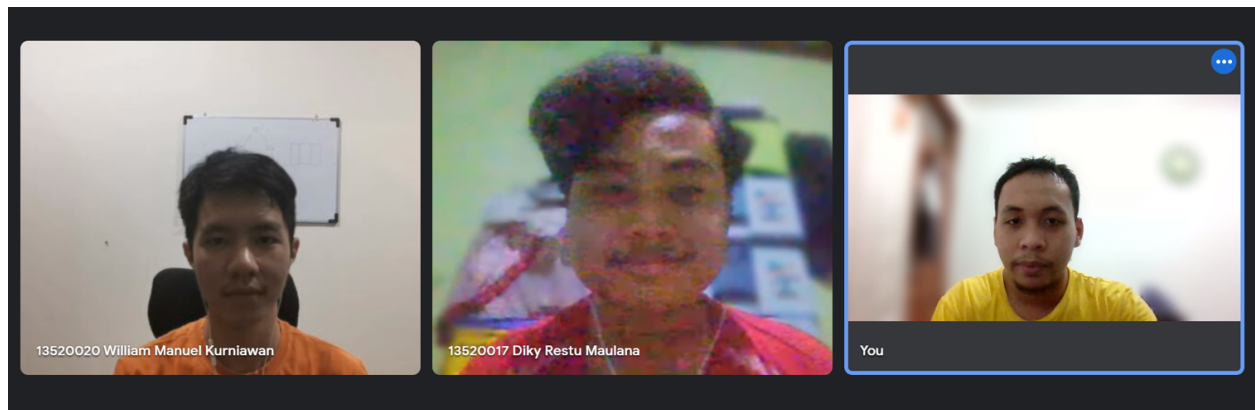


**Laporan Tugas Besar 2**  
**IF2211 Strategi Algoritma**

**Pengaplikasian Algoritma BFS dan DFS dalam  
Implementasi Folder Crawling**



Disusun Oleh:

Kelompok 9 - CariFile.com

Diky Restu Maulana 13520017

William Manuel Kurniawan 13520020

Fawwaz Anugrah Wiradhika Dharmasatya 13520086

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

**2022**

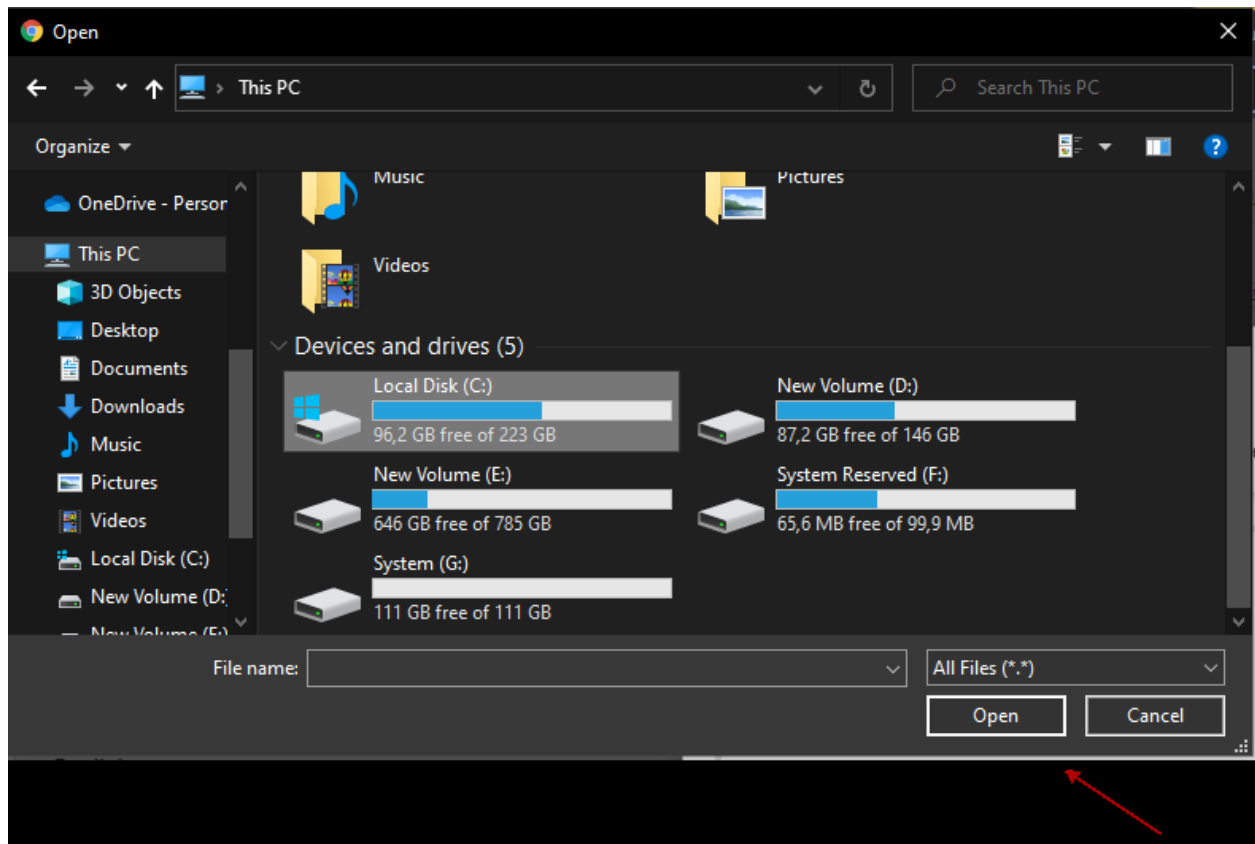
## DAFTAR ISI

|  |           |
|--|-----------|
| <b>BAB I: Deskripsi Tugas</b>                                  | <b>3</b>  |
| <b>BAB II: Landasan Teori</b>                                  | <b>10</b> |
| Traversal Graf   | 10        |
| Algoritma Pencarian Solusi                                     | 10        |
| Representasi Graf dalam Proses Pencarian                       | 10        |
| <i>Breadth First Search</i>                                    | 10        |
| <i>Depth First Search</i>                                      | 12        |
| <i>C# desktop application development</i>                      | 13        |
| <b>BAB III: Analisis Pemecahan Masalah</b>                     | <b>14</b> |
| Langkah-Langkah Pemecahan Masalah                              | 14        |
| BFS  | 14        |
| DFS  | 14        |
| Mapping Persoalan Menjadi Elemen -Elemen Algoritma BFS dan DFS | 15        |
| Ilustrasi Kasus Lain   | 15        |
| <b>BAB IV: Implementasi dan Pengujian</b>                      | <b>17</b> |
| Implementasi Program   | 17        |
| Struktur Data  | 29        |
| Result   | 29        |
| Spesifikasi Program  | 29        |
| Tata Cara Penggunaan Program                                   | 30        |
| <i>Interface</i>   | 30        |
| Fitur  | 31        |
| Cara Penggunaan  | 31        |
| Hasil Pengujian  | 31        |
| Analisis Desain Solusi   | 45        |
| <b>BAB V: Kesimpulan, Saran, dan Link <i>Source Code</i></b>   | <b>46</b> |
| Kesimpulan   | 46        |
| Saran  | 46        |
| Tautan <i>Source Code</i>                                      | 46        |
| <b>DAFTAR PUSTAKA</b>  | <b>47</b> |

## BAB I: 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.



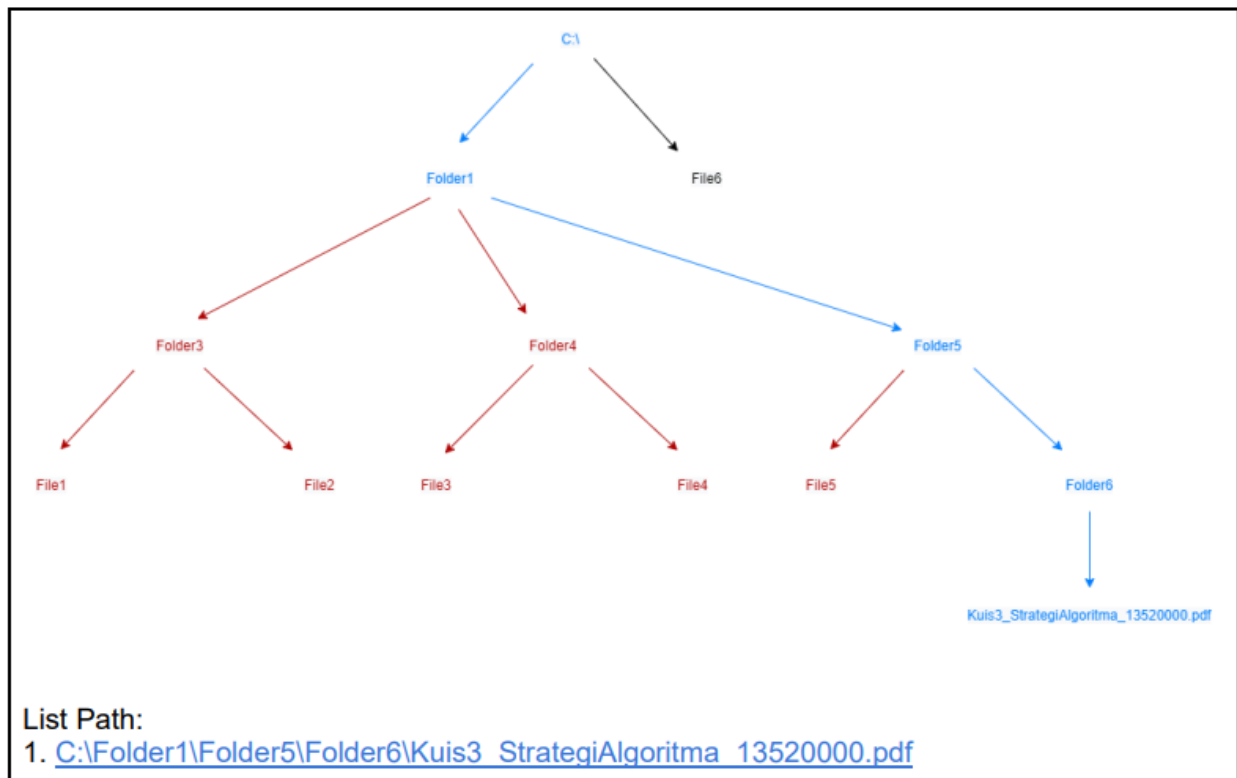
Input Starting Directory

Kuis3\_StrategiAlgoritma\_13520000.pdf

Search

Input Nama File

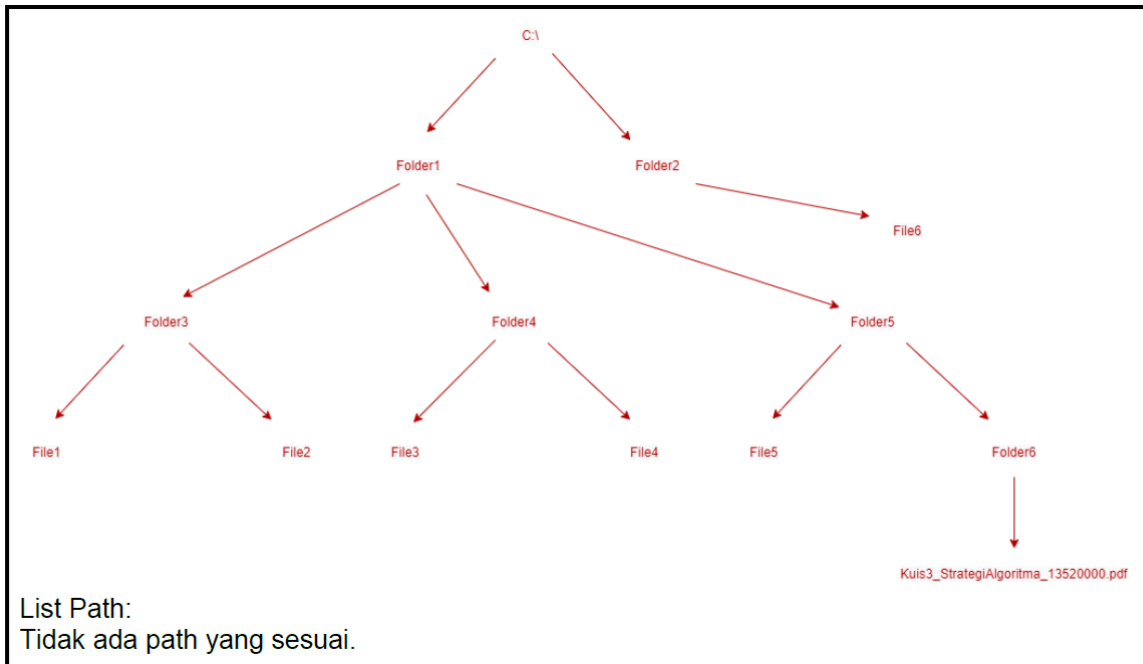
Contoh output aplikasi:



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.

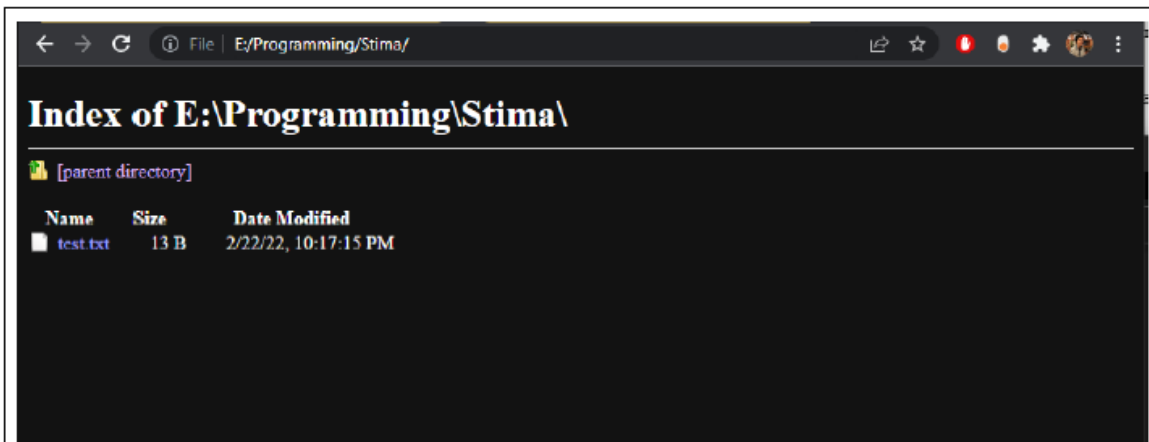


*Gambar 4. Contoh output program jika file tidak ditemukan*

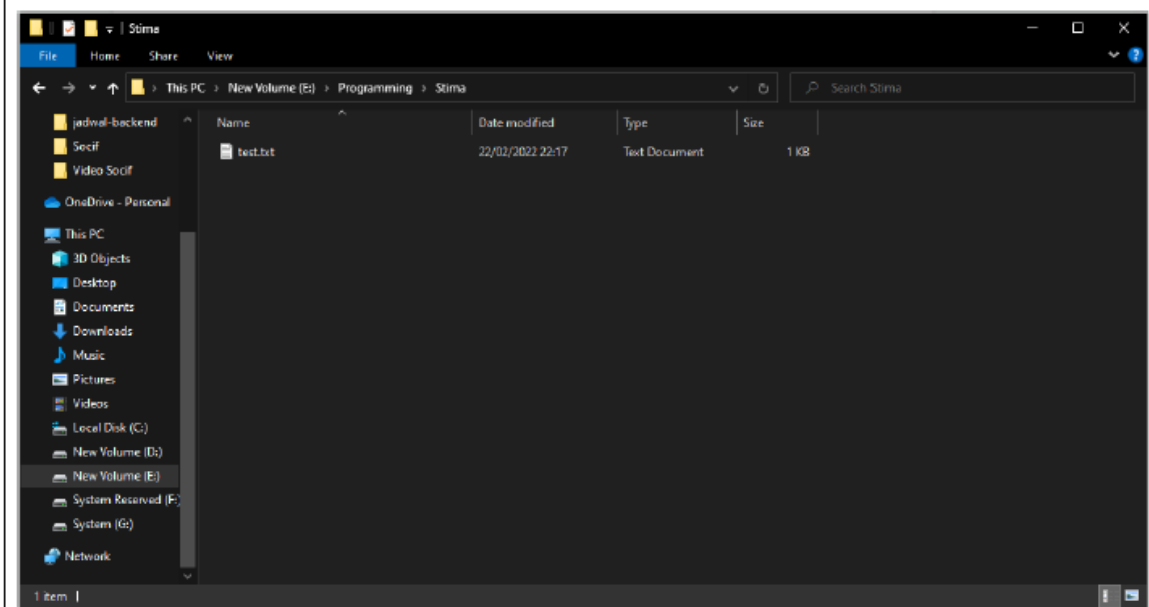
Jika file yang ingin dicari pengguna tidak ada pada direktori file, misalnya saat pengguna mencari Kuis3Probststat.pdf, maka path pencarian DFS adalah sebagai berikut: C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → 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 4. Contoh ketika hyperlink di-klik*

### 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

☐

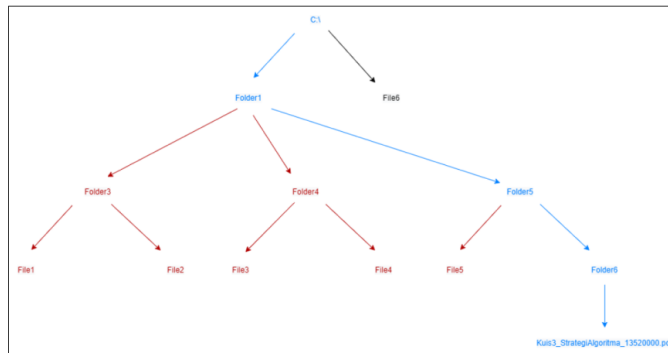
BFS

☒

DFS

Search

### Output



Path File :

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

Time spent: 20.02s

Gambar 9. 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 kaskas yang tersedia. Sebagai referensi, salah satu kaskas 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/1XhFSpHU028Gaf7YxkmdbluLkQgVl3MY6gt1t-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

### ***Traversal Graf***

Traversal graf (juga dikenal sebagai *search graph*) mengacu pada proses mengunjungi (memeriksa dan/atau memperbarui) setiap simpul dalam graf. Lintasan tersebut diklasifikasikan menurut urutan simpul yang dikunjungi. Traversal pohon adalah kasus khusus dari traversal graf. Ada dua algoritma traversal graf, yaitu BFS dan DFS dengan asumsi graf terhubung. Graf sebagai representasi persoalan dan traversal graf sebagai pencarian solusi dari persoalan tersebut.

### ***Algoritma Pencarian Solusi***

Algoritma pencarian solusi atas sebuah persoalan yang dapat dimodelkan dalam bentuk graf dapat dibagi menjadi dua, yaitu

1. Tanpa Informasi (*uninformed/blind search*)  
Pencarian dilakukan tanpa tambahan informasi sejak awal. Contoh algoritma yang termasuk *uninformed search* adalah *Breadth First Search* (BFS), *Depth First Search* (DFS), *Iterative Deepening Search* (IDS), dan *Uniform Cost Search*.
2. Dengan informasi (*informed search*)  
Pencarian berbasis heuristik dan sudah diketahui *non-goal state* sejak awal. Contoh algoritma yang termasuk *informed search* adalah *Best First Search* dan A\*.

### ***Representasi Graf dalam Proses Pencarian***

Dalam pencarian solusi, terdapat dua pendekatan, yaitu

1. Graf Statis  
Graf yang sudah terbentuk sebelum proses pencarian dilakukan. Dalam pendekatan ini, graf direpresentasikan sebagai struktur data.
2. Graf Dinamis  
Graf yang terbentuk saat proses pencarian dilakukan. Dalam pendekatan ini, graf tidak tersedia sebelum pencarian dan dibangun selama proses pencarian solusi

### ***BFS***

BFS (Breadth First Search) merupakan sebuah metode pencarian traversal yang dilakukan secara melebar. Pencarian dilakukan dengan mengunjungi semua simpul anak yang dimiliki simpul terlebih dahulu sebelum mulai menelusuri simpul anak yang ditelusuri. Dalam proses pencarian, akan dibuat sebuah antrian untuk menyimpan urutan simpul yang akan dicari dan menggunakan rekursi.

Algoritma BFS dimulai dari simpul yang dipilih, misalnya x.

- Membuat antrian awal berisi x
- Kunjungi simpul x
- Kunjungi semua simpul anak dari simpul x dan masukkan simpul ke dalam antrian di belakang simpul terakhir dalam antrian
- Hapus simpul x dari antrian dan ulangi pencarian dengan simpul pertama pada antrian
- Pencarian berakhir bila tidak ada simpul di dalam antrian dan semua simpul sudah diperiksa

### Pseudocode

```
Procedure BFS(input v : integer)
{ Traversal graf dengan algoritma pencarian BFS.
  Masukan : v adalah simpul awal kunjungan
  Keluaran : semua simpul yang dikunjungi dicetak di layar
}

Deklarasi :
  w : integer
  q : 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 q pada posisi belakang }

  procedure HapusAntrian(input/output q : antrian, output v : integer)
  { menghapus v dari kepala antrian q }

  function AntrianKosong(input q : antrian) -> boolean
  { true jika antrian q kosong, false jika sebaliknya }

Algoritma :
  BuatAntrian(q)

  write(v)
  dikunjungi[v] ← true

  MasukAntrian(q,v)

  while not AntrianKosong(q) do
    HapusAntrian(q,v)

    for tiap simpul w yang bertetangga dengan simpul v do
      if not dikunjungi [w] then
        write(w)
        MasukAntrian(q,w)
```

```

        dikunjungi[w] ← true
    endif
endfor
endwhile

```

## **DFS**

DFS (Depth First Search) merupakan sebuah algoritma untuk menelusuri graf berhingga. DFS mengunjungi simpul anak terlebih dahulu sebelum mengunjungi simpul saudara yang memiliki kedalaman yang sama dengan suatu simpul. Pada dasarnya, DFS akan memprioritaskan kedalaman daripada memperlebar jelajah. Rekursi seringkali digunakan saat mengimplementasikan algoritma ini.

Algoritma DFS dimulai dari simpul yang dipilih, misalnya  $v$ .

- Kunjungi simpul  $v$
- Kunjungi simpul  $w$  yang bertetangga dengan  $v$
- Ulangi DFS dari simpul  $w$
- 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
- Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi

## Pseudocode

```

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 layar
)

Deklarasi:
     $w$  : integer
Algoritma:
    write( $v$ )
    dikunjungi( $v$ ) ← true
    for  $w=1$  to  $n$  do
        if  $A[v,w]=1$  then {simpul  $v$  dan  $w$  bertetangga}
            if not dikunjungi( $w$ ) then
                DFS( $w$ )
            endif
        endif
    endfor

```

### ***C# Desktop Application Development***

C# merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti Java, Delphi, Visual Basic, dan lain-lain) dengan beberapa penyederhanaan. C# banyak digunakan dalam pengembangan aplikasi berbasis web, aplikasi desktop, aplikasi mobile, dan games.

## **BAB III: Analisis Pemecahan Masalah**

### **Langkah-Langkah Pemecahan Masalah**

#### ***BFS***

1. Terima masukan folder awal pencarian, nama file yang ingin dicari, dan pilihan (mencari kemunculan pertama atau seluruh kemunculan)
2. Buat sebuah array sebagai antrian, masukkan folder awal ke dalam antrian tersebut
3. Ambil semua folder dan file yang berada di bawah folder awal tersebut
4. Masukkan semua folder ke dalam antrian untuk diperiksa lebih lanjut (file tidak perlu diperiksa karena tidak akan mengandung file / folder lain)
5. Hapus folder pertama dalam antrian
6. Cek nama setiap file / folder yang dicari. Jika nama sesuai, maka ada 2 kemungkinan
  - Jika ingin menemukan kemunculan pertama, lompat ke langkah 7
  - Jika ingin menemukan seluruh kemunculan, ulangi langkah 3-5 sampai antrian habis dan semua file sudah diperiksa
7. Pencarian dihentikan, cetak traversal pohon yang dihasilkan, dan tuliskan semua jalur yang mungkin untuk mendapatkan file yang ingin dicari

#### ***DFS***

1. Terima masukan folder awal pencarian, nama file yang ingin dicari, dan pilihan (mencari kemunculan pertama atau seluruh kemunculan)
2. Ambil semua folder dan file yang berada langsung di bawah folder tersebut
3. Urutkan folder dan file berdasarkan abjad sesuai dengan prinsip DFS yang mendahulukan nama simpul yang memiliki abjad lebih awal
4. Jadikan folder pertama dalam urutan yang belum dikunjungi sebagai masukan berikutnya
5. Ulangi langkah 2-4 hingga ditemukan folder kosong atau file
6. Cek semua nama file yang ditemukan. Apabila namanya sama dengan file yang ingin dicari, selanjutnya ada dua kemungkinan
  - Jika ingin menemukan kemunculan pertama, lompat ke langkah 10
  - Jika ingin menemukan seluruh kemunculan, lanjutkan ke langkah 7
7. Runut balik ke folder terakhir yang isinya belum dikunjungi
8. Ambil folder dalam urutan yang belum dikunjungi dan jadikan masukan berikutnya
9. Ulangi langkah 2-6 hingga seluruh folder atau file yang berada di dalam folder awal pencarian telah dikunjungi
10. Pencarian dihentikan, cetak traversal pohon yang dihasilkan, dan tuliskan semua jalur yang mungkin untuk mendapatkan file yang ingin dicari

## Mapping Persoalan Menjadi Elemen -Elemen Algoritma BFS dan DFS

Hierarki folder dan file dimodelkan dalam bentuk pohon.

- Simpul menyatakan folder/file
  - Simpul akar menyatakan direktori pertama pencarian dimulai
  - Simpul parent merupakan sebuah folder
  - Simpul child merupakan folder/file di dalam simpul parentnya
  - Simpul daun merupakan file atau folder kosong
- Sisi yang mengarah ke bawah menyatakan folder/file yang ada di dalamnya



### Ilustrasi Kasus Lain

Contoh : mencari file test3.txt dengan folder awal F:\pertama



|   |                  |             |
|---|------------------|-------------|
|  delapan | 25/03/2022 14:04 | File folder |
|  kedua   | 13/03/2022 14:47 | File folder |
|  kedua_2 | 13/03/2022 14:48 | File folder |

Gambar 3.1. Isi Folder F:\pertama

Isi folder F:\pertama\delapan kosong

|   |                  |               |
|---|------------------|---------------|
|  empat | 13/03/2022 20:21 | File folder   |
|  tiga  | 13/03/2022 20:21 | File folder   |
|  test  | 13/03/2022 14:47 | Text Document |

Gambar 3.2. Isi Folder F:\pertama\kedua

|   |                  |               |
|---|------------------|---------------|
|  lima  | 18/03/2022 13:07 | File folder   |
|  test3 | 14/03/2022 20:59 | Text Document |

Gambar 3.3. Isi Folder F:\pertama\kedua\_2

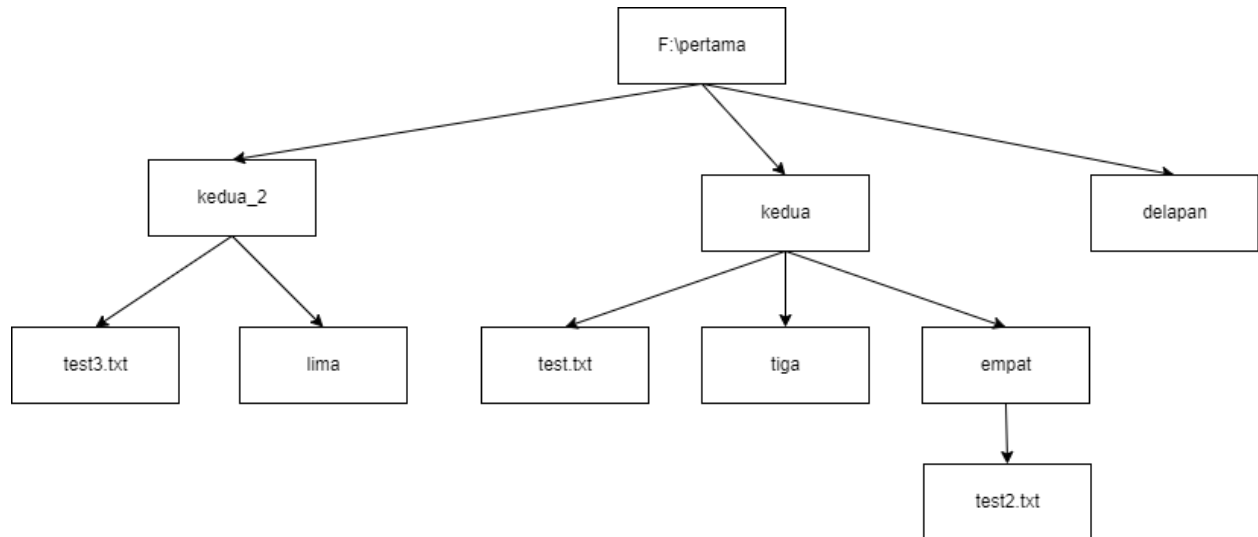
|   |                  |               |
|---|------------------|---------------|
|  test2 | 13/03/2022 20:21 | Text Document |
|---|------------------|---------------|

Gambar 3.4. Isi Folder F:\pertama\kedua\empat

Isi folder F:\pertama\kedua\tiga kosong

Isi folder F:\pertama\kedua\_2\lima kosong

Dari informasi tersebut, dibuat pohon yang menggambarkan isi folder awal sebagai berikut



Gambar 3.5. Pohon yang dibuat dari penelusuran folder awal

Penelusuran menggunakan BFS akan menghasilkan urutan :

delapan → kedua → kedua\_2 → empat → tiga → test.txt → lima → test3.txt

Penelusuran menggunakan DFS akan menghasilkan urutan :

delapan → kedua → empat → test2.txt → tiga → test.txt → kedua-2 → lima → test3.txt



# BAB IV: Implementasi dan Pengujian

## Implementasi Program

Garis Besar Program Utama:

Procedure Main ()

( Mencari sebuah file dimulai dari starting directory lalu menampilkan pohon pencarian, tautan ke folder parent file yang dicari, serta waktu pencarian  
Constraint: tombol untuk memilih algoritma BFS dan DFS mutual exclusive satu sama lain. Jika sudah menekan BFS(isUsingBFS=true), maka yang DFS secara otomatis false. Begitupun sebaliknya  
)

**Deklarasi:**

```
startingDirectory : String
filename: String
isSearchAllOccurence: boolean
isUsingBFS: boolean
isUsingDFS: boolean
stopwatch: Stopwatch { kelas untuk membantu mencatat waktu }
res: Result { struktur data untuk menyimpan hasil pencarian }
graph: GViewer { graf hasil pencarian}
listPath: array of String { daftar path file-file yang ditemukan }
timeElapsed: integer { waktu pencarian dalam ms }
```

**Algoritma:**

```
startingDirectory <- OpenFileDialog().filename
filename <- fileNameTextBox.Text
isSearchAllOccurence <- findAllOccurenceButton.Checked
isUsingBFS <- BFSbutton.Checked
isUsingDFS <- DFSbutton.Checked
resetPanelControl() { menghapus bekas pencarian sebelumnya}

if(isUsingBFS)then {Mencari menggunakan algoritma BFS }
    stopwatch.Start(){ Memulai perhitungan waktu}
    res <- BFS(startingDirectory,filename,isSearchAllOccurence)
    graph <- res.graph
    stopwatch.Stop(){ Mengakhiri perhitungan waktu}
    timeElapsed <- stopwatch.ElapsedMilliseconds
    write(timeElapsed, " ms")
    displayGraph(graph) { menampilkan pohon pencarian ke layar }
    listPath <- res.listOfPath
    displayListPath(listPath) { Menampilkan path ke file yang ditemukan
    beserta hyperlink untuk mengakses folder parentnya }
    else { isUsingBFS bernilai false}
        if(isUsingDFS)then { Mencari menggunakan algoritma DFS }
            stopwatch.Start(){ Memulai perhitungan waktu}
            res <- DFS(startingDirectory,filename,isSearchAllOccurence)
            graph <- res.graph
```

```

        stopwatch.Stop(){ Mengakhiri perhitungan waktu}
        timeElapsed <- stopwatch.ElapsedMilliseconds
        write(timeElapsed, " ms")
        displayGraph(graph) { menampilkan pohon pencarian ke layar }
        listPath <- res.listOfPath
        displayListPath(listPath) { Menampilkan path ke file yang ditemukan
        else { tombol memilih algoritma tidak ada yang ditekan }
        { tidak melakukan apa-apa }

```

## Algoritma BFS

{ Catatan: di implementasi source code kami, algoritma BFS untuk mencari 1 file dan mencari semua file merupakan fungsi yang berbeda karena masalah implementasi. Namun di pseudocode ini, akan dijadikan satu fungsi saja dengan menerima parameter tambahan berupa nilai boolean apakah pencarian dilakukan untuk kemunculan pertama saja atau untuk semua file/folder}

**Function** BFS(startingDirectory: String, fileName: String, isSearchAllOccurence:boolean)->Result

### **Deklarasi:**

```

inputDiskArr : array of String
hasilBFS : array of String
matrixBFS : array of array of String
pathReference : array of array of String

```

### **Algoritma:**

```

inputDiskArr <- { startingDirectory }
hasilBFS <- cariAll(inputDiskArr)
pathReference <- getPathId(hasilBFS, startingDirectory)
if (isSearchAllOccurence) then
    matrixBFS <- getMatrixNode(hasilBFS, filename, startingDirectory)
    res.graph <- displayGraph(pathReference, matrixBFS)

{ displayGraph tidak digunakan pada BFS tetapi isi bagian mendapatkan BFS
sama dengan fungsi displayGraph yang digunakan pada DFS }

    res.listOfPath <- getDirectory(listPath, fileName)

    else
        matrixBFS <- getMatrixNodeOneFound(hasilBFS, filename,
startingDirectory)
        res.graph <- displayGraph(pathReference, matrixBFS)

{ displayGraph tidak digunakan pada BFS tetapi isi bagian mendapatkan BFS
sama dengan fungsi displayGraph yang digunakan pada DFS }

```

```

    res.listOfPath <- getDirectoryOneFound(listPath, fileName)

->res

Function cariAll(antrian : array of String) -> array of String
{ mencari semua path file/folder dari folder awal menggunakan metode
pencarian BFS. Folder awal ditaruh dalam variabel antrian }

Deklarasi:
searchFolderTemp: IList<string>
searchFileTemp: IList<string>
searchFolder: array of String
searchFile: array of String
searchAll : array of String

Algoritma:
    Directory.GetDirectories(firstDir + "\\").ToList().ForEach(s =>
searchFolderTemp.Add(s))
        catch(UnauthorizedAccessException)
        { }
        { Mengambil file di bawah firstDir }
    Try
        Directory.GetFiles(firstDir + "\\").ToList().ForEach(s =>
searchFileTemp.Add(s))
        catch(UnauthorizedAccessException)
        { }
        {Urutkan sesuai abjad }
        searchFolder <- searchFolderTemp.ToArray()
        searchFile <- searchFileTemp.ToArray()
        Array.Sort(searchFolder, StringComparer.OrdinalIgnoreCase)
        Array.Sort(searchFile, StringComparer.OrdinalIgnoreCase)
        SearchAll <- searchFolder.Concat(searchFile).ToArray()
        antrian <- antrian.Concat(searchFolder).ToArray()
        { menambahkan elemen pada antrian }
        antrian <- antrian.Where((source, index) => index != 0).ToArray()
        { menghapus elemen pertama dari antrian }
        -> SearchAll.Concat(cariAll(antrian)).ToArray()
        { melakukan rekursi }

function getMatrixNode(listPath : array of String, fileName : string,
firstDir : string)

Deklarasi:
matrix: array of array of String
listPreviousPath: List<string>
pathReference:array of array of String

```

```

row: integer
arrLeaf: array of String
leaf: String
previousPath: String
previousId: String
leafId: String
hasilGabung: array of String
arrFolder : array of String
arrPath : array of String
Algoritma:
  pathReference <- getPathId(listPath, firstDir)
  row <- 0
  foreach (path in listPath)
    arrLeaf <- path.Split('\\')
    leaf <- arrLeaf[arrLeaf.Length - 1]
    previousPath <- path.Substring(0, path.Length - leaf.Length - 1)
    previousId <- ""
    foreach (pathId in pathReference)
    { menentukan id untuk previousPath }
      if(pathId[0] = previousPath)then
        previousId <- pathId[1]

    leafId <- ""
    foreach (pathId in pathReference) {menentukan id untuk path}
      if (pathId[0] = path)then
        leafId <- pathId[1]

    hasilGabung[0] <- previousId
    hasilGabung[1] <- leafId
    hasilGabung[2] <- previousPath
    hasilGabung[3] <- leaf
    hasilGabung[4] <- "0" { memberi warna hitam pada simpul }
    matrix[row] <- hasilGabung

    if (leaf = filename) then
      matrix[row][4] = 1 { mengubah warna menjadi biru }
      listPreviousPath.Add(previousPath)
      { simpul ditambahkan ke list }

    else
      matrix[row][4] = 2 { mengubah warna menjadi merah }

  row <- row + 1

```

```

{ Mengubah path menjadi nama file/folder }
foreach (node in matrix)
  previousPath <- node[2]
  arrFolder <- previousPath.split('\\')
  node[2] <- arrFolder[arrFolder.length-1]

{ Mengiterasi semua folder di atas file yang ditemukan }
foreach (path in listPreviousPath)
  arrPath <- path.split('\\')

  i traversal [0..arrPath.length-2]
  edge <- [ arrPath[i], arrPath[i + 1] ]
  foreach (node in matrix)
    if (node[2] = edge[0] and node[3] = edge[1]) then
      node[4] <- "1"
-> matrix

```

```

function getMatrixNodeOneFound(listPath : array of String, fileName :
string, firstDir : string)

```

**Deklarasi:**

```

matrix: array of array of String
listPreviousPath: List<string>
pathReference:array of array of String
row: integer
arrLeaf: array of String
leaf: String
previousPath: String
previousId: String
leafId: String
hasilGabung: array of String
arrFolder : array of String
arrPath : array of String
isFound : integer

```

**Algoritma:**

```

  pathReference <- getPathId(listPath, firstDir)
  row <- 0
  isFound <- 0
  foreach (path in listPath)
    arrLeaf <- path.Split('\\')
    leaf <- arrLeaf[arrLeaf.Length - 1]
    previousPath <- path.Substring(0, path.Length - leaf.Length - 1)
    previousId <- ""

```

```

foreach (pathId in pathReference)
  { menentukan id untuk previousPath }
  if(pathId[0] = previousPath)then
    previousId <- pathId[1]

leafId <- ""
foreach (pathId in pathReference) {menentukan id untuk path}
  if (pathId[0] = path)then
    leafId <- pathId[1]

hasilGabung[0] <- previousId
hasilGabung[1] <- leafId
hasilGabung[2] <- previousPath
hasilGabung[3] <- leaf
hasilGabung[4] <- "0" { memberi warna hitam pada simpul }
matrix[row] <- hasilGabung

if (leaf = filename and isFound = 0) then
  matrix[row][4] = 1 { mengubah warna menjadi biru }
  listPreviousPath.Add(previousPath)
  isFound <- 1 { menandakan file/folder sudah ditemukan }

if (leaf != filename and isFound = 0) then
  matrix[row][4] = 2 { mengubah warna menjadi merah }

row <- row + 1

{ Mengubah path menjadi nama file/folder }
foreach (node in matrix)
  previousPath <- node[2]
  arrFolder <- previousPath.split('\\')
  node[2] <- arrFolder[arrFolder.length-1]

{ Mengiterasi semua folder di atas file yang ditemukan }
foreach (path in listPreviousPath)
  arrPath <- path.split('\\')

  i traversal [0..arrPath.length-2]
  edge <- [ arrPath[i], arrPath[i + 1] ]
  foreach (node in matrix)
    if (node[2] = edge[0] and node[3] = edge[1]) then
      node[4] <- "1"

```

```

-> matrix

{ fungsi lain yang mendukung kerja algoritma}
Function getPathId(listPath : array of String, firstDir : String) ->
array of array of String
{ menambahkan string id untuk setiap path yang ada dalam listPath }

Function getDirectory(listPath : array of String, fileName : String) ->
array of String
{ menghasilkan list path yang merupakan path ke nama file/folder yang
sama dengan fileName dari listPath}

Function getDirectoryOneFound(listPath : array of String, fileName :
String) -> array of String
{ menghasilkan list path yang merupakan path ke nama file/folder yang
sama dengan fileName dari listPath. Akan dikeluarkan 1 path saja yaitu
path pertama yang ditemukan dalam listpath}

```

## Algoritma DFS

```

Function DFS(firstDir: String, fileName: String,
isSearchAll:boolean)->Result
{ Fungsi untuk melakukan folder crawling dengan algoritma DFS dan
mengembalikan pohon pencarian serta daftar path ke file yang dicari }
Deklarasi:
    listPath: array of String
    pathReference: array of array of String
    res: Result
    matrixNodeAll: array of array of String
    matrixNodeOnce: array of array of String
Algoritma:
    listPath <- getListPath(firstDir)
    pathReference <- getPathId(listPath,firstDir)
    if(isSearchAll)then { cari semua kemunculan }
        matrixNodeAll <- getMatrixNodeAllOccurrence(listPath, firstDir,
fileName)
        res.graph <- displayGraph(pathReference, matrixNodeAll)
        res.listOfPath <- getDirectory(listPath, fileName)
    else { cari kemunculan pertama saja }
        matrixNodeOnce <- getMatrixNodeOnce(listPath, firstDir, fileName)
        res.graph <- displayGraph(pathReference, matrixNodeOnce)
        res.listOfPath <- getDirectoryOneFound(listPath, fileName)
    ->res

```

```

Function getListPath(firstDir: String)->array of String
{Menghasilkan daftar path yang mungkin dari sebuah direktori }
Deklarasi:
searchFolderTemp: IList<string>
searchFileTemp: IList<string>
searchFolder: array of String
searchFile: array of String
listPath: array of String
folderResult: array of String
Algoritma:
    if(firstDir.Length=0)then { nama direktori kosong }
        -> []
    else { nama direktori ada, Mengambil direktori di bawah firstDir }
        try
            Directory.GetDirectories(firstDir + "\\").ToList().ForEach(s =>
searchFolderTemp.Add(s))
            catch(UnauthorizedAccessException)
                { }
            { Mengambil file di bawah firstDir }
            try
                Directory.GetFiles(firstDir + "\\").ToList().ForEach(s =>
searchFileTemp.Add(s))
                catch(UnauthorizedAccessException)
                    { }
            {Urutkan sesuai abjad }
            searchFolder <- searchFolderTemp.ToArray()
            searchFile <- searchFileTemp.ToArray()
            Array.Sort(searchFolder, StringComparer.OrdinalIgnoreCase)
            Array.Sort(searchFile, StringComparer.OrdinalIgnoreCase);
            { Rekursi untuk mendapatkan semua file dan folder }
            foreach (folder in searchFolder)
                folderResult[0] <- folder
                listPath<- listPath.Concat(folderResult).ToArray()
                listPath<- listPath.Concat(getListPath(folderResult[0])).
ToArray()
                { Menghapus semua reference null }
                listPath<- listPath.Concat(searchFile).ToArray()
                listPath<- listPath.Where(s => s != null).ToArray()
            ->listPath

```

```

Function getMatrixNodeOnce(listPath: array of String,firstDir:
String,fileName: String)->array of array of String
{ Mendapatkan matrix node untuk kemunculan fileName yang pertama }
Deklarasi:

```



```

matrix: array of array of String
listPreviousPath: List<string>
pathReference:array of array of String
row: integer
arrLeaf: array of String
leaf: String
previousPath: String
previousId: String
leafId: String
hasilGabung: array of String
prevPath: String
arrFolder:array of String
arrPath: array of String
i: integer
edge: array of String
Algoritma:
    pathReference <- getPathId(listPath, firstDir)
    row <- 0
    foreach (path in listPath)
        arrLeaf <- path.Split('\\')
        leaf <- arrLeaf[arrLeaf.Length - 1]
        previousPath <- path.Substring(0, path.Length - leaf.Length - 1)
        previousId <- ""
        foreach (pathId in pathReference)
            if(pathId[0] = previousPath)then
                previousId <- pathId[1]
                break
        leafId <- ""
        foreach (pathId in pathReference)
            if (pathId[0] = path)then
                leafId <- pathId[1]
                break
        { mewarnai simpul awal dengan warna hitam }
        hasilGabung[0] <- previousId
        hasilGabung[1] <- leafId
        hasilGabung[2] <- previousPath
        hasilGabung[3] <- leaf
        hasilGabung[4] <- "0"
        Matrix[row] <- hasilGabung
        row <- row + 1
    { Karena ada kemungkinan terdapat path yang belum diperiksa karena
break di foreach awal, maka dilakukan iterasi kembali }
    row <- 0
    foreach (path in listPath)

```

```

arrLeaf <- path.Split('\\')
leaf <- arrLeaf[arrLeaf.Length - 1]
previousPath <- path.Substring(0, path.Length - leaf.Length - 1)
previousId <- ""
foreach (pathId in pathReference)
    if(pathId[0] = previousPath)then
        previousId <- pathId[1]
        break
leafId <- ""
foreach (pathId in pathReference)
    if (pathId[0] = path)then
        leafId <- pathId[1]
        break
{ mewarnai simpul awal dengan warna hitam }
hasilGabung[0] <- previousId
hasilGabung[1] <- leafId
hasilGabung[2] <- previousPath
hasilGabung[3] <- leaf
hasilGabung[4] <- "0"
Matrix[row] <- hasilGabung
if (leaf = fileName) then { merupakan path ke file }
    matrix[row][4] <- "1" { Kalau merupakan path ke file atau file
itu sendiri, warnai biru }
    listPreviousPath.Add(previousPath)
    break
else { bukan merupakan path ke file yang dicari }
    matrix[row][4] <- "2" { Kalau tidak ketemu, warnai merah }
    row <- row + 1

{ Mengubah path menjadi nama file/folder }
foreach (node in matrix)
    prevPath <- node[2];
    arrFolder <- prevPath.Split('\\')
    node[2] <- arrFolder[arrFolder.Length - 1]

{ Mengiterasi semua folder di atas file yang ditemukan }
foreach (path in listPreviousPath)
    arrPath <- path.Split('\\')
    i traversal[0..arrPath.Length - 2]
    edge[0] <- arrPath[i]
    edge[1] <- arrPath[i + 1]
    foreach (node in matrix)
        if (node[2] = edge[0] and node[3] = edge[1])then
            node[4] <- "1"

```

**break**

->matrix

**Function** getMatrixNodeAllOccurence(listPath: array of String, firstDir: String, fileName: String) -> array of array of String

{ Mendapatkan matrix node untuk semua kemunculan fileName }

**Deklarasi:**

matrix: array of array of String

listPreviousPath: List<string>

pathReference: array of array of String

row: integer

arrLeaf: array of String

leaf: String

previousPath: String

previousId: String

leafId: String

hasilGabung: array of String

prevPath: String

arrFolder: array of String

arrPath: array of String

i: integer

edge: array of String

**Algoritma:**

pathReference <- getPathId(listPath, firstDir)

row <- 0

{ Pasti diperiksa semua sehingga hanya perlu 1 foreach loop}

foreach (path in listPath)

arrLeaf <- path.Split('\\')

leaf <- arrLeaf[arrLeaf.Length - 1]

previousPath <- path.Substring(0, path.Length - leaf.Length - 1)

previousId <- ""

foreach (pathId in pathReference)

if (pathId[0] = previousPath) then

previousId <- pathId[1]

**break**

leafId <- ""

foreach (pathId in pathReference)

if (pathId[0] = path) then

leafId <- pathId[1]

**break**

{ mewarnai simpul awal dengan warna hitam }

hasilGabung[0] <- previousId

hasilGabung[1] <- leafId

hasilGabung[2] <- previousPath

```

    hasilGabung[3] <- leaf
    hasilGabung[4] <- "0"
    Matrix[row] <- hasilGabung
    if (leaf = fileName) then { merupakan path ke file }
        matrix[row][4] <- "1" { Kalau merupakan path ke file atau file
itu sendiri, warnai biru }
        listPreviousPath.Add(previousPath)
        break
    else { bukan merupakan path ke file yang dicari }
        matrix[row][4] <- "2" { Kalau tidak ketemu, warnai merah }
    row <- row + 1

{ Mengubah path menjadi nama file/folder }
foreach (node in matrix)
    prevPath <- node[2];
    arrFolder <- prevPath.Split('\\')
    node[2] <- arrFolder[arrFolder.Length - 1]

{ Mengiterasi semua folder di atas file yang ditemukan }
foreach (path in listPreviousPath)
    arrPath <- path.Split('\\')
    i traversal[0..arrPath.Length - 2]
        edge[0] <- arrPath[i]
        edge[1] <- arrPath[i + 1]
        foreach (node in matrix)
            if (node[2] = edge[0] and node[3] = edge[1])then
                node[4] <- "1"
            break
->matrix

{ Fungsi dan Prosedur Lain yang Mendukung Algoritma }
function getPathId(listPath:array of String, firstDir: String)-> array of
array of String
{ mendapatkan id unik dari sebuah path }
function displayGraph(pathReference:array of array of String,
matrixNode: array of array of String)->GViewer
{ Mengolah matriks node untuk membentuk graf berisi pohon pencarian
dengan bantuan library MSAGL }
function getDirectory(listPath:array of String, fileName: String)
->array of String
{ mengembalikan daftar path menuju file/folder yang dicari }
function getDirectoryOneFound(listPath:array of String,
fileName:String)->array of String

```

```
{mengembalikan path dari kemunculan pertama file / folder yang dicari }
```

## Struktur Data

### Result

Struktur data ini diimplementasikan dalam bentuk kelas. Struktur data ini berguna untuk menyimpan hasil pencarian yang terdiri dari GViewer yang berisi pohon pencarian yang siap ditampilkan dan array string yang berisi path dari file-file yang ditemukan.

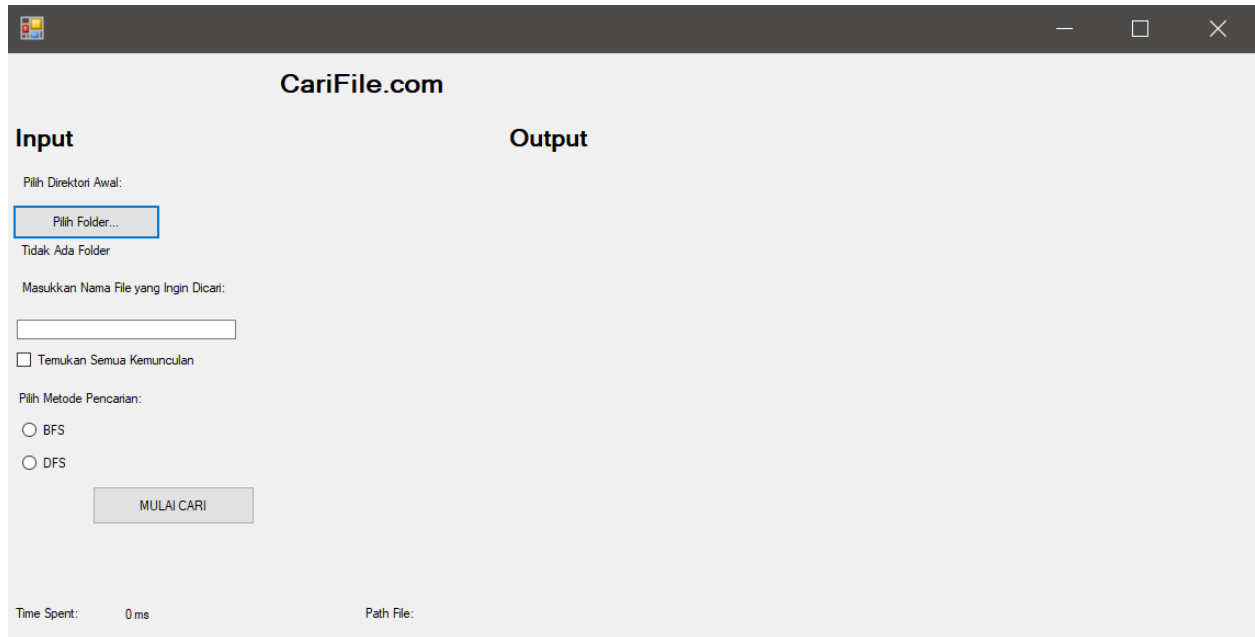
```
//class buat nyimpen hasil pencarian
public class Result
{
    public string[] listOfPath { get; set; }
    public Microsoft.Msagl.GraphViewerGdi.GViewer graph { get; set; }
}
```

Gambar 4.1. Definisi kelas Result

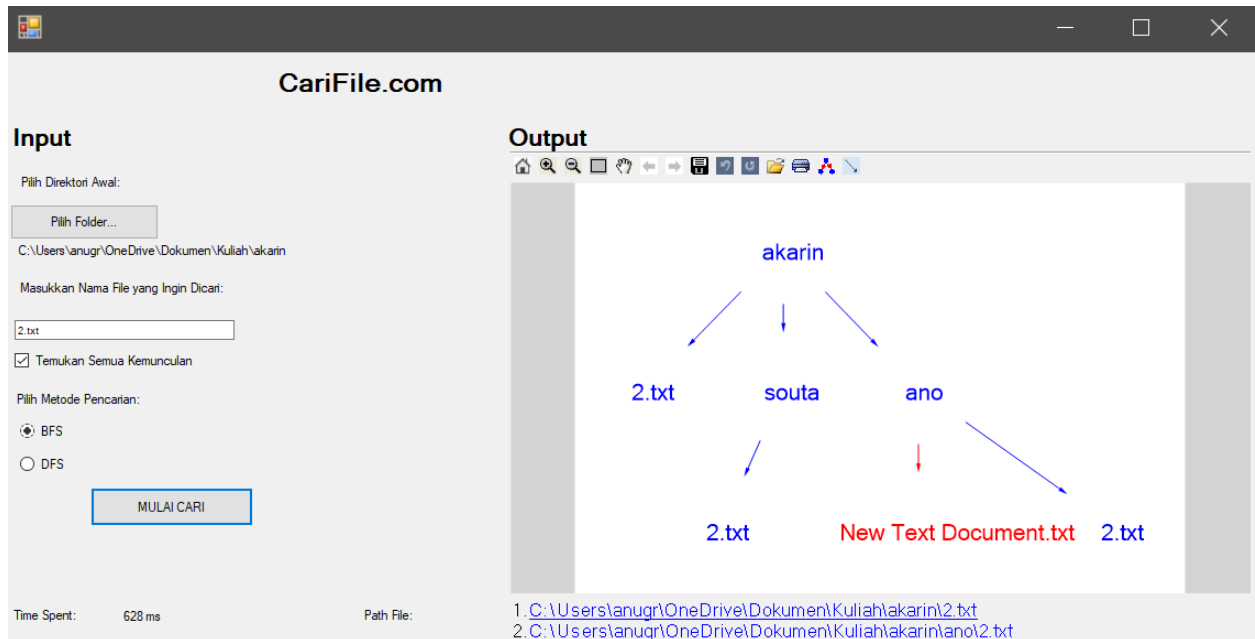
### Spesifikasi Program

1. Program dapat menerima input folder dan query nama file.
2. Program dapat menerima input pilihan untuk mencari satu kemunculan saja atau semua kemunculan.
3. Program dapat menerima input 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. Program dapat menampilkan hasil pencarian berupa rute/path (bisa lebih dari satu jika memilih menemukan semua file) serta durasi waktu algoritma.

## Tata Cara Penggunaan Program Interface



Gambar 4.2. Tampilan Awal Aplikasi



Gambar 4.3. Tampilan Hasil Pencarian

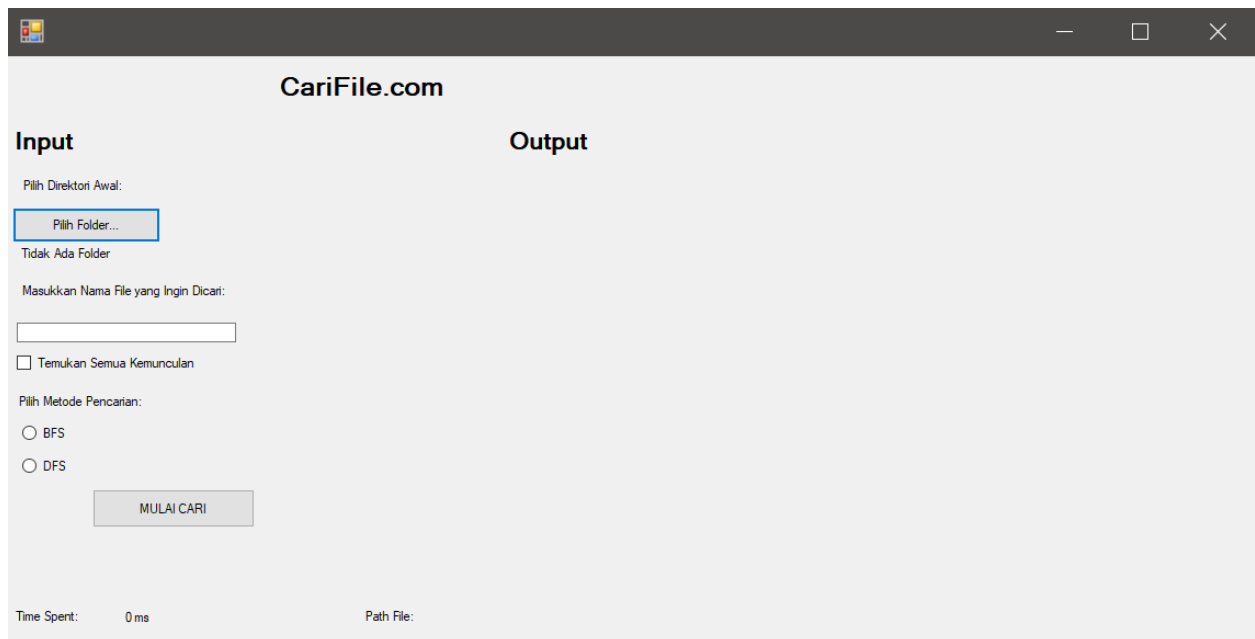
## Fitur

- Dapat memasukkan folder awal yang diinginkan menggunakan GUI open file dialog.
- Dapat memilih algoritma yang digunakan untuk mencari file/folder.
- Dapat memilih antara hanya mencari kemunculan pertama file/folder saja atau mencari semua kemunculan file/folder yang dicari.
- Dapat menampilkan pohon pencarian dengan warna untuk ilustrasi node yang menjadi path ke file, tidak menjadi path ke file, dan node yang belum diperiksa.
- Pada hasil pencarian, ditampilkan daftar path file yang dicari. Jika ditekan, maka akan membuka folder parent file/folder tersebut.

## Cara Penggunaan

1. Double klik pada file **CariFile.com.exe** di folder **bin**
2. Klik **Pilih Folder...**
3. Masukkan nama file yang ingin dicari
4. Klik Temukan semua kemunculan jika ingin menemukan semua kemunculan file
5. Pilih salah satu metode pencarian (BFS atau DFS)
6. Klik MULAI CARI
7. Tunggu hingga pencarian selesai dan hasil ditampilkan

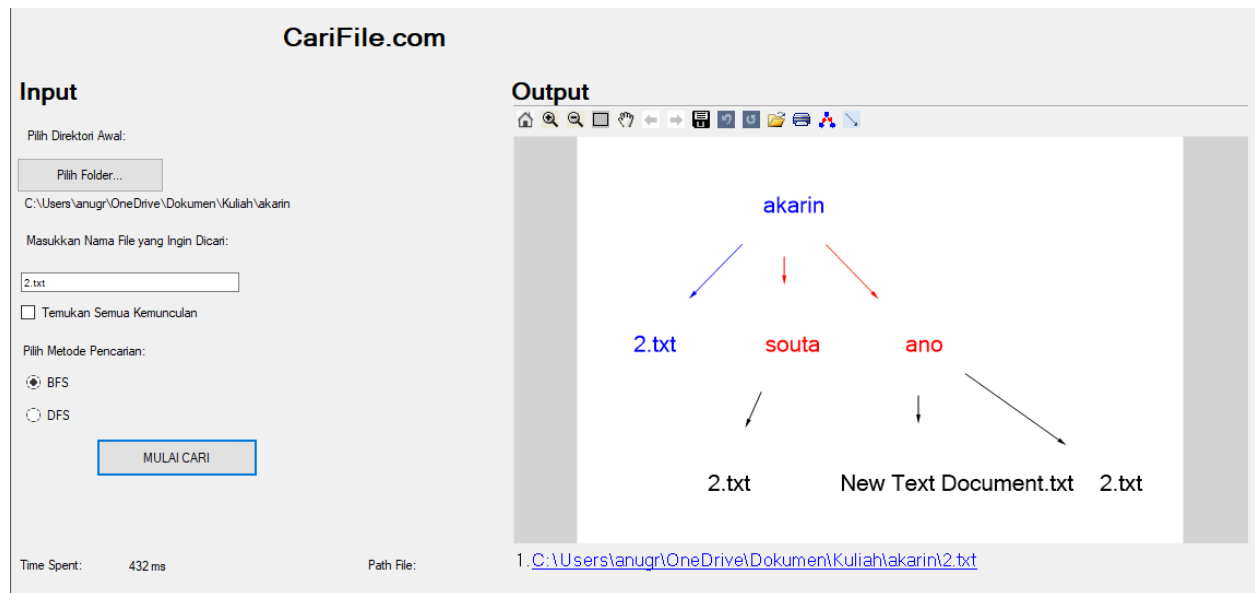
## Hasil Pengujian



Gambar 4.4. Tatap Muka Program

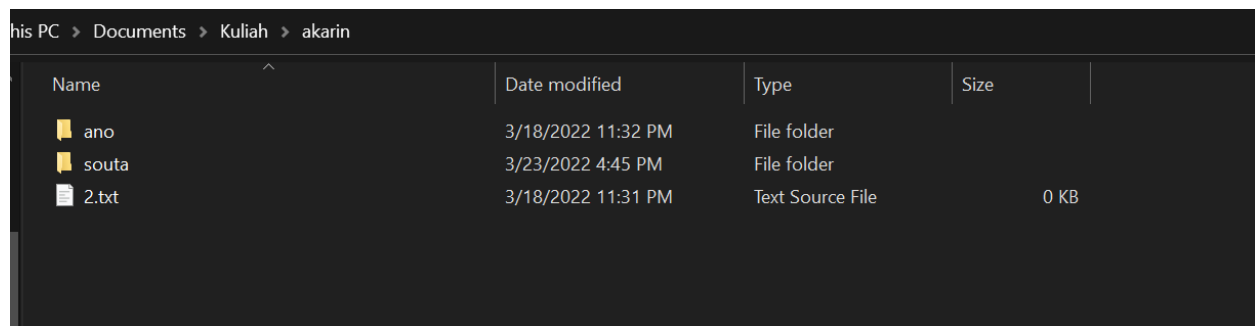
#### A. TC 1 ( FILE DITEMUKAN , KEMUNCULAN PERTAMA, BFS )

Deskripsi: Mencari kemunculan pertama sebuah file bernama “2.txt” dimulai dari folder “C:\Users\anugr\OneDrive\Dokumen\Kuliah\akaran” menggunakan algoritma BFS



Gambar 4.5. Hasil Pencarian TC1

Jika nomor 1 dibuka:

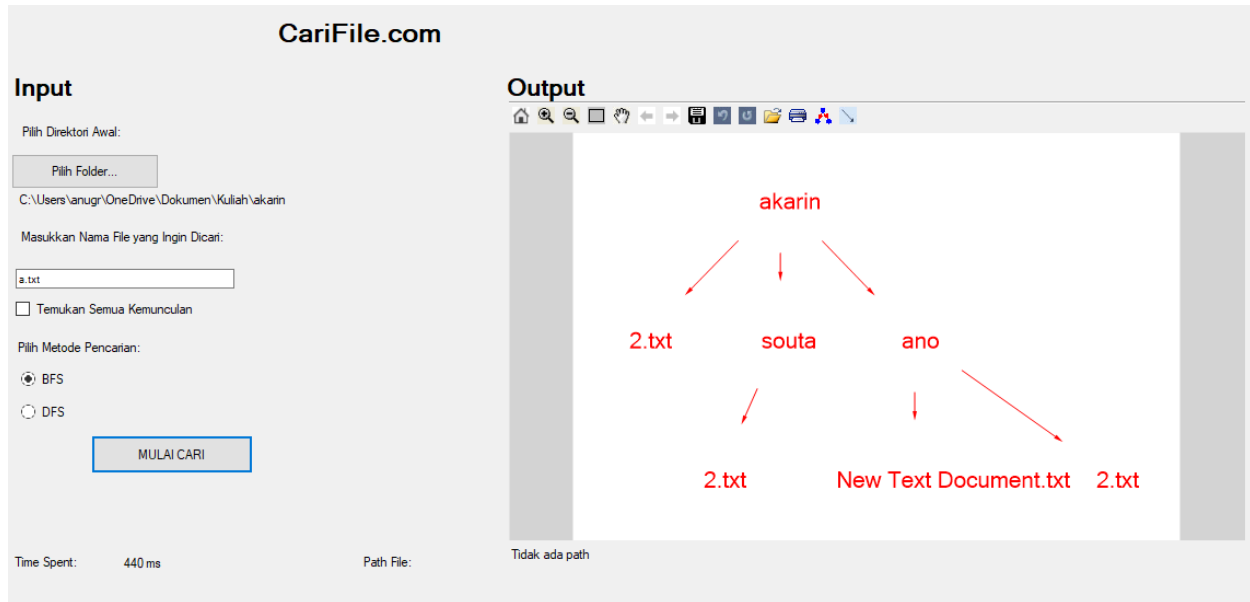


Gambar 4.6. Contoh Path Yang Dibuka

#### B. TC 2 ( FILE TIDAK DITEMUKAN , KEMUNCULAN PERTAMA, BFS )

Deskripsi: Mencari kemunculan pertama sebuah file bernama “a.txt” dimulai dari folder “C:\Users\anugr\OneDrive\Dokumen\Kuliah\akaran” menggunakan algoritma BFS

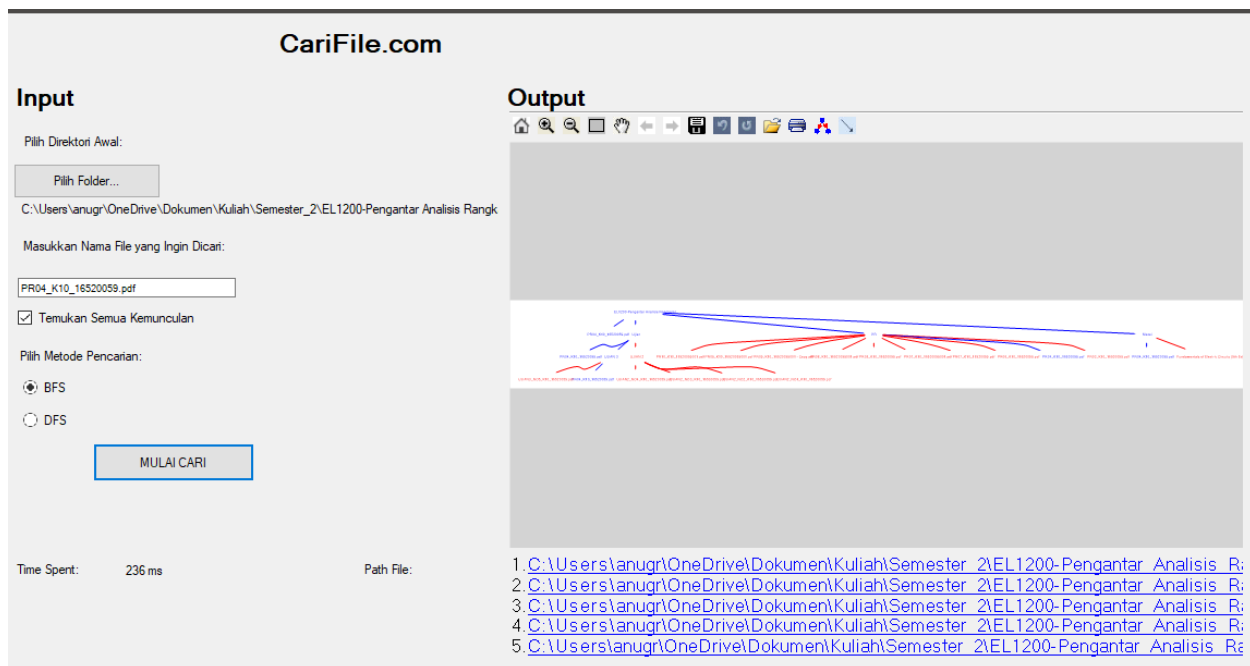




Gambar 4.7. Hasil Pencarian TC2

### C. TC 3 ( FILE DITEMUKAN , SEMUA KEMUNCULAN, BFS )

Deskripsi: Mencari semua kemunculan file bernama “PR04\_K10\_16520059.pdf” dimulai dari folder “C:\Users\anugr\OneDrive\Dokumen\Kuliah\Semester\_2\EL1200-Pengantar Analisis Rangkaian” menggunakan algoritma BFS



Gambar 4.8. Hasil Pencarian TC3

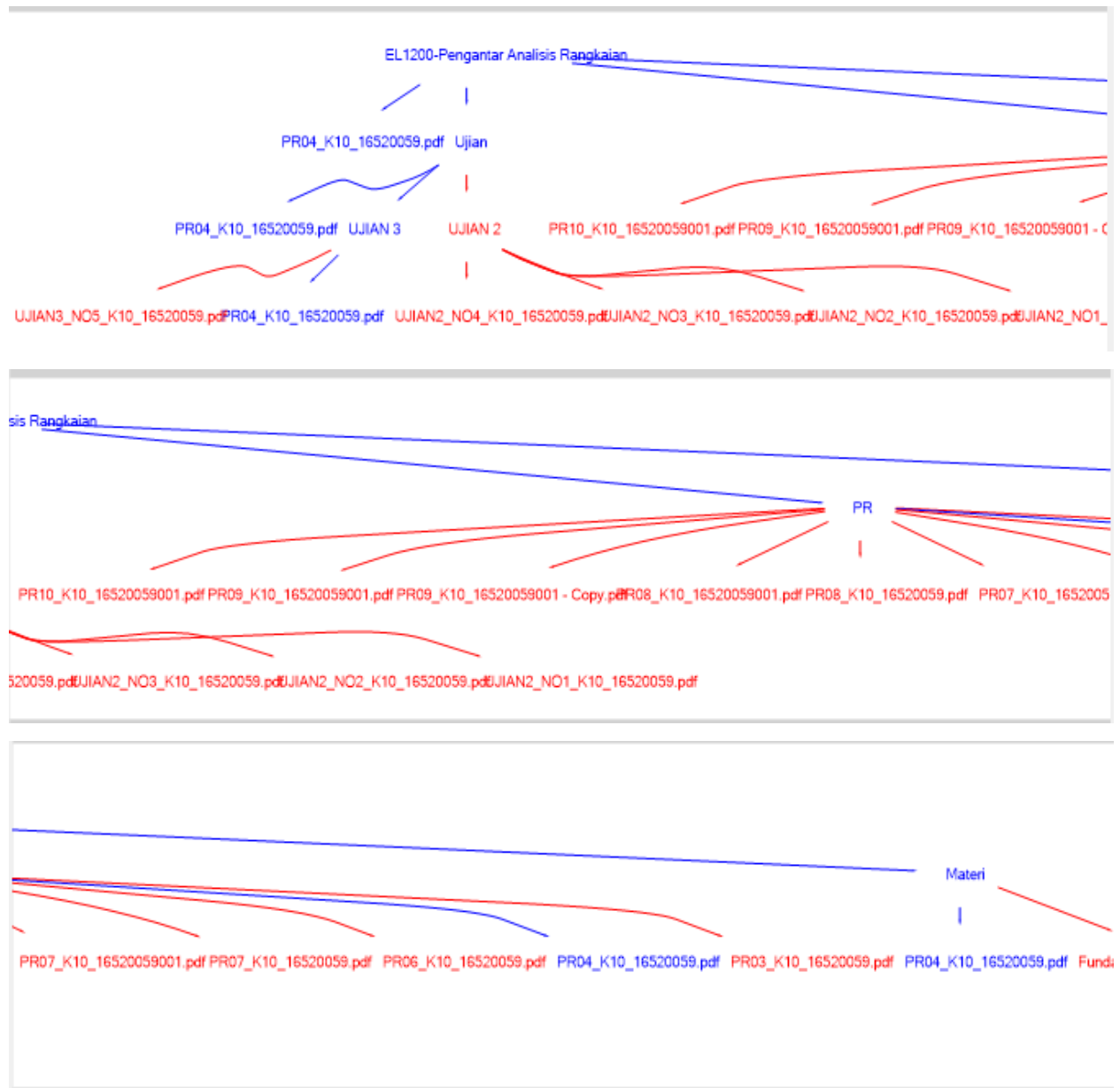
Jika nomor 2 dibuka:

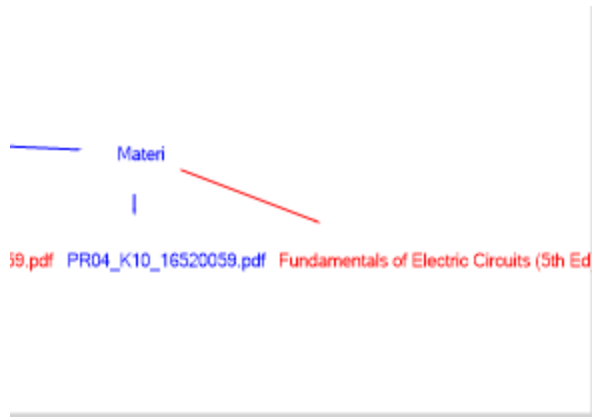
This PC > Documents > Kuliah > Semester\_2 > EL1200-Pengantar Analisis Rangkaian > Materi

| Name  | Date modified     | Type                  | Size      |
|---|-------------------|-----------------------|-----------|
| Fundamentals of Electric Circuits (5th Ed)(gnv64... | 5/5/2021 5:38 PM  | Adobe Acrobat Docu... | 23,783 KB |
| PR04_K10_16520059.pdf                               | 3/11/2021 7:45 PM | Adobe Acrobat Docu... | 1,753 KB  |

Gambar 4.9. Contoh Path Yang Dibuka

Grafnya jika di zoom:

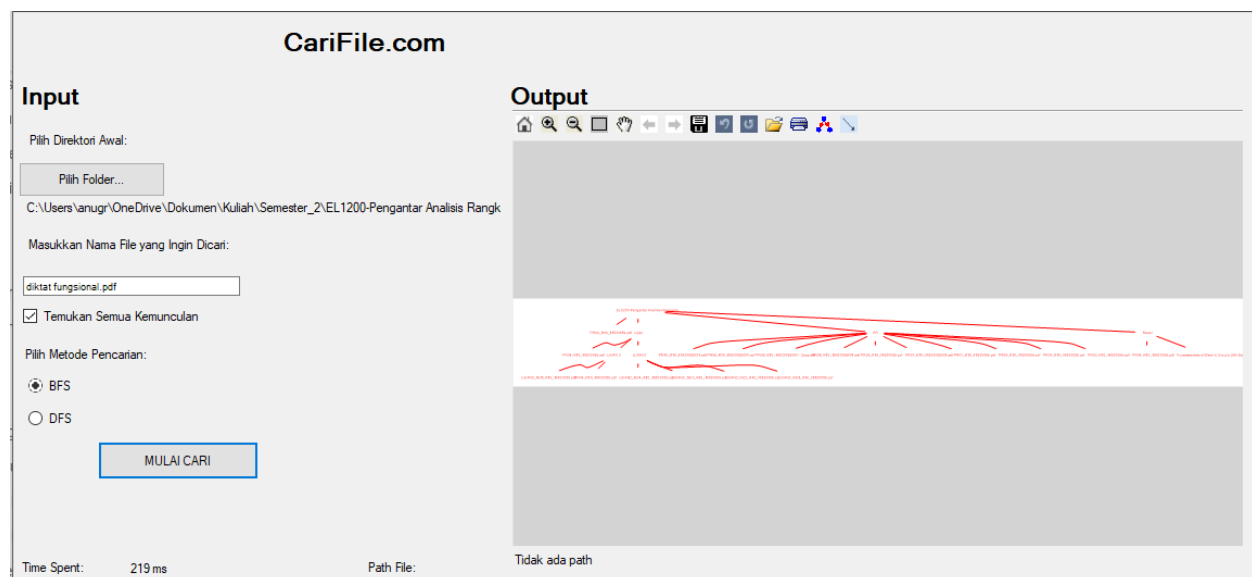




Gambar 4.9 Pohon Pencarian TC 3

D. TC 4 ( **FILE TIDAK DITEMUKAN , SEMUA KEMUNCULAN , BFS** )

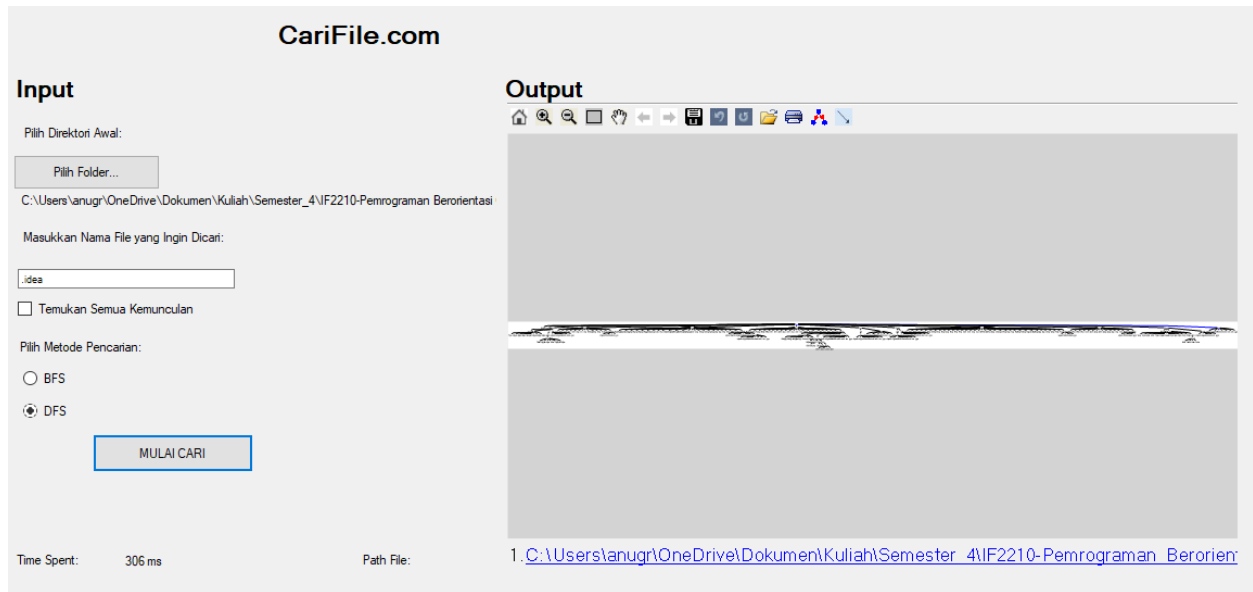
Deskripsi: Mencari semua kemunculan sebuah file bernama “diktat fungsional.pdf” dimulai dari folder “C:\Users\anugr\OneDrive\Dokumen\Kuliah\Semester\_2\EL1200-Pengantar Analisis Rangkaian” menggunakan algoritma BFS



Gambar 4.10. Hasil Pencarian TC4

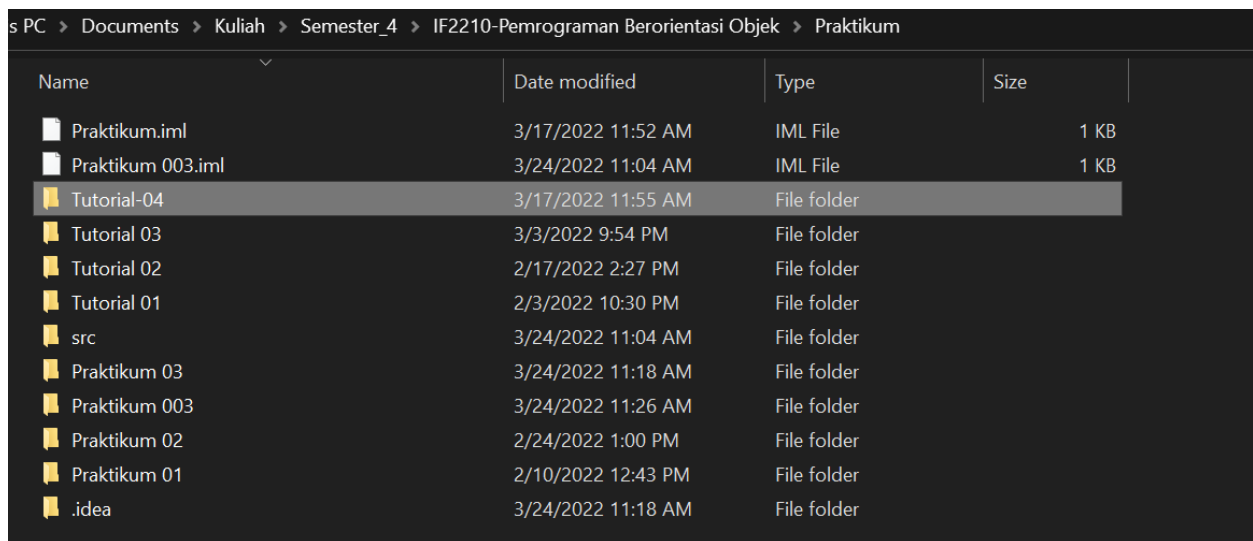
E. TC 5 ( **FILE DITEMUKAN , KEMUNCULAN PERTAMA, DFS** )

Deskripsi: Mencari kemunculan pertama sebuah folder bernama “.idea” dimulai dari folder “C:\Users\anugr\OneDrive\Dokumen\Kuliah\Semester\_4\IF2210-Pemrograman Berorientasi Objek\Praktikum” menggunakan algoritma DFS



Gambar 4.11. Hasil Pencarian TC5

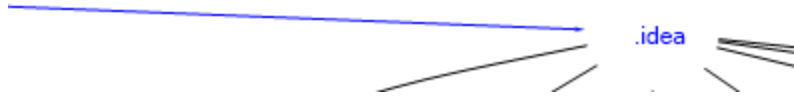
Jika nomor 1 dibuka:



Gambar 4.12. Contoh Path Yang Dibuka

Path dari pohon yang mengarah ke solusi:

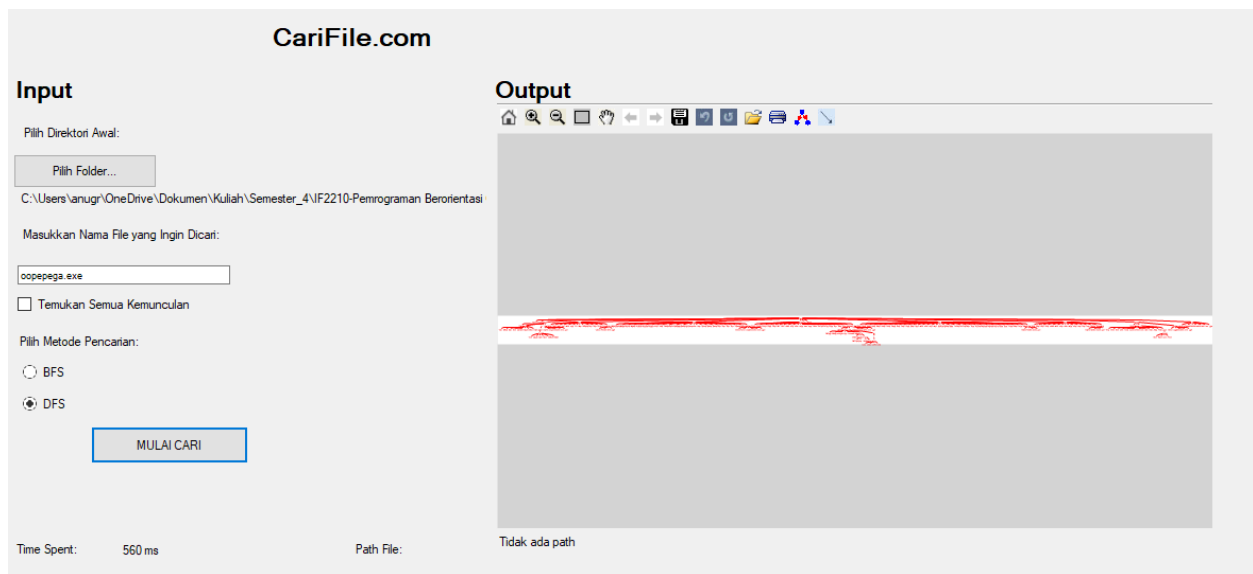




Gambar 4.13. Path ke Solusi

**F. TC 6 ( FILE TIDAK DITEMUKAN , KEMUNCULAN PERTAMA, DFS )**

Deskripsi: Mencari kemunculan pertama sebuah file bernama “oopepega.exe” dimulai dari folder “C:\Users\anugr\OneDrive\Dokumen\Kuliah\Semester\_4\IF2210-Pemrograman Berorientasi Objek\Praktikum” menggunakan algoritma DFS



Gambar 4.14. Hasil Pencarian TC6

**G. TC 7 ( FILE DITEMUKAN , SEMUA KEMUNCULAN, DFS )**

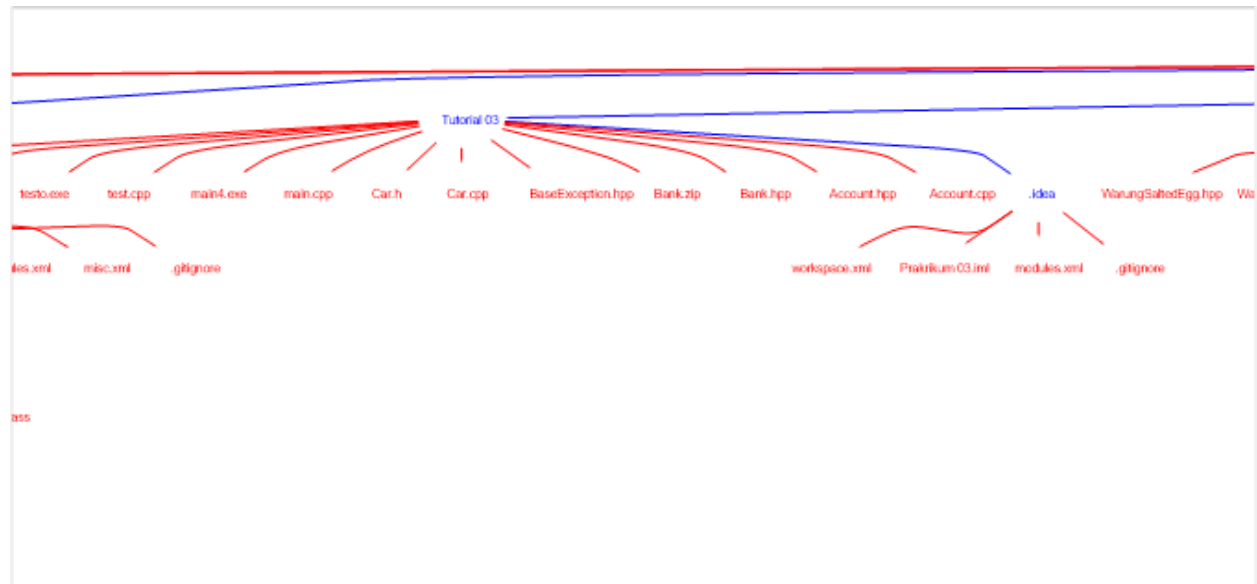
Deskripsi: Mencari semua kemunculan folder bernama “.idea” dimulai dari folder “C:\Users\anugr\OneDrive\Dokumen\Kuliah\Semester\_4\IF2210-Pemrograman Berorientasi Objek\Praktikum” menggunakan algoritma DFS



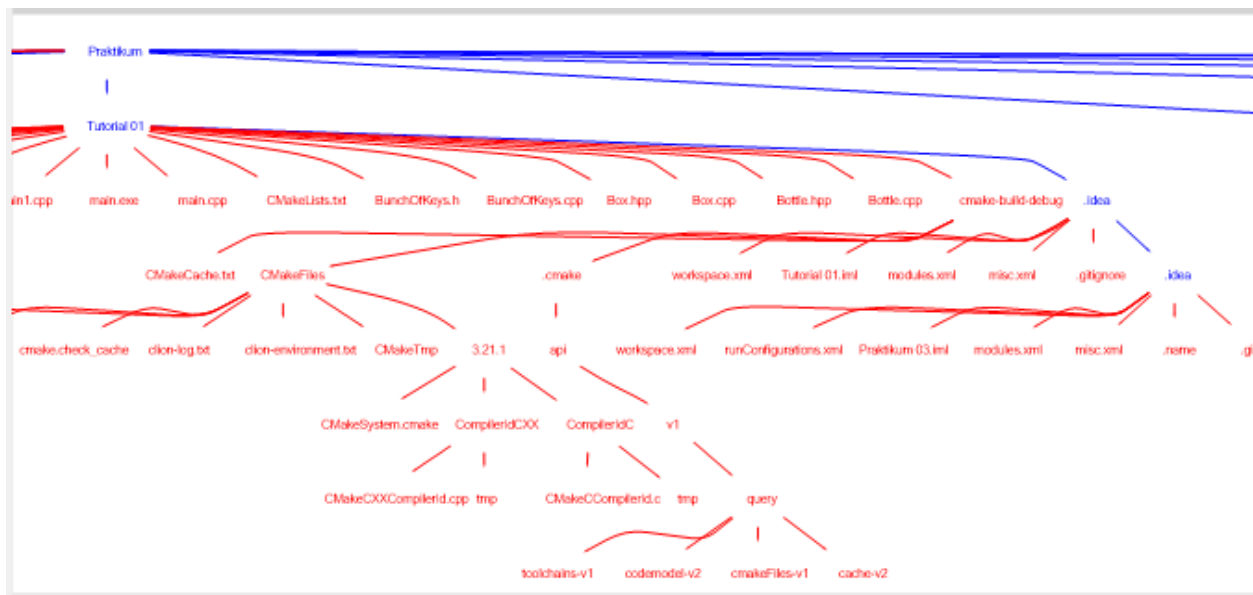
Gambar 4.15. Hasil Pencarian TC7

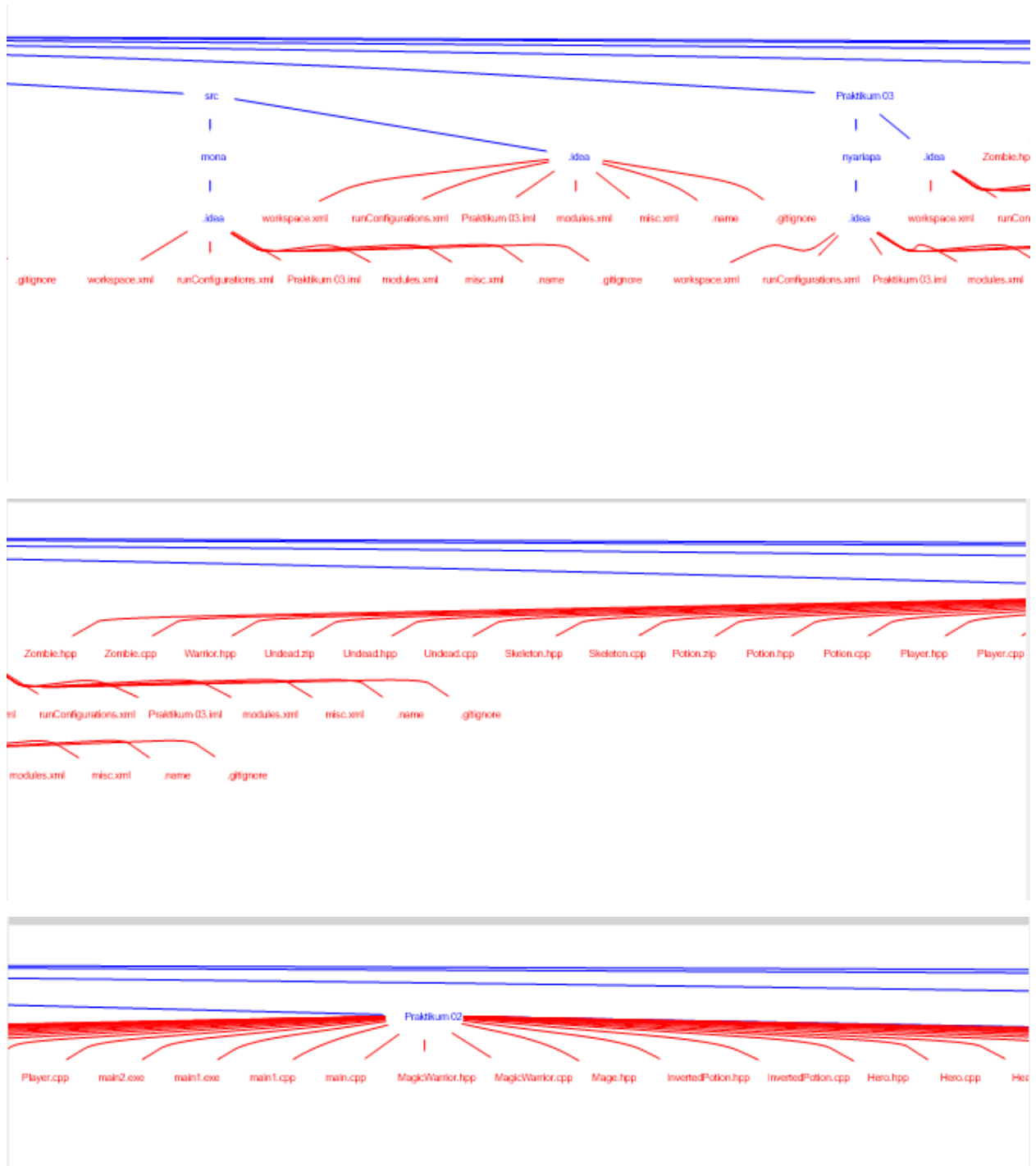
Jika nomor 3 dibuka:

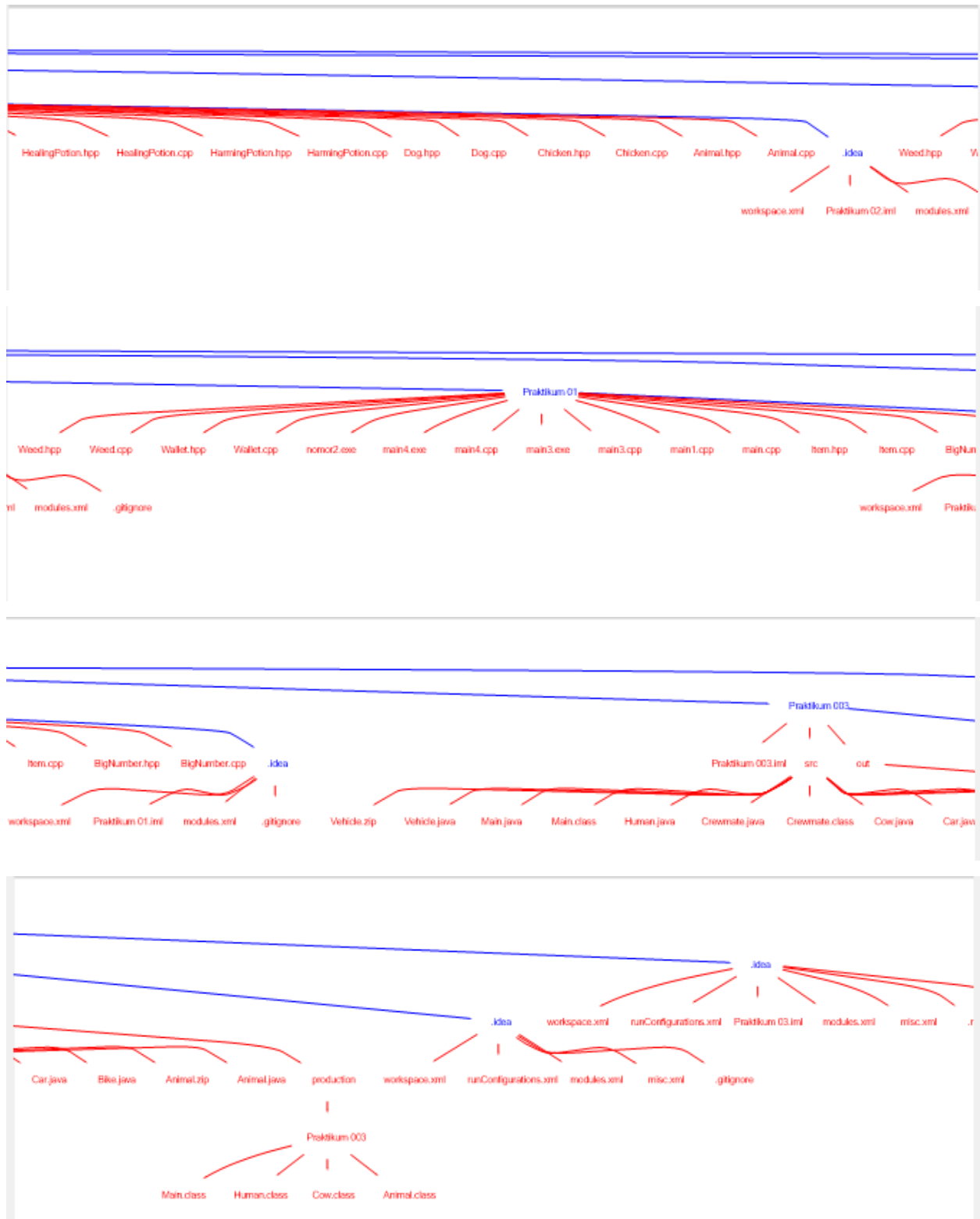










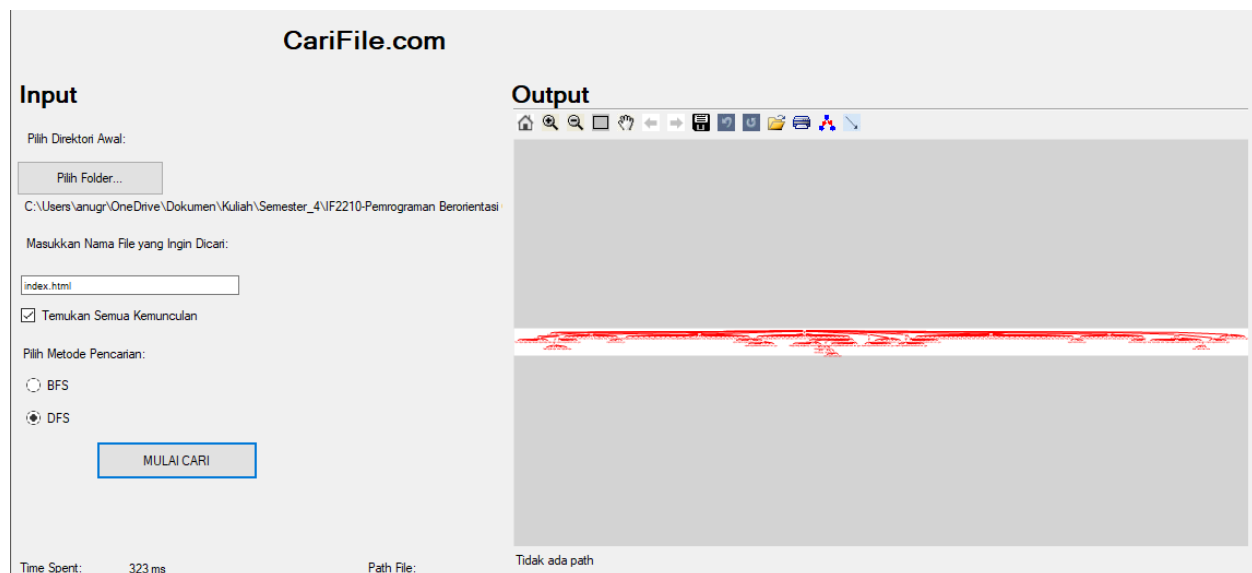




Gambar 4.17 Pohon Pencarian TC 7

#### H. TC 8 ( FILE TIDAK DITEMUKAN , SEMUA KEMUNCULAN , DFS )

Deskripsi: Mencari sebuah kemunculan file bernama “index.html” dimulai dari folder “C:\Users\anugr\OneDrive\Dokumen\Kuliah\Semester\_4\IF2210-Pemrograman Berorientasi Objek\Praktikum” menggunakan algoritma DFS



Gambar 4.18. Hasil Pencarian TC8

## **Analisis Desain Solusi**

Pada dasarnya, algoritma BFS dan DFS tidak memiliki perbedaan jauh dalam kompleksitas tetapi hasil kecepatan pencarian dapat berbeda berdasarkan cara folder dan file diatur. DFS lebih baik digunakan ketika mencari file/folder yang berada di bagian lebih dalam pada gambar dari hasil pencarian sedangkan BFS lebih baik digunakan ketika mencari file/folder yang berada di bagian lebih awal. Hal ini disebabkan program memeriksa lebih sedikit file/folder untuk menemukan input dari pengguna. Contoh dari hasil pengujian dapat dilihat pada TC1 dari gambar 4.5. Pada gambar tersebut, pencarian BFS akan memeriksa 3 barang sebelum memberhentikan pemeriksaan. Jika digunakan DFS pada TC1, maka harus diperiksa semua barang yang terdapat pada gambar tersebut karena ketika menggunakan DFS, akan diperiksa semua barang yang terdapat pada folder “ano” terlebih dahulu sebelum memeriksa folder “souta”. Ketika BFS dan DFS diminta untuk memeriksa semua kemunculan, kedua pencarian akan memiliki hasil kecepatan yang mirip karena harus memeriksa jumlah barang yang sama.

## **BAB V: Kesimpulan, Saran, dan Link *Source Code***

### **Kesimpulan**

Tugas Besar 2 IF2211 Strategi Algoritma Semester 2 Tahun 2021/2022 tentang “Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling” telah diselesaikan dengan hasil yang memuaskan. Kami, kelompok 9 dengan nama CariFile.com yang beranggotakan Diky, William, dan Fawwaz, telah berhasil membuat program dalam bahasa C# yang dapat mencari file dari direktori masukan. Program yang telah dibuat juga memiliki Graphical User Interface (GUI) yang memudahkan penggunaanya sekaligus membuat program ini menjadi jauh lebih menarik. GUI yang telah dirancang dilengkapi dengan fitur tampilan graf berwarna untuk menunjukkan jalur penelusuran yang telah dilakukan. Semua fitur yang ada ditujukan untuk kenyamanan *user* ketika menggunakan program.

### **Saran**

Saran-saran yang dapat kami berikan untuk Tugas Besar IF2211 Strategi Algoritma Semester 2 Tahun 2021/2022 adalah

1. Algoritma yang digunakan dalam Tugas Besar ini masih memiliki banyak kekurangan dan sangat memungkinkan untuk dilakukan efisiensi, misalnya penggunaan rekursi yang bisa saja mengakibatkan *stack overflow* jika pemanggilan fungsinya terlalu banyak.
2. Program ini dapat dikembangkan lebih lanjut dari segi UI/UX supaya semakin *user-friendly* dan memiliki desain yang lebih menarik
3. Memperjelas spesifikasi dan batasan-batasan setiap program pada file tugas besar untuk mencegah adanya multitafsir dan kesalahpahaman pada proses pembuatan program
4. Menimbang fungsionalitas dari program pada Tugas Besar ini, sebaiknya program ini bisa dipublikasikan setelah dikembangkan lebih lanjut supaya memiliki kebermanfaatan yang lebih luas

### **Tautan *Source Code***

[https://github.com/dikyrest/Tubes2\\_13520017](https://github.com/dikyrest/Tubes2_13520017)

## DAFTAR PUSTAKA

1. Rinaldi Munir dan Nur Ulfa Maulidevi, *Breadth First Search (BFS) dan Depth First Search (DFS) Bagian 1*, 2021, diakses melalui <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>
2. Rinaldi Munir dan Nur Ulfa Maulidevi, *Breadth First Search (BFS) dan Depth First Search (DFS) Bagian 2*, 2021, diakses melalui <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>
3. *C# docs - get started, tutorials, reference*, diakses melalui <https://docs.microsoft.com/en-us/dotnet/csharp/>
4. *Panduan MSAGL untuk Visualisasi Graph*, diakses melalui <https://docs.google.com/document/d/1XhFSpHU028Gaf7YxkmdbluLkQgVl3MY6gt1t-PL30LA/edit>