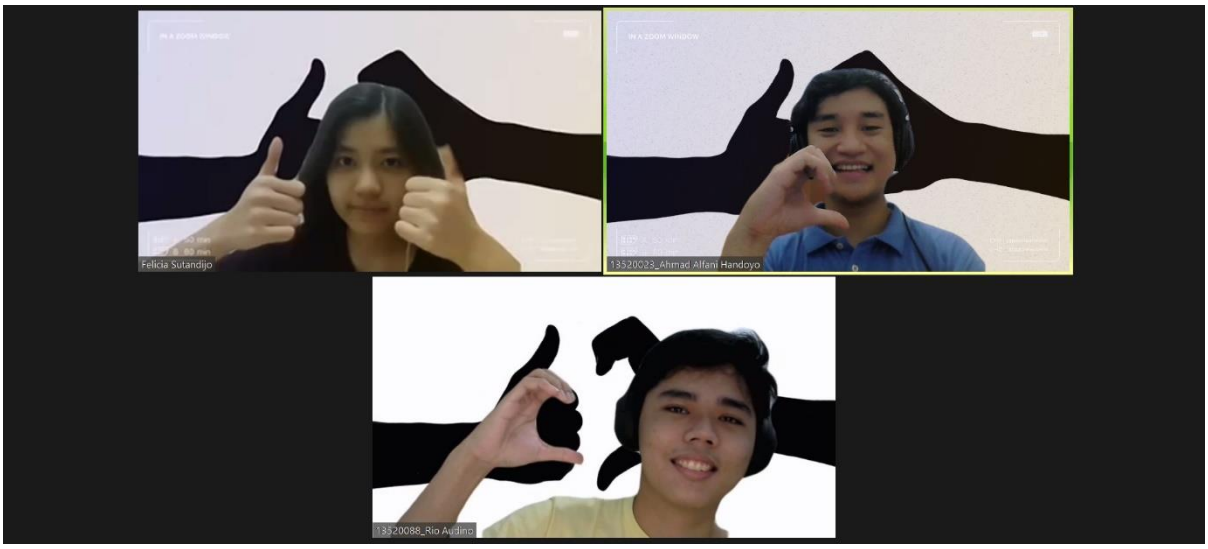
LAPORAN TUGAS BESAR II
IF2211 STRATEGI ALGORITMA

Pengaplikasian Algoritma BFS dan DFS
dalam Implementasi *Folder Crawling*



Disusun oleh:

13520023 Ahmad Alfani Handoyo

13520050 Felicia Sutandijo

13520088 Rio Alexander Audino

TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

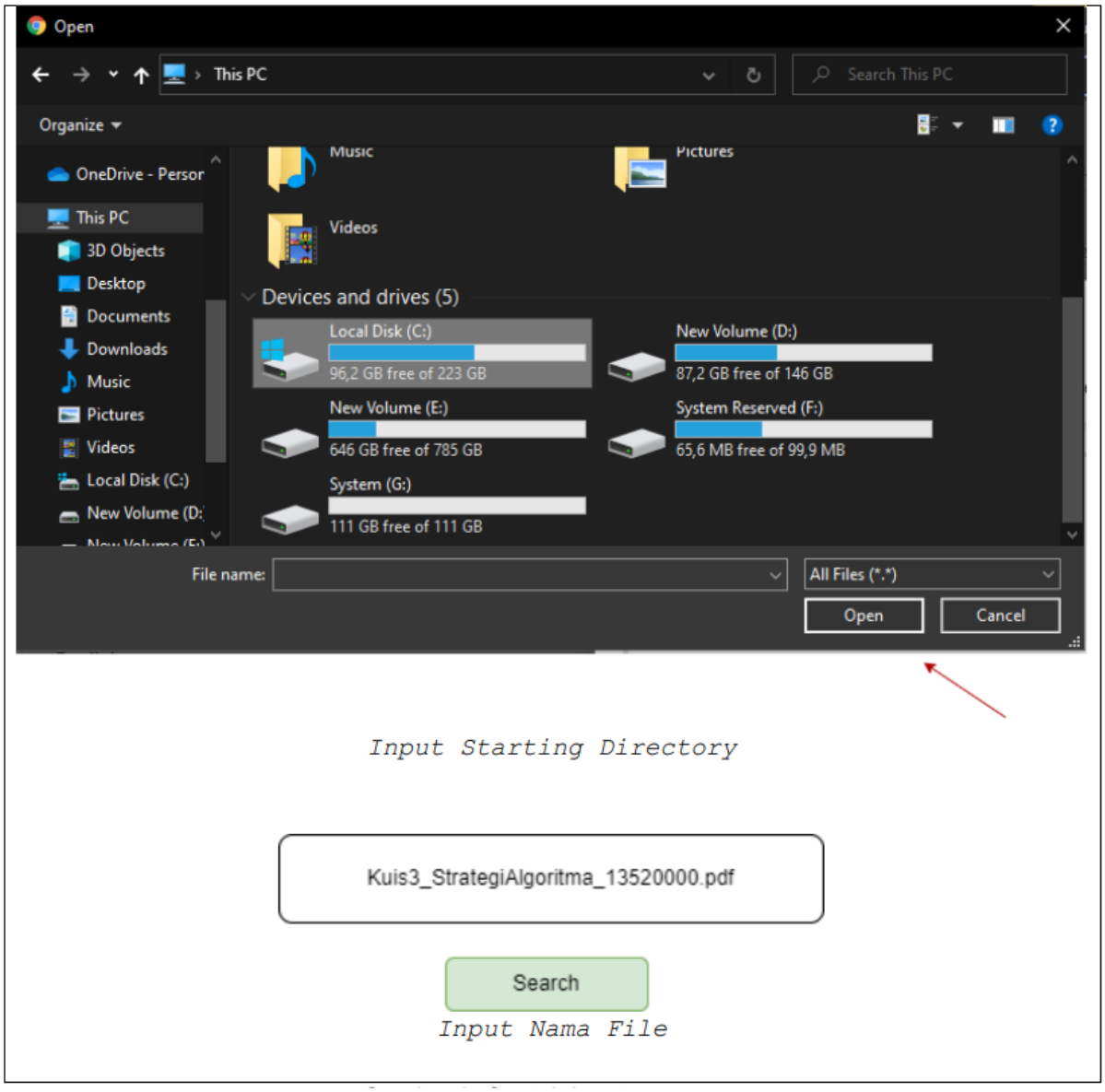
SEMESTER II TAHUN 2021/2022

Daftar Isi

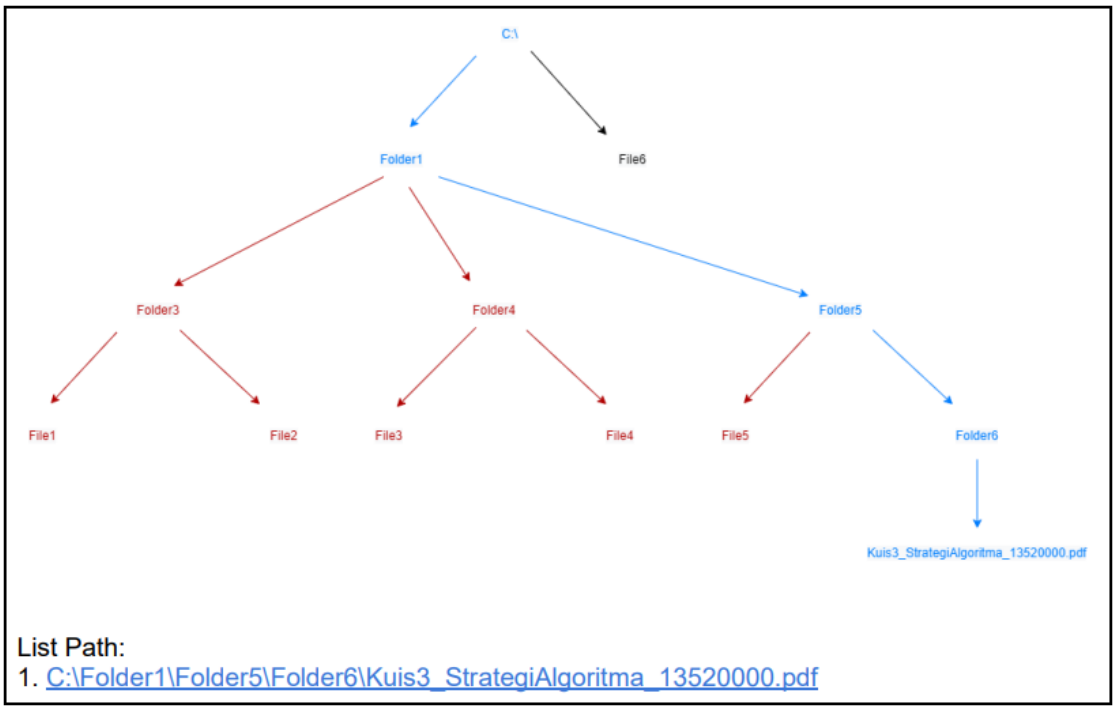
BAB 1
DESKRIPSI TUGAS

Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari file explorer pada sistem operasi, yang pada tugas ini disebut dengan Folder Crawling. Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), Anda dapat menelusuri folder-folder yang ada pada direktori untuk mendapatkan direktori yang Anda inginkan. Anda juga diminta untuk memvisualisasikan hasil dari pencarian folder tersebut dalam bentuk pohon.

Selain pohon, Anda diminta juga menampilkan list path dari daun-daun yang bersesuaian dengan hasil pencarian. Path tersebut diharuskan memiliki hyperlink menuju folder parent dari file yang dicari, agar file langsung dapat diakses melalui browser atau file explorer. Contoh hal-hal yang dimaksud akan dijelaskan di bawah ini.

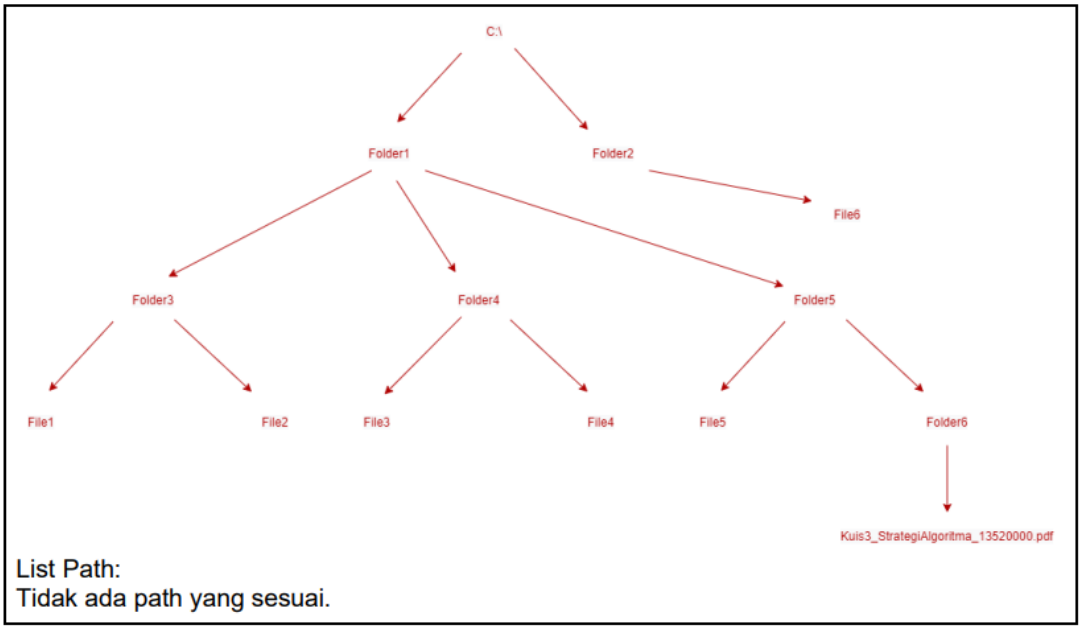


Gambar 1.1 Contoh input program



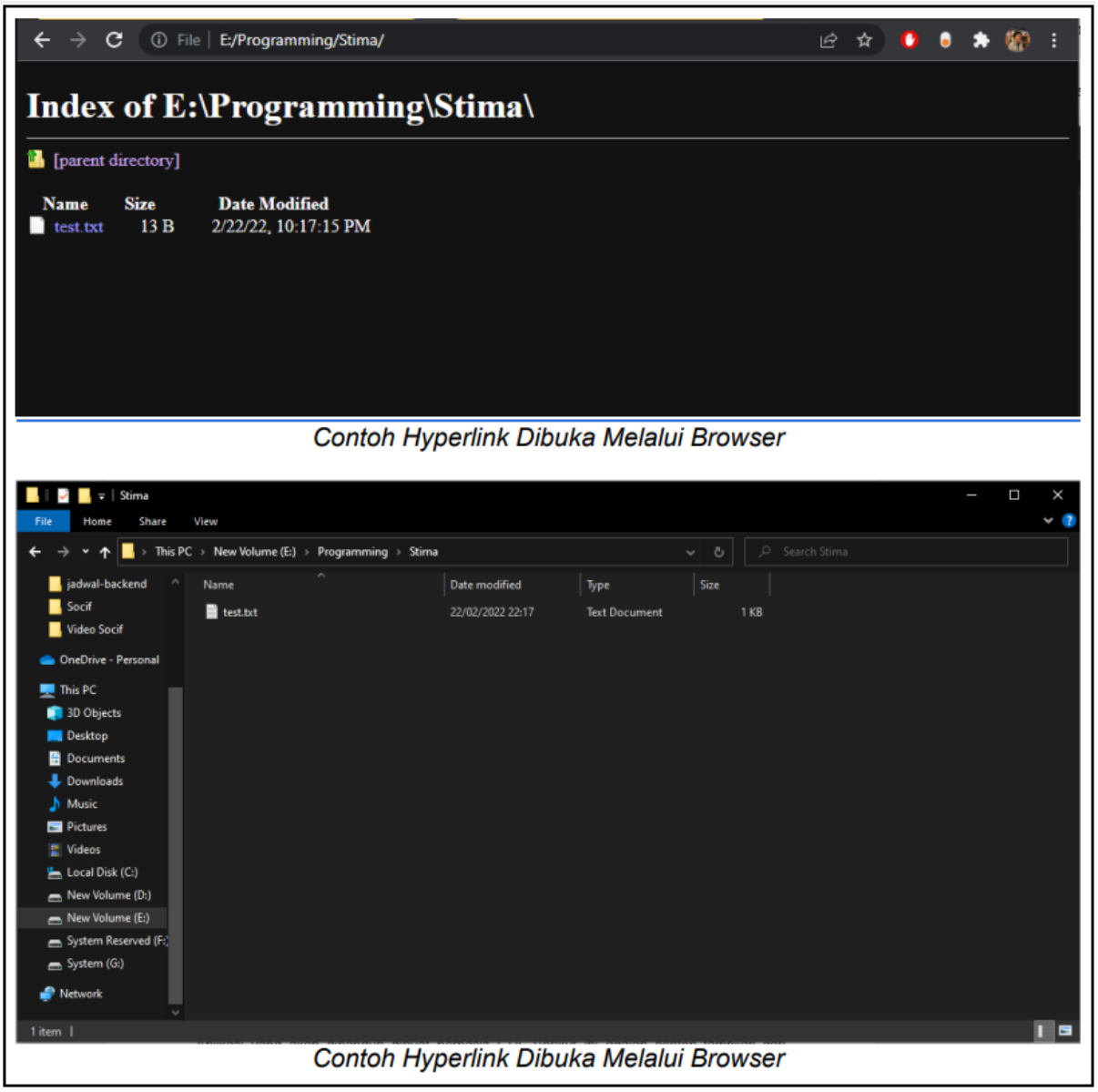
Gambar 1.2 Contoh output program

Misalnya pengguna ingin mengetahui langkah folder crawling untuk menemukan file Kuis3_StrategiAlgoritma_13520000.pdf. Maka, path pencarian DFS adalah sebagai berikut. C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3_StrategiAlgoritma_13520000.pdf. Pada gambar di atas, rute yang dilewati pada pencarian DFS diwarnai dengan warna merah. Sedangkan, rute untuk menuju tempat file berada diberi warna biru. Rute yang masuk antrian tapi belum diperiksa diberi warna hitam. Anda bebas menentukan warnanya asalkan dibedakan antara ketiga hal tersebut.



Gambar 1.3 Contoh output program jika file tidak ditemukan

Jika file yang ingin dicari pengguna tidak ada pada direktori file, misalnya saat pengguna mencari Kuis3Probstat.pdf, maka path pencarian DFS adalah sebagai berikut: C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3_StrategiAlgoritma_13520000.pdf → Folder6 → Folder5 → Folder1 → C:\ → Folder2 → File6. Pada gambar di atas, semua simpul dan cabang berwarna merah yang menandakan seluruh direktori sudah selesai diperiksa semua namun tidak ada yang mengarah ke tempat file berada.



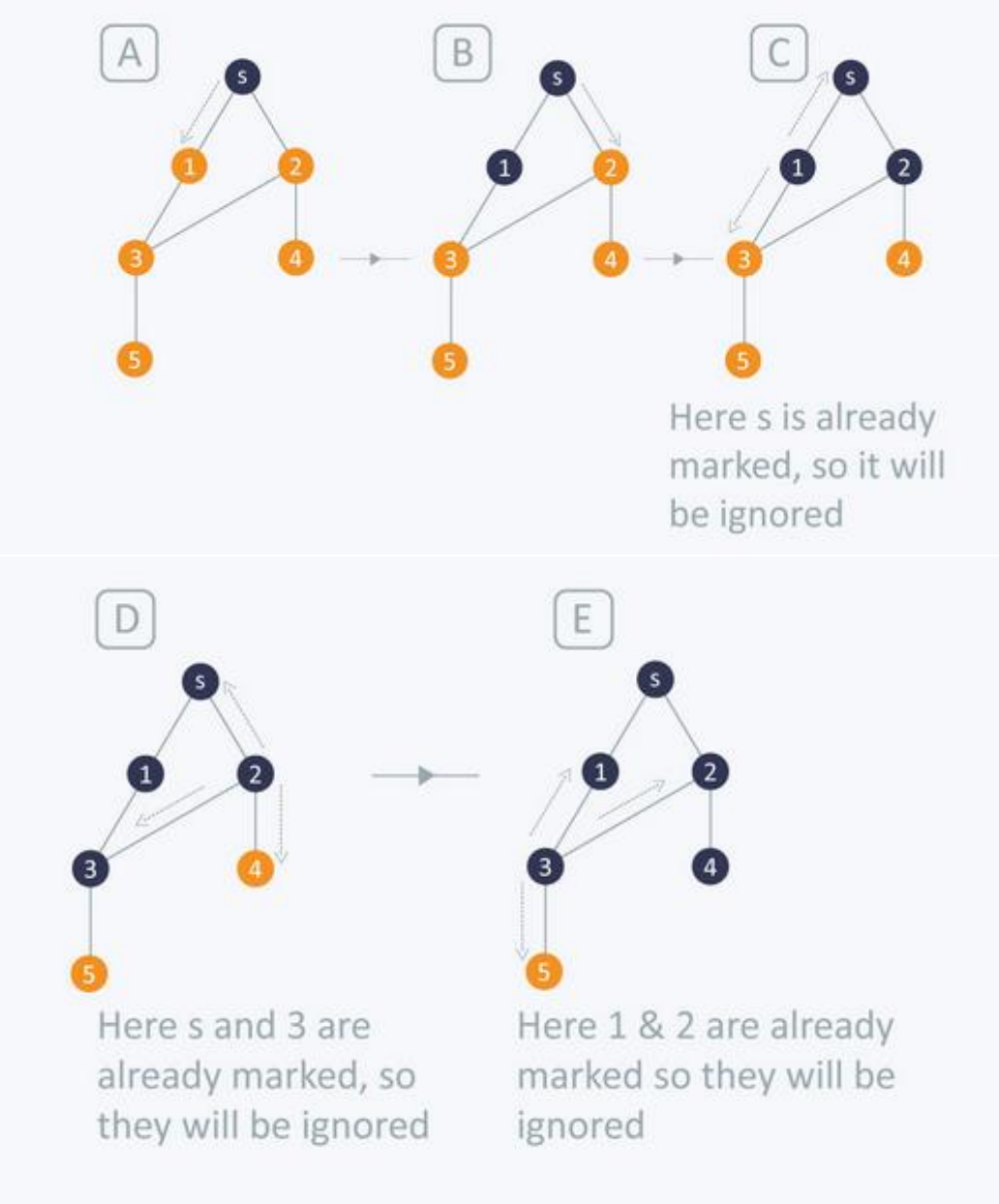
Gambar 1.4 Contoh ketika hyperlink di-klik

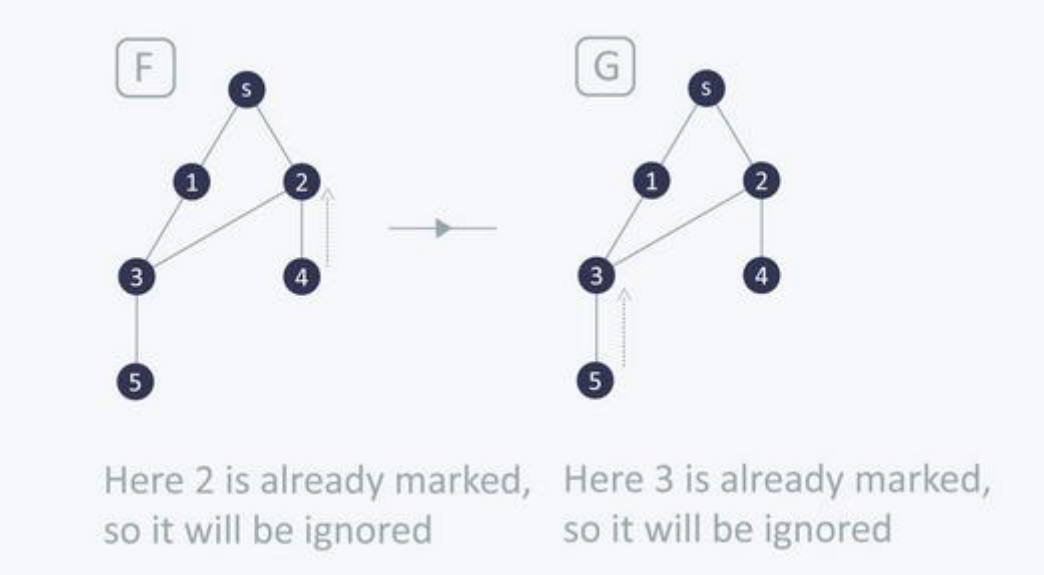
BAB 2
LANDASAN TEORI

2.1 Algoritma BFS

BFS merupakan singkatan dari Breadth First Search, yang berarti melakukan pencarian melebar terlebih dahulu. BFS adalah salah satu algoritma yang digunakan untuk melakukan pencarian di dalam sebuah graf. Algoritma BFS dilakukan dengan cara:

1. Kunjungi simpul v.
2. Kunjungi seluruh simpul yang bertetangga dengan v.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.





Gambar 2.1 Ilustrasi Algoritma BFS

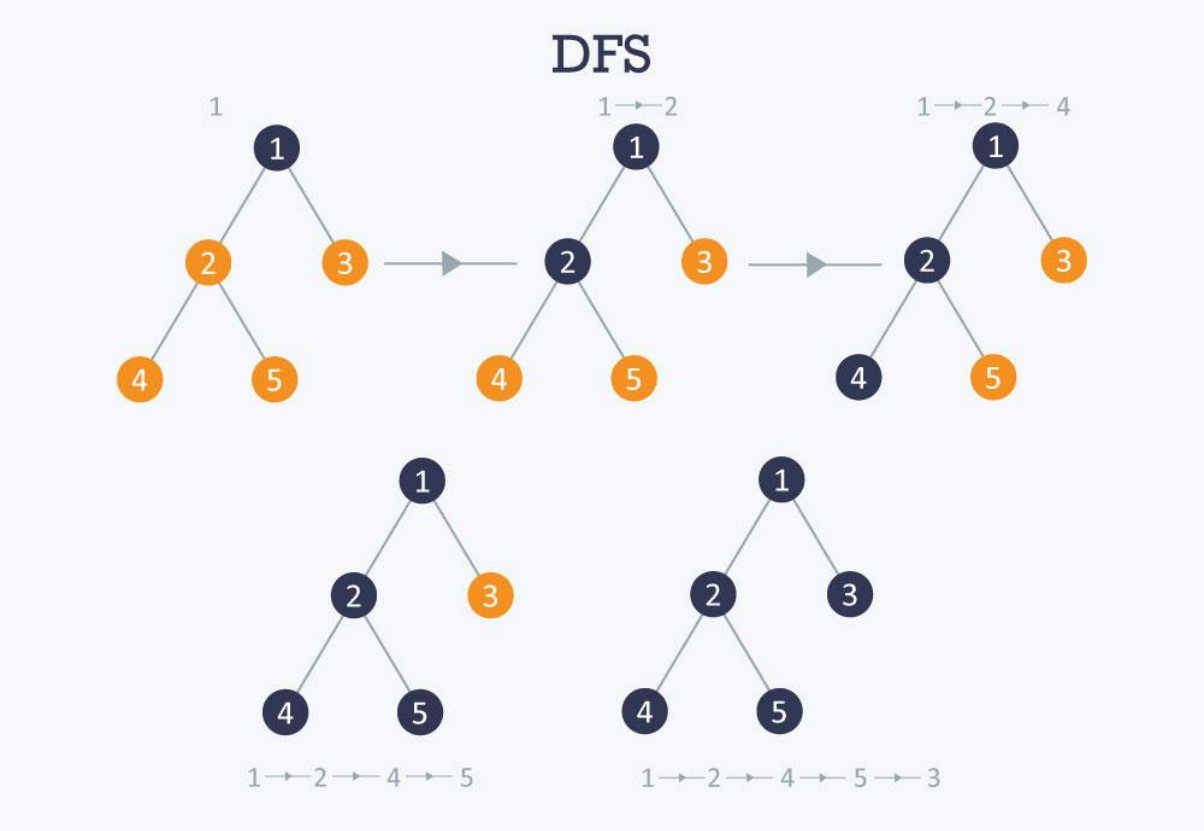
Secara *pseudocode*, algoritma BFS dapat ditulis sebagai berikut.

```
procedure BFS (input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.
Masukan: v edalah simout awal kunjungar
Keluaran: semue simpul yang dixunjungi dicetat ke layar}

Deklarasi
       w : integer
       g : antrian
       procedure BuatAntrian (input/output q : antrian)
       { membuat antrian kosong, kepala(q) diisi 0 }
       procedure MasukAntrian (input/output q:antrian, input v:integer)
       { memasukkan v ke dalam antrian g pada posisi belakang }
       procedure HapusAntrian (input/output q:antrian, output v:integer)
       { menghapus y dari kepala antrian q }
       function AntrianKosong (input q:antrian) → booleen
       { true jike antrian q kosong, false jike sebaliknye }

Algoritma
BuatAntrian(q)               { buat antrian kosong }
write(v)                     { cetak simpul awal yang dikunjungi }
dikunjungi[v] ← true         { simpul v telah dikunjungi, tandai dengan true }
MasukAntrian(q,v)            { masukkan simpul awal kunjungan ke dalam
                               antrian }

{ kunjungi semua simpul graf selama antrian belun kosong }
while not AntrianKosong(q) do
       HapusAntrian(q,v)     { simpul v telah dikunjungi, hapus dari antrian }
       for tiap simpul w yang bertetangga dengan simpul v do
              if not dikungungi[w] then
                     write(w) { cetak simpul yang dikunjungi }
                     MasukAntrian (q,w)
                     dikunjungi[w] ← true
              endif
       endfor
endwhile
{ Antriankosong(q) }
```

2.2 Algoritma DFS

DFS merupakan singkatan dari Depth First Search, yang berarti melakukan pencarian mendalam terlebih dahulu. DFS adalah salah satu algoritma lain yang digunakan dalam pencarian dalam graf. Algoritma DFS dilakukan dengan cara:

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.



Gambar 2.2 Ilustrasi Algoritma DFS
(sumber: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/)

Secara *pseudocode*, algoritma DFS dapat ditulis sebagai berikut.

```
procedure DFS (input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpul yang dikunjungi ditulis ke layer
}

Deklarasi
        w : integer

Algoritma
        write(v)
        dikunjungi[v] ← true
```

```
        for w ← 1 to n do
                if A[v,w]=1 then { simpul v dan simpul w bertetangga }
                        if not dikunjungi[w] then
                                DFS(w)
                        endif
                endif
        endfor
```

2.3 Graph Traversal

Graf merupakan salah satu struktur matematika yang melambangkan sekumpulan objek serta keterhubungan antar objek-objek tersebut. Graf dapat dituliskan sebagai G=(V,E) di mana V merupakan simpul-simpul dan E merupakan sisi dari graf.

Graph traversal merupakan algoritma pencarian dalam graf dengan cara mengunjungi simpul-simpul di dalam graf secara sistematis. Dengan merepresentasikan sebuah persoalan sebagai graf, traversal graf merupakan pencarian solusi untuk permasalahan tersebut.

Algoritma traversal graf sebagai pencarian solusi dapat dibagi menjadi dua, yaitu tanpa informasi, seperti DFS dan BFS, atau dengan informasi, seperti Best First Search dan A*.

2.4 C# Desktop Apps Development

C# (dibaca: C *sharp*) merupakan sebuah bahasa pemrograman berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. Pemrograman menggunakan Bahasa C# dapat digunakan untuk membangun sebuah aplikasi perangkat lunak desktop. Untuk mendesain GUI dari perangkat lunak, winform atau windows form dapat dimanfaatkan dengan bantuan aplikasi Visual Studio

BAB 3
ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Pada tugas Folder-Crawling kali ini, terdapat beberapa komponen yang harus diperhatikan, seperti ketepatan algoritma yang dipilih, GUI yang lengkap dan mudah digunakan, visualisasi hasil pencarian berupa graf pohon, serta pengintegrasian berbagai komponen. Tiap-tiap komponen tentu memiliki sub-permasalahannya masing-masing. Algoritma BFS/DFS yang dipilih nantinya harus dipastikan sudah sesuai dengan algoritma yang ada (bisa dicocokan dengan materi di kelas). GUI yang dibuat haruslah lengkap memuat semua kebutuhan spesifikasi dan mudah dibutuhukan. Sistem visualisasi harus terinegrasi dengan hasil pencarian. Oleh karena itu, berikut telah kami petakan langkah-langkah dalam pemecahan masalah Folder-Crawling menggunakan algoritma BFS/DFS,

a. Membuat algoritma BFS/DFS
b. Mengintegrasikan algoritma BFS/DFS dengan sistem direktori windows
c. Memetakan permasalahan Folder-Crawling pada elemen algoritma BFS/DFS
d. Mengaplikasikan algoritma BFS/DFS pada elemen-elemen Folder-Crawling yang sudah terpetakan
e. Membuat sistem graf pohon dan memvisualisasikan hasil pencarian
f. Membuat GUI sebagai kontrol program nantinya
g. Mengintegrasikan GUI dengan algoritma BFS/DFS dan visualisai graf pohon
h. Melakukan uji eksekusi program

3.2 Mapping Algoritma BFS dan DFS pada Persoalan

Dalam pemecahan masalah menggunnakan algoritma BFS/DFS, terdapat beberapa elemen yang harus dipetakan. Dimulai dari graf pohon, graf pohon menggambarkan sistem percabangan dari suatu direktori yang dipilih. Tiap-tiap node menggambarkan folder maupun file dan tiap-tiap edge mengambarkan hubungan folder dengan folder atau folder dengan file.

Pada algoritma pencarian file menggunakan BFS, pencarian akan dimulai secara traversal dari root folder (root node). Selanjutnya akan dipetakan tiap-tiap folder/file dalam root folder. Lalu tiap-tiap sub-folder akan dipetakan kembali tiap-tiap subfolde/filenya. Proses ini terus diulang secara rekursif sampai program menemukan file yang sesuai.

Pada algoritma pencarian file menggunakan DFS, pencarian akan dimulai dari root folder juga. Perbedaannya adalah pada proses traversal folder, pada algoritma DFS akan ditelusuri tiap sub-folder sampai mentok/tidak menemukan sub-folder lagi. Ketika sudah mentok, akan dilakukan backtracking ke sub-folder yang tersedia. Proses akan berhenti ketika semua file sudah tervalidasi atau file sudah ditemukan.

3.3 Alternatif Solusi BFS/DFS

Selain permasalahan Folder-Crawling, algoritma BFS/DFS juga dapat digunakan dalam berbagai masalah lainnya, seperti automasi pembuatan sitasi dalam suatu makalah. Dalam permasalahan automasi setiap kata yang dicari akan menjadi sebuah node dan tiap kata yang berhubungan akan menjadi edge. Proses traversal akan dilakukan sama seperti permasalahan sebelumnya.

Terdapat juga permasalahan seperti Web Spider, proses pencarian halaman-halaman web secara traversal menggunakan algoritma BFS/DFS. Mirip dengan proses folder crawling, tiap-tiap web akan direpresentasikan sebagai sebuah node dan hubungan antar web akan direpresentasikan sebagai edge. Setelah graf pohon terbentuk dari node dan edge, proses pencarian akan dilakukan menggunakan algoritma BFS/DFS.

Sebenarnya masih banyak lagi contoh implementasi dari BFS/DFS. Setiap problem pencarian solusi yang dapat kita petakan ke dalam sebuah graf pohon dapat kita selesaikan menggunakan algoritma BFS/DFS. Kita hanya perlu memetakan permasalahan menjadii elemen algoritma BFS/DFS (node dan edge). Lalu menggunakan algoritma BFS/DFS akan mentranversal tiap node sampai menemukan solusi atau traversal sudah habis.

BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma BFS/DFS

Pada program yang kami buat, implementasi algoritma BFS/DFS dapat dibagi ke dalam tiga tahapan, yaitu memilih tipe algoritma, membuat larik direktori berdasarkan jenis algoritma yang sudah dipilih, dan terkahir mengecek file dalam tiap direktori. Berikut adalah penjelasan lengkap dari tiap langkah utama beserta fungsi terkait.

a. Memilih tipe algoritma

Proses memilih tipe algoritma adalah proses yang mudah, karena proses hanya menerima sebuah masukan dari user yang lalu menjalakan jenis algoritma sesuai. Pada program kami, proses pemilihan ditangani oleh sebuah variabel integer (0:Algoritma BFS, 1:Algoritma DFS).

fungsi di dalam *source-code :* *BFSorDFSAlgo.BFSorDFS()*

b. Membuat larik direktori

Proses membuat larik direktori adalah proses pembuatan sebuah larik berdasarkan proses iterasi algoritma. Larik yang terbentuk akan secara terurut sesuai dengan proses pencarian file, baik BFS atau DFS. Berikut adalah penjelasan rinci tiap-tiap algoritma,

- Algoritma DFS
    1. Menambahkan semua nama files dan folder yang ada di root folder
    2. Melakukan iterasi dari element pertama
    3. Jika elemen yang diiterasi adalah sebuah folder, menambahkan semua nama file dan folder yang ada di elemen tersebut
    4. Proses penambahan dilakukan tepat setelah indeks yang dilakukan iterasi
    5. Iterasi elemen sudah selesai (semua elemen sudah diiterasi)
    6. Menambahkan root path ke indeks pertama

- Algoritma BFS
    1. Menambahkan root path
    2. Melakukan iterasi dari element pertama
    3. Jika elemen yang diiterasi adalah sebuah folder, menambahkn semua nama file dan folder yang ada di elemen tersebut
    4. Proses penambahan dilakukan di akhir larik
    5. Iterasi elemen sudah selesai (semua elemen sudah diiterasi)

fungsi di dalam *source-code :* *BFSorDFSAlgo. getAllDirsBFS ()*
BFSorDFSAlgo. getAllDirsDFS ()

c. Mengecek file tiap direktori

Sesuai namanya, proses mengecek direktori adalah proses yang melakukan iterasi dari setiap direktori dalam larik direktori yang telah dibuat

sebelumnya. Pada setiap iterasi, proses ini akan mengecek keberadaan file yang dicari. Proses ini juga menangani kasus ketika ingin menemukan semua kemunculan file.

fungsi di dalam *source-code :* *Main.FileChecker ()*
Main.CheckFileInsideFolder()

4.2 Pseudocode

a. BFSorDFS(int algorithm, string fileName, string rootPath, ref string[] allDirPath, ref string[] allRootsPath, Boolean findAllOccurrence)

```
procedure BFSorDFS (input integer algorithm, input strinig fileName, input string
rootPath, input/output string[] allDirPath, input/output string[] allRootsPath, input
boolean findAllOccurrence)

{Prosedur utama dalam proses implementasi algoritma BFS/DFS, pada prosedur ini akan
dipilih jenis algoritma yang dijalankan}

KAMUS LOKAL

ALGORITMA

        if (algorithm == 0) then

                getAllDirsBFS(rootPath, allDirPath, allRootsPath)

        if (algorithm == 1) then

                getAllDirsDFS(rootPath, allDirPath, allRootPath)
```

b. getAllDirsDFS(String rootPath, ref String[] allDirs, ref String[] allRoots)

```
procedure getAllDirsDFS(input string rootPath, input/output string[] allDirs,
input/output string[] allRoots)

{Prosedur yang menghasilkan larik pencarian file berdasarkan algoritma DFS}

KAMUS LOKAL

        tempDirs, tempRoots, isDir, tempRoots        : list of String[]

        i                                            : integer


ALGORITMA

        {Instantiate required variables}

        tempDirs <- {get all directory in rootPath}

        tempFiles <- {get all files in rootPath}

        tempRoots <- {get all root path from tempDirs and tempFiles}

        isDir <- {get all type files or directory from tempDiris and tempFiles}


        tempDirs <- {Conncatenate tempFiles and tempDirs}


        {Iterating through each elements in tempDirs}

        i <- 0

        while (i < tempDirs.Length)
```

```
        {If current element is a file, skip current element}
        if (!idDir[i]) then
            i++
            continue

        {Add all file and folder to tempDirs, make sure to insert elements at
        index (i+1)}
        tempDirs <- {add all files and folder from current elements}
        tempRoots <- {get all root path from tempDirs }
        isDir <- {get all type files or directory from tempDiris}
        i++
    {Finished iterating through each elements}
    tempDirs <- {insert root folder into the first index}
    tempRoots <- {insert root path into the first index }
```

c. getAllDirsBFS(String rootPath, ref String[] allDirs, ref String[] allRoots)

```
procedure getAllDirsBFS(input string rootPath, input/output string[] allDirs,
input/output string[] allRoots)
{Prosedur yang menghasilkan larik pencarian file berdasarkan algoritma BFS}
KAMUS LOKAL
    tempDirs, tempRoots, isDir    : list of String[]
    i                             : integer
ALGORITMA
    {Instantiate required variables}
    tempDirs <- {add rootPath}
    tempRoots <- {add rootPath}
    isDir <- {add true because first element is a root folder}

    {Iterating through each elements in tempDirs}
    i <- 0
    while (i < tempDirs.Length)
        {If current element is a file, skip current element}
        if (!idDir[i]) then
            i++
            continue

        {Add all file and folder to tempDirs, make sure to insert elements on
        the last index}
        tempDirs <- {add all files and folder from current elements}
```

```
        tempRoots <- {get all root path from tempDirs }

        isDir <- {get all type files or directory from tempDiris}

        i++

{Finished iterating through each elements}
```

d. fileChecker(string fileName, string[] allDirPath, ref string[] targetPath, bool findAllOccurrence)

```
procedure fileChecker(input string fileName, input string[] allDirPath, input/output
string[] targetPath, input bool findAllOccurrence)

{Prosedur yang mengecek file tiap direktori}

KAMUS LOKAL

ALGORITMA

        {Iterate through each dir in allDirPath}

        foreach (string dir in allDirPath) do
                if ({mengecek apakah dir merupakan file}) then
                        {dir merupakan file}
                        continue
                else
                        if (CheckFileInsideFolder(fileName, dir))
                                targetPath <- {menambahkan dir ke dalam targetPath}
                                if (not findAllOccurrence) then
                                        {Cukup menemukan satu file}
                                        break
```

e. CheckFileInsideFolder(String fileName, String path)

```
function CheckFileInsideFolder(input string fileName, input string path) -> bool

{Fungsi mengembalikan true jika terdapatt file dengan nama filenName, jika tidak
mengembalikan false}

KAMUS LOKAL

        files : List of String[]

ALGORITMA

        {Mengiterasi tiap file dalam folder path}

        files <- {Menambahkan semua file dalam folder path}

        foreach (string file in files)

                if (file == fileName) then

                        -> true

        {Iterasi selesai dan tidak menemukan file yang sama}

        -> false
```

4.3 Struktur Data

Pada implementasi algoritma BFS/DFS kali ini, kami menggunakan algoritma berbasis objek, struktur data didefinisikan sebagai kelas (*class*). Kelas utama berada di dalam

file Main.cs, yang peran utamanya adalah menjalakan algoritma dan mengubahnya ke dalam sebuah graf pohon. Adapun kelas-kelas pendukung lainnya seperti BFSorDFSAlgo.cs dan TreeNode.cs. Berikut adalah penjelasan tiap-tiap kelas.

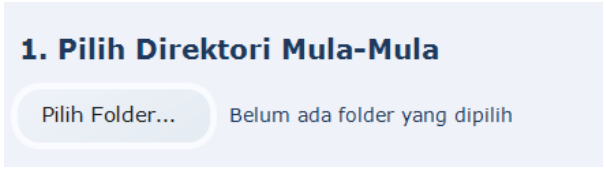
No	Nama Kelas	Penjelasan
1	Main	Kelas ini merupakan kelas utama yang dalam implementasi algoritma BFS/DFS. Kelas Main akan memanggil fungsi seperti, menjalankan algoritma, mengecek keberadaan file, dan mengubah larik menjadi sebuah graf pohon. Di dalam kelas main sudah terdapat method-method pendukung algoritma.
2	BFSorDFSAlgo	Kelas ini merupakan kelas yang menangani proses pencarian file menggunakan algoritma BFS/DFS.
3	TreeNode	Kelas ini merupakan kelas yang menangani pohon graf. Di dalamnya terdapat struktur dari sebuah pohon graf dan method inisiasinya.

4.4 Tata Cara Penggunaan Program

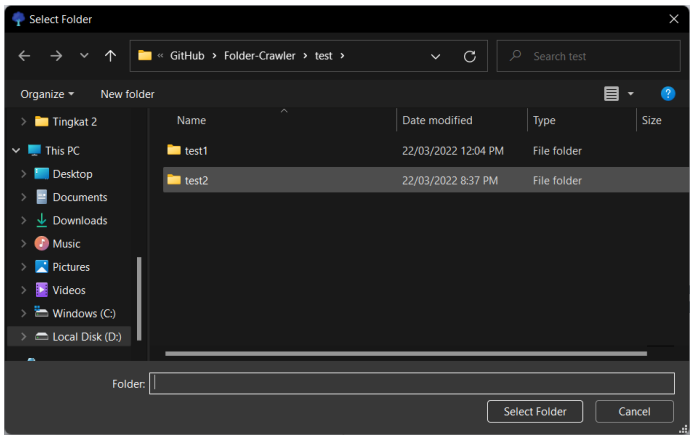
Program dijalankan dengan menjalankan file *Folder-Crawler.exe*. Bila program dijalankan pada komputer yang sebelumnya belum pernah menjalankan program yang membutuhkan .NET Desktop Runtime maka unduh dan install terlebih dahulu pada pop-up yang muncul. Setelah itu, ada kemungkinan muncul pop-up yang kedua untuk menginstall framework-framework yang dibutuhkan untuk .NET. Oleh karena itu, install terlebih dahulu. Bila kedua hal tersebut sudah dilakukan, maka program dapat dijalankan dan muncul tampilan seperti berikut.

Gambar 4.1 Tampilan Awal Program

Pada tampilan awal tersebut ada 4 bagian yang harus diisi oleh pengguna untuk memulai program pada bagian kiri tampilan.

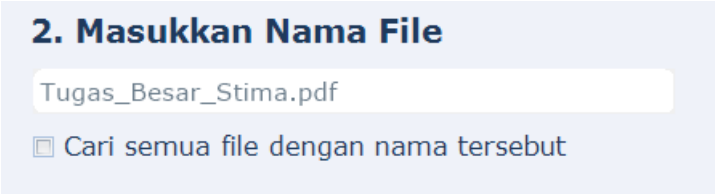


Gambar 4.2 Bagian Pemilihan Direktori Mula-Mula



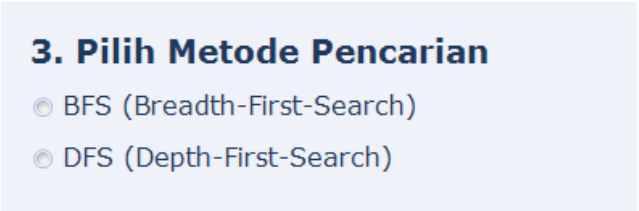
Gambar 4.3 Dialog Memilih Folder atau Direktori

Pada bagian pertama pengguna diminta untuk memilih folder atau direktori mula-mula dimana algoritma BFS/DFS dijalankan. Menekan tombol “Pilih Folder...” akan membuka dialog baru untuk memilih folder pada File Explorer. Pada dialog tersebut pengguna dapat memilih folder. Bila sudah, pengguna menekan “Select Folder”.



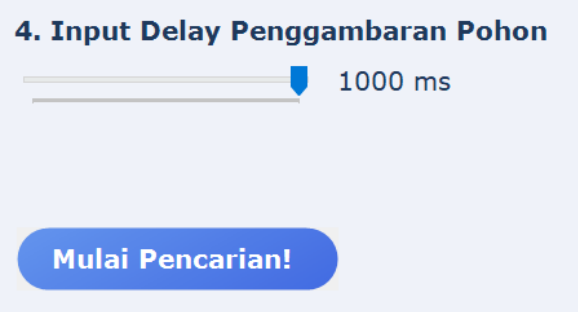
Gambar 4.4 Bagian Pengisian Nama File

Pada bagian kedua, pengguna diminta untuk mengetik nama file beserta extensionnya yang akan dicari pada algoritma. Lalu, terdapat checkbox yang dapat dipilih agar algoritma mencari semua file dengan nama tersebut, atau berhenti ketika menemukan satu file dengan nama tersebut.

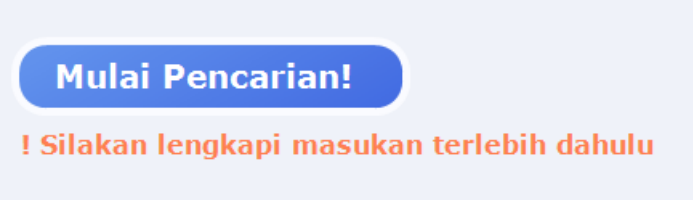


Gambar 4.4 Bagian Pemilihan Algoritma BFS atau DFS

Pada bagian ketiga, pengguna diminta untuk memilih apakah penelusuran menggunakan algoritma BFS atau DFS.



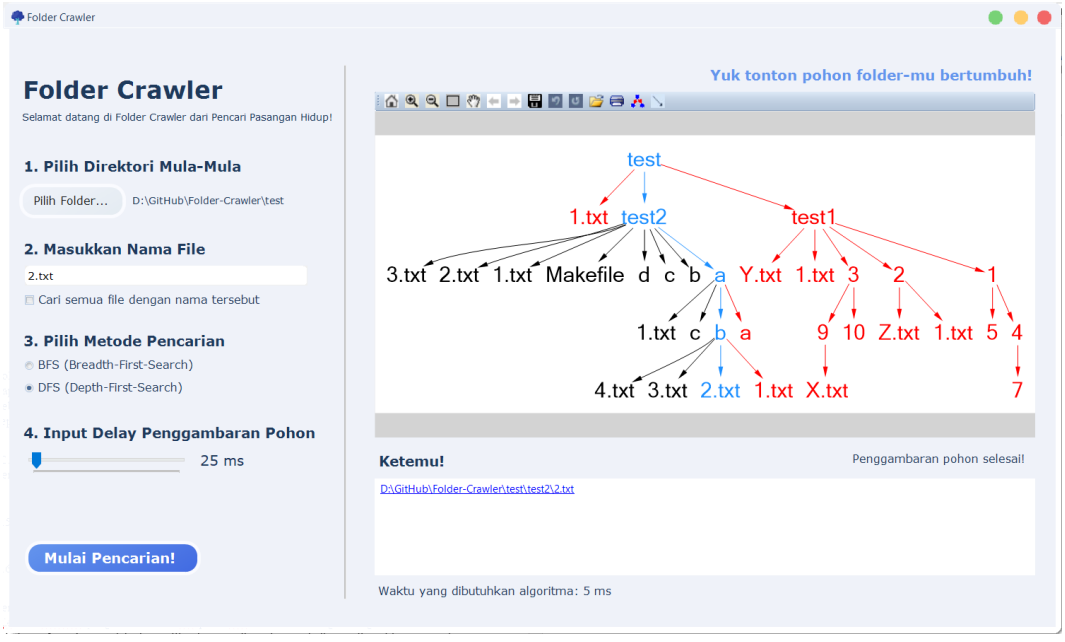
Gambar 4.4 Bagian Slider Delay Penggambaran Pohon



Gambar 4.4 Tombol Mulai Pencarian dan Peringatan Input Kosong

Pada bagian keempat, pengguna diminta untuk memilih delay untuk menggambar pohon direktori pada suatu slider dengan interval 5 milisekon ke 1 sekon. Selanjutnya, user dapat memulai algoritma dengan menekan tombol “Mulai Pencarian!”. Bila salah satu atau beberapa dari bagian input sebelumnya ada yang belum terisi, maka akan muncul peringatan seperti pada Gambar 4.4.

Misal pengguna memilih suatu direktori “D:\GitHub\Folder-Crawler\test\”, nama file “2.txt” dengan checkbox cari semua file tidak tercentang, dengan algoritma DFS, dan delay penggambaran pohon sebesar 25ms, maka akan muncul tampilan sebagai berikut.

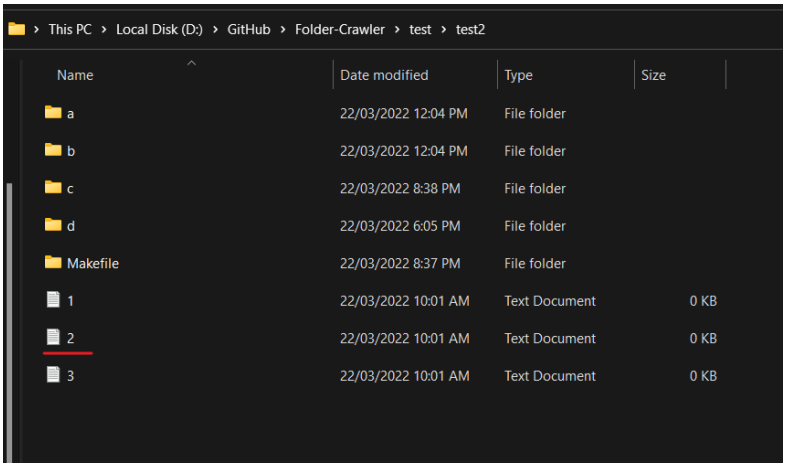


Gambar 4.4 Tampilan Program dengan Input Terisi

Gambar dari pohon dan direktori-direktori file dimana ditemukan file dengan nama yang dicari ditampilkan. Pada pohon, simpul dan sisi yang menyambungkannya diwarnai menjadi hitam, merah, dan biru. Simpul dan sisi berwarna hitam berarti file/folder itu baru saja ditambahkan ke antrean pengecekan pada algoritma. Simpul dan sisi berwarna merah

berarti file/folder sudah dicek kebenarannya terhadap nama file yang dicari dan tidak sesuai. Simpul dan sisi file/folder yang berwarna biru adalah simpul dan sisi yang sudah dicek kebenarannya terhadap nama file yang dicari dan sesuai sehingga menjadi solusi. Perlu dicatat bahwa pewarnaan simpul dan sisi solusi berwarna biru dilakukan hingga mencapai simpul teratas yaitu direktori akar.

Gambar 4.4 Tampilan Hasil Pencarian



Gambar 4.4 File Explorer ketika Menekan Hyperlink Solusi

Solusi berupa direktori file-file yang sudah ditemukan ditampilkan sebagai hyperlink yang dapat diklik oleh pengguna untuk membuka File Explorer pada folder yang memegang file tersebut. Waktu total algoritma pun ditampilkan yang pada kasus ini adalah 5 milisekon.

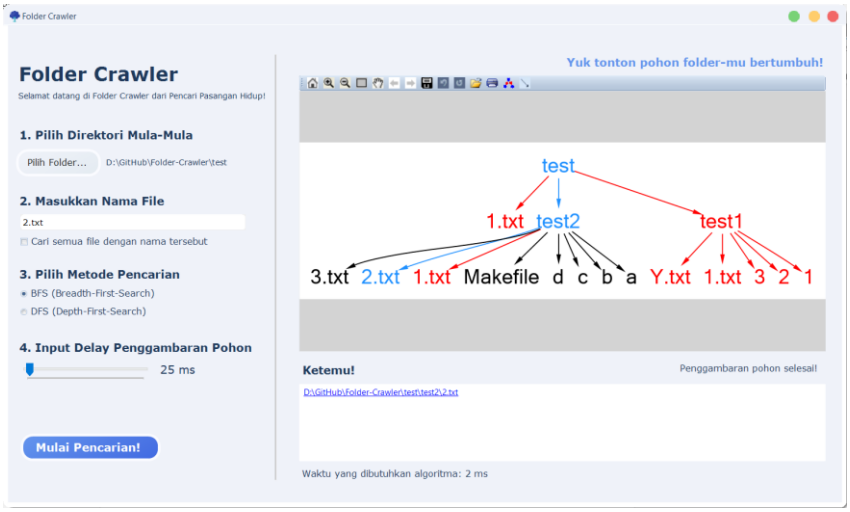
4.5 Hasil Pengujian

Program diuji dengan menggunakan struktur dan konten direktori suatu folder “test” sebagai berikut.

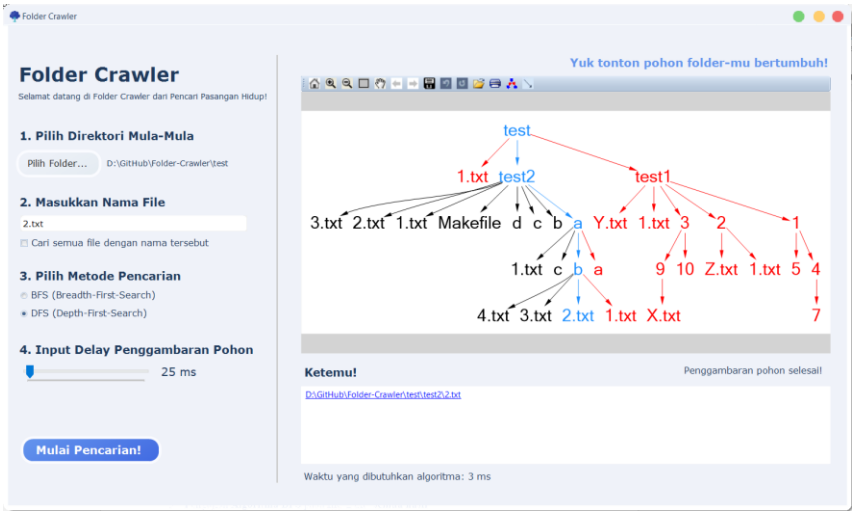
```
test
|   1.txt
|
+---test1
|   |   1.txt
|   |   Y.txt
|   |
|   +---1
|   |   +---4
|   |   |   \---7
|   |   \---5
|   +---2
|   |       1.txt
|   |       Z.txt
|   |
|   \---3
|       +---10
```

```
|       \---9
|               X.txt
|
\---test2
    |   1.txt
    |   2.txt
    |   3.txt
    |
    +---a
    |   |   1.txt
    |   |
    |   +---a
    |   +---b
    |   |       1.txt
    |   |       2.txt
    |   |       3.txt
    |   |       4.txt
    |   |
    |   \---c
    +---b
    |   |   1.txt
    |   |
    |   +---a
    |   +---b
    |   |       Makefile
    |   |
    |   +---c
    |   \---d
    +---c
    |   +---a
    |   +---b
    |   |       1.txt
    |   |       2.txt
    |   |       3.txt
    |   |       4.txt
    |   |
    |   \---Makefile
    +---d
    |   +---a
    |   |       Makefile
    |   |
    |   +---b
    |   +---c
    |   \---d
    \---Makefile
```

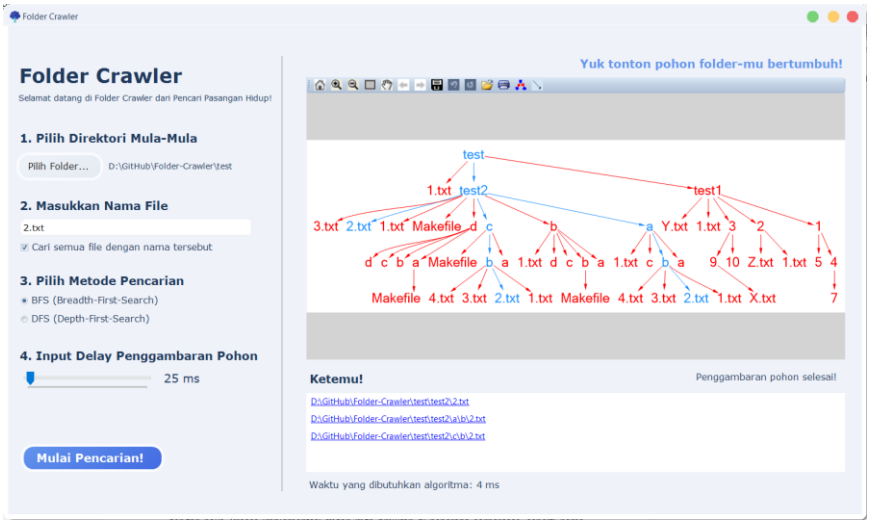
1. Pengujian **Algoritma BFS** pada file '2.txt', **hasil pertama**



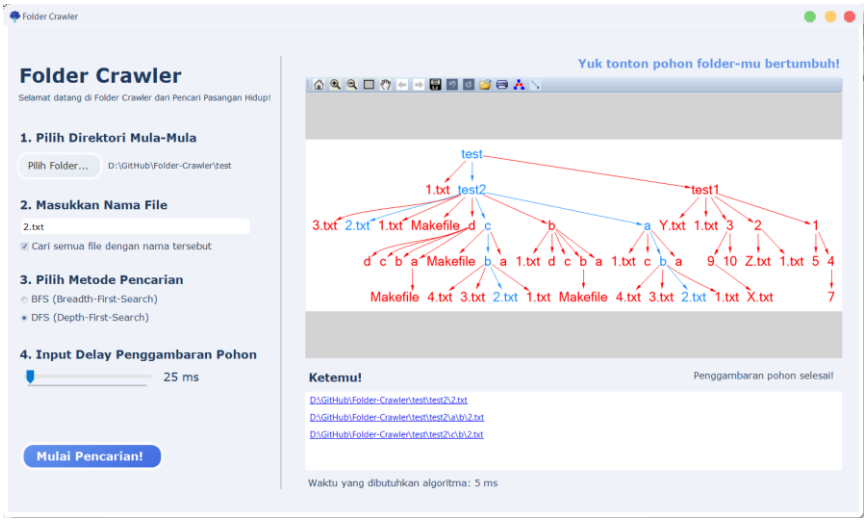
2. Pengujian **Algoritma DFS** pada file '2.txt', **hasil pertama**



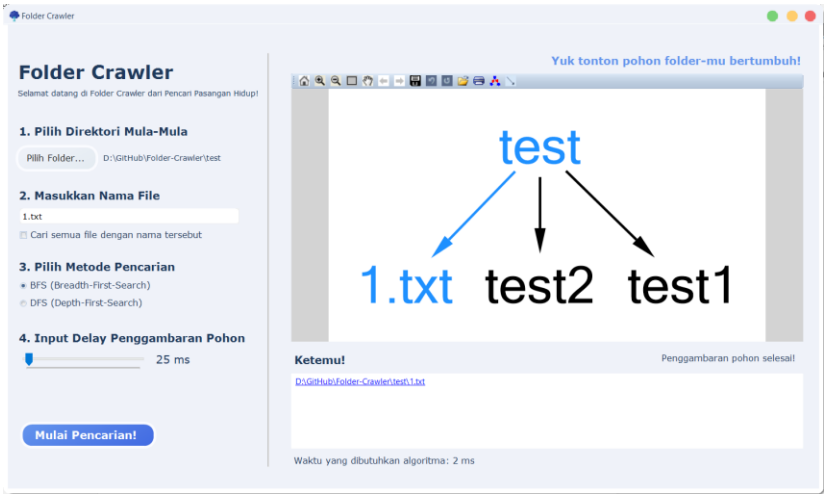
3. Pengujian **Algoritma BFS** pada file '2.txt', **semua hasil**



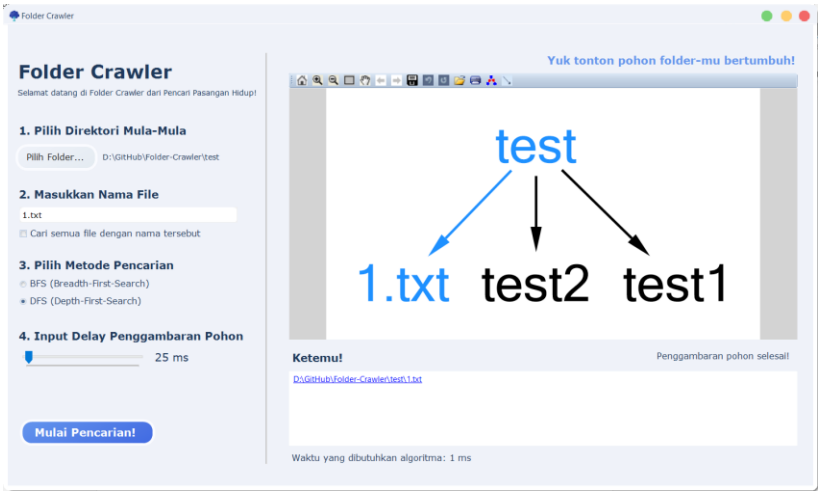
4. Pengujian **Algoritma DFS** pada file '2.txt', **semua hasil**



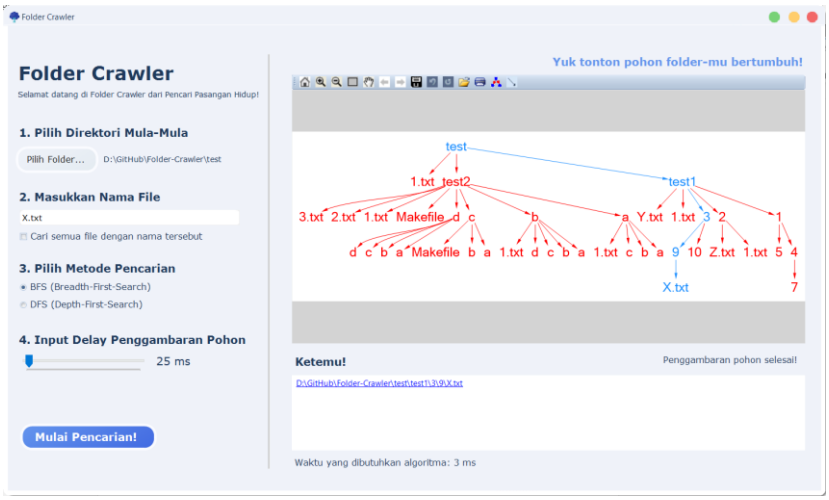
5. Pengujian **Algoritma BFS** pada file '1.txt' yang **terdapat pada folder root**, hasil pertama



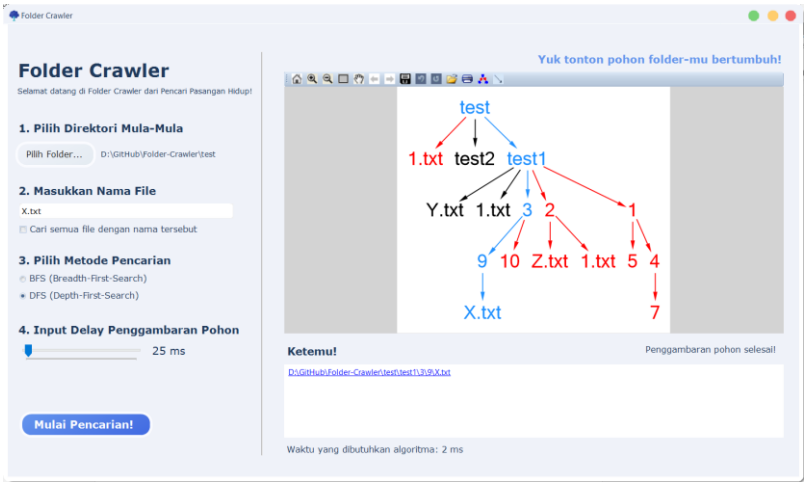
6. Pengujian **Algoritma DFS** pada file '1.txt' yang **terdapat pada folder root**, hasil pertama



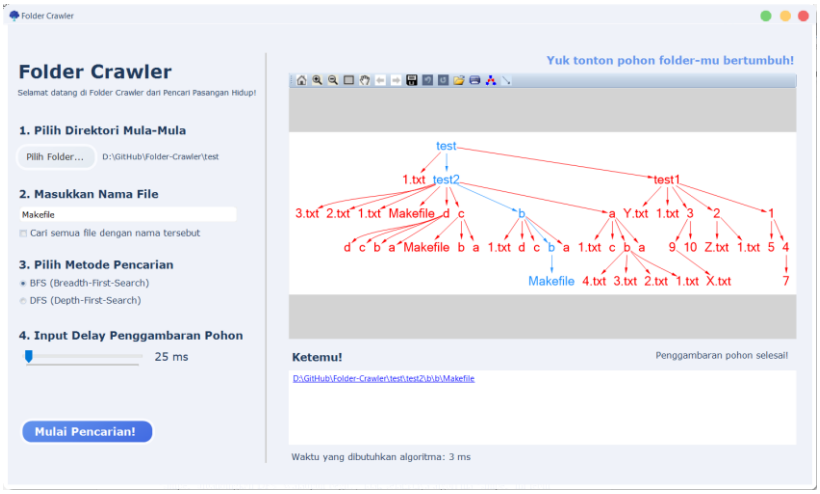
7. Pengujian **Algoritma BFS** pada file 'X.txt' yang **tidak terdapat pada folder root**, hasil pertama



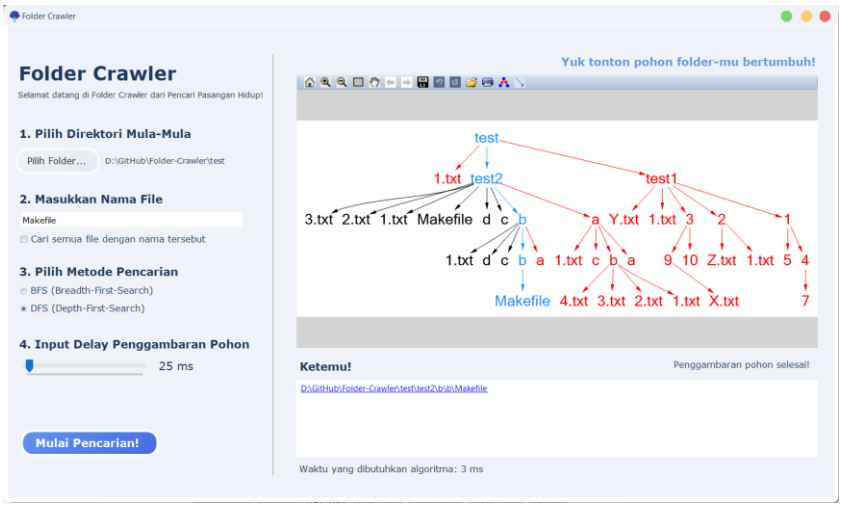
8. Pengujian **Algoritma DFS** pada file 'X.txt' yang **tidak terdapat pada folder root**, hasil pertama



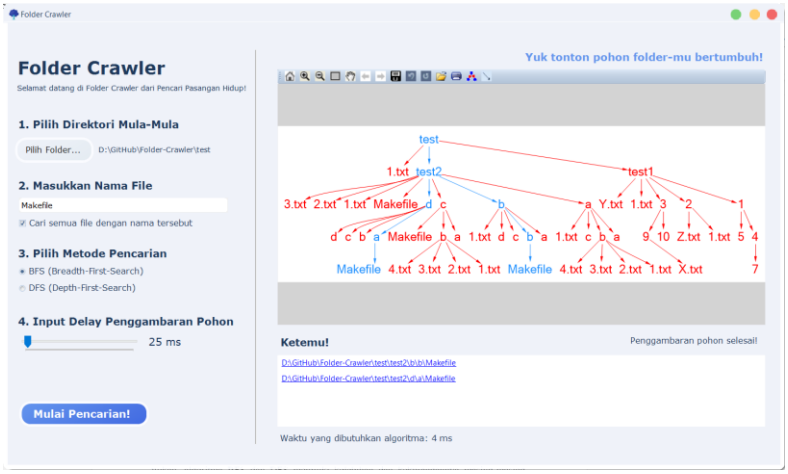
9. Pengujian **Algoritma BFS** pada file 'Makefile', **file tanpa extension dan bernama sama seperti folder**, **hasil pertama**



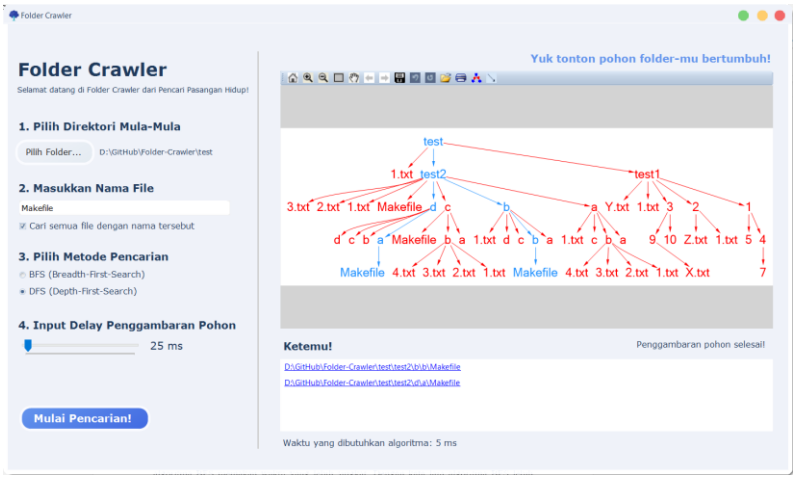
10. Pengujian **Algoritma DFS** pada file 'Makefile', **file tanpa extension dan bernama sama seperti folder**, **hasil pertama**



11. Pengujian **Algoritma BFS** pada file 'Makefile', **file tanpa extension dan bernama sama seperti folder**, **semua hasil**



12. Pengujian **Algoritma DFS** pada file 'Makefile', **file tanpa extension dan bernama sama seperti folder**, **semua hasil**



4.6 Studi Analisis Implementasi Algoritma BFS/DFS

Implementasi algoritma BFS/DFS yang kami kembangkan sudah dapat berjalan dengan baik. Proses implementasi sudah juga diujikan di beberapa cornercase, seperti nama file dan folder yang sama, letak file pada root folder, dll. Pengembangan GUI berbasis windows form juga sudah dapat diimplementasikan dengan baik. Hasil perhitungan pada bagian algoritma sudah sesuai dengan tampilan pada GUI.

Jika kita membandingkan kedua algoritma BFS dan DFS, proses pengembangan algoritma BFS memakan waktu yang lebih singkat. Dengan kata lain algoritma BFS lebih ‘simpel’ dibandingkan DFS. Walaupun begitu, tidak selamanya algoritma ‘simpel’ ini lebih lambat dibandingkan algoritma DFS. Terdapat beberapa kasus ketika waktu komputasi algoritma BFS lebih cepat dibandingkan algoritma DFS. Dari sekian kasus yang pernah kita ujikan, algoritma BFS dan DFS memiliki kelebihan dan kekurangannya masing-masing. Dapat kita simpulkan bahwa, Algoritma BFS akan lebih cocok untuk mencari file dengan level kedalaman yang kecil, sedangkan algoritma DFS akan lebih cocok untuk mencari file dengan level kedalaman yang lebih dalam.

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Algoritma BFS dan DFS berhasil diterapkan dalam aplikasi *folder crawler* dengan GUI dan dapat digunakan untuk mencari suatu file tertentu dari direktori mula-mula. Aplikasi *folder crawler* juga dapat mencari semua kemunculan file maupun hanya satu kemunculan file. Bila file berhasil ditemukan, aplikasi akan memberikan link yang menuju ke file tersebut dan dapat dibuka di file explorer.

5.2 Saran

Beberapa saran yang didapatkan dari pengerjaan aplikasi *folder crawler* adalah sebagai berikut.

1. Pembuatan GUI sebaiknya menggunakan suatu palet basis yang dapat digunakan untuk mengubah tampilan secara terpusat agar memudahkan penggantian-penggantian yang mungkin terjadi.
2. Eksplorasi aplikasi Visual Studio dan Bahasa pemrograman C# perlu dilakukan sebelum pengerjaan tugas agar pengerjaan lebih lancar.
3. Diperlukan laptop/computer yang memadai untuk melakukan pengembangan aplikasi menggunakan IDE Visual Studio, mengingat aplikasi tersebut cukup berat.

LAMPIRAN

Repository Github:

https://github.com/blueguy42/Folder-Crawler

Video Kelompok:

https://youtu.be/XCkLPBznWRM

DAFTAR PUSTAKA

1. Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1) https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf

2. Dokumentasi MSAGL

   https://github.com/microsoft/automatic-graph-layout