Laporan Tugas Besar 2

Tugas Besar II IF2211 Strategi Algoritma

Semester II Tahun 2020/2021

Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling

Oleh kelompok filePedia:

Maharani Ayu Putri Irawan (13520019)

Rizky Ramadhana P. K. (13520151)

Vito Ghifari (13520153)

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

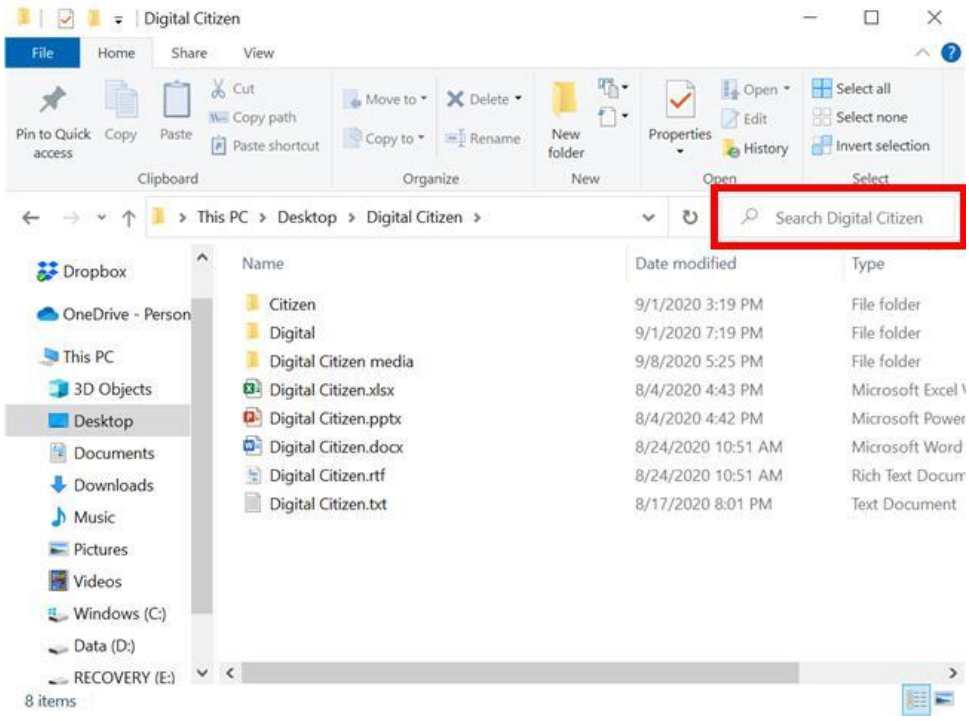
2022

Daftar Isi

BAB 1

Deskripsi Masalah

Pada saat kita ingin mencari file spesifik yang tersimpan pada komputer kita, seringkali *task* tersebut membutuhkan waktu yang lama apabila kita melakukannya secara manual. Bukan saja harus membuka beberapa folder hingga dapat mencapai directory yang diinginkan, kita bahkan dapat lupa di mana kita meletakkan file tersebut. Sebagai akibatnya, kita harus membuka berbagai folder secara satu persatu hingga kita menemukan file yang diinginkan. Hal ini pastinya akan sangat memakan waktu dan energi.



Gambar 1. Fitur Search pada Windows 10 File Explorer

(Sumber: https://www.digitalcitizen.life/wp-content/uploads/2020/10/explorer_search_10.png)

Meskipun demikian, kita tidak perlu cemas dalam menghadapi persoalan tersebut sekarang. Pasalnya, hampir seluruh sistem operasi sudah menyediakan fitur search yang dapat digunakan untuk mencari file yang kita inginkan. Kita cukup memasukkan query atau kata kunci pada kotak pencarian, dan komputer akan mencarikan seluruh file pada suatu starting directory (hingga seluruh children-nya) yang berkorespondensi terhadap query yang kita masukkan.

Fitur ini diimplementasikan dengan teknik folder crawling, di mana mesin komputer akan mulai mencari file yang sesuai dengan query mulai dari starting directory hingga seluruh children dari starting directory tersebut sampai satu file pertama/seluruh file ditemukan atau tidak ada file yang ditemukan. Algoritma yang dapat dipilih untuk melakukan crawling tersebut pun dapat bermacam-macam dan setiap algoritma akan memiliki teknik dan konsekuensinya sendiri. Oleh karena itu, penting agar komputer memilih algoritma yang tepat sehingga hasil yang diinginkan dapat ditemukan dalam waktu yang singkat.

BAB 2

Landasan Teori

2.1 Graph Traversal

Graph traversal atau penelusuran graf merupakan metode untuk mengunjungi simpul-simpul pada suatu graf secara sistematis. Algoritma *graph traversal* dapat digunakan sebagai pencarian solusi dari suatu masalah dengan merepresentasikan masalah tersebut sebagai graf. Pada dasarnya, algoritma pencarian solusi dengan graf terbagi menjadi dua, yaitu pencarian tanpa informasi (*uninformed search/blind search*) dan pencarian dengan informasi (*informed search*). Beberapa jenis algoritma pencarian solusi dengan *graph traversal* tanpa informasi adalah DFS, BFS, *Depth Limited Search, Iterative Deepening Search, Uniform Cost Search*. Sementara itu, salah satu algoritma pencarian dengan informasi adalah algoritma A* (A-star). Algoritma ini digunakan sebagai pencarian solusi berbasis heuristik yang memanfaatkan graf berbobot. Masing-masing algoritma pencarian solusi dengan graf mempunyai pendekatan dan cara tersendiri. Namun, pada makalah ini, algoritma yang dibahas adalah algoritma BFS dan DFS.

2.2 Algoritma Penelusuran Pada Graf

2.2.1 Breadth-First Search (BFS)

Breadth-First Search merupakan algoritma pencarian simpul pada graf secara *layer per layer*. Mula-mula simpul yang menjadi akar akan dikunjungi, lalu masukkan semua anak dari simpul tersebut ke antrean berupa ADT Queue. Tidak ada pembatasan bagaimana urutan memasukkan anak-anak dari simpul tersebut ke dalam Queue, hanya saja sering digunakan aturan sesuai urutan abjad. Simpan informasi mengenai simpul yang baru saja dikunjungi agar simpul tersebut tidak dikunjungi dua kali. Lalu kunjungi simpul berikutnya sesuai dengan antrean pada Queue dan aturan *first in first out.* Pencarian selesai ketika tidak ada simpul yang tersisa pada Queue atau simpul yang dicari telah ditemukan.

2.2.2 Depth-First Search (DFS)

Depth-First Search merupakan algoritma pencarian simpul pada graf yang terus menelusuri sampai tidak terdapat anak yang bisa ditelusuri lagi lalu penelusuran dilakukan ke

upagraf yang lain. Mula-mula simpul yang menjadi akar akan dikunjungi, lalu setiap anak dari simpul tersebut dimasukkan ke dalam antrean berupa ADT Stack. Simpul yang baru saja dikunjungi harus dicatat sehingga simpul tersebut tidak dikunjungi dua kali. Simpul berikutnya yang akan dikunjungi dipilih berdasar antrean yang disimpan dalam Stack dan aturan *last in first out.*

2.3 Pengembangan Aplikasi Desktop Berbasis C#

Salah satu implementasi algoritma BFS dan DFS adalah dengan membuat aplikasi desktop yang memanfaatkan kedua hal tersebut. Pengembangan aplikasi desktop dapat dilakukan dengan berbagai cara, salah satunya adalah dengan menggunakan bahasa pemrograman C#. C# merupakan Bahasa pemrograman yang dikembangkan oleh Microsoft yang berjalan di atas kerangka kerja .NET. Bahasa pemrograman ini memiliki paradigma berorientasi objek dimana satu file mewakili satu kelas. Untuk mengembangkan aplikasi dengan Bahasa C#, digunakan lingkungan pengembangan terintegrasi (*integrated development environment*) Visual Studio. IDE tersebut memungkinkan pengguna untuk mengelola file di suatu folder sampai menjalankannya tanpa perlu menginstalasi *compiler* nya secara individual.

Pada bahasa pemrograman C# dapat dibuat antarmuka visual untuk mendapatkan input dari pengguna dan menampilkan output ke pengguna. Antarmuka visual tersebut dibuat menggunakan Winforms. Antarmuka yang dibuat bisa beragam jenisnya, mulai dari form untuk menerima input berupa teks, *radio button* untuk menerima pilihan, sampai tombol. Antarmuka yang dibuat juga bisa menampilkan output dari program yang dijalankan. Untuk menggambar keluaran program berupa pohon pencarian, digunakan pustaka Microsoft Automatic Graph Layout (MSAGL).

MSAGL merupakan pustaka .NET untuk membuat *layout* graf secara otomatis. Cara kerja pembuatan graf dengan MSAGL adalah dengan membuat objek graf. Setelah itu, objek graf tersebut dapat diubah dengan melakukan penambahan *node* ataupun penambahan *edge*. Penambahan *edge* untuk *node* yang belum ada akan membuat *node* secara otomatis. Di lain sisi, *node* juga dapat dicari berdasarkan ID nya agar *node* tersebut dapat diubah *properties*-nya. Pada konteks *crawling* file, *node* digunakan untuk menandakan eksistensi sebuah file atau folder, sedangkan *edge* berarah digunakan untuk menandakan hubungan suatu file atau folder terhadap *parent folder*-nya.

BAB 3

Analisis Pemecahan Masalah

3.1 Langkah Pemecahan Masalah

Mula-mula diminta nama file yang akan dicari dan folder tempat pencarian bermula. Setelah itu akan diinstansiasi sebuah objek Graph dengan folder tempat pencarian bermula menjadi salah satu atributnya. Terdapat dua pilihan pada proses pencarian, yang pertama pencarian akan berhenti sesaat setelah file ditemukan. Sedangkan pilihan kedua akan meneruskan pencarian sampai semua file ditemukan dalam folder tempat pencarian bermula atau folder-folder di bawahnya.

Dalam proses pencarian dapat digunakan dua pilihan algoritma yaitu *breadth first search* (BFS) dan *depth first search* (DFS). Pilihan algoritma tersebut menentukan folder mana yang akan dikunjungi terlebih dahulu. Untuk setiap folder yang dikunjungi akan diperiksa seluruh file yang berada pada folder tersebut apakah namanya sama dengan nama file yang hendak dicari. Kemudian nama-nama folder anak pada folder yang dikunjungi saat ini akan dimasukkan ke dalam antrean untuk selanjutnya dikunjungi juga. Digunakan aturan *first in first out* untuk algoritma BFS dan *last in first out* untuk algoritma DFS.

3.2 Pemetaan Persoalan ke Elemen BFS dan DFS

Terdapat tiga elemen dalam algoritma BFS dan DFS yaitu matriks ketetanggaan, antrean simpul, dan tabel boolean. Pertama-tama harus didefinisikan apa yang menjadi simpul graf dalam permasalahan pencarian file ini. Yang akan menjadi simpul pohon dalam permasalahan ini adalah folder. Satu folder mewakili satu simpul. Folder yang berada di dalam folder lain maka akan menjadi anak dari simpul folder orang tua nya.

Elemen pertama, matriks ketetanggaan, menyatakan simpul mana yang bisa diakses dari suatu simpul tertentu. Pada permasalahan ini tidak digunakan matriks ketetanggaan tetapi setiap kali mengunjungi sebuah folder akan dicari folder apa saja yang berada di dalam folder saat ini menggunakan *method* GetDirectories yang dimiliki oleh kelas DirectoryInfo pada Bahasa C#. Pada intinya, *method* ini juga bisa menghasilkan simpul mana saja yang bisa diakses dari simpul saat ini walaupun bentuknya berbeda dengan matriks ketetanggaan.

Elemen kedua, antrean simpul, menentukan simpul mana yang akan dikunjungi berikutnya. Untuk algoritma BFS, antrean diimplementasikan dengan sebuah ADT Queue yang

disimpan dalam variabel q. Untuk algoritma DFS, antrean diimplementasikan dengan sebuah ADT Stack yang disimpan dalam sebuah variabel bernama s.

Elemen ketiga, tabel boolean, menyimpan informasi mengenai simpul mana saja yang telah dikunjungi. Pada permasalahan ini, elemen tabel boolean tidak diimplementasikan mengingat struktur folder-folder yang akan ditelusuri identik dengan struktur sebuah pohon yang tidak mengandung sirkuit. Sehingga semua folder yang akan dikunjungi berikutnya bisa dipastikan belum pernah dikunjungi sebelumnya.

3.3 Contoh Ilustrasi Kasus Lain

Salah satu contoh lain dimana proses penyelesaiannya menggunakan algoritma DFS dan BFS juga adalah web crawler. Web crawler menerima suatu halaman yang ingin dicari dan halaman tempat memulai pencarian. Sama seperti permasalahan pencarian file ini yang menerima nama file yang ingin dicari dan folder tempat pencarian bermula. Suatu web crawler juga akan mengunjungi semua halaman yang tautannya berada pada halaman yang saat ini sedang dikunjungi. Sama dengan permasalahan pencarian file yang akan mengunjungi semua folder yang menjadi anak dari folder yang saat ini dikunjungi.

BAB 4

Implementasi dan Pengujian

A. Implementasi Program

```
function BFS(root, searched, mode) → array of string
    q ← {}
    result ← {}
    enqueue(q,root)
    found ← false
    while isNotEmpty(q) and not found do
        expand ← dequeue(q)
        for each child in expand
            if isFile(child) and child=searched then
                result ← result ∪ concat(expand,child)
                if mode=”first” then
                    found ← true
            else
                enqueue(q,child)
    → result

function DFS(root, searched, mode) → array of string
    s ← {}
    result ← {}
    push(s,root)
    found ← false
    while isNotEmpty(s) and not found do
        expand ← pop(s)
        for each file in expand
            result ← result ∪ concat(expand,file)
            if file=searched and mode=”first” then
                found ← true

        for each directory in expand
            push(s,directory)
    → result

function buttonSearch_Click ():
    clear listBox
    results ← {}
    graph ← {}

    if (labelFolder = "No Folder Chosen"):
        show error message
    if (fileName = ""):
        show error message
    if (pilih BFS):
        stopwatch.Start()
        if (findAllOccurences):
            results ← graph.BFS("all", fileName, labelFolder)
        else:
            results ← graph.BFS("first", fileName, labelFolder)
        stopwatch.Stop()
    else if (pilih DFS):
        stopwatch.Start()
        if (findAllOccurences):
            results ← graph.DFS("all", fileName, labelFolder)
        else:
```

```
            results ← graph.DFS("first", fileName, labelFolder)
        stopwatch.Stop()

    labelTE ← stopwatch.ElapsedMilliseconds + "ms"

    for each (string res in results):
        add to listBox

Viewer1.Graph = graph.GetGraph()
```

B. **Struktur Data**

Pada program *folder crawler* yang telah dibuat, dibuat sebuah kelas Graph yang menggunakan struktur data berikut:

1. Atribut

   graph : graph Microsoft MSAGL khusus untuk kelas Graph

2. Method
   a. GetGraph() : *getter* untuk data privat graph
   b. BFS() : method untuk mencari filename dengan mode tertentu (mencari semua kemunculan atau tidak) memanfaatkan algoritma *Breadth First Search*
      1) Queue : Queue q digunakan untuk menyimpan daftar urutan pengecekan simpul dalam pencarian suatu simpul filename.
      2) List : List of string results digunakan untuk menyimpan *full path* dari file yang dicari, baik ketika mode pencarian untuk menemukan seluruh file atau hanya file pertama yang ditemukan.
      3) Array : Array of string dirsNfiles digunakan untuk menyimpan nama seluruh direktori dan dokumen yang menjadi anak langsung simpul saat ini.
      4) List : List of strirng foundFilePath digunakan untuk menyimpan *path* dari file dicari yang ditemukan.
      5) Array : Array of node digunakan untuk menyimpan node pada graf
      6) Array : Array of string pathSplit digunakan untuk memisahkan path berdasar simbol “\\”
   c. DFS : method untuk mencari filename dengan mode tertentu (mencari semua kemunculan atau tidak) memanfaatkan algoritma *Depth First Search*
      1) Stack : Stack of string s digunakan untuk menyimpan riwayat pengecekan simpul dalam pencarian simpul filename.

2) List : List of string results digunakan untuk menyimpan *full path* dari file yang dicari, baik ketika mode pencarian untuk menemukan seluruh file atau hanya file pertama yang ditemukan.
3) Array : Array of DirectoryInfo digunakan untuk menyimpan direktori keberaadaan folder.
4) Array : Array of fileInfo mengembalikan FileInfo dari file yang terdapat pada suatu direktori.
5) List : List of strirng foundFilePath digunakan untuk menyimpan *path* dari file dicari yang ditemukan.
6) Array : Array of pathSplit digunakan untuk memisahkan path berdasar simbol "\\"

Pada *on-click function* yang didefinisikan pada Form, digunakan struktur data sebagai berikut:

1. List : List of string results digunakan untuk menyimpan *full path* dari file yang dicari, baik ketika mode pencarian untuk menemukan seluruh file atau hanya file pertama yang ditemukan.
2. Graph : *class* Graph

C. **Spesifikasi Program**

Berikut merupakan spesifikasi program yang telah dibuat:

1. Masukan

   Program harus menerima masukan berupa:

   a. *Starting Directory* pencarian file. Perlu diperhatikan bahwa folder anak dari *starting directory* harus dapat diakses.
   b. Nama file yang hendak dicari. Ekstensi file juga perlu diinputkan. Perlu diperhatikan bahwa nama file yang dimasukkan bersifat *case-sensitive*.
   c. Pilihan mencari semua keberadaan file. Jika mencentang, akan dicari seluruh keberadaan file.
   d. Pilihan algoritma yang digunakan dalam pencarian file. Diberikan pilihan algoritma BFS daan DFS.

2. Keluaran

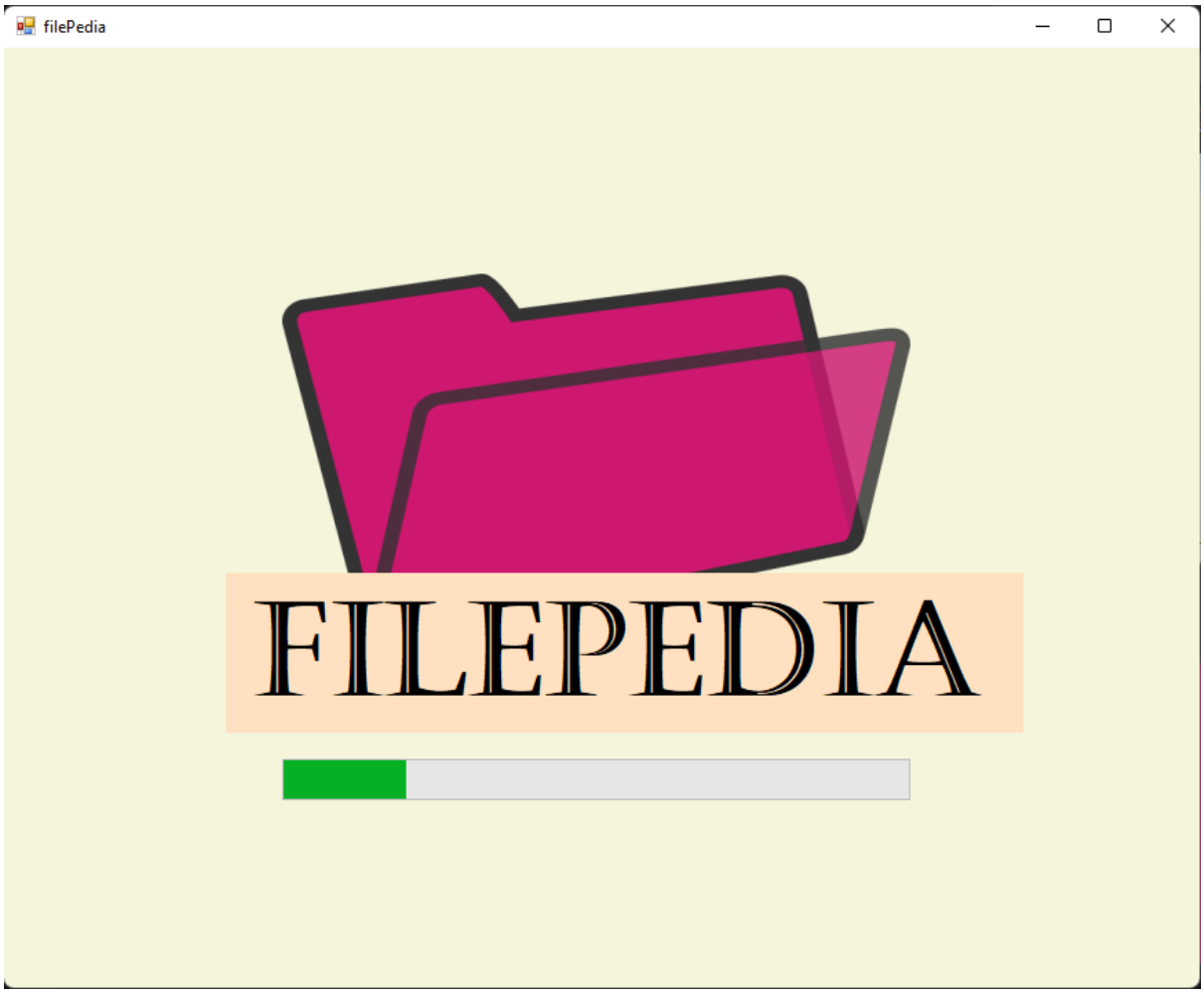
   Program akan menghasilkan keluaran berupa:

a. List *full path* penemuan file yang dicari. Dapat discroll untuk melihat list *full path* apabila lebih dari 1.
b. Link untuk membuka lokasi file. Dengan *double click full path,* program akan membuka *file explorer* lokasi file yang dicari.
c. Waktu yang dibutuhkan untuk melakukan pencarian dan pembuatan graf. Ditampilkan dalam satuan *milliseconds*.
d. Graf pencarian sesuai pilihan algoritma pencarian.

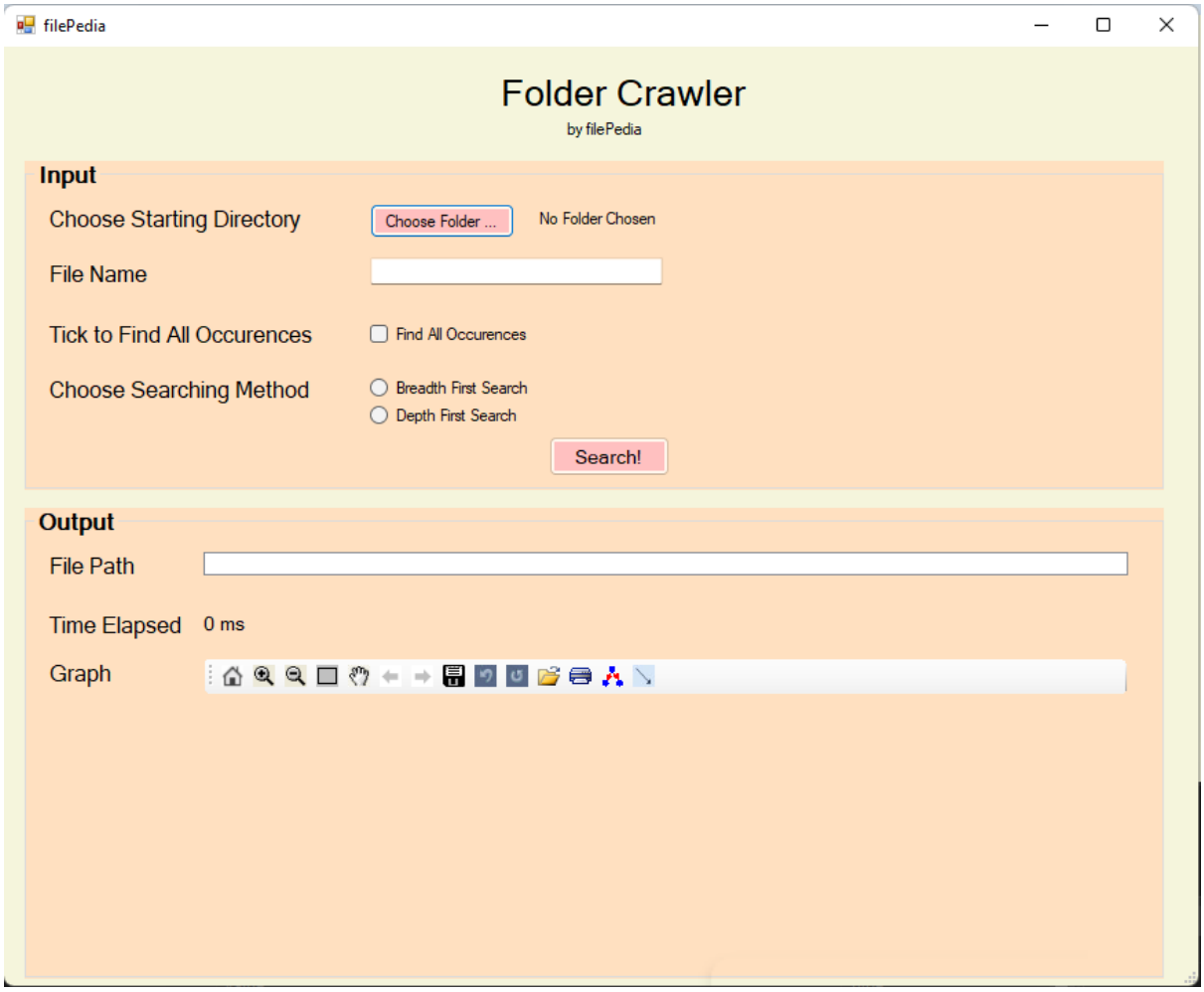
D. Tata Cara Penggunaan Program

1. Interface

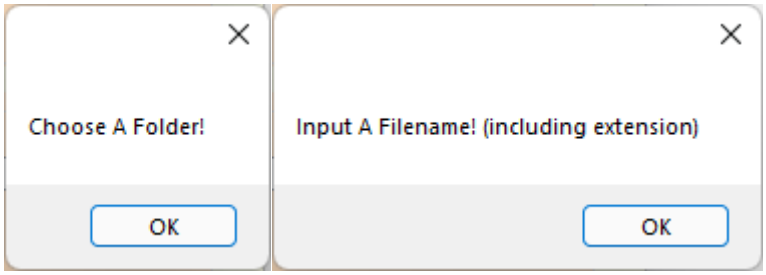
Berikut merupakan tampilan pengantar program,



Berikut merupakan tampilan program pencarian file,



Berikut merupakan tampilan pesan kesalahan,



2. Fitur yang Tersedia

   Berikut merupakan fitur-fitur yang tersedia pada program filePedia:

   a. Fitur pencarian file, baik hanya kemunculan pertama atau seluruh kemunculan, dengan algoritma BFS dan DFS
   b. Fitur navigasi file yang dicari. Dengan melakukan *double click* pada full path file, akan terbuka *file explorer* pada path tersebut.
   c. Fitur tampilan graf pencarian. Digunakan library MSAGL dalam menampilkan graf ini.

3. Tata Cara Penggunaan

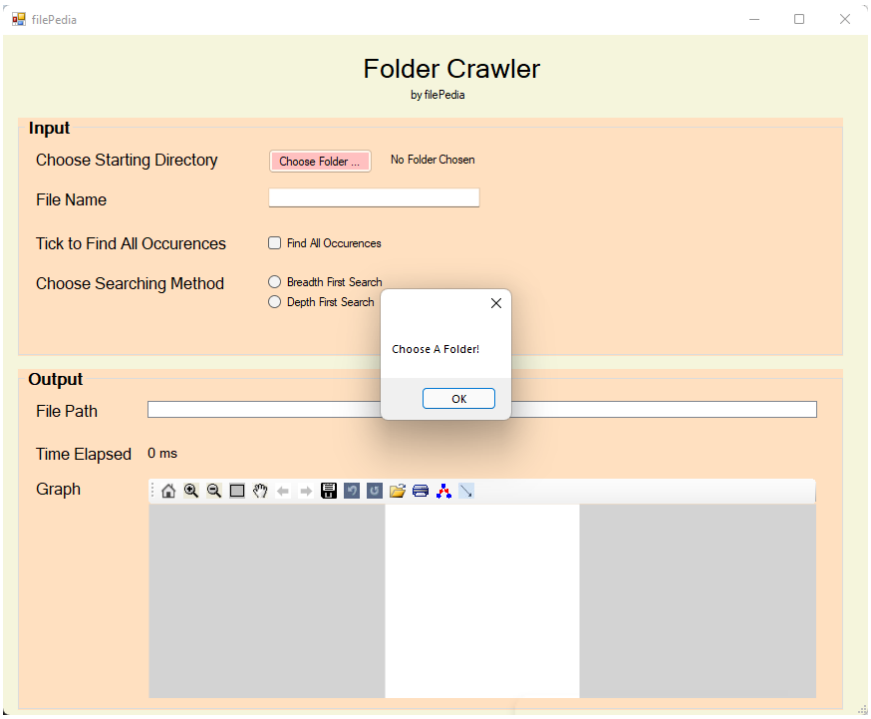
Berikut merupakan tata cara penggunaan program filePedia setelah meng-*clone* atau mendownload file dari *repository*,

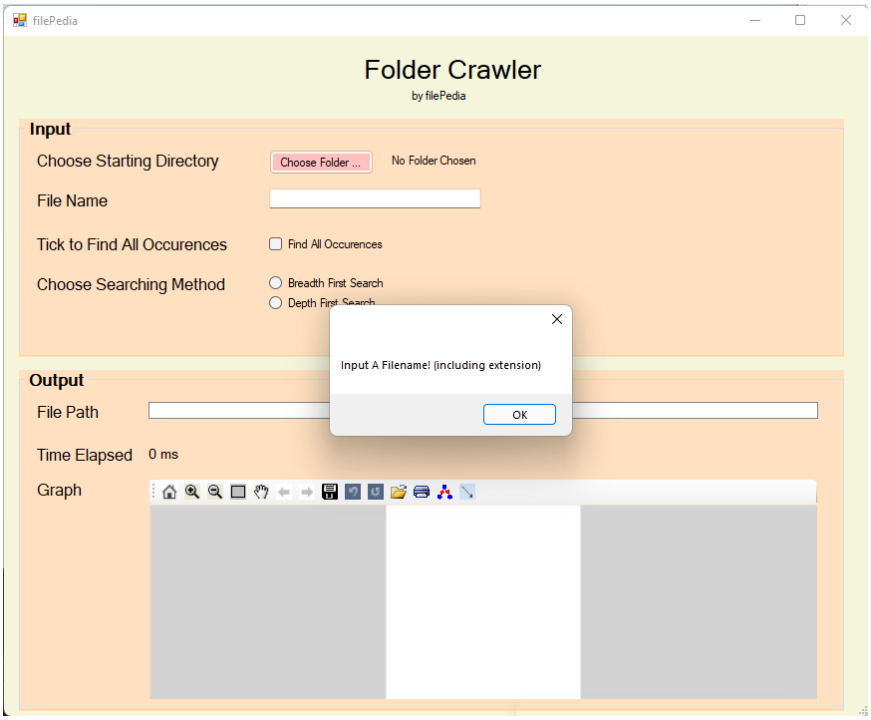
a. Untuk menjalankan program secara langsung, jalankan *executable file* stima-filePedia.exe yang terdapat pada folder stima-filePedia\bin
b. Untuk melakukan kompilasi, buka *project solution* stima-filePedia.sln pada Visual Studio, lalu tekan *icon* Start pada menu bar.

E. Hasil dan Analisis Hasil Pengujian

Berikut merupakan pengujian-pengujian yang dilakukan dan hasilnya:

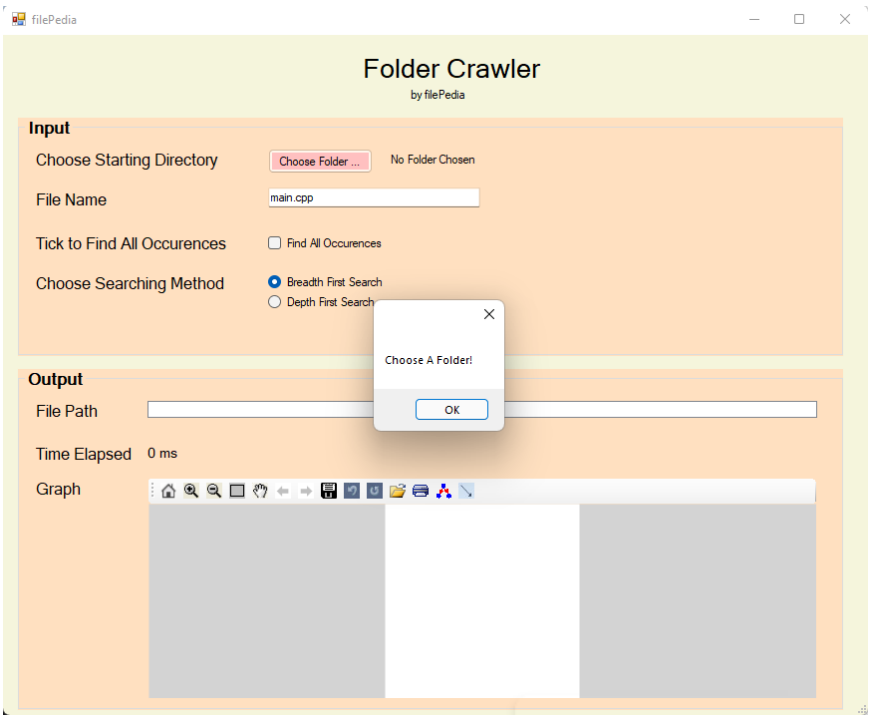
1. Tidak memasukkan *Starting Directory* dan nama file





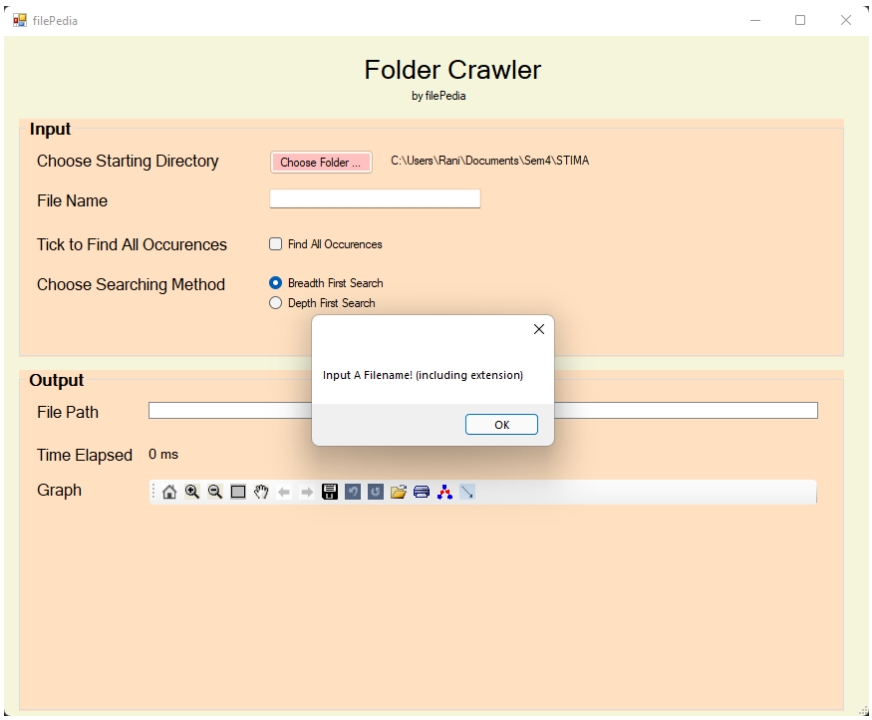
Pada kasus ini, tidak dilakukan pencarian baik dengan algoritma BFS maupun DFS sehingga tidak perlu dilakukan analisis implementasi algoritma.

2. Tidak memasukkan *Starting Directory*



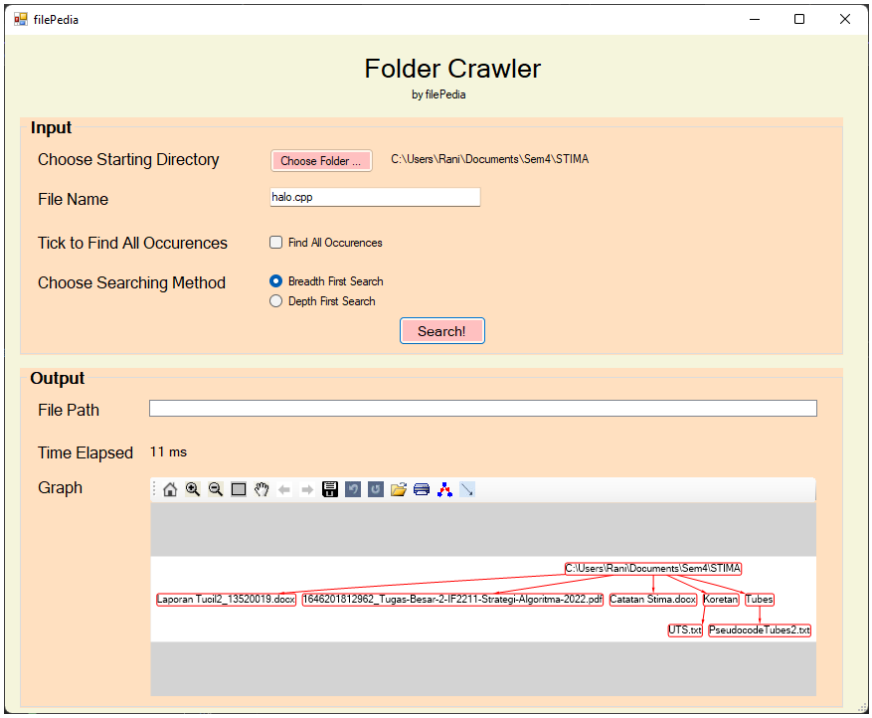
Pada kasus ini, tidak dilakukan pencarian baik dengan algoritma BFS maupun DFS sehingga tidak perlu dilakukan analisis implementasi algoritma.

3. Tidak memasukkan nama file



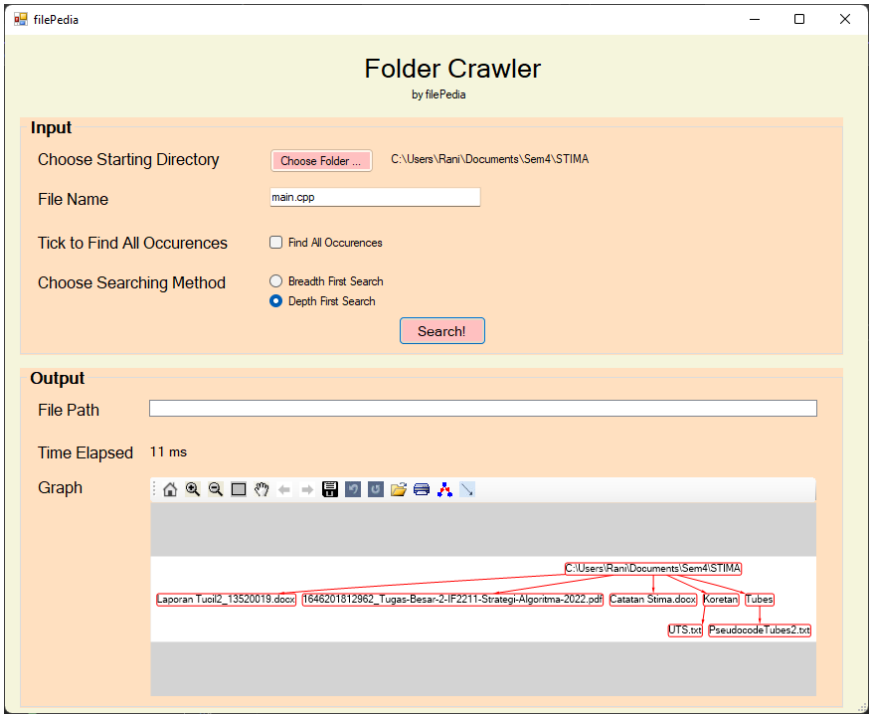
Pada kasus ini, tidak dilakukan pencarian baik dengan algoritma BFS maupun DFS sehingga tidak perlu dilakukan analisis implementasi algoritma.

4. File yang dicari tidak ada pada folder maupun subfolder *StartingDirectory* dengan algoritma BFS



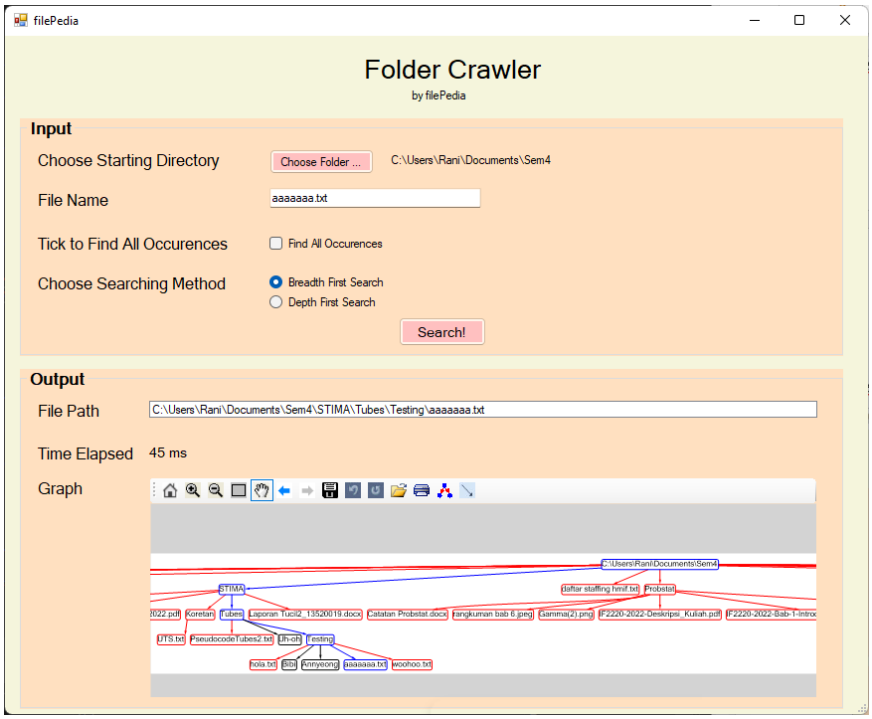
Pencarian file yang tidak ada dengan algoritma BFS memerlukan traversal ke seluruh node yang terdapat pada root folder. Hal ini menyebabkan waktu pencarian yang dibutuhkan lama.

5. File yang dicari tidak ada pada folder maupun subfolder *StartingDirectory* dengan algoritma DFS



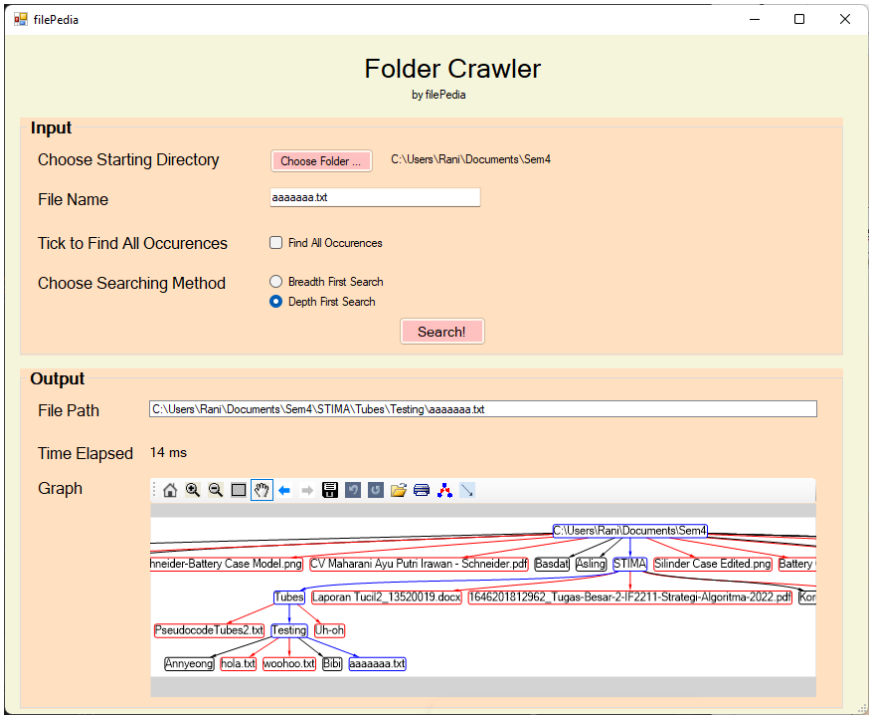
Sama halnya dengan pencarian file yang tidak ada dengan algoritma BFS, pencarian dengan algoritma DFS juga perlu mencari pada seluruh node yang ada. Dalam hal ini, jumlah langkah yang diambil dengan algoritma DFS sama dengan BFS.

6. Pencarian file tunggal menggunakan algoritma BFS



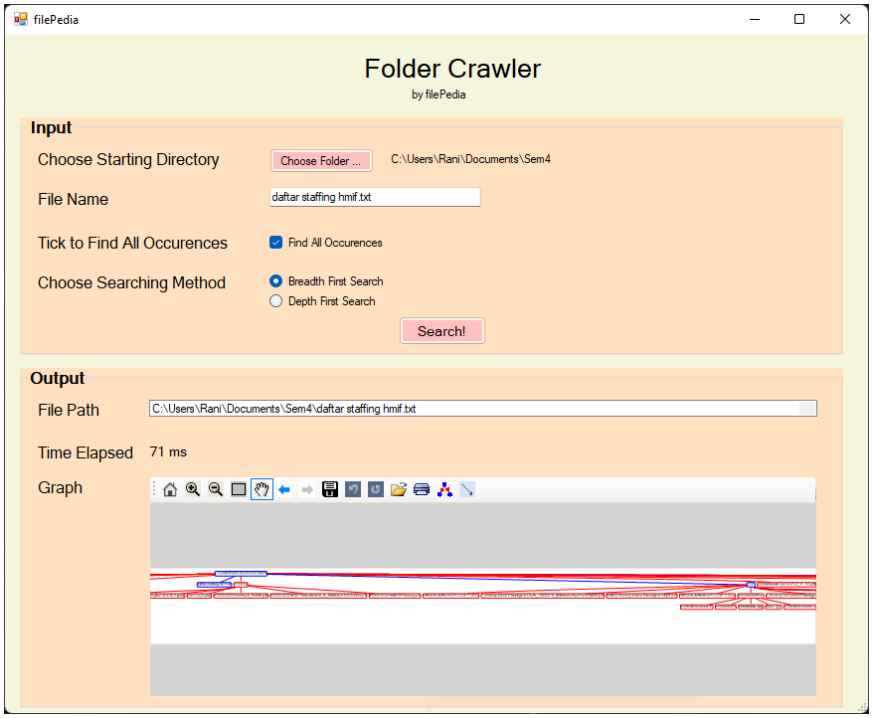
Pencarian satu file dengan algoritma BFS dalam kasus ini membutuhkan pencarian hingga level ke-4. Algoritma ini terbukti tidak efisien, dengan banyaknya node yang perlu dicari dan dibuktikan dengan waktu pencarian yang relatif lama.

7. Pencarian file tunggal menggunakan algoritma DFS



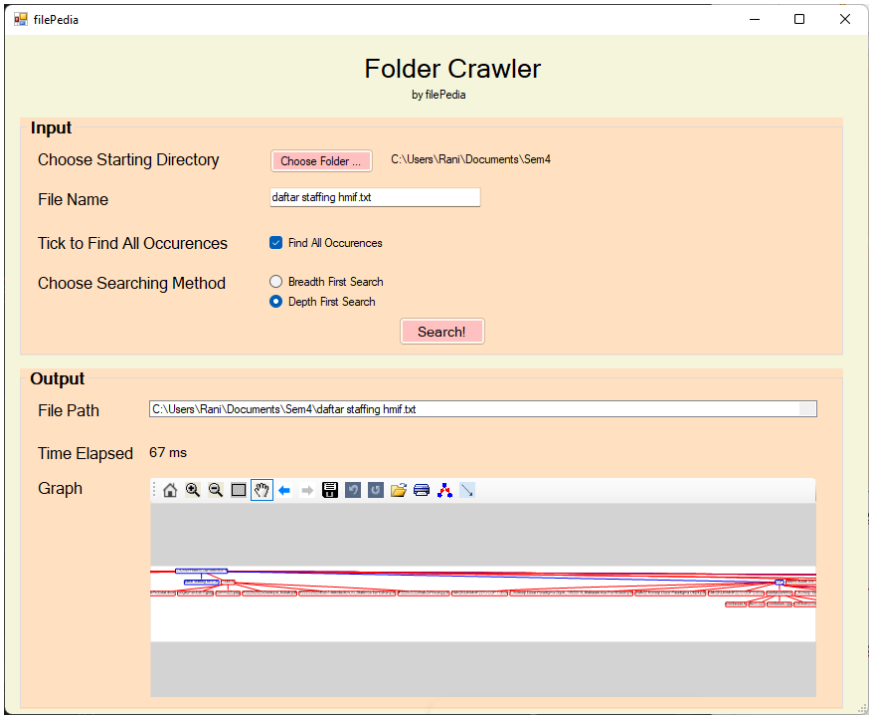
Pencarian satu file dengan algoritma DFS dalam kasus ini lebih efektif dibandingkan algoritma BFS, karena *parent folder* yang berisi file yang dicari relatif awal dari perspektif *root folder* sehingga DFS mencari hanya ke folder dan subfolder dari *parent folder* file yang dicari hingga *parent folder* itu sendiri. Waktu eksekusi juga membuktikan bahwa algoritma DFS, dalam hal ini memakan waktu kurang lebih 1/3 kali waktu yang dibutuhkan algoritma BFS

8. Pencarian banyak file menggunakan algoritma BFS



Pencarian seluruh kemungkinan keberadaan file dengan algoritma BFS dalam hal ini dilakukan dengan menelusuri seluruh node folder dan subfolder yang terdapat pada *root folder*. Hal ini menyebabkan waktu pencarian yang dibutuhkan relatif lama.

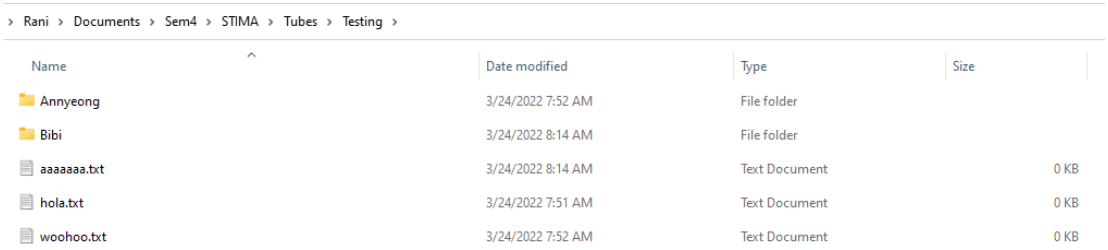
9. Pencarian banyak file menggunakan algoritma DFS



Pencarian seluruh kemungkinan keberadaan file dengan algoritma DFS juga membutuhkan penelusuran ke seluruh node folder dan subfolder yang terdapat pada *root folder*. Oleh karena itu, dalam kasus ini, algoritma DFS maupun BFS

membutuhkan jumlah langkah yang sama. Hal ini dibuktikan dengan selisih waktu eksekusi yang terpaut sedikit.

10. Membuka lokasi file pada *file explorer*



Pada kasus ini, tidak dilakukan pencarian baik dengan algoritma BFS maupun DFS sehingga tidak perlu dilakukan analisis implementasi algoritma.

Bab 5

Kesimpulan dan Saran

5.1 Kesimpulan

Proses pencarian file di dalam suatu susunan folder dapat direpresentasikan sebagai sebuah permasalahan penelusuran pohon. Adapun strategi penelusurannya dapat menggunakan algoritma *Depth First Search* (DFS) ataupun *Breadth First Search* (BFS).

Algoritma DFS akan menelusuri anak suatu folder sampai tidak ada lagi anak yang bisa ditelusuri, lalu baru menelusuri saudara dari node yang belum dikunjungi seperti yang dapat dilihat pada graf yang dihasilkan. Sedangkan algoritma BFS akan menelusuri seluruh anak dari suatu folder terlebih dahulu sebelum lanjut ke tingkat berikutnya seperti yang dapat dilihat pada graf yang dihasilkan.

5.2 Saran

Saran untuk pengembangan aplikasi yang telah dibuat adalah sebagai berikut

1. Pembuatan antarmuka visual yang responsif terhadap ukuran window.
2. Pembuatan grafik *tree* untuk struktur folder dan file selain menggunakan pustaka MSAGL.
3. Melakukan pencarian file dengan pendekatan lain seperti algoritma DLS (*Depth Limited Search*)

Daftar Pustaka

Munir, Rinaldi (2022). “Breadth/Depth First Search (BFS/DFS) (Bagian 1)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf. Diakses tanggal 20 Maret 2022.

Munir, Rinaldi (2021). “Breadth/Depth First Search (BFS/DFS) (Bagian 2)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf. Diakses tanggal 20 Maret 2022.

https://docs.microsoft.com/en-us/dotnet/csharp/

Lampiran

https://github.com/rannnayy/Tubes2_13520019

https://youtu.be/Pz17_znL1TI