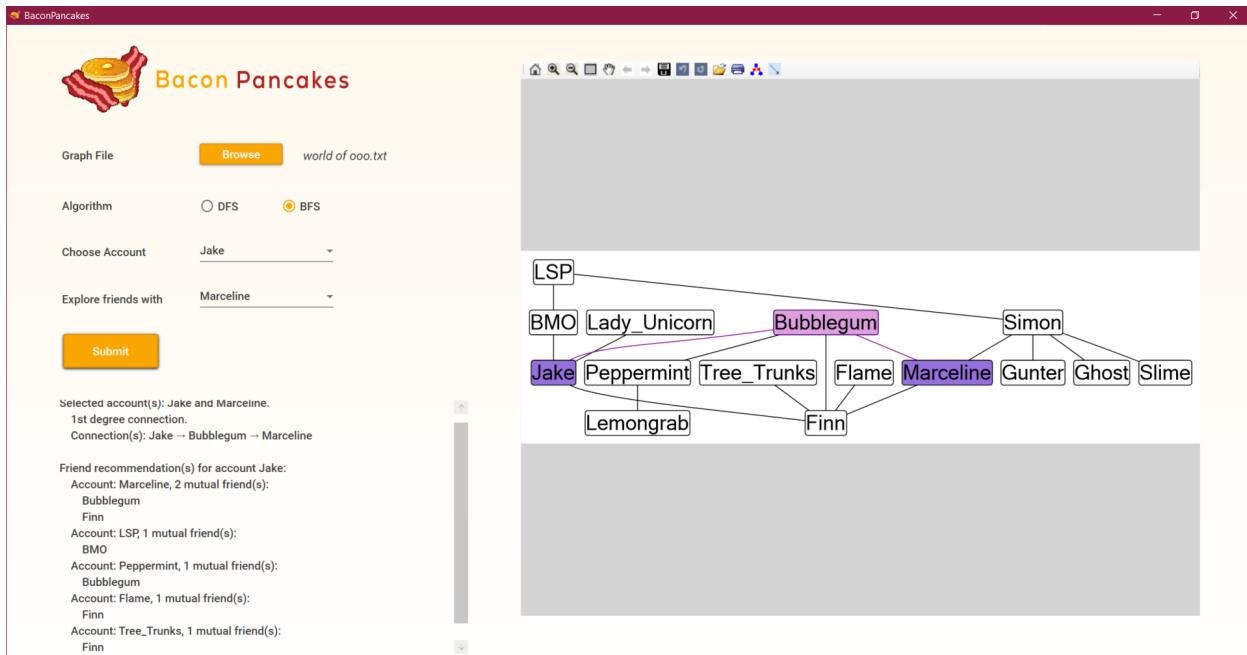


Gambar 4.4.1.4 DFS Graf Tidak Berarah (4)

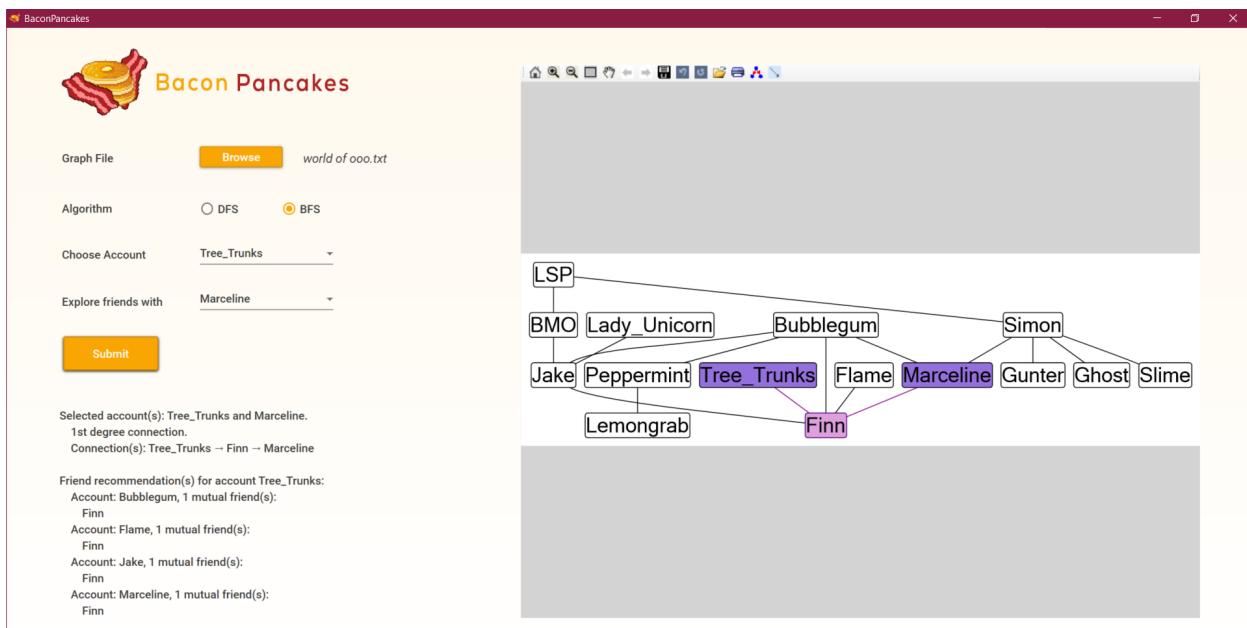


Gambar 4.4.1.5 DFS Graf Tidak Berarah (5)

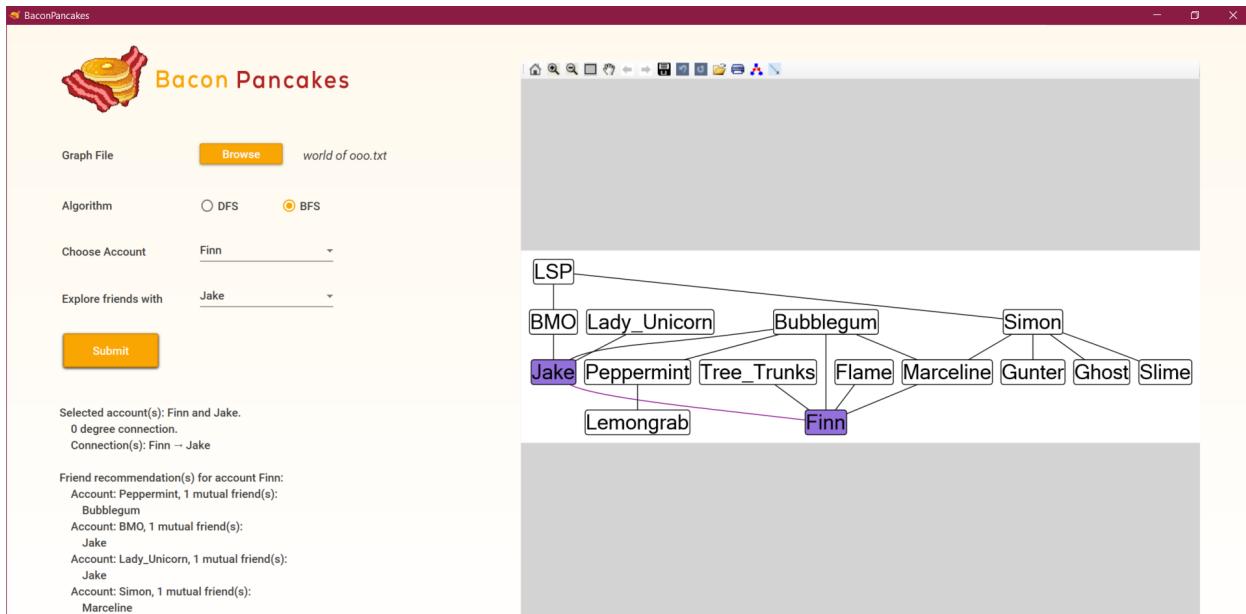
- BFS



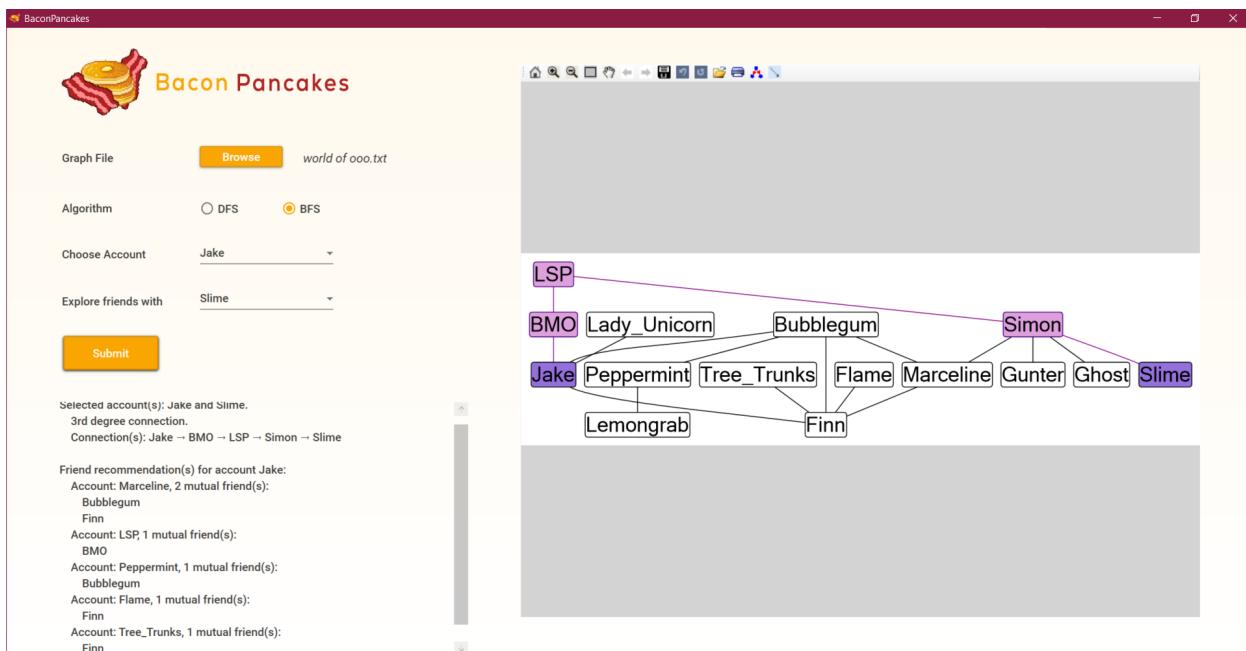
Gambar 4.4.1.6 BFS Graf Tidak Berarah (1)



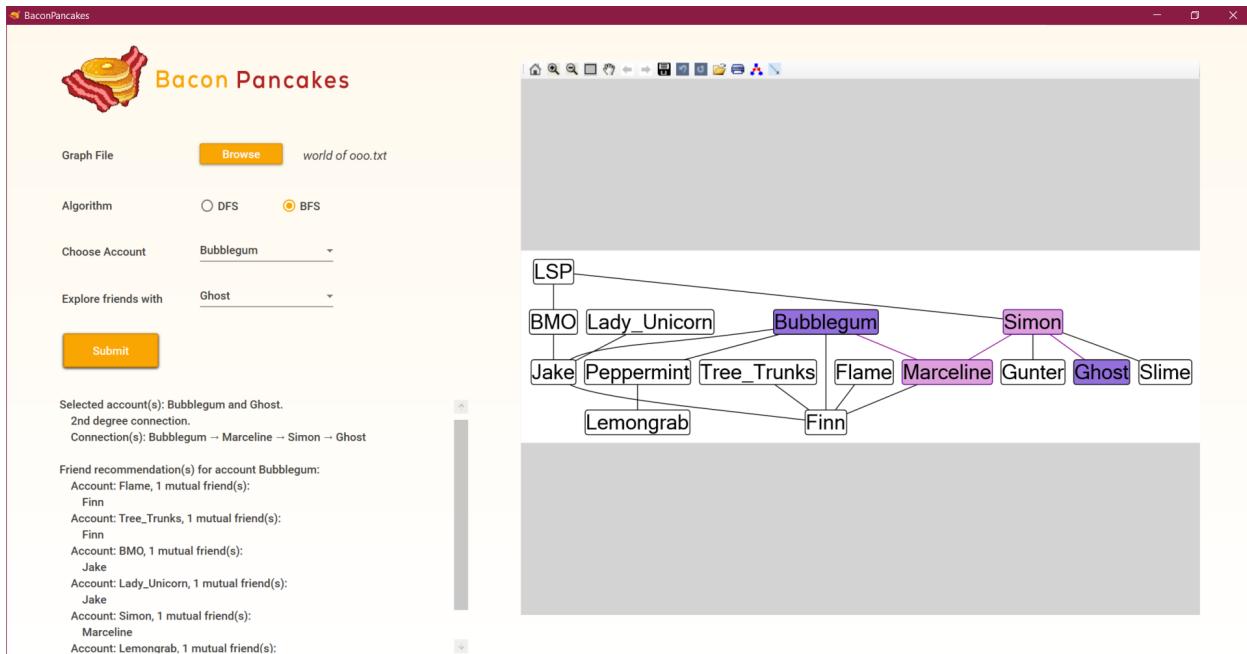
Gambar 4.4.1.7 BFS Graf Tidak Berarah (2)



Gambar 4.4.1.8 BFS Graf Tidak Berarah (3)



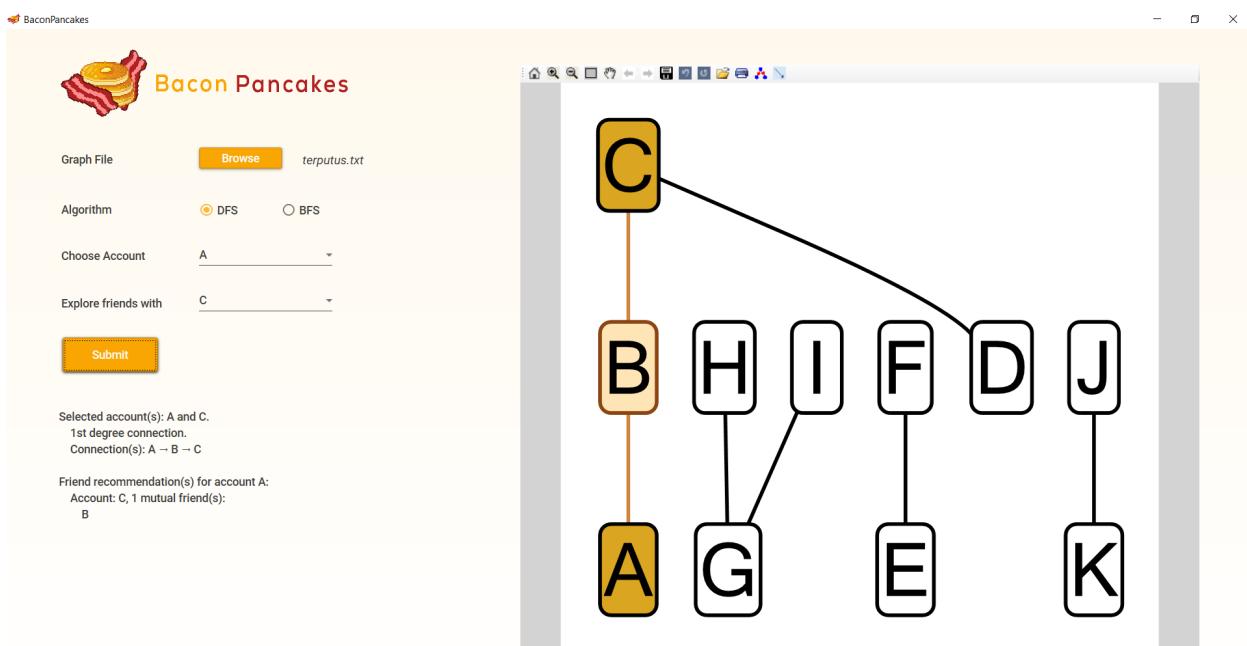
Gambar 4.4.1.9 BFS Graf Tidak Berarah (4)



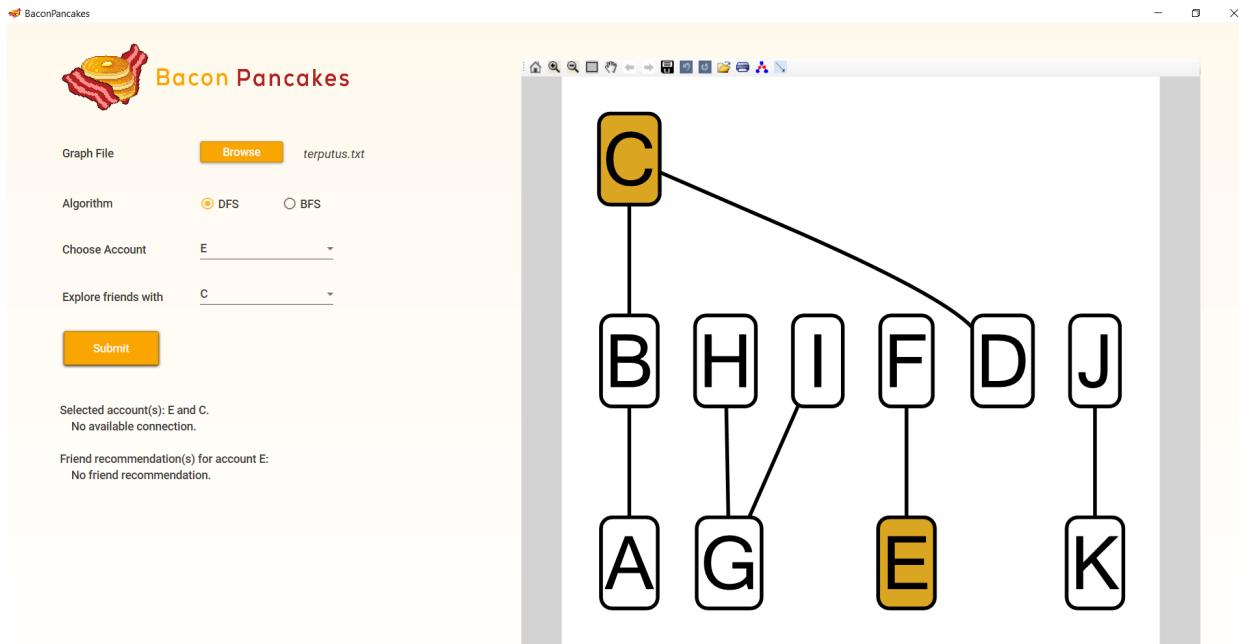
Gambar 4.4.1.10 BFS Graf Tidak Berarah (5)

4.4.2 Hutan

- DFS

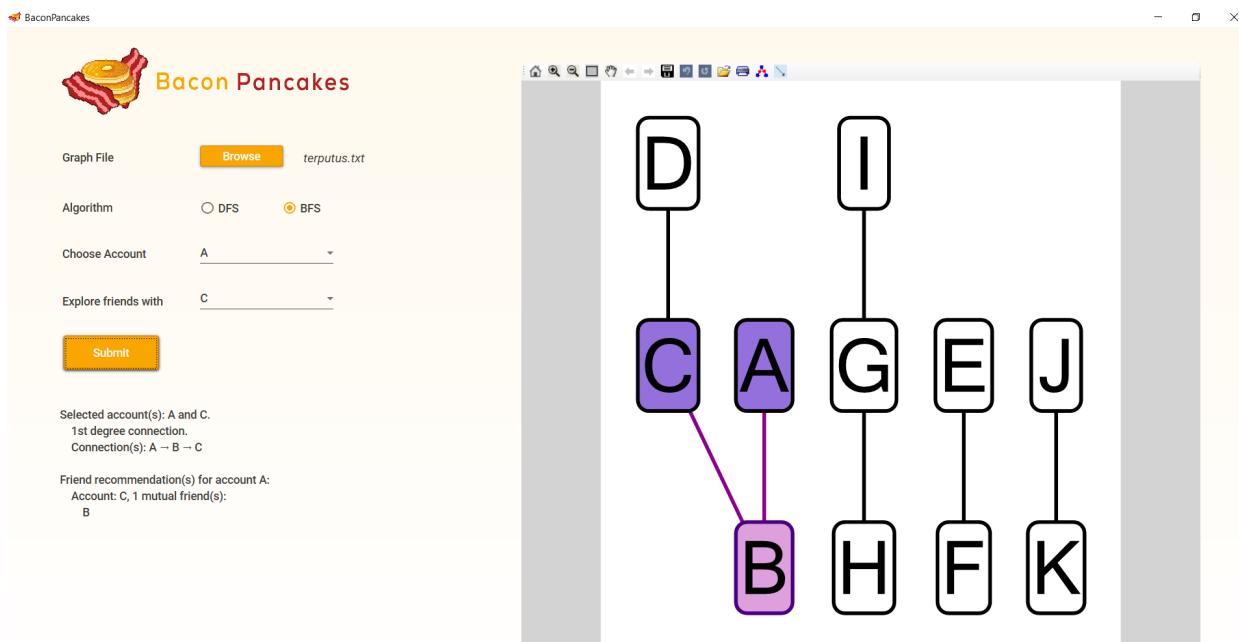


Gambar 4.4.2.1 DFS Hutan (1)

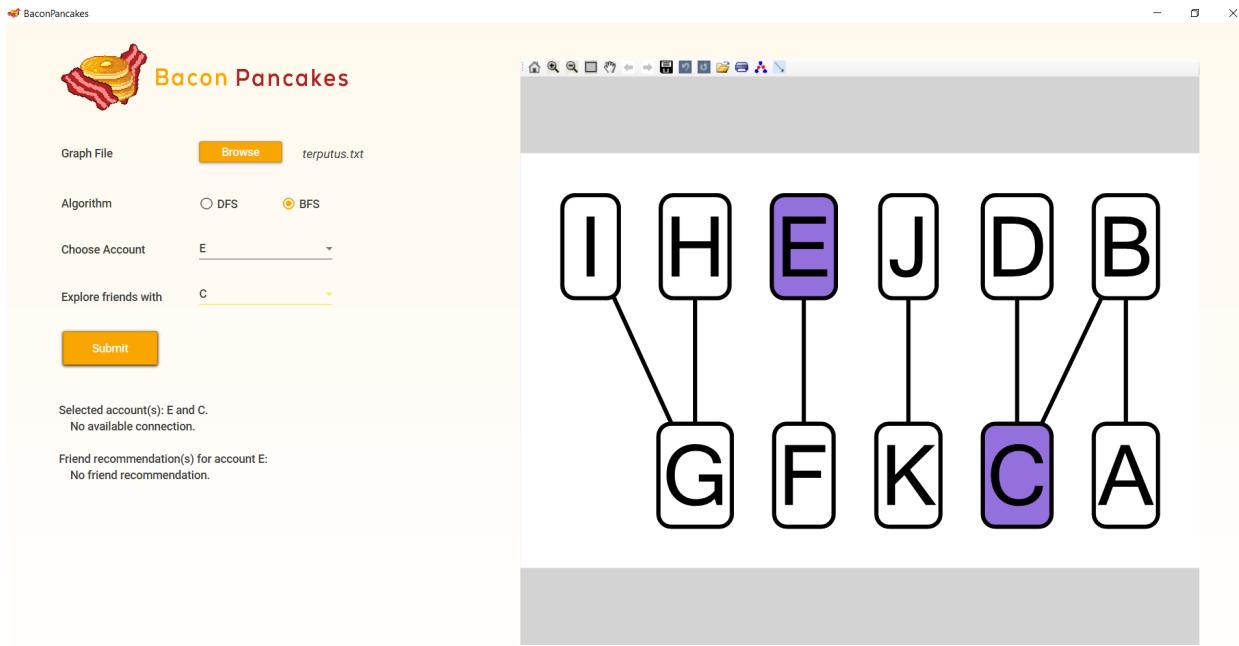


Gambar 4.4.2.2 DFS Hutan (2)

- BFS



Gambar 4.4.2.3 BFS Hutan (1)



Gambar 4.4.2.4 BFS Hutan (2)

4.5 Analisis Desain Solusi Algoritma BFS dan DFS

Dengan menggunakan 2 ilustrasi kasus, yaitu graf tidak berarah dan berupa hutan, didapatkan kesimpulan bahwa algoritma BFS dan DFS tidak mempunyai pengaruh yang besar untuk ilustrasi hutan, dengan contoh Gambar 4.4.2.1 dengan Gambar 4.4.2.3, dan Gambar 4.4.2.2 dengan Gambar 4.4.2.4. Kedua pasangan tersebut tidak memiliki perbedaan untuk jalurnya. Sedangkan, untuk ilustrasi graf tidak berarah dapat dilihat bahwa terdapat perbedaan jalur untuk algoritma BFS dan DFS. Untuk Gambar 4.4.1.1 dengan Gambar 4.4.1.6, algoritma DFS memiliki *3rd degree connection* dan algoritma BFS hanya memiliki *1st degree connection*. Gambar 4.4.1.2 (algoritma DFS) memiliki *6th degree connection*, memiliki perbedaan dengan Gambar 4.4.1.7 (algoritma BFS), hanya memiliki *1st degree connection*. Untuk jalur dari Finn ke Jake, tidak ada perbedaan yang signifikan (algoritma DFS memiliki *1st degree connection* dan algoritma BFS memiliki *0 degree connection*) dan jalur dari Jake ke Slime tidak memiliki perbedaan, dengan keduanya memiliki *3rd degree connection* dan jalur yang sama. Kasus yang terakhir, menggunakan Gambar 4.4.1.5 dan Gambar 4.4.1.10, juga memiliki perbedaan yang dapat dilihat, yaitu algoritma DFS memiliki *5th degree connection* dan algoritma BFS hanya memiliki *2nd degree connection*.

Secara keseluruhan, dapat disimpulkan bahwa algoritma BFS dapat menghasilkan jalur yang lebih pendek dibandingkan dengan algoritma DFS, hal ini dikarenakan strategi BFS mencari node-node yang merupakan tetangganya terlebih dahulu dan DFS akan mencari node sampai ke ujung. Dengan menggunakan strategi DFS, program tidak menemukan node target sampai ujungnya dan harus melakukan *backtracking* sampai node-node sebelumnya yang tetangga-tetangganya belum diperiksa dan menggunakan strategi BFS berarti program tidak akan melakukan *backtracking*. Hal tersebut mempengaruhi jalur yang dihasilkan untuk setiap algoritma.

Tidak hanya strategi BFS yang memiliki keunggulan, tetapi strategi DFS juga memiliki keunggulan untuk beberapa kasus. Salah satu contohnya adalah pada Gambar 4.4.2.1 (strategi DFS) dengan Gambar 4.4.2.6 (strategi BFS), jalur dari Jake dan Marceline menggunakan strategi DFS tidak memerlukan *backtracking* sama sekali, yang berarti lebih menghemat memori, sedangkan strategi BFS akan lebih memakan memori karena harus mengecek setiap node-node yang ada di sekitarnya sampai menemukan tujuannya.

Kedua strategi memiliki keunggulannya masing-masing dan untuk pemilihan strategi yang ingin digunakan dapat bergantung pada keinginan pengguna, yaitu ingin mendapatkan jalur terpendek atau yang lebih hemat secara langkah. Jumlah node pada graf yang digunakan sebagai contoh masih relatif sedikit dan mungkin saja akan ada kasus dengan graf yang memiliki node relatif jauh lebih banyak dibandingkan contoh yang telah digunakan. Alangkah baiknya jika ditentukan terlebih dahulu apa yang diharapkan (jalur terpendek atau penghematan memori) sebelum menentukan strateginya.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Tugas besar ini dapat dikerjakan dengan baik jika sudah mengerti alur penggerjaan strategi DFS dan BFS dengan baik. Aplikasi sederhana mengenai hubungan pertemanan antara orang-orang dapat dibuat dengan menggunakan bahasa pemrograman C# dengan kedua strategi tersebut dan kedua strategi tersebut dapat menghasilkan jalur dan *degree connection* yang berbeda-beda, dan juga mencari teman yang direkomendasi. Pemilihan strategi dapat bergantung pada pengguna jika ingin mendapatkan jalur terpendek (algoritma BFS) atau memori yang relatif lebih optimal (algoritma DFS).

5.2 Saran

Agar dapat mempermudah penggerjaan tugas besar, disarankan untuk melatih membaca dokumentasi, karena C# merupakan hal yang baru dan dokumentasi mengenai C# akan sangat membantu. Penggerjaan juga sebaiknya dikerjakan bersama agar dapat menyelesaiannya dengan relatif sangat cepat dan mudah. Selain itu, jika spesifikasi PC atau *laptop* tidak cukup untuk *download* Visual Studio, tidak akan mengerjakan GUI, atau operasi sistem yang digunakan bukan Windows, lebih baik *install* .Net untuk kebutuhan *compile*.

5.3 Refleksi dan Komentar

Alangkah baiknya jika *deadline* tugas besar 2 Strategi Algoritma tidak ber-'tabrak'-an dengan *deadline* tugas besar yang lainnya, karena menyebabkan ke-keos-an, dengan sisi positifnya menjadi dapat belajar manajemen waktu.

DAFTAR PUSTAKA

https://share.cocalc.com/share/5bc399842fed9875d75f21d8e9c505aad184b36d/LectureNotes/8_3-GraphTraversal.ipynb#:~:text=Graph%20Traversal%20merupakan%20proses%20menunjungi,titik%20asal%20ke%20titik%20tujuan diakses pada Maret 2021

<http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>
diakses pada Maret 2021

<https://www.techrepublic.com/article/explore-alternatives-to-visual-studio-net/> diakses pada Maret 2021

<https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/visual-studio-ide?view=vs-2019>
diakses pada Maret 2021