

# LAPORAN TUGAS BESAR 2

IF2211 Strategi Algoritma

Pengaplikasian Algoritma BFS dan DFS dalam Implementasi *Folder Crawling*



Disusun oleh:

Kelompok 20

1. Afrizal Sebastian (13520120)
2. Johannes Winson Sukiatmodjo (13520123)
3. Frederik Imanuel Louis (13520163)

PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2022

# DAFTAR ISI

<b>DESKRIPSI TUGAS</b>	<b>5</b>
1.1 Latar belakang	5
1.2 Deskripsi tugas	6
1.3 Spesifikasi program	10
<b>LANDASAN TEORI</b>	<b>14</b>
2.1 Graph traversal, BFS, dan DFS	14
2.2 C# desktop application development	15
<b>ANALISIS PEMECAHAN MASALAH</b>	<b>16</b>
3.1 Langkah-langkah pemecahan masalah	16
3.2 Proses mapping persoalan	16
3.3 Contoh ilustrasi kasus lain	17
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>21</b>
4.1 Implementasi program	21
4.2 Struktur data dan spesifikasi program	25
4.3 Tata cara penggunaan program	27
4.4 Hasil pengujian beserta analisisnya	27
<b>KESIMPULAN DAN SARAN</b>	<b>34</b>
5.1 Kesimpulan	34
5.2 Saran	34
<b>DAFTAR PUSTAKA</b>	<b>35</b>

## DAFTAR GAMBAR

**Gambar 1.1** Fitur Search pada Windows 10 File Explorer

**Gambar 1.2** Contoh input program

**Gambar 1.3** Contoh output program

**Gambar 1.4** Contoh output program jika file tidak ditemukan

**Gambar 1.5** Contoh ketika hyperlink di-klik

**Gambar 1.6** Tampilan layout dari aplikasi desktop yang dibangun

**Gambar 2.1** Tampilan Winforms (Windows Forms) Visual Studio 2019

**Gambar 3.1** Root Folder

**Gambar 3.2** SubFolder 1

**Gambar 3.3** SubFolder 2

**Gambar 3.4** SubFolder 3

**Gambar 3.5** SubFolder 4

**Gambar 3.6** Pencarian satu file dengan BFS

**Gambar 3.7** Pencarian satu file dengan DFS

**Gambar 3.8** Pencarian semua file dengan BFS

**Gambar 3.9** Pencarian semua file dengan DFS

**Gambar 4.1** Antarmuka program file crawler

**Gambar 4.2** Hasil pengujian test case 1 dengan BFS

**Gambar 4.3** Hasil pengujian test case 1 dengan DFS

**Gambar 4.4** Hasil pengujian test case 2 dengan BFS

**Gambar 4.5** Hasil pengujian test case 2 dengan DFS

**Gambar 4.6** Hasil pengujian test case 3 dengan BFS yang mencari semua kemunculan file

**Gambar 4.7** Hasil pengujian test case 3 dengan DFS yang mencari semua kemunculan file

**Gambar 4.8** Hasil pengujian test case 4 dengan BFS

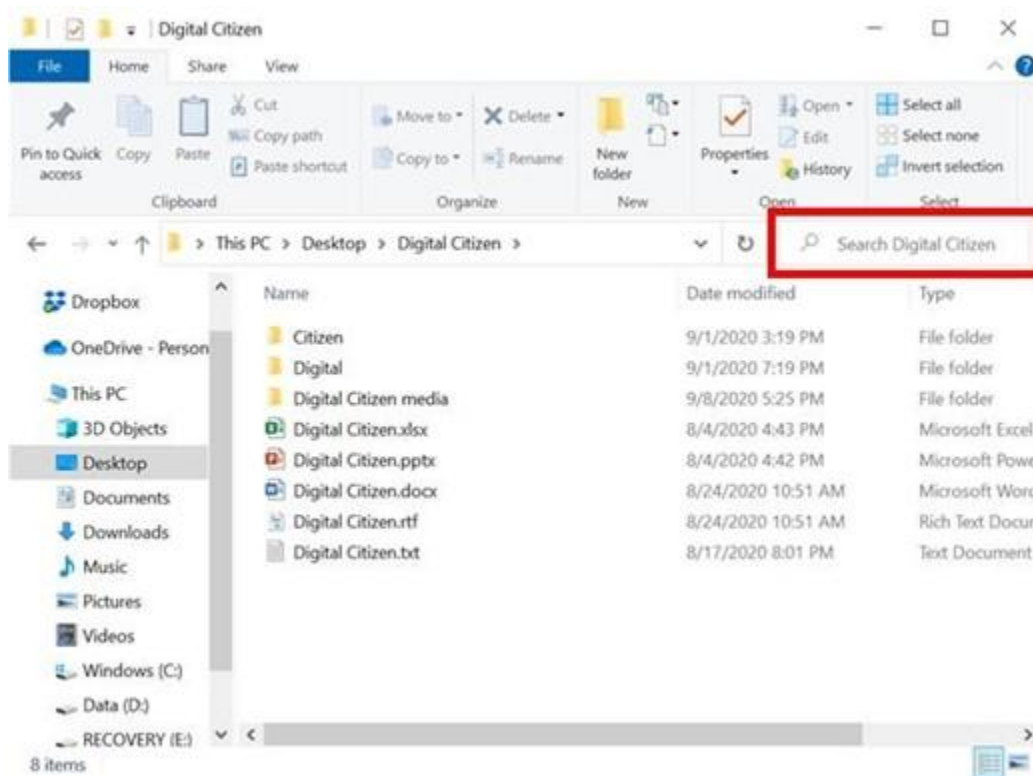
**Gambar 4.9** Hasil pengujian test case 4 dengan DFS

# BAB I

## DESKRIPSI TUGAS

### 1.1 Latar belakang

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 1.1** Fitur Search pada Windows 10 File Explorer

(Sumber: [https://www.digitalcitizen.life/wp-content/uploads/2020/10/explorer\\_search\\_10.png](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 mencari 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.

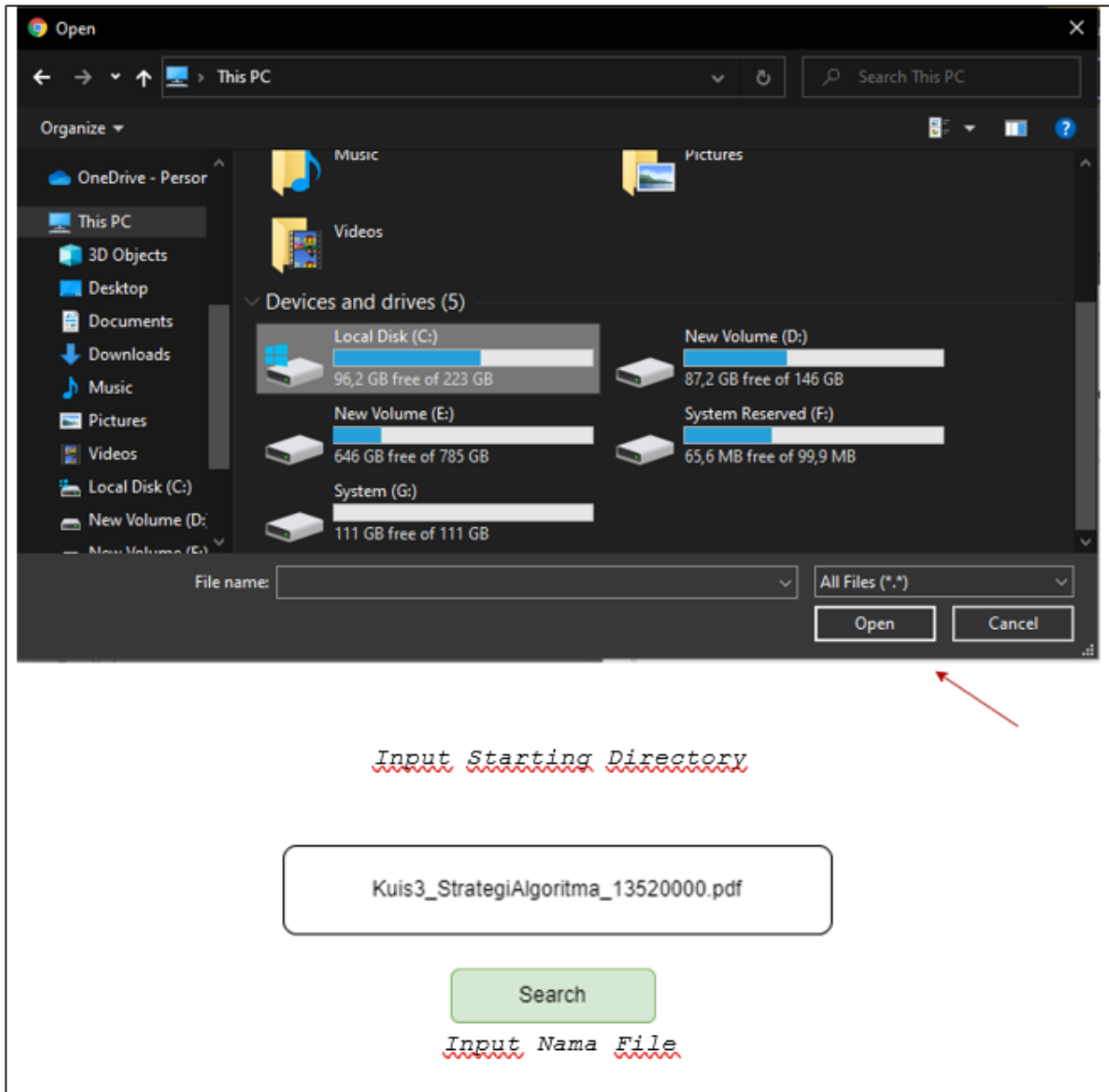
## 1.2 Deskripsi tugas

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.

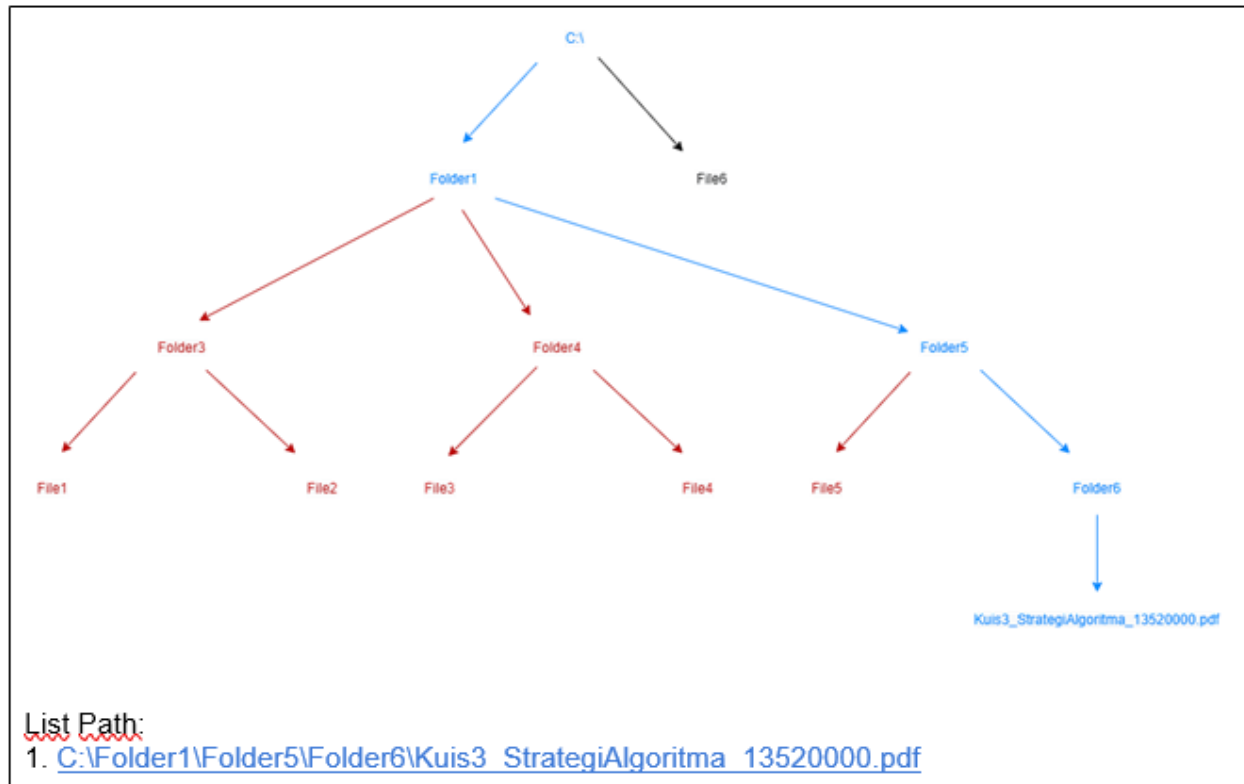
### Contoh Input dan Output Program

Contoh masukan aplikasi:



**Gambar 1.2** Contoh input program

Contoh output aplikasi:



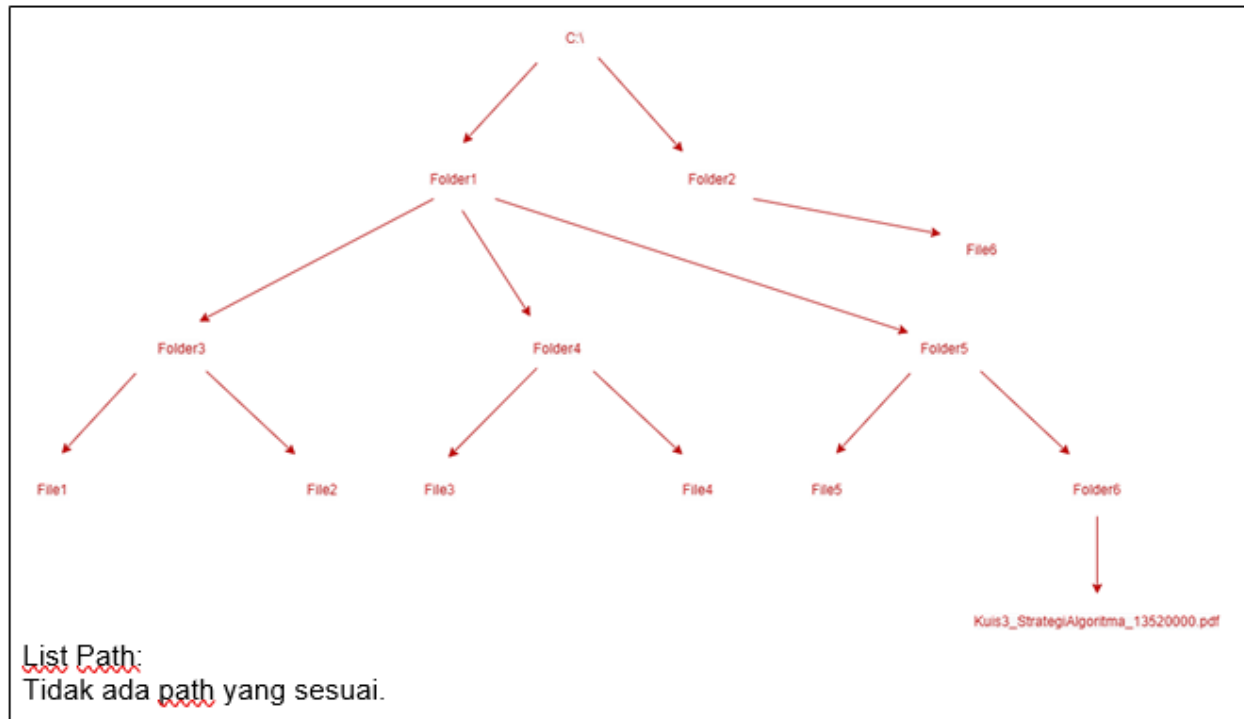
**Gambar 1.3** Contoh output program

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.



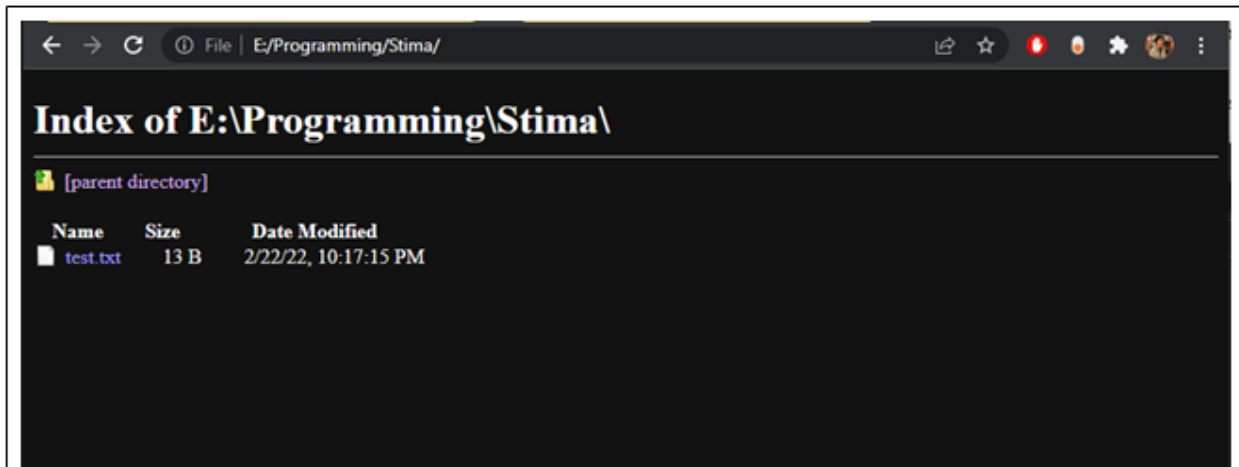


**Gambar 1.4** Contoh output program jika file tidak ditemukan

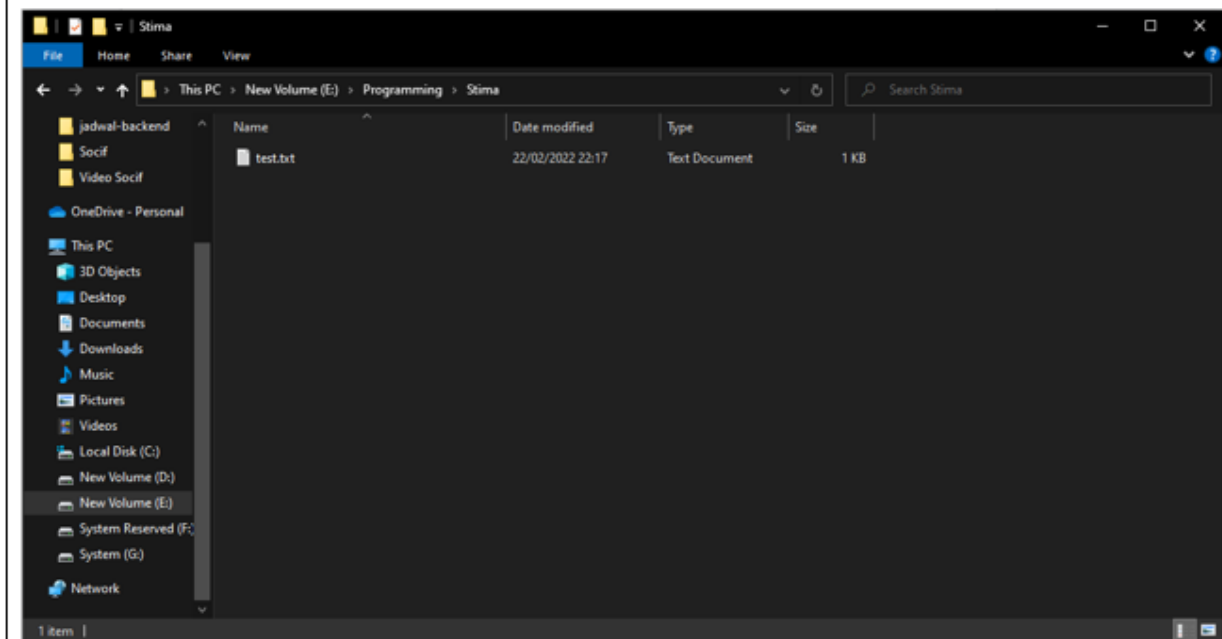
Jika file yang ingin dicari pengguna tidak ada pada direktori file, misalnya saat pengguna mencari Kuis3Probststat.pdf, maka path pencarian DFS adalah sebagai berikut: C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → 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.

### Contoh Hyperlink Pada Path:



Contoh Hyperlink Dibuka Melalui Browser



Contoh Hyperlink Dibuka Melalui Browser

**Gambar 1.5** Contoh ketika hyperlink di-klik

### 1.3 Spesifikasi program

Aplikasi yang akan dibangun dibuat berbasis GUI. Berikut ini adalah contoh tampilan dari aplikasi GUI yang akan dibangun.

## Folder Crawling

### Input

Choose Starting Directory

Choose Folder... No File Chosen

Input File Name

e.g. "word.pdf"

☐ Find all occurrence


Input Metode Pencarian

☐ BFS

☒ DFS

Search

### Output



## Folder Crawling

### Input

Choose Starting Directory

Change Folder... C:/

Input File Name

Kuis3\_StrategiAlgoritma\_13520000.pdf

☐ Find all occurrence

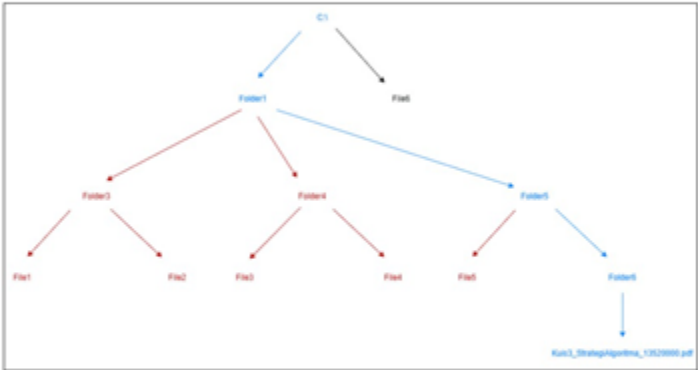
Input Metode Pencarian

☐ BFS

☒ DFS

Search

### Output



Path File :

- [C:/Folder1/Folder5/Folder6/Kuis3\\_StrategiAlgoritma\\_13520000.pdf](#)

Time spent: 20.02s

**Gambar 1.6** Tampilan layout dari aplikasi desktop yang dibangun

**Catatan:** Tampilan diatas hanya berupa salah satu contoh layout dari aplikasi saja, untuk design layout aplikasi dibebaskan dengan syarat mengandung seluruh input dan output yang terdapat pada spesifikasi.

Spesifikasi GUI:

1. Program dapat menerima input folder dan query nama file.
2. Program dapat memilih untuk menampilkan satu hasil saja atau menemukan semua file yang memiliki nama file sama persis dengan input query
3. Program dapat memilih algoritma yang digunakan.
4. Program dapat menampilkan pohon hasil pencarian file tersebut dengan memberikan keterangan folder/file yang sudah diperiksa, folder/file yang sudah masuk antrian tapi belum diperiksa, dan rute folder serta file yang merupakan rute hasil pertemuan.
5. **(Bonus)** Program dapat menampilkan progress pembentukan pohon dengan menambahkan node/simpul sesuai dengan pemeriksaan folder/file yang sedang berlangsung.
6. Program dapat menampilkan hasil pencarian berupa rute/path (bisa lebih dari satu jika memilih menemukan semua file) serta durasi waktu algoritma.
7. GUI dapat dibuat **sekreatif** mungkin asalkan memuat 5(6 jika mengerjakan bonus) spesifikasi di atas.

Program yang dibuat harus memenuhi **spesifikasi wajib** sebagai berikut:

- 1) Buatlah program dalam bahasa **C#** untuk melakukan penelusuran *Folder Crawling* sehingga diperoleh hasil pencarian file yang diinginkan. Penelusuran harus memanfaatkan algoritma **BFS dan DFS**.
- 2) Awalnya program menerima sebuah input folder pada direktori yang ada dan nama file yang akan dicari oleh program.
- 3) Terdapat dua pilihan pencarian, yaitu:
  - a. Mencari 1 file saja  
Program akan memberhentikan pencarian ketika sudah menemukan file yang memiliki nama sama persis dengan input nama file.
  - b. Mencari semua kemunculan file pada folder root  
Program akan berhenti ketika sudah memeriksa semua file yang terdapat pada folder root dan program akan menampilkan daftar semua rute file yang memiliki nama sama persis dengan input nama file
- 4) Program kemudian dapat menampilkan **visualisasi pohon pencarian file** berdasarkan informasi direktori dari folder yang di-input. Pohon hasil pencarian file ini memiliki root adalah folder yang di-input dan setiap daunnya adalah file yang ada di folder root tersebut. Setiap folder/file direpresentasikan sebagai sebuah node atau

simpul pada pohon. Cabang pada pohon menggambarkan folder/file yang terdapat di folder *parent*-nya.

Visualisasi pohon juga harus disertai dengan **keterangan** node yang sudah diperiksa, node yang sudah masuk antrian tapi belum diperiksa, dan node yang bagian dari rute hasil penemuan.

Proses visualisasi ini boleh memanfaatkan pustaka atau kakas yang tersedia. Sebagai referensi, salah satu kakas yang tersedia untuk melakukan visualisasi adalah **MSAGL** (<https://github.com/microsoft/automatic-graph-layout>) Berikut ini adalah panduan singkat terkait penggunaan MSAGL oleh tim asisten yang dapat diakses pada:

<https://docs.google.com/document/d/1XhFSpHU028Gaf7YxkmdbluLkQgVI3MY6gt1t-PL30LA/edit?usp=sharing>

- 5) Program juga dapat menyediakan *hyperlink* pada setiap hasil rute yang ditemukan. *Hyperlink* ini akan membuka folder parent dari file yang ditemukan. Folder hasil *hyperlink* dapat dibuka dengan *browser* atau *file explorer*.
- 6) Mahasiswa **tidak diperkenankan** untuk melihat atau menyalin library lain yang mungkin tersedia bebas terkait dengan pemanfaatan BFS dan DFS. Tapi untuk algoritma lainnya seperti *string matching* dan akses *directory*, diperbolehkan menggunakan library jika ada.

## **BAB II**

### **LANDASAN TEORI**

#### **2.1 Graph traversal, BFS, dan DFS**

Graph traversal adalah suatu algoritma pencarian solusi dari sebuah graf terhubung yang merepresentasikan persoalan dengan mengunjungi tiap simpul/node dengan cara yang sistematis. Graph traversal dapat dibagi menjadi dua metode, yaitu:

1. Pencarian melebar (Breadth First Search / BFS)

Breadth First Search (BFS) adalah metode traversing yang digunakan dalam grafik dengan menggunakan antrian untuk menyimpan simpul yang dikunjungi. Dalam metode ini, yang ditekankan adalah pada simpul grafik. Satu simpul dipilih pada awalnya kemudian dikunjungi dan ditandai. Verteks yang berdekatan dengan verteks yang dikunjungi kemudian dikunjungi dan disimpan dalam antrian berurutan. Demikian pula, simpul yang disimpan kemudian diperlakukan satu per satu, dan simpul yang berdekatan dikunjungi. Sebuah node sepenuhnya dieksplorasi sebelum mengunjungi node lain dalam grafik. Dengan kata lain, node tersebut melintasi node yang belum dieksplorasi paling dangkal terlebih dahulu. Berikut merupakan alur jalan algoritma BFS.

- a. Kunjungi simpul v.
- b. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
- c. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

BFS dapat berguna dalam menemukan apakah grafik memiliki komponen yang terhubung atau tidak, sekaligus juga dapat digunakan dalam mendeteksi grafik bipartit. BFS juga lebih baik dalam menemukan jalur terpendek dalam grafik yang dapat dilihat sebagai jaringan.

2. Pencarian mendalam (Depth First Search / DFS)

Metode traverse Depth First Search (DFS) menggunakan stack untuk menyimpan simpul yang dikunjungi. DFS adalah metode berbasis tepi dan bekerja secara rekursif di mana simpul dieksplorasi di sepanjang jalur (tepi). Eksplorasi sebuah node ditangguhkan segera setelah simpul lain yang belum dijelajahi ditemukan dan simpul-simpul yang belum dijelajahi yang terdalam ditelusuri pada titik terdepan. DFS melintasi/mengunjungi setiap simpul tepat sekali dan setiap tepi diperiksa tepat dua kali. Berikut merupakan alur jalan algoritma BFS.

- a. Kunjungi simpul v.
- b. Kunjungi simpul w yang bertetangga dengan simpul v.
- c. Ulangi DFS mulai dari simpul w.
- d. 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.

- e. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

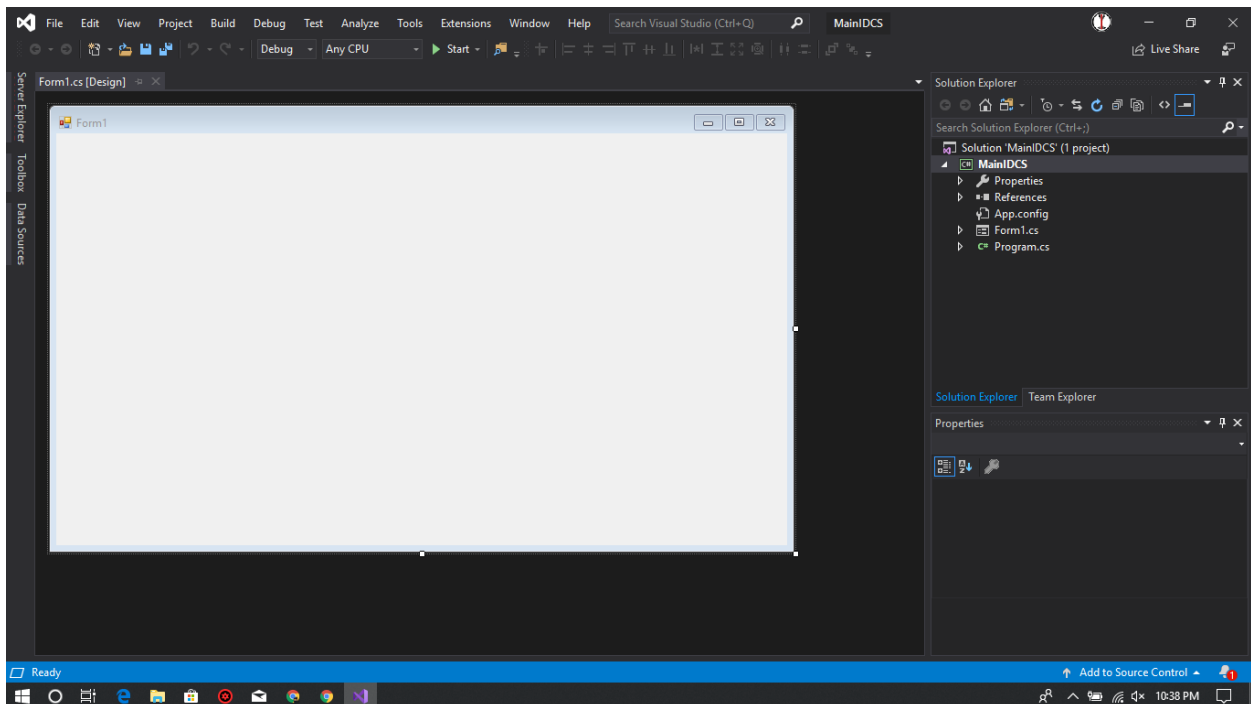
Aplikasi DFS mencakup pemeriksaan dua graf yang terhubung ke tepi, graf yang terhubung kuat, grafik asiklik, dan urutan topologi.

## 2.2 C# desktop application development

C# desktop application development adalah suatu kegiatan pengembangan aplikasi desktop dengan menggunakan bahasa pemrograman C#. Terdapat beberapa framework yang digunakan untuk pengembangan aplikasi desktop, salah satunya adalah Winforms yang merupakan kerangka kerja dalam .NET framework yang terutama digunakan untuk mengembangkan aplikasi GUI sederhana.

Microsoft .NET Framework adalah sebuah perangkat lunak pada komputer (software komputer) yang digunakan untuk memudahkan pengembangan dan eksekusi program dari berbagai macam bahasa pemrograman dan sekumpulan library agar sistem operasi Windows dapat menjalankan sebuah aplikasi. Fungsi utama dari .NET Framework pada Windows adalah sebagai penerjemah dan pengeksekusi sebuah perangkat lunak agar dapat berjalan pada sistem operasi Windows. Sebenarnya tidak semua aplikasi membutuhkan .NET framework versi terbaru atau tidak perlu melakukan install .NET framework. Akan tetapi pada beberapa kasus pada saat melakukan instalasi aplikasi, kita diwajibkan untuk melakukan instalasi .NET framework terlebih dahulu agar bisa menjalankan atau melakukan instalasi aplikasi tersebut.

Winforms atau Windows Forms merupakan salah satu jenis Class Library yang diciptakan Microsoft untuk membuat suatu aplikasi atau software berbasis GUI (Graphical User Interface). Winforms ini tergabung dalam framework .NET. Bahasa pemrograman yang dapat digunakan untuk membuat aplikasi berbasis Winforms adalah C# dan Visual Basic.



Gambar 2.1 Tampilan Winforms (Windows Forms) Visual Studio 2019

## BAB III

### ANALISIS PEMECAHAN MASALAH

#### 3.1 Langkah-langkah pemecahan masalah

Dari penjabaran permasalahan pada Bab I Deskripsi Tugas, dilakukan langkah pemecahan masalah sebagai berikut :

1. Memahami permasalahan pada Deskripsi Tugas.
2. Memahami algoritma BFS dan DFS dalam mencari goal node pada pohon pencarian.
3. Memetakan permasalahan menjadi elemen-elemen BFS dan DFS.
4. Mempelajari pemrograman C# dan penggunaan Visual Studio.
5. Mengimplementasikan algoritma BFS dan DFS dalam pemecahan masalah dengan menggunakan bahasa pemrograman C#.
6. Pembuatan GUI dan Visualisasi Tree menggunakan *library* MSAGL.

#### 3.2 Proses mapping persoalan

Berikut merupakan proses mapping persoalan menjadi elemen-elemen algoritma BFS dan DFS.

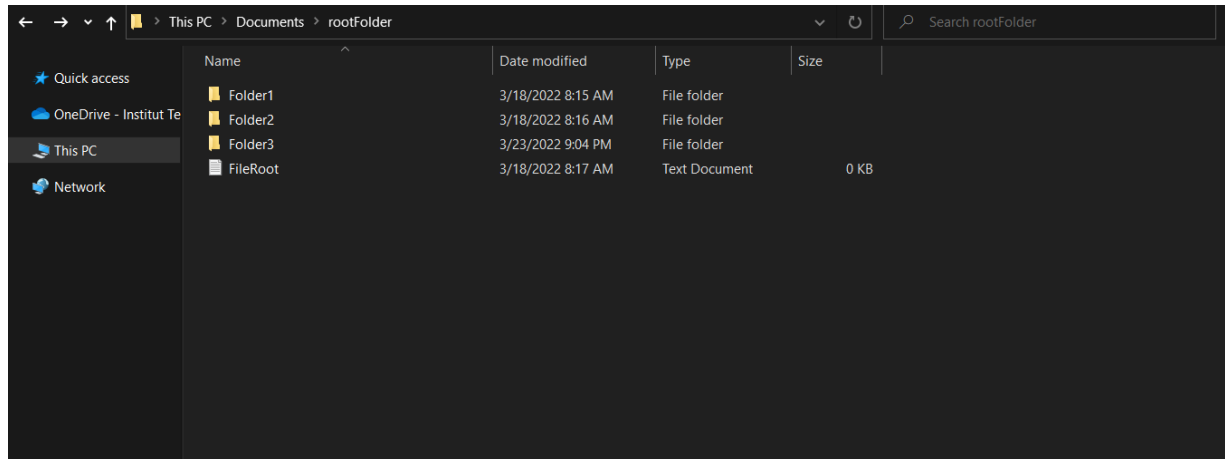
1. Pencarian sebuah file dengan nama tertentu dalam root folder  
Root folder akan bertindak sebagai simpul awal dan semua subfolder dan file yang ada di dalam root folder akan bertindak sebagai child.
  - a. Problem State : Mencari sebuah file dengan nama tertentu yang berada di dalam root folder
  - b. Initial State : root folder
  - c. Solution State : File yang memiliki nama yang sesuai
  - d. State Space : Himpunan semua nama folder dan file yang ada di root folder
  - e. State Space Tree : Pohon dari semua folder dan file pada State Space
  - f. Solution Space : Himpunan file di root folder
  - g. Solution : File yang memiliki nama yang sesuai dengan yang dicari
2. Pencarian semua file dengan nama tertentu dalam root folder  
Root folder akan bertindak sebagai simpul awal dan semua subfolder dan file yang ada di dalam root folder akan bertindak sebagai child.
  - a. Problem State : Mencari sebuah file dengan nama tertentu yang berada di dalam root folder.
  - b. Initial State : root folder
  - c. Solution State : Semua file yang memiliki nama yang sesuai
  - d. State Space : Himpunan semua nama folder dan file yang ada di root folder
  - e. State Space Tree : Pohon dari semua folder dan file pada State Space
  - f. Solution Space : Himpunan file di root folder.
  - g. Solution : Semua file yang memiliki nama yang sesuai dengan yang dicari



### 3.3 Contoh ilustrasi kasus lain

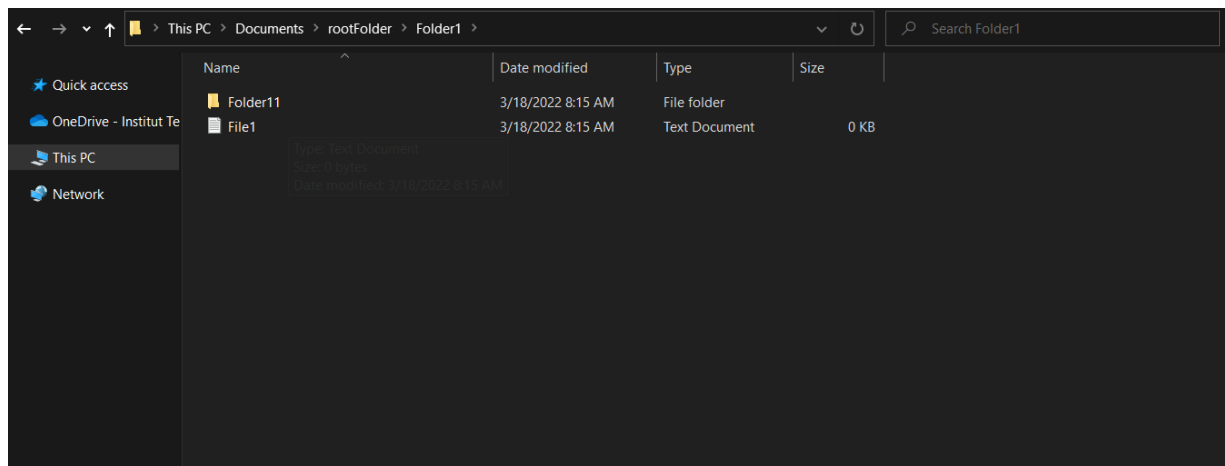
Berikut diberikan contoh sebuah root Folder.

#### Root Folder



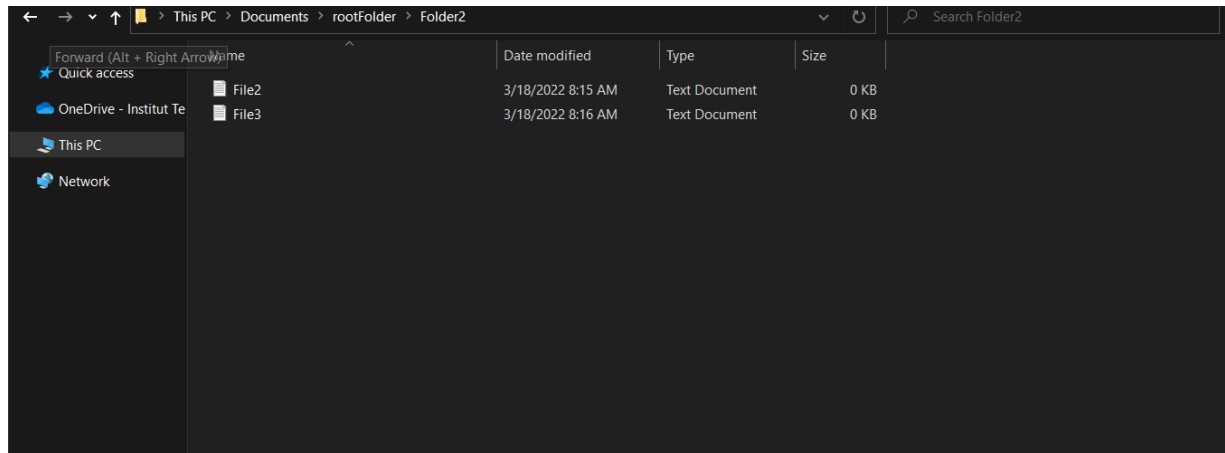
**Gambar 3.1** Root Folder

#### Folder1



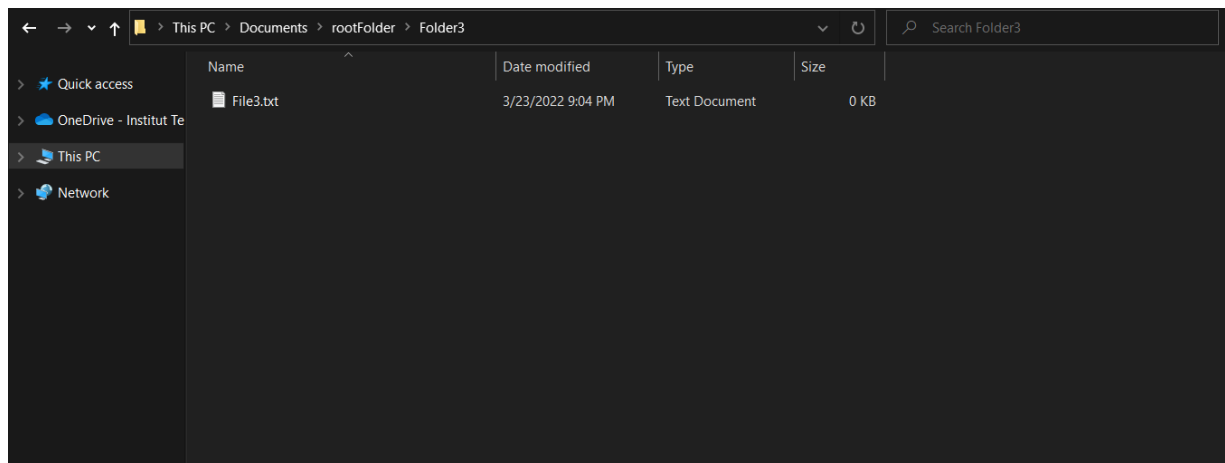
**Gambar 3.2** SubFolder 1

## Folder2



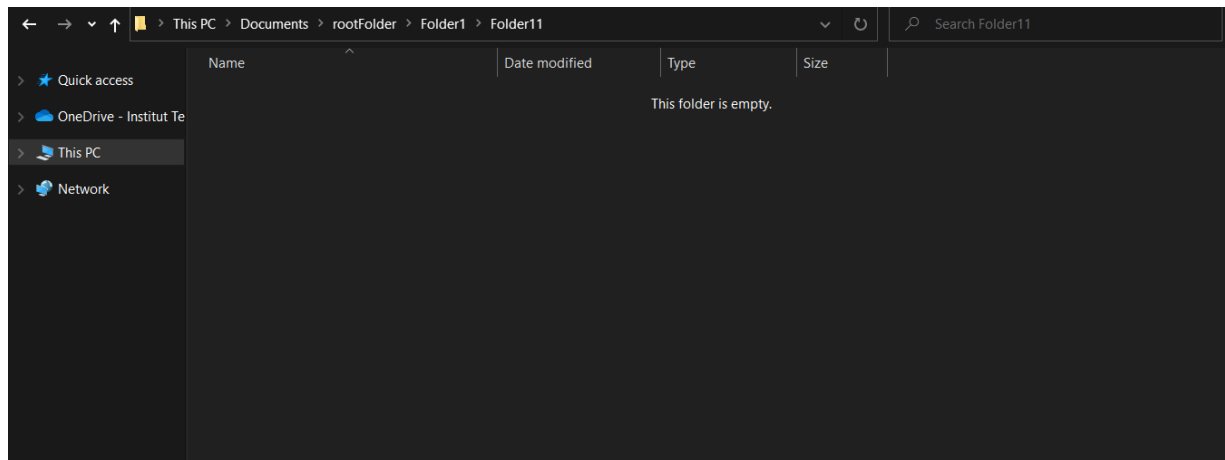
**Gambar 3.3** SubFolder 2

## Folder3



**Gambar 3.4** SubFolder 3

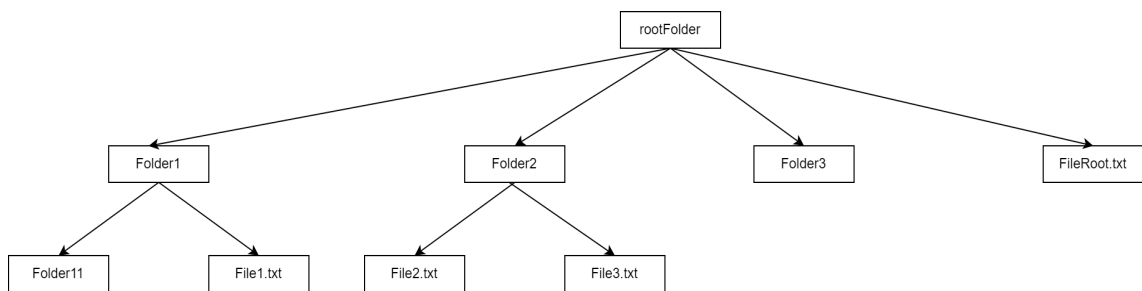
## Folder11



**Gambar 3.5** SubFolder 4

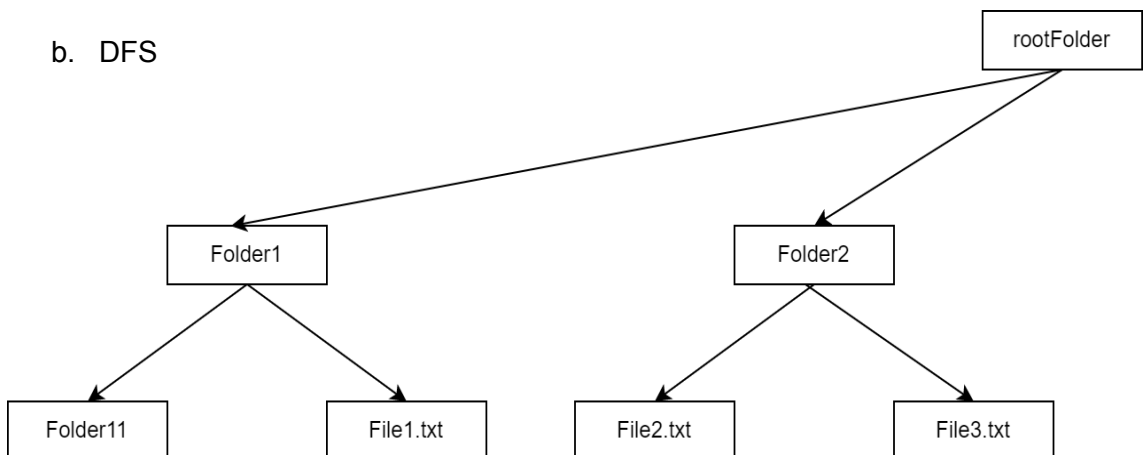
Misalkan rootFolder bertindak sebagai Root Folder dan yang ingin dicari adalah File3.txt, maka visualisasi dari permasalahan tersebut adalah sebagai berikut.

1. Satu File Saja
  - a. BFS



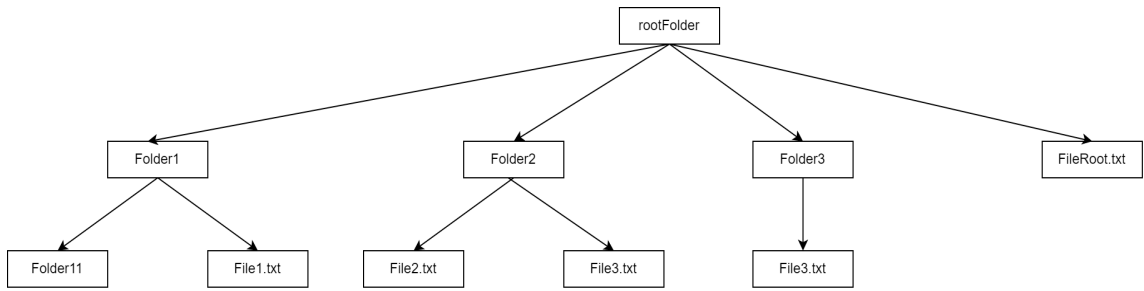
**Gambar 3.6** Pencarian satu file dengan BFS

- b. DFS



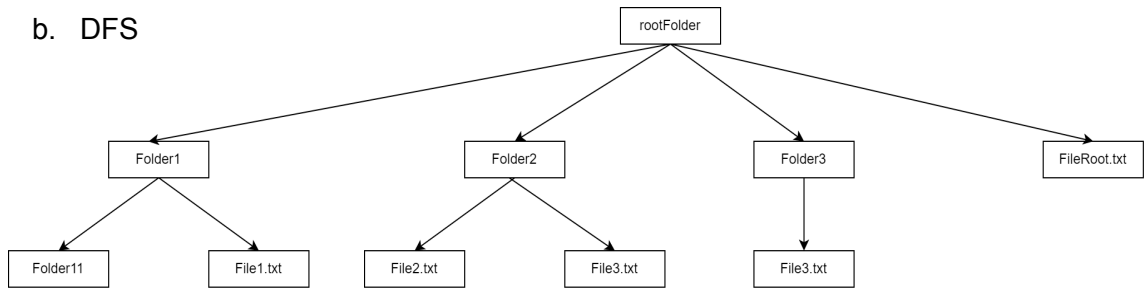
**Gambar 3.7** Pencarian satu file dengan DFS

2. Semua File  
a. BFS



**Gambar 3.8** Pencarian semua file dengan BFS

b. DFS



**Gambar 3.9** Pencarian semua file dengan DFS

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi program

##### 1. Main.cs

Create forms object  
Run forms

##### 2. Forms.cs

Forms class:

Attributes:  
    RootFolder  
    SearchType  
    AllOccurrences  
    TargetFile

Constructor:  
    Create form buttons and boxes

Choose root folder: //Event Handler  
    RootFolder = Input

Choose search type: //Event Handler  
    SearchType = Input

Choose occurrences: //Event Handler  
    AllOccurrences = Input

Choose target file: //Event Handler  
    TargetFile = Input

Search button click: //Event Handler  
    If (one attribute is not valid):  
        Print error message  
    Else:  
        If (SearchType is "DFS"):  
            Create DFS object  
            If (AllOccurrences is True):  
                Call DFS.ManyFiles  
                Output results  
        Else:  
            Call DFS.OneFiles

```
        Output results
    If (SearchType is "BFS"):
        Create BFS object
        If (AllOccurrences is True):
            Call BFS.BFSManyFiles
            Output results
        Else:
            Call BFS.BFSOneFile
            Output results
```

### 3. DFS.cs

DFS Class:

Attribute:

Blue //List of string  
Black //List of string  
AdjList //List of List of string  
Parent //Dictionary  
PathToName //Dictionary

GetName (Path): //maps path to generated name to handle duplicate folders

If (Path not in PathToName):  
    PathToName[Path] = Name //Generated name  
Return PathToName[Path]

DFSOneFile(Parent, Target):

Ret = "File tidak ditemukan" //return value  
Directories = Parent.Directories //uses get name  
Files = Parent.Files  
Tree.insert(Parent, {Files,Directories})

For file in Files:

    If (Ret is not "File tidak ditemukan"):  
        Black.insert(file)  
    Else If (file is target):  
        Blue.insert(target.Name)  
        Ret = target.FullName

For directory in Directories:

    If (Ret is not "File tidak ditemukan"):  
        Black.insert(GetName(directory))  
    Else:  
        Parent[GetName(directory)] = GetName(directory)  
        Ret = DFSOneFile(directory, Target)

If (Ret is not "File tidak ditemukan"):  
    Blue.insert(GetName(Parent))

Return Ret

DFSManyFiles(Parent,Target,Answers):

Directories = Parent.Directories //uses get name  
Files = Parent.Files  
Tree.insert(Parent, {Files,Directories})

For file in Files:

    If (file is target):  
        Blue.insert(target.Name)  
        Answers.insert(target.FullName)

```

For directory in Directories:
    Parent[GetName(directory)] = GetName(directory)
    Cur = Answers.length
    Ret = DFSOneFile(directory, Target, Answers)
    If (Cur < Answers.length):
        Blue.insert(GetName(directory))

```

#### 4. BFS.cs

BFS Class:

Attribute:

```

Blue //List of string
Black //List of string
AdjList //List of List of string
Parent //Dictionary
PathToName //Dictionary

```

```

GetName (Path): //maps path to generated name to handle duplicate folders
    If (Path not in PathToName):
        PathToName[Path] = Name //Generated name
    Return PathToName[Path]

```

```

BFSOneFile(Root, Target):
    Ret = "File tidak ditemukan" //return value
    Queue.push(Root)
    parent[Root] = Root

```

```

While (Queue is not empty):
    Parent = Queue.pop()
    Directories = Parent.Directories //uses get name
    Files = Parent.Files
    Tree.insert(Parent, {Files,Directories})

```

```

For file in Files:
    If (Ret is not "File tidak ditemukan"):
        Black.insert(file)
    Else If (file is target):
        P = getName(Parent)
        Blue.insert(target.Name)
        Blue.insert(P) //create blue path
        While (Parent[P] is not P):
            P = Parent[P]
            Blue.insert(P)
        Ret = target.FullName

```

```

For folder in Folders:
    Parent[getName(folder)] = getName(Parent)
    Queue.push(folder.FullName)

```



```

    If (Ret is not "File tidak ditemukan"):
        While (Queue is not empty): //blacks
            Black.insert(getName(Queue.pop()))

    Return Ret

BFSManyFile(Root, Target, Answer):
    Queue.push(Root)
    parent[Root] = Root

    While (Queue is not empty):
        Parent = Queue.pop()
        Directories = Parent.Directories //uses get name
        Files = Parent.Files
        Tree.insert(Parent, {Files,Directories})

    For file in Files:
        If (file is target):
            P = getName(Parent)
            Blue.insert(target.Name)
            Blue.insert(P) //create blue path
            While (Parent[P] is not P):
                P = Parent[P]
                Blue.insert(P)
            Answer.insert(file.FullName)

    For folder in Folders:
        Parent[getName(folder)] = getName(Parent)
        Queue.push(folder.FullName)

```

## 4.2 Struktur data dan spesifikasi program

Program menggunakan empat buah kelas, yaitu kelas Main, Forms, BFS, dan DFS. Kelas BFS dan DFS masing-masing digunakan untuk penelusuran filesystem dengan algoritma BFS dan DFS. BFS atau DFS diinstansiasi menjadi objek pada kelas Forms tiap kali search dilakukan. Kelas Forms melakukan seluruh proses frontend dari program, mulai dari penerimaan input user, sampai output graph dengan MSAGL, serta pengeluaran hyperlink file yang ditemukan. Kelas Main menginstansiasi Forms menjadi sebuah objek, dan alur program dimulai dari method Main pada kelas Main.

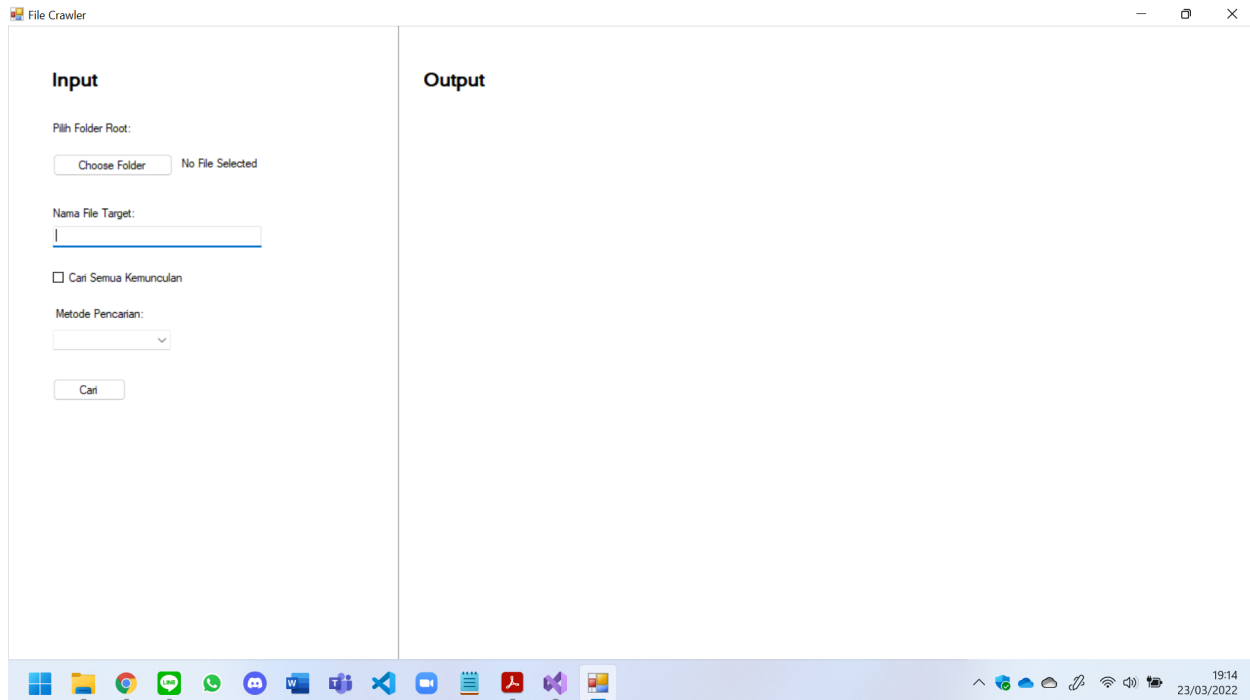
Kelas BFS memiliki atribut pairNode, pathToFile, pairNode, blue, dan black. Atribut pairNode memiliki tipe List of Tuple of String, List of String, dan digunakan sebagai adjacency list dari pohon yang akan dibentuk. Atribut pathToFile memiliki tipe dictionary of string to string yang digunakan untuk menyimpan mapping path ke filename. Hal tersebut menangani nama folder yang sama pada path yang berbeda. Atribut blue dan black memiliki tipe List of String yang masing-masing menyimpan daftar nama file yang akan diwarnai biru dan hitam pada graph. Semua file lain yang tidak ada pada kedua list tersebut akan diwarnai merah. Kelas ini

juga memiliki tiga getters, yaitu `getPairNode`, `getBlue`, dan `getBlack` yang masing-masing mengembalikan atribut `pairNode`, `blue`, dan `black`. Kelas ini juga memiliki dua method utility, yaitu `getNameDirectory` yang berfungsi sebagai setter dan getter pada dictionary `pathToFile`, serta `convertNameToList` yang berfungsi membantu membentuk adjacency list.

Selain itu, kelas BFS memiliki dua method utama, yaitu `BFSOneFile` dan `BFSManyFile`, yang masing-masing bertanggung jawab untuk BFS sampai file ditemukan, dan BFS sampai semua kemunculan file ditemukan. `BFSOneFile` menerima parameter folder root dan file target, dan mengeluarkan string yaitu path lengkap file target apabila target ditemukan. Method tersebut juga akan mengubah isi atribut objek BFS yang diinstansiasi. `BFSManyFile` bekerja dengan prinsip yang sama, tetapi memiliki satu masukan tambahan yaitu list of string yang berfungsi menyimpan tiap path file yang ditemukan, serta tidak memiliki keluaran. Keluaran akan dicek dari isi list of string yang diberikan. Kelas DFS memiliki atribut dan method utility yang serupa dengan kelas BFS, tetapi ia memiliki method utama `DFSOneFile` dan `DFSManyFile`, yang bekerja dengan cara yang serupa dengan method utama kelas BFS, tetapi dengan menggunakan algoritma DFS.

Kelas `Forms` merupakan kelas turunan dari library `WinForms` dan memiliki berbagai atribut yang merupakan elemen dari tampilan forms, seperti textbox, drop down list, button, label, dan sebagainya. Selain itu, kelas tersebut juga memiliki atribut `RootFolder`, `FileName`, `AllOccurrences`, dan `SearchType` yang berfungsi menyimpan masukan user. Kelas ini juga memiliki berbagai method yang menangani tampilan forms, seperti `initializeComponent` yang mengatur komponen-komponen Forms yang dibuat. Selain itu, ia juga memiliki beberapa method yang bersifat event based, seperti saat tombol dipencet. Salah satu method tersebut yang penting adalah event click tombol search, yang menginstansiasi objek BFS atau DFS, dan memanggil fungsi file search sesuai dengan masukan User. Kelas `Main` menginstansiasi `Forms` sebagai objek pada method `Main`, dan method tersebutlah yang berfungsi menjadi entry-point program. Kelas `main` tidak memiliki atribut atau method lain.

### 4.3 Tata cara penggunaan program



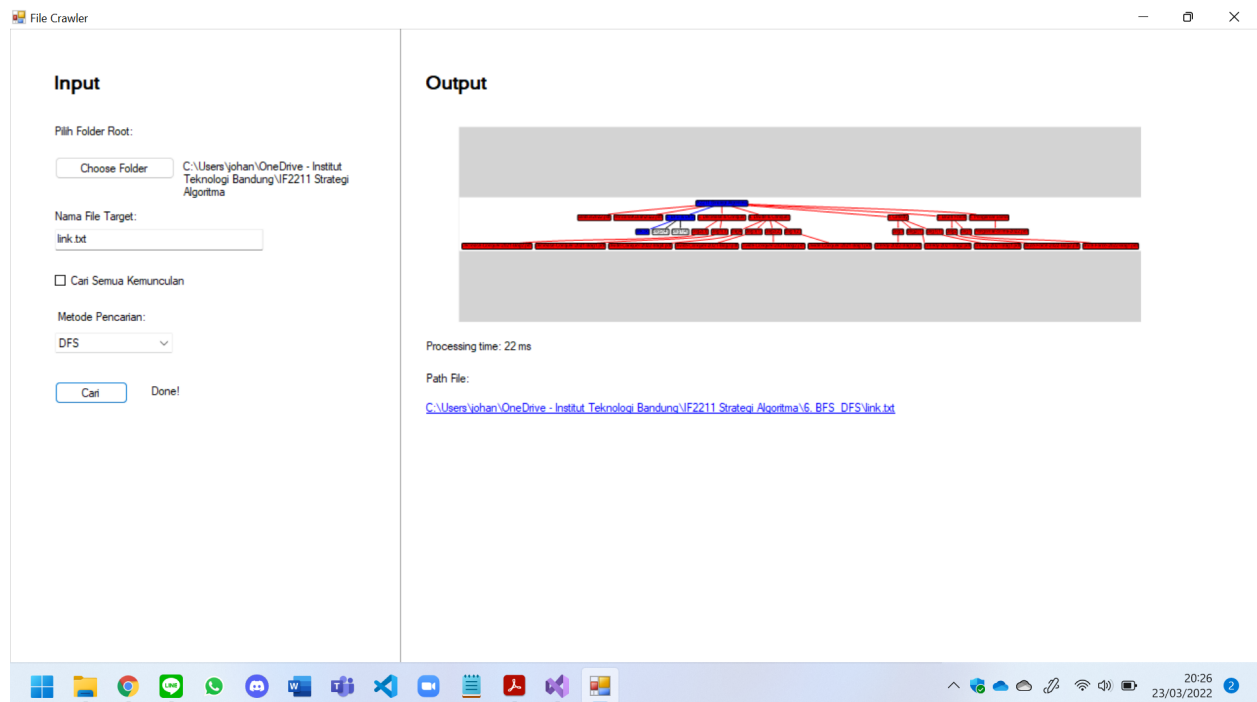
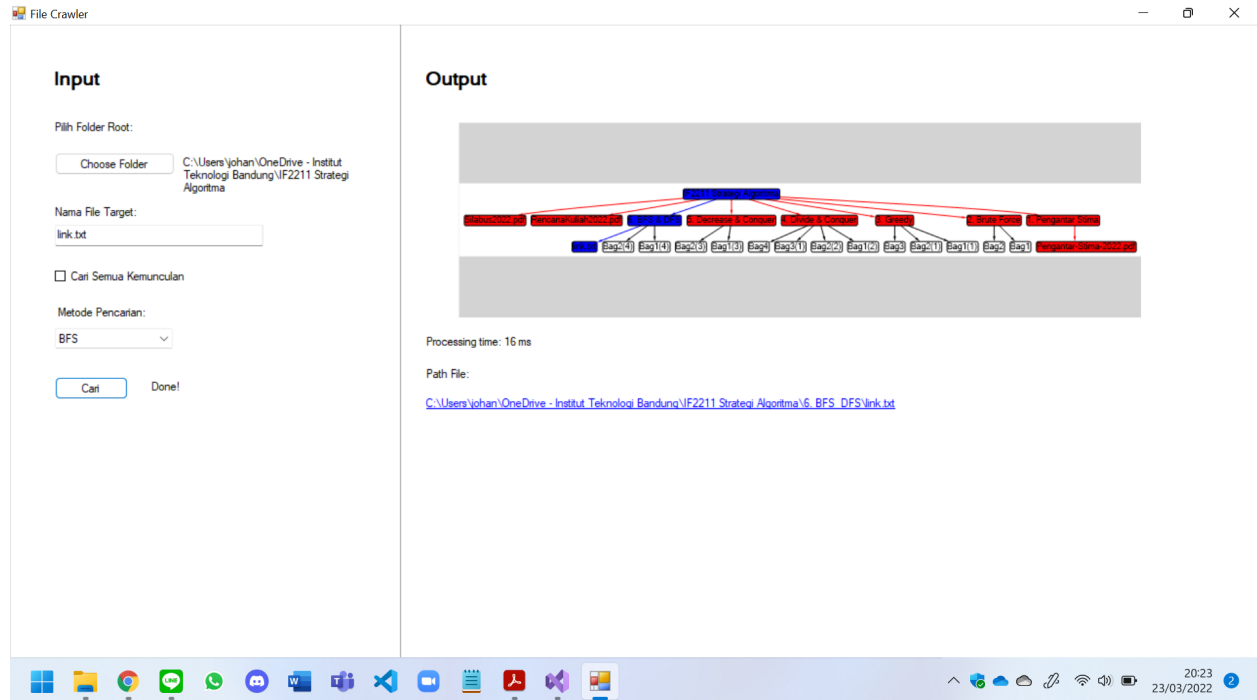
**Gambar 4.1** Antarmuka program file crawler

Gambar di atas merupakan interface dari program file crawler kami. Segala hal yang berhubungan dengan input berada di sisi kiri interface program, sedangkan segala hal yang berhubungan dengan output berada di sisi kanan interface program. Fitur-fitur yang terdapat pada bagian input program ini yaitu tombol pencarian folder yang akan ditelusuri, textbox nama file yang ingin dicari, checkbox mencari semua kemunculan file, pilihan metode pencarian yang akan digunakan, dan tombol “Cari”. Untuk bagian output, akan ditampilkan visualisasi pohon pencarian file, durasi waktu pencariannya, dan hyperlink untuk menuju file tersebut jika file tersebut ditemukan. Berikut merupakan tata cara penggunaan program file crawler ini.

1. Pilih nama folder yang akan ditelusuri
2. Masukkan nama file yang ingin dicari
3. Centang checkbox jika ingin mencari semua kemunculan file
4. Pilih metode pencarian yang akan digunakan
5. Tekan tombol “Cari”

### 4.4 Hasil pengujian beserta analisisnya

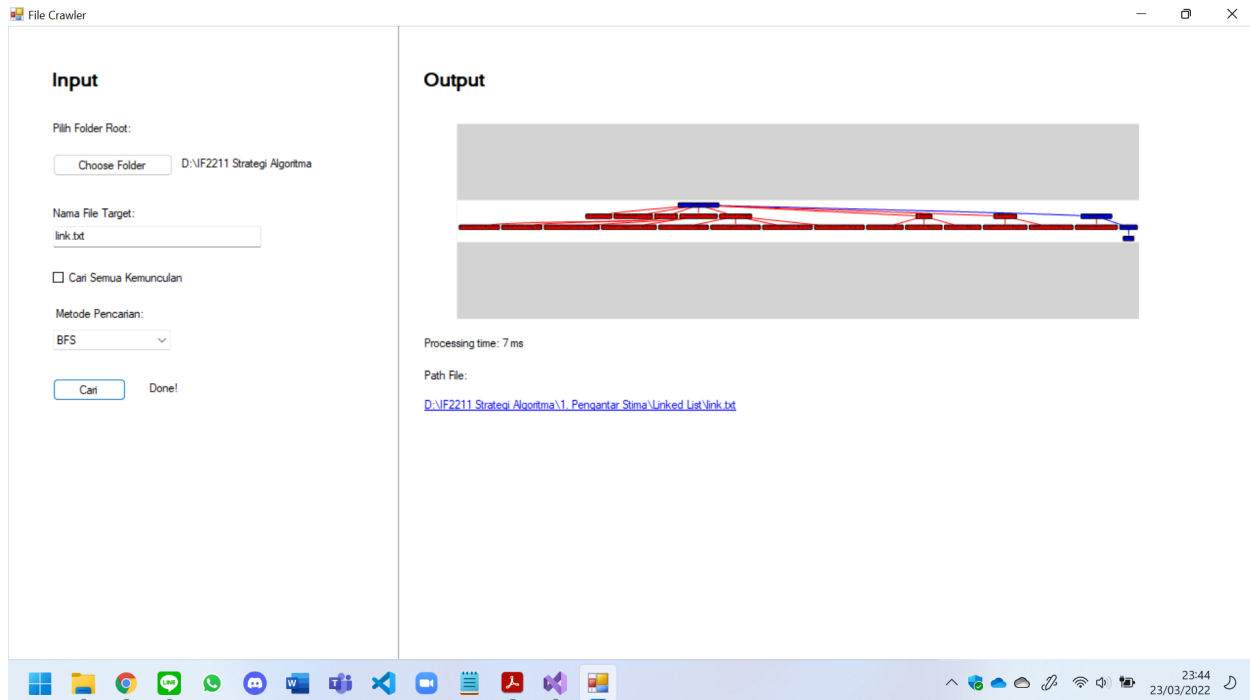
1. Test Case 1



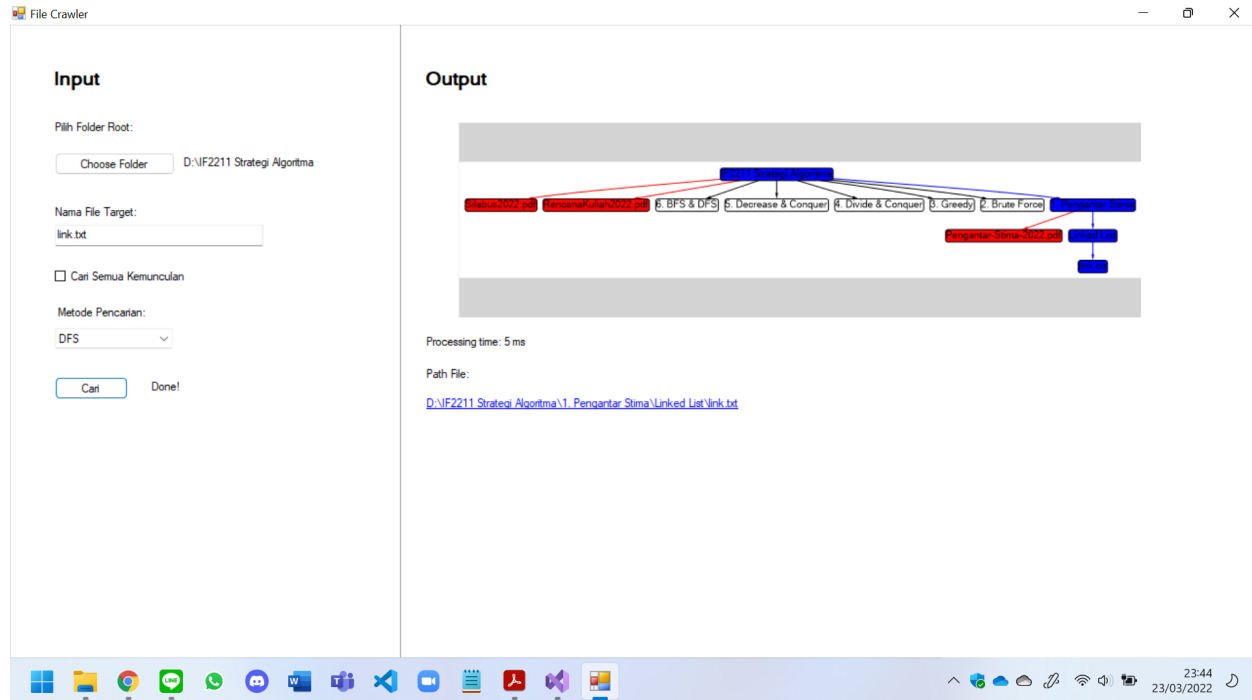
Pada test case 1 ini, dapat dilihat bahwa metode pencarian BFS lebih baik daripada DFS. Hal ini disebabkan karena file yang ingin dicari berada pada urutan folder keenam dengan tingkat 2. Di dalam folder urutan kedua sampai kelima terdapat beberapa folder lagi sehingga memiliki tingkat 3. Alhasil, dengan metode pencarian DFS yang mengecek semua tingkat dari

urutan pertama akan menelusuri terlebih dahulu folder-folder dari urutan pertama hingga menelusuri ke setiap tingkat 3. Metode pencarian tersebut kurang efisien terhadap test case ini karena file yang ingin dicari berada pada tingkat 2. Oleh karena itu, metode pencarian BFS lebih cocok untuk digunakan pada test case ini mengingat BFS menelusuri terlebih dahulu ke setiap tingkatan yang ada. Kesimpulan yang dapat diambil dari kasus ini adalah jika kita ingin mencari file dengan tingkat yang rendah, maka metode pencarian yang paling cocok untuk digunakan adalah dengan menggunakan BFS.

## 2. Test Case 2



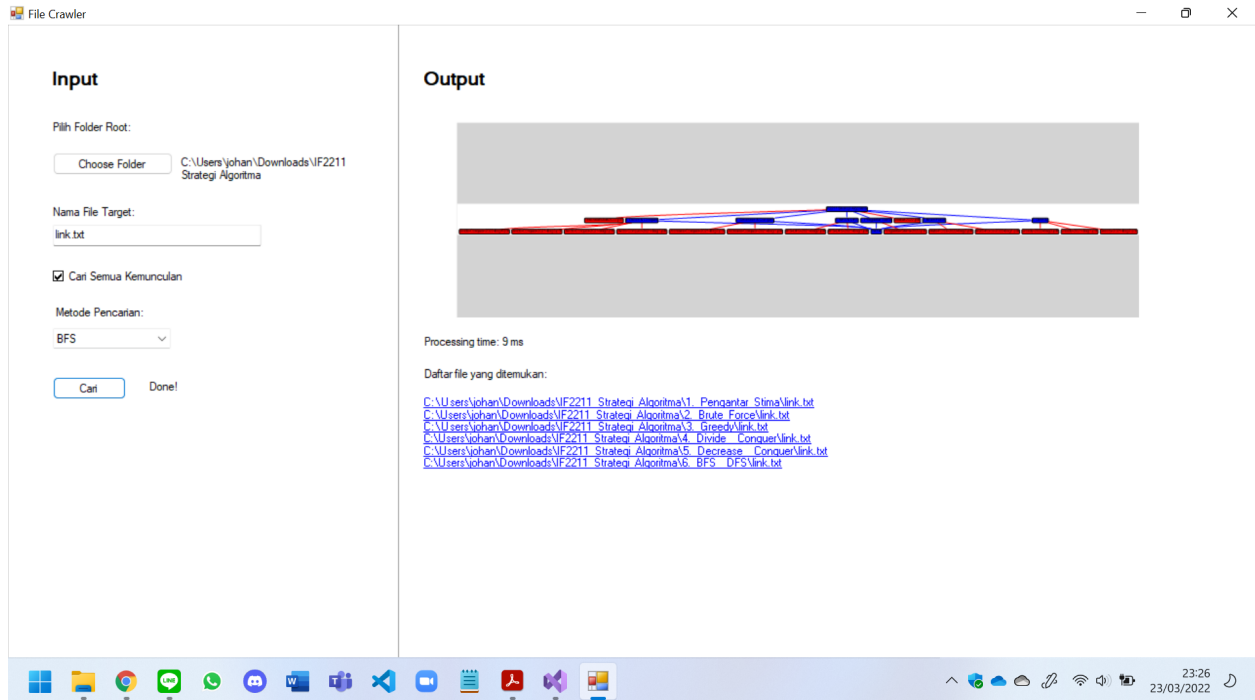
**Gambar 4.4** Hasil pengujian test case 2 dengan BFS



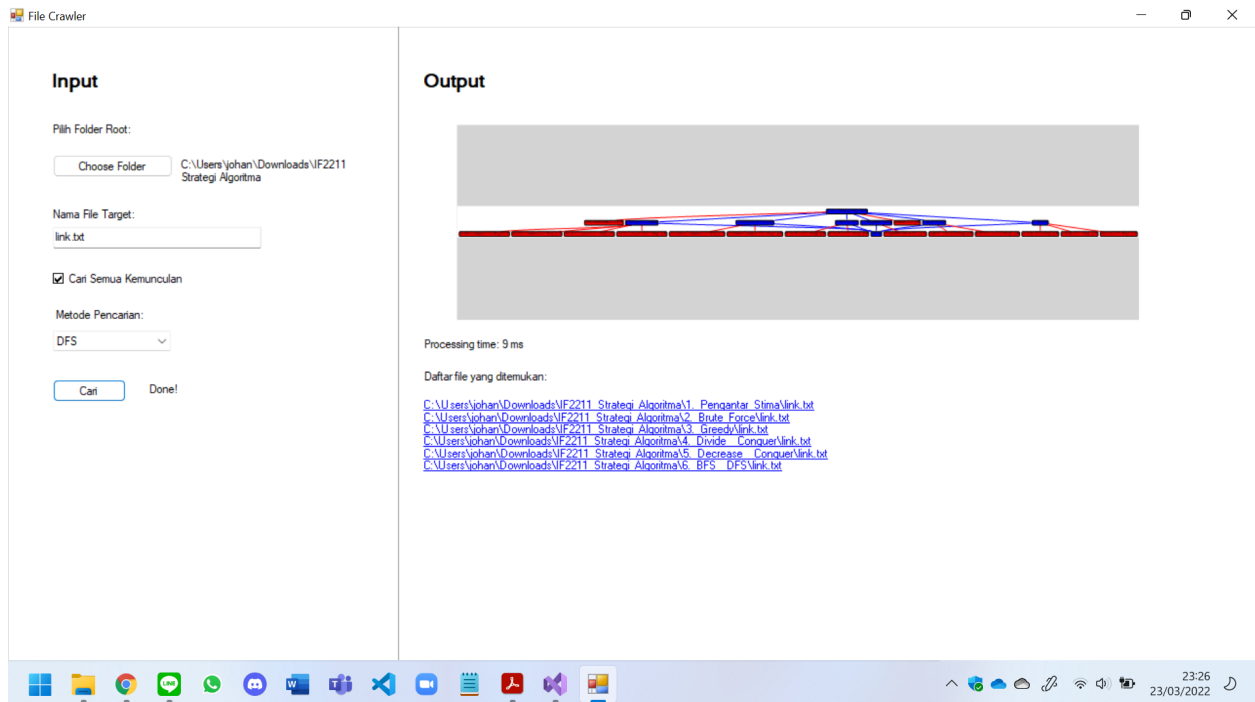
**Gambar 4.5** Hasil pengujian test case 2 dengan DFS

Pada test case 2 ini, dapat dilihat bahwa metode pencarian DFS lebih baik daripada BFS. Hal ini disebabkan karena file yang ingin dicari berada pada urutan folder pertama dengan tingkat 3. Alhasil, dengan metode pencarian BFS akan mengecek semua yang berada pada tingkat 2 terlebih dahulu. Metode pencarian tersebut kurang efisien terhadap test case ini karena file yang ingin dicari berada pada tingkat 3 dan berada pada folder yang memiliki nama yang diawali dengan abjad awal. Oleh karena itu, metode pencarian DFS lebih cocok untuk digunakan pada test case ini mengingat DFS menelusuri terlebih dahulu setiap folder yang memiliki nama yang diawali dengan abjad awal sampai ke tingkat dasar. Kesimpulan yang dapat diambil dari kasus ini adalah jika kita ingin mencari file dengan tingkat yang tinggi serta nama yang diawali dengan abjad awal, maka metode pencarian yang paling cocok untuk digunakan adalah dengan menggunakan DFS.

### 3. Test Case 3



**Gambar 4.6** Hasil pengujian test case 3 dengan BFS yang mencari semua kemunculan file

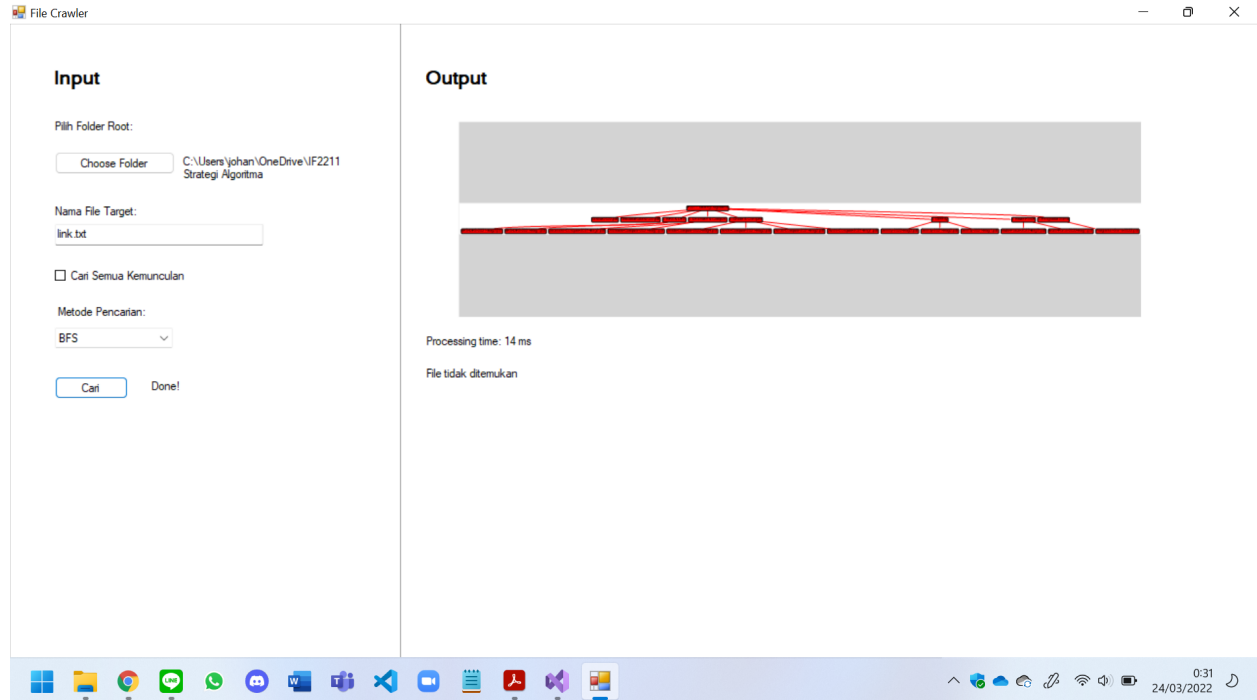


**Gambar 4.7** Hasil pengujian test case 3 dengan DFS yang mencari semua kemunculan file

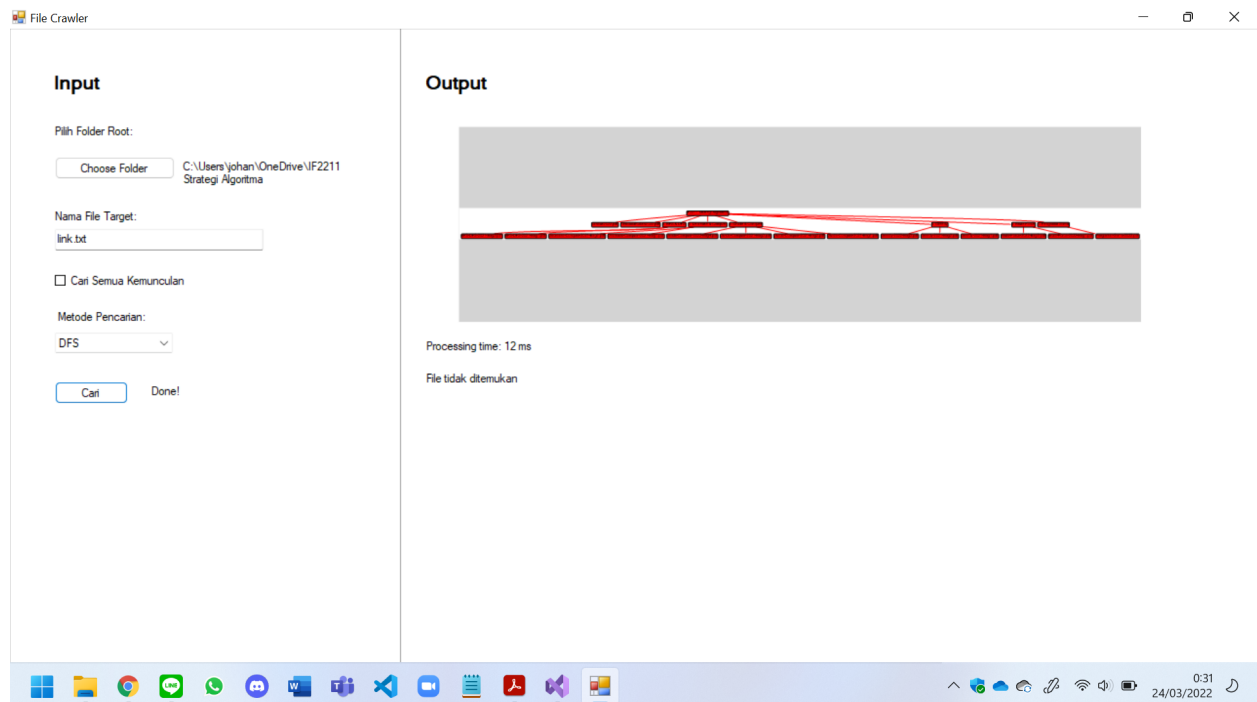
Untuk pencarian semua kemunculan file, metode pencarian BFS maupun DFS memberikan visualisasi pohon pencarian file yang sama. Hal ini dikarenakan kedua metode pencarian tersebut menelusuri semua rute yang ada dalam folder tersebut yang bertujuan agar dapat menemukan semua file yang ingin dicari. Namun pada saat proses pencariannya, kedua

metode pencarian tersebut memiliki cara kerjanya masing-masing dan berbeda satu sama lain. BFS dengan cara menelusuri setiap tingkatannya terlebih dahulu, sedangkan DFS dengan cara menelusuri sesuai urutan abjad sampai ke dasar tingkatannya.

#### 4. Test Case 4



**Gambar 4.8** Hasil pengujian test case 4 dengan BFS



**Gambar 4.9** Hasil pengujian test case 4 dengan DFS



Pada test case 4 ini, input file yang diberikan tidak ditemukan dalam folder yang ditelusuri, sehingga visualisasi pohon pencarian file-nya berwarna merah semua. Baik BFS maupun DFS memberikan visualisasi pohon pencarian file yang sama karena sama-sama mencari ke seluruh penjuru direktori. Namun pada saat proses pencariannya, kedua metode pencarian tersebut memiliki cara kerjanya masing-masing dan berbeda satu sama lain. BFS dengan cara menelusuri setiap tingkatannya terlebih dahulu, sedangkan DFS dengan cara menelusuri sesuai urutan abjad sampai ke dasar tingkatannya.

## BAB V

### KESIMPULAN DAN SARAN

#### 5.1 Kesimpulan

Berdasarkan hasil pengujian di atas, dapat disimpulkan bahwa struktur pohon (tree) dapat menggambarkan susunan file dan folder dalam suatu *directory*. Dengan menggambarkan suatu *directory* sebagai sebuah pohon, kita akan mudah dalam melakukan pencarian file dengan menggunakan pencarian BFS dan DFS.

Pencarian dengan menggunakan DFS pada umumnya akan terlihat lebih singkat jika file target berada pada subfolder pertama pada *directory* karena DFS akan mencari terlebih dahulu pada suatu kedalaman, sedangkan dengan BFS akan terlihat lebih panjang karena harus mencari terlebih dahulu di suatu tingkatan sebelum masuk ke kedalaman berikutnya. Dalam pencarian pada kasus hanya satu file, pencarian akan diberhentikan pada saat file target telah ditemukan, sedangkan dalam kasus mencari semua file, harus mentraversal semua folder, sehingga pada kasus ini BFS dan DFS akan memberikan hasil yang sama dengan metode pencarian yang berbeda. Dalam hal ini, program kami telah dapat menerapkan metode BFS dan DFS dalam *Folder Crawling*.

#### 5.2 Saran

Saran-saran yang dapat kami berikan untuk tugas besar IF2211 Strategi Algoritma semester 2 tahun 2021/2022 adalah sebagai berikut.

1. Penulisan *pseudocode* tampaknya kurang perlu pada laporan karena kelengkapan program dapat dilihat dari *source code* secara langsung.
2. Spesifikasi tugas yang diberikan seharusnya sudah dijelaskan secara detail agar tidak menimbulkan banyak pertanyaan dalam sheet QnA.
3. Landasan teori mengenai C# desktop application development dirasa tidak perlu dijadikan sebagai landasan teori karena hal itu dianggap sudah kami pahami. Landasan teori seharusnya berisi teori dasar yang digunakan dalam pembuatan program folder crawling ini.

## DAFTAR PUSTAKA

1. [https://www.digitalcitizen.life/wp-content/uploads/2020/10/explorer\\_search\\_10.png](https://www.digitalcitizen.life/wp-content/uploads/2020/10/explorer_search_10.png)
2. <https://github.com/microsoft/automatic-graph-layout>
3. <https://docs.google.com/document/d/1XhFSpHU028Gaf7YxkmdbluLkQgVI3MY6gt1t-PL30LA/edit?usp=sharing>
4. <http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>
5. <https://id.gadget-info.com/difference-between-bfs>
6. <https://www.teorikomputer.com/2016/04/pengertian-dan-fungsi-microsoft-net.html>
7. <https://idcsharp.com/2019/04/28/mengenal-tampilan-winform-pada-visual-studio-2019/>

Link repository → <https://github.com/johannes-ws/Tubes-2-Stima>

Link video → <https://www.youtube.com/watch?v=9rw5M5faM64>