

LAPORAN TUGAS BESAR

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

Ditujukan untuk memenuhi salah satu tugas besar mata kuliah IF2211 Strategi Algoritma pada
Semester I Tahun Akademik 2021/2022



Disusun oleh:

Rheza Rizqullah Ecaldy	13520060
Muhammad Gilang Ramadhan	13520137
Muhammad Gerald Akbar Giffera	13520143

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2021**

DAFTAR ISI

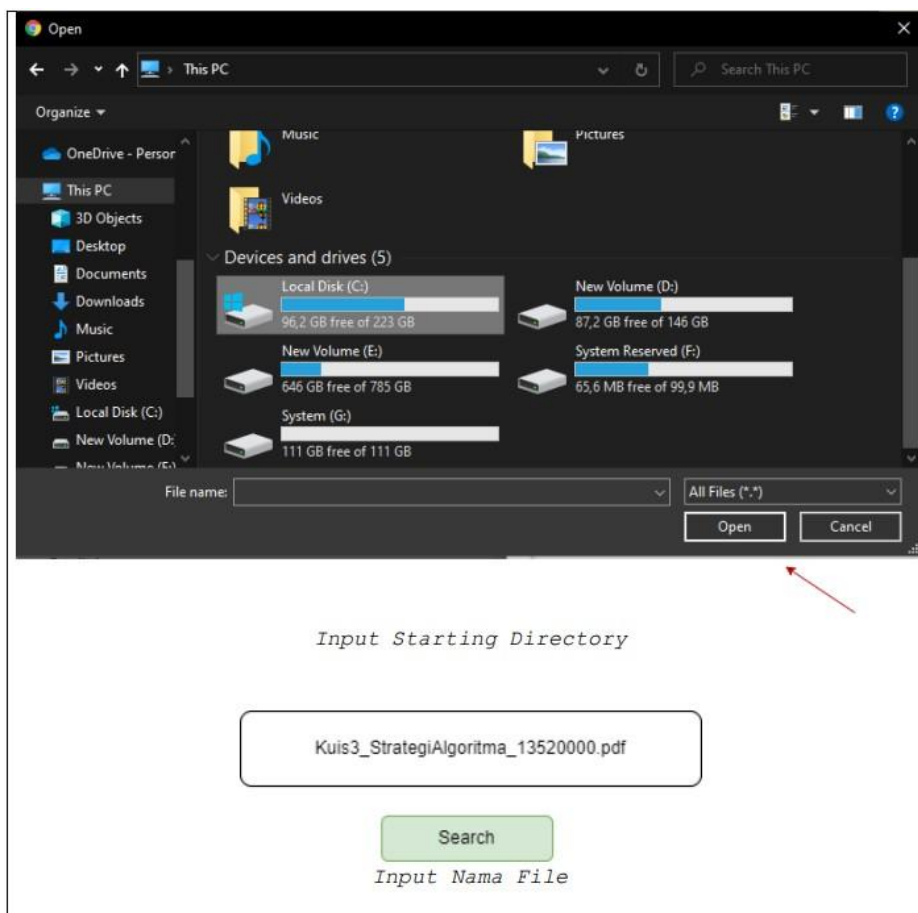
BAB I DESKRIPSI TUGAS	3
BAB II LANDASAN TEORI	6
BAB III ANALISIS PEMECAHAN MASALAH	9
BAB IV IMPLEMENTASI DAN PENGUJIAN	12
BAB V KESIMPULAN DAN SARAN	27
DAFTAR PUSTAKA	28
LAMPIRAN	29

BAB I

DESKRIPSI TUGAS

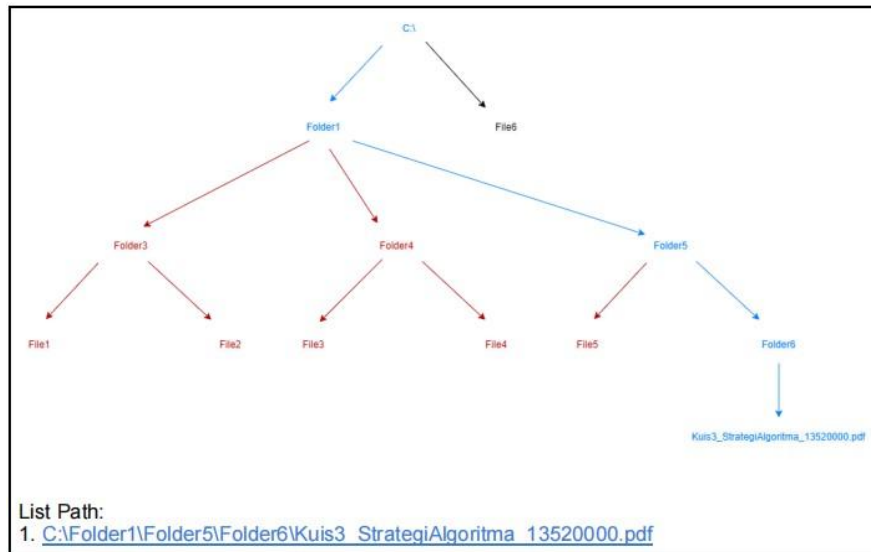
Dalam tugas besar ini, Setiap kelompok akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari *file explorer* pada sistem operasi, yang pada tugas ini disebut dengan *Folder Crawling*. Dengan memanfaatkan algoritma *Breadth First Search* (BFS) dan *Depth First Search*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

Selain pohon, aplikasi yang dibuat juga harus bisa 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.



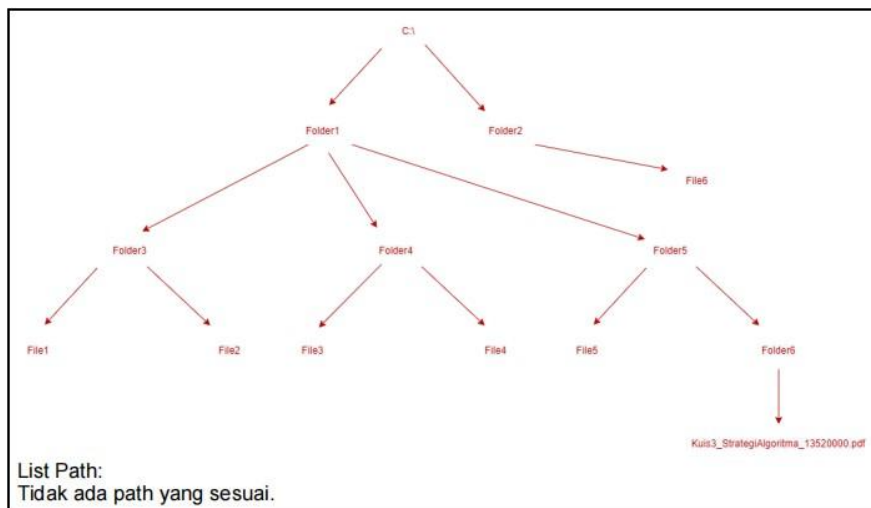
Gambar 1.1 Contoh input program

Contoh output aplikasi :



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. Pemilihan warna disebabkan asalkan dibedakan antara ketiga hal tersebut.



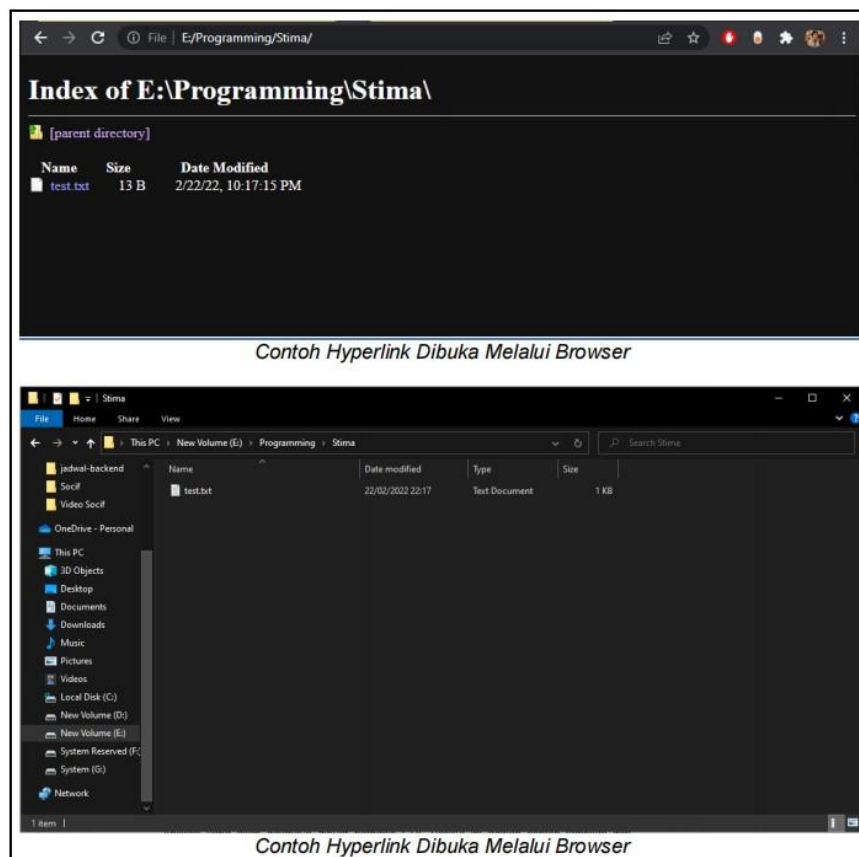
Gambar 1.3 Contoh output program jika file tidak ditemukan

Laporan Tugas Besar 2

IF2211–Kelompok Tukang Lipat

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 :



Gambar 1.4 Contoh ketika hyperlink di-klik

BAB II

LANDASAN TEORI

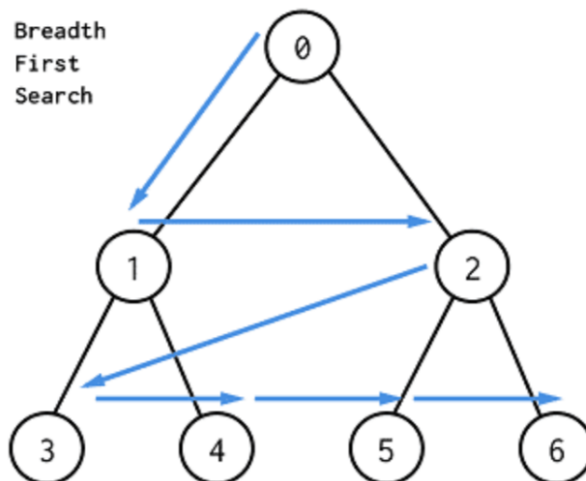
1. Graph Traversal

Graph transversal adalah proses penelusuran simpul-simpul dalam graf secara sistematis. Proses penelusuran graf dengan algoritma ini dilakukan sesuai aturan dan ketentuan tertentu. Terdapat 2 jenis algoritma graf transversal yaitu *breadth first search (BFS)* dan *depth first search (DFS)*.

2. Algoritma BFS

Breadth First Search (BFS) adalah metode penelusuran graf secara transversal secara melebar. Pencarian dimulai dari sebuah simpul v yang selanjutnya akan “diperlebar” ke simpul-simpul yang bertetangga dengan simpul v . Langkah-langkah yang dilakukan pada *BFS* adalah sebagai berikut:

- Kunjungi simpul v
- Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu
- Untuk setiap simpul yang dikunjungi tersebut kunjungi simpul yang belum dikunjungi dan bertetangga simpul tersebut
- Ulangi langkah 3 hingga semua simpul dikunjungi

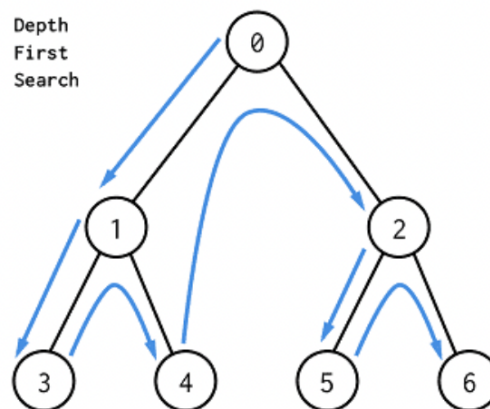


Gambar 2.2.1 Ilustrasi Algoritma Graf Transversal BFS

3. Algoritma DFS

Depth First Search (DFS) adalah metode penelusuran graf secara transversal secara mendalam. Penelusuran akan dimulai dari sebuah simpul v yang selanjutnya akan diekspansi secara “mendalam”, sehingga penelusuran akan dilakukan ke simpul-simpul daun terlebih dahulu, jika sudah tidak ada simpul yang dapat dikunjungi maka akan dilakukan proses *backtracking*, hal ini bertujuan untuk mencari simpul selanjutnya yang belum dikunjungi. Berikut langkah-langkah dari algoritma DFS:

- Kunjungi simpul v
- Kunjungi simpul w yang merupakan simpul tetangga dari v
- Ulangi algoritma DFS secara rekursif dengan simpul awal adalah simpul w
- Apabila proses pencarian mencapai suatu simpul u sehingga tidak ada lagi simpul tetangga yang belum dikunjungi, dilakukan pencarian runut-balik ke simpul terakhir yang dikunjungi sebelum simpul u dan memiliki simpul tetangga yang belum dikunjungi.
- Pencarian berakhir apabila semua simpul telah dikunjungi atau tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai.



Gambar 2.3.1 Ilustrasi Algoritma DFS

4. C# Desktop Application Development

Desktop application merupakan perangkat lunak yang berjalan secara lokal pada komputer pribadi atau laptop pengguna, berbeda dengan *web application* yang berjalan pada *web browser* maupun *mobile application* yang berjalan pada *smartphone*. Desktop application dapat hanya berbentuk CLI (*Command Line Interface*) atau memiliki GUI (*Graphical User Interface*).

Salah satu bahasa yang populer digunakan dalam pengembangan aplikasi *desktop* adalah C#. C# merupakan bahasa pemrograman yang dikembangkan oleh Microsoft dan berjalan pada *framework* .NET. Bahasa C# tergolong bahasa pemrograman berorientasi objek. Dalam proses pengembangan aplikasi *desktop* berbahasa C#, *programmer* biasa menggunakan sebuah IDE buatan Microsoft yang bernama Visual Studio untuk mempermudah proses pengembangan.

BAB III

ANALISIS PEMECAHAN MASALAH

1. Langkah-langkah Pemecahan Masalah

Untuk menyelesaikan masalah yang telah dibahas di atas, kami melalui langkah-langkah sebagai berikut.

- a. Memahami masalah yang diberikan
- b. Memahami konsep DFS dan BFS dalam graph traversal
- c. Memahami dasar bahasa C#
- d. Memahami library yang berhubungan dengan pengaksesan drive, directory, dan files
- e. Memetakan persoalan menjadi elemen-elemen DFS dan BFS
- f. Mengimplementasikan algoritma DFS dan BFS dalam bahasa C# untuk menyelesaikan persoalan
- g. Memvisualisasikan hasil persoalan dalam bentuk graf dengan menggunakan MSAGL
- h. Membuat GUI
- i. Menghubungkan GUI dengan program utama yang telah dibuat

2. Proses Mapping Persoalan

Sesuai yang sudah dinyatakan pada bagian 3.1, persoalan akan dipetakan terlebih dahulu menjadi elemen-elemen DFS dan BFS sebelum solusi masalah diimplementasikan dalam bahasa C#. Elemen-elemen yang terdapat dalam algoritma DFS adalah simpul graf, matriks ketetanggaan, dan tabel boolean untuk menandai apakah suatu simpul sudah dikunjungi. Elemen-elemen yang terdapat dalam algoritma BFS adalah simpul graf, matriks ketetanggaan, tabel boolean, dan antrian. Persoalan akan dipetakan menjadi elemen-elemen tersebut sebagai berikut.

- a. Simpul graf
Simpul graf yang dibentuk adalah setiap file/directory yang berada di dalam sebuah simpul(directory) tertentu.
- b. Matriks ketetanggaan
Ketetanggaan direpresentasikan dengan memanfaatkan sebuah fungsi bawaan C#, dimana fungsi tersebut akan memberikan semua simpul(directory/file) yang bertetangga dengan simpul tertentu.

c. Tabel boolean

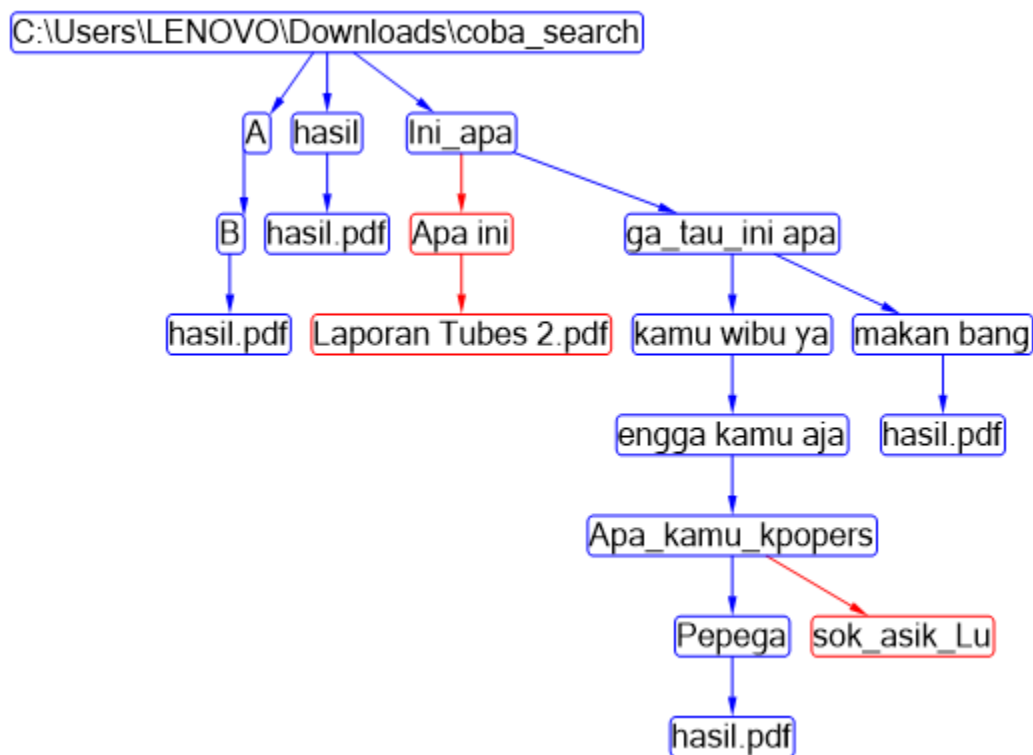
Karena pada persoalan ini graf belum terbentuk sebelumnya, melainkan graf secara teori akan dibentuk seiring dengan berjalannya algoritma (graf dinamis), maka untuk tabel boolean kami rasa akan redundan, dan tidak diperlukan untuk diperiksa apakah sebuah node sudah dikunjungi.

d. Antrian(khusus BFS)

Antrian dalam algoritma BFS diimplementasikan dengan struktur data Queue yang disediakan oleh library C#.

3. Contoh Ilustrasi Kasus

Pertama-tama masing-masing directory tersebut diconversi menjadi sebuah node, kemudian ditampung dalam sebuah graf. Adapun untuk *edge* dibuat berdasarkan representasi arah dari satu directory ke directory lain tersebut, dimana directory yang menampung directory tersebut akan dibuat edge berarah dari directory penampung tersebut ke directory di dalamnya, begitu seterusnya sampai ujung directory (tidak ada directory lagi). Berikut adalah salah satu contoh kasus folder crawling, dimana file tujuan yang ingin dicari adalah hasil.pdf.



Gambar 3.3.1 Contoh Ilustrasi Kasus

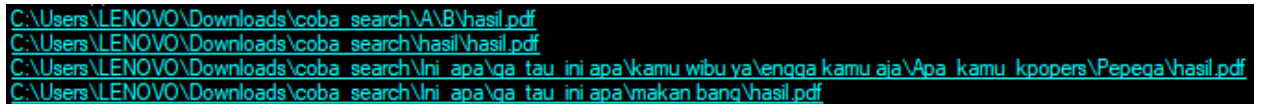
Laporan Tugas Besar 2

IF2211–Kelompok Tukang Lipat

1. Solusi Depth First Search

Dari visualisasi graf di atas, untuk mencari file .pdf dengan penelusuran DFS, penelusuran dilakukan secara mendalam pada suatu directory (sampai tidak ditemukan directory lagi pada masing-masing subdirectory) sesuai dengan Depth First Search. Adapun dari definisi tersebut dapat dideduksikan penelusuran graf yang memenuhi adalah Root -> A -> B -> hasil.pdf (ditemukan) -> hasil -> hasil.pdf (ditemukan) -> Ini_apa -> Apa_in_i -> Laporan Tubes 2.pdf -> ga_tau_in_i apa -> kamu wibu ya -> engga kamu aja -> Apa_kamu_kpopers -> pepega -> hasil.pdf (ditemukan) -> sok_asik_Lu -> makan bang -> hasil.pdf (ditemukan)

Sehingga sesuai dengan jalurnya dari root, diperoleh path solusi sebagai berikut.



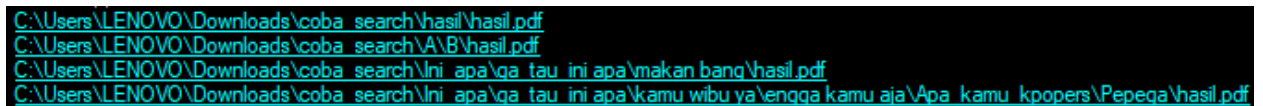
```
C:\Users\LENOVO\Downloads\coba_search\A\B\hasil.pdf  
C:\Users\LENOVO\Downloads\coba_search\hasil\hasil.pdf  
C:\Users\LENOVO\Downloads\coba_search\Ini_apa\ga_tau_in_i_apa\kamu_wibu_ya\engga_kamu_aja\Apa_kamu_kpopers\Pepega\hasil.pdf  
C:\Users\LENOVO\Downloads\coba_search\Ini_apa\ga_tau_in_i_apa\makan_bang\hasil.pdf
```

Gambar 3.3.1.1 Path Solusi DFS

2. Solusi Breadth First Search

Dari visualisasi graf di atas, untuk mencari file .pdf dengan penelusuran BFS, penelusuran secara melebar pada suatu directory sesuai dengan Breadth First Search yaitu menelusuri tetangga-tetangga dari satu directory sampai selesai dahulu kemudian baru ke tetangga dari tetangga tersebut ditelusuri, berikut seterusnya. Adapun dari definisi tersebut dapat dideduksikan penelusuran graf yang memenuhi adalah Root -> A -> hasil -> Ini_apa -> B -> hasil.pdf (ditemukan) -> Apa_in_i -> ga_tau_in_i apa -> hasil.pdf (ditemukan) -> Laporan tubes 2.pdf -> kamu wibu ya -> makan bang -> engga kamu aja -> hasil.pdf (ditemukan) -> Apa_kamu_kpopers -> Pepega -> sok_asik_Lu -> hasil.pdf (ditemukan).

Sehingga sesuai dengan jalurnya dari root, diperoleh path solusi sebagai berikut.



```
C:\Users\LENOVO\Downloads\coba_search\hasil\hasil.pdf  
C:\Users\LENOVO\Downloads\coba_search\A\B\hasil.pdf  
C:\Users\LENOVO\Downloads\coba_search\Ini_apa\ga_tau_in_i_apa\makan_bang\hasil.pdf  
C:\Users\LENOVO\Downloads\coba_search\Ini_apa\ga_tau_in_i_apa\kamu_wibu_ya\engga_kamu_aja\Apa_kamu_kpopers\Pepega\hasil.pdf
```

Gambar 3.3.2.1 Path Solusi BFS

BAB IV

IMPLEMENTASI DAN PENGUJIAN

1. Pseudocode Program Utama

1.1. BFS

procedure searchFilePathBFS(string rootDir, string filename, bool findAll)

KAMUS

q: Queue<string>
// Node merupakan tipe bawaan library MSGSL
nodeQueue: Queue<Node>
directories: List<string>
currentParentNode: Node
dir: string

function directoryValid(string dirName) -> boolean yang menyatakan jika directory dengan nama dirName masih bisa ditelusuri

function directoryHasFiles(string dirName) -> boolean yang menyatakan jika directory mempunyai file

function fileFound(string: dirName, FileName) -> boolean yang menyatakan apakah sebuah file dengan nama filename berada di sebuah directory bernama dirName

ALGORITMA

q.enqueue(rootDir)
directoryR ← rootDir

nodeQueue.enqueue(GraphFile.R)
currentParentNode.setColorRed()

while (findAll **or** this.solutionPath.isEmpty()) **and** q.Count > 0
do

 directoryR ← q.Dequeue()
 currentParentNode = nodeQueue.Dequeue()
 Directories ← GetDirectories(directoryR)

foreach directory **in** directories **do**
 GraphFile.AddEdgeBlack(currentParentNode, directory)

```
Files ← GetFiles(directoryR)
foreach file in Files do
    GraphFile.AddEdgeBlack(currentParentNode, file)

foreach file in Files do
if GetFileName(file) = filename then
    this.setSolution(file)
    GraphFile.TurnBlue(PathToFile(file))

    if not findAll then
        break
else
    GraphFile.TurnRed(PathToFile(file))

foreach directory in directories
    Node N = redEdge(currentParentNode, directory)
    if isdirectoryValid(directory) then
        q.Enqueue(directory)
        nodeQueue.Enqueue(N)
```

1.2. DFS

procedure searchFilePathDFS(string rootDir, string filename,
bool Node startNode, bool findAll)

KAMUS

-

ALGORITMA

```
if (not findAll and solutionPath.Count > 0) then
    return;
```

```
else
```

```
    files ← null
    subDirectories ← null
    root ← new DirectoryInfo()
```

```
if startNode = null then
    startNode = fileGraph.R
```

```
try
```

```
    files ← GetFiles(rootDir)
```

```
catch(Unauthorized access e)
```

```
    log.Add(e.Message)
```

```
// Kalo file tidak null
```

```
    if (files != null)
```

Laporan Tugas Besar 2

IF2211–Kelompok Tukang Lipat

```
// Ubah node menjadi warna merah jika warna awalnya bukan
merah

        if (StartNode.Attr.Color !=
Microsoft.Msagl.Drawing.Color.Blue)

            StartNode.Attr.Color =
Microsoft.Msagl.Drawing.Color.Red;

        // Tambahkan node semua folder start directory
SubDirecs = root.GetDirectories();

        // Reverse
Array.Reverse(SubDirecs);

        foreach (System.IO.DirectoryInfo dirInfo in
SubDirecs)

            Microsoft.Msagl.Drawing.Node subdirectory =
fileGraph.AddEdgeBlack(StartNode, dirInfo.Name);

        Array.Reverse(SubDirecs);

        // Tambahkan node semua file start directory
Array.Reverse(files);
        foreach (System.IO.FileInfo file in files)

            Microsoft.Msagl.Drawing.Node subdirectory =
fileGraph.AddEdgeBlack(StartNode, file.Name);

        Array.Reverse(files);

        // Tampilkan pohon
fileGraph.VisualizeGraph(viewer, Delay);

        foreach (System.IO.FileInfo fi in files)

            // Cek apakah file merupakan file yang ingin
dicari

            if (SearchFile.Equals(fi.Name))

                this.addSolution(fi.FullName);
```

```
fileGraph.TurnBlue(fileGraph.dirToList(fi.FullName));

// Show pohon
fileGraph.VisualizeGraph(viewer, Delay);

if (!findAll)

    break;
else

    Microsoft.Msagl.Drawing.Node N =
fileGraph.ColorEdgeRed(StartNode, fi.Name);
    Graph.ColorNodeRed(N);

// Show pohon
fileGraph.VisualizeGraph(viewer, Delay);

// Continue DFS
if (this.solutionPath.Count == 0 || findAll)

    foreach (System.IO.DirectoryInfo DirectoryInfo
in SubDirecs)

        Microsoft.Msagl.Drawing.Node NextNode =
fileGraph.ColorEdgeRed(StartNode, DirectoryInfo.Name);
        searchFilePathDFS(DirectoryInfo.FullName,
SearchFile, NextNode, Delay, findAll);
```

1.3. Graph

```
// Make Graph
procedure Graph(string rootDirectory)
KAMUS
-
ALGORITMA
setRoot(rootDirectory);

// Set Root
procedure setRoot(string rootDirectory)
KAMUS
-
ALGORITMA
graph = new Microsoft.Msagl.Drawing.Graph();
```

Laporan Tugas Besar 2

IF2211–Kelompok Tukang Lipat

```
R = new
Microsoft.Msagl.Drawing.Node(graph.NodeCount.ToString());
R.LabelText = rootDirectory;
graph.AddNode(R);

// turn node A to color red
procedure ColorNodeRed(Microsoft.Msagl.Drawing.Node X)
KAMUS
    -
ALGORITMA
X.Attr.Color = Microsoft.Msagl.Drawing.Color.Red;

// turn node A to color blue
procedure ColorNodeBlue(Microsoft.Msagl.Drawing.Node X)
KAMUS
    -
ALGORITMA
X.Attr.Color = Microsoft.Msagl.Drawing.Color.Blue;

// Turn edge(A,B) to color blue, A and B must be conected
Microsoft.Msagl.Drawing.Node
ColorEdgeBlue(Microsoft.Msagl.Drawing.Node A, String B)
KAMUS
    -
ALGORITMA
Microsoft.Msagl.Drawing.Node N = null;
var el = A.OutEdges.ToArray();
foreach (var e in el)
    if (e.TargetNode.LabelText.Equals(B))
        e.Attr.Color = Microsoft.Msagl.Drawing.Color.Blue;
        N = e.TargetNode;
return N;

// Turn edge(A,B) to color red, A and B must be conected
Microsoft.Msagl.Drawing.Node
ColorEdgeRed(Microsoft.Msagl.Drawing.Node A, String B)
KAMUS
    -
ALGORITMA
Microsoft.Msagl.Drawing.Node N = null;
var el = A.OutEdges.ToArray();
foreach (var e in el)
    if (e.TargetNode.LabelText.Equals(B) && e.Attr.Color !=
Microsoft.Msagl.Drawing.Color.Blue)
        e.Attr.Color = Microsoft.Msagl.Drawing.Color.Red;
```


Laporan Tugas Besar 2

IF2211–Kelompok Tukang Lipat

```
        N = e.TargetNode;
    return N;

// Turn all node root to node goal to color blue. Following the
// directory path in an array of string (node label)
procedure TurnBlue(List<string> L)
KAMUS
    -
ALGORITMA
    ColorNodeBlue(R);
    Microsoft.Msagl.Drawing.Node N = R;
    foreach (String name in L)
        N = ColorEdgeBlue(N, name);
        ColorNodeBlue(N);

// Add new node B and add egde(A,B) with black color
Microsoft.Msagl.Drawing.Node
AddEdgeBlack(Microsoft.Msagl.Drawing.Node A, String B)
KAMUS
    -
ALGORITMA
    Microsoft.Msagl.Drawing.Node N = new
    Microsoft.Msagl.Drawing.Node(graph.NodeCount.ToString());
    N.LabelText = B;
    graph.AddNode(N);
    graph.AddEdge(A.Id, N.Id).Attr.Color =
    Microsoft.Msagl.Drawing.Color.Black;
    return N;

// Convert directory ke List
List<string> dirToList(string directory)
KAMUS
    -
ALGORITMA
    List<string> list = new List<string>();
    while(directory != this.R.Label.Text)
        list.Insert(0, System.IO.Path.GetFileName(directory));
        directory =
        System.IO.Directory.GetParent(directory).FullName;
    return list;

// Visualize GRAPH
procedure VisualizeGraph(Microsoft.Msagl.GraphViewerGdi.GViewer
viewer, int delay)
KAMUS
```

ALGORITMA

```
viewer.Graph = graph;
System.Windows.Forms.Application.DoEvents();
System.Threading.Thread.Sleep(delay);
```

2. Struktur Data

Kelas BFS
Deskripsi Kelas: Implementasi pencarian file dengan penelusuran BFS
Atribut Kelas: <ol style="list-style-type: none"> 1. fileGraph: graph yang menggambarkan path yang sudah ditelusuri 2. solutionPath: array string berisi path-path file yang dicari 3. viewer: visualisasi dari fileGraph
Method Kelas: <ol style="list-style-type: none"> 1. searchFilePathBFS: mencari file dengan metode BFS 2. directoryValid: memeriksa apakah sebuah directory masih bisa ditelusuri 3. fileFound: memeriksa apakah file yang dicari sudah ditemukan

Kelas DFS
Deskripsi Kelas: Implementasi pencarian file dengan penelusuran DFS
Atribut Kelas: <ol style="list-style-type: none"> 1. fileGraph: graph yang menggambarkan path yang sudah ditelusuri 2. log: kumpulan string yang berisi path yang tidak diizinkan aksesnya 3. solutionPath: array string berisi path-path file yang dicari 4. viewer: visualisasi dari fileGraph
Method Kelas: <ol style="list-style-type: none"> 1. DFS : konstruktor dari kelas DFS 2. getSolutionPath: mengembalikan list solutionPath 3. addSolution: menambahkan sebuah path ke dalam list solutionPath 4. searchFilePathDFS: mencari file dengan metode DFS dan disimpan

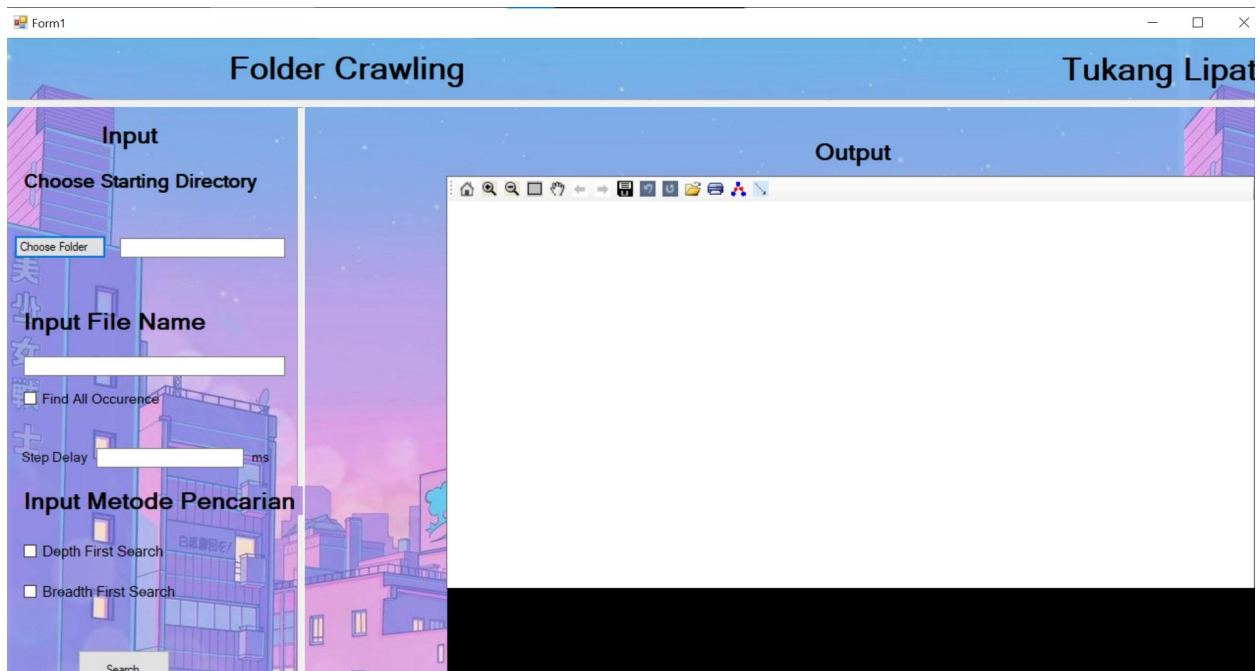
Kelas Graph
Deskripsi Kelas:

Representasi Graf dan visualisasi graf
Atribut Kelas: <ol style="list-style-type: none">1. Graph : Atribut yang menyimpan graph dalam penelusuran directory2. Node : Atribut yang menyimpan node/directory
Method Kelas: <ol style="list-style-type: none">1. Graph : Konstruktor dari kelas graph2. setRoot : Mengeset Sebuah root dari suatu directory3. ColorNodeRed : Mewarnai suatu Node menjadi merah4. ColorNodeBlue : Mewarnai suatu Node menjadi biru5. ColorEdgeBlue : Mewarnai suatu Edge menjadi biru6. ColorEdgeRed : Mewarnai suatu Edge menjadi merah7. TurnBlue : Mewarnai suatu edge dan node pada suatu jalur penelusuran file yang ingin dicari menjadi biru8. AddEdgeBlack : Memasukkan edge dengan warna default (belum ditelusuri) berwarna hitam9. dirToList : Mengonversi kumpulan directory menjadi list of string10. VisualizeGraph : Memvisualisasikan graph

3. Tata Cara Penggunaan Program

3.3.1 Antarmuka Program

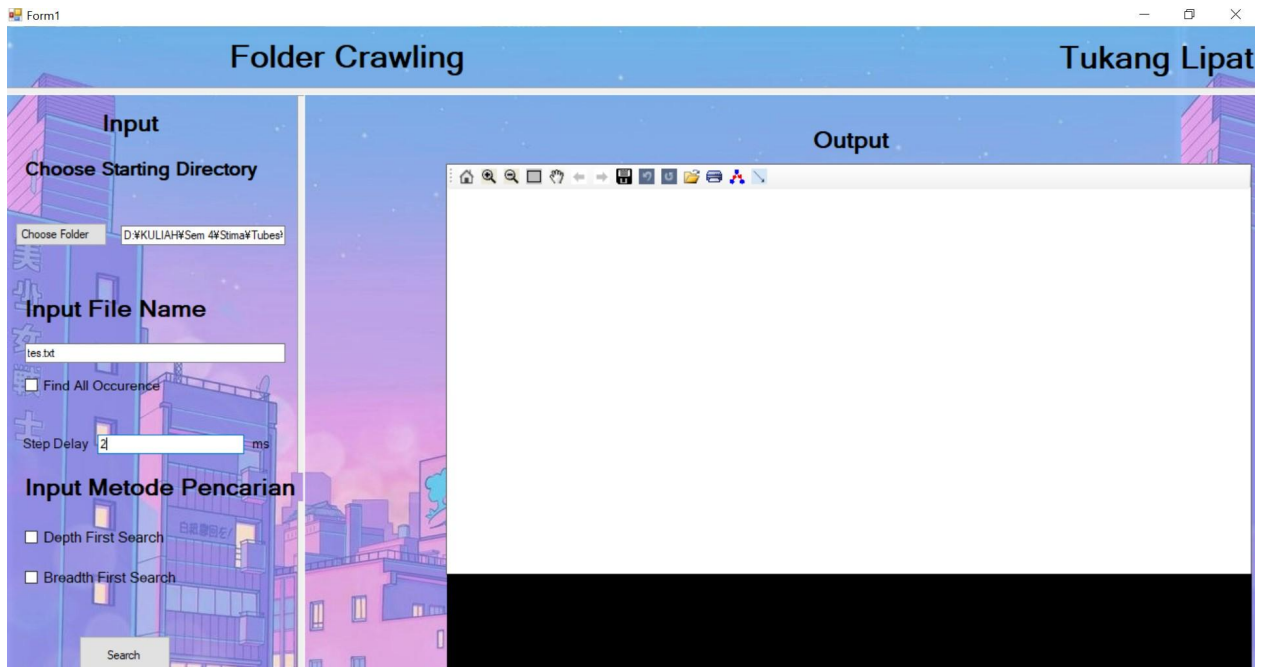
Untuk menjalankan program, jalankan file Folder_Crawling.exe yang terdapat dalam file bin/Debug. Setelah program berhasil dijalankan, program akan menampilkan tampilan sebagai berikut.



Gambar 3.3.1 Antarmuka Aplikasi

3.3.2 Proses Input

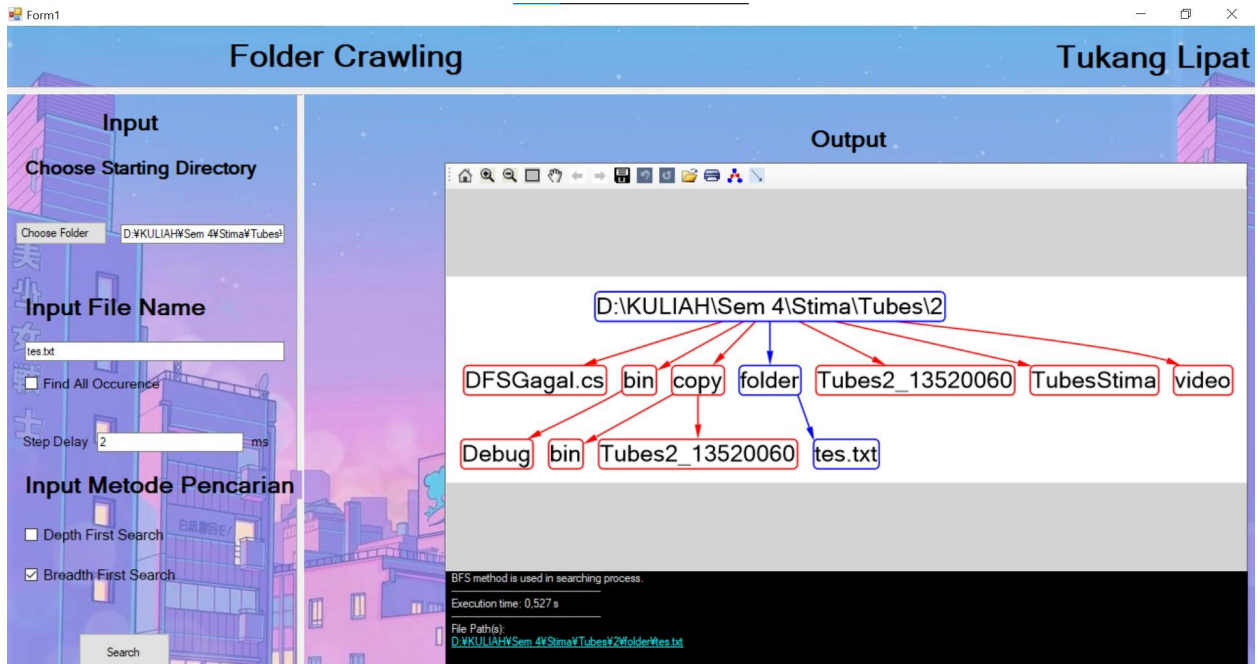
Untuk input awal, tekan tombol “choose folder” untuk memilih folder yang dijadikan sebagai root/starting folder. Kemudian isi nama file yang ingin dicari pada form di bawah tulisan “Input File Name”. Isi juga step delay pada form di sebelah tulisan “Step Delay” untuk menentukan seberapa lama jeda setiap langkah.



Gambar 3.3.2 Pengisian input awal

3.3.3 Memilih Jenis Penelusuran

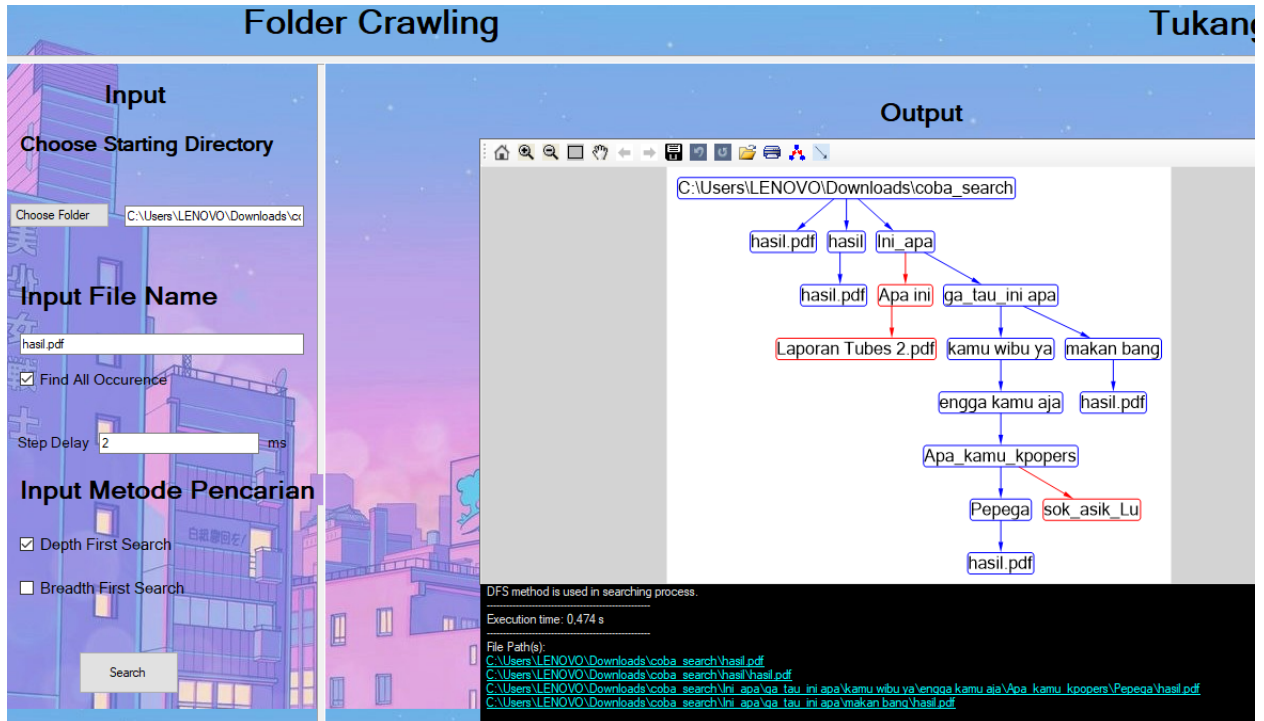
Untuk memilih jenis penelusuran yang akan dilakukan, tekan checklist di sebelah tulisan “Depht First Search” atau “Breadth First Search”. Setelah memilih jenis penelusuran, tekan tombol “Search” untuk memulai proses pencarian. Hasil pencarian akan divisualisasikan dalam bentuk graf pada bagian output program dan hyperlink akan ditampilkan di bawahnya.



Gambar 3.3.3 Pemilihan jenis penelusuran

3.3.4 Menggunakan fitur FindAll

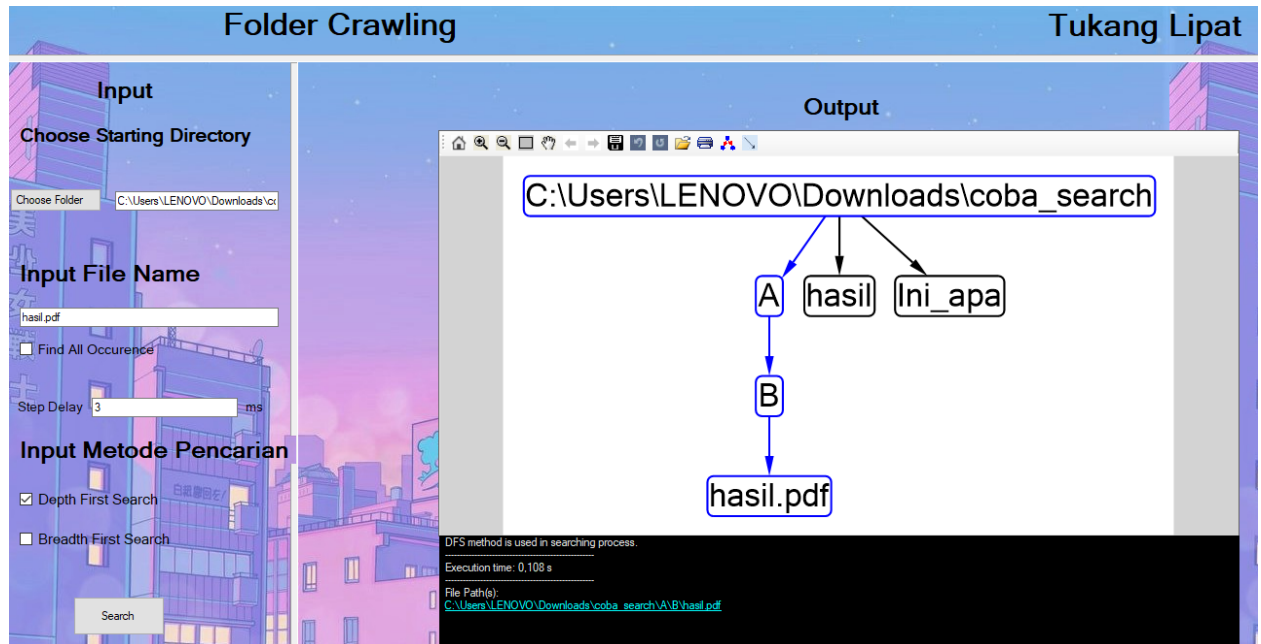
Untuk menggunakan fitur FindAll, setelah anda melakukan langkah 3.3.1 hingga 3.3.3, tekan checklist di sebelah tulisan “Find All Occurence”. Kemudian, tekan tombol “Search” untuk memulai proses pencarian. Hasil pencarian akan divisualisasikan dalam bentuk graf pada bagian output program dan seluruh hyperlink akan ditampilkan di bawahnya.



Gambar 3.3.4 Penggunaan fitur FindAll

4. Hasil Pengujian dan Analisis

Input : C:\Users\LENOVO\Downloads\coba_search
File yang ingin dicari : hasil.pdf
Step Delay : 3 ms
Find All Occurence : No
Metode Pencarian : DFS



Gambar 4.1 Hasil Uji 1

Analisis :

Dari visualisasi graf di atas, untuk mencari file .pdf dengan penelusuran DFS, penelusuran secara mendalam pada suatu directory (sampai tidak ditemukan directory lagi pada masing-masing subdirectory) sesuai dengan Depth First Search, sampai ditemukan satu file saja, namun jika tidak ditemukan, maka pasti seluruh node ditelusuri. Dengan metode DFS, adapun penelusurannya sebagai berikut.

Root -> A -> B -> hasil.pdf (ditemukan).

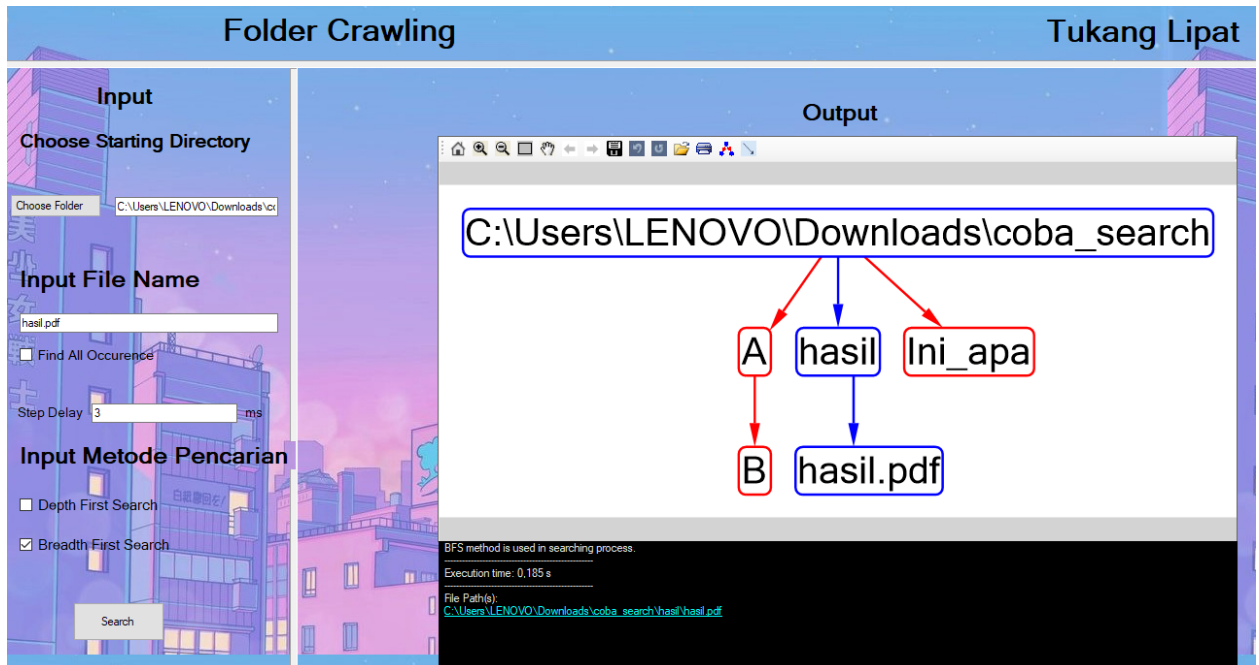
Input : C:\Users\LENOVO\Downloads\coba_search

File yang ingin dicari : hasil.pdf

Step Delay : 3 ms

Find All Occurrence : No

Metode Pencarian : BFS



Gambar 4.2 Hasil uji 2

Analisis :

Dari visualisasi graf di atas, untuk mencari file .pdf dengan penelusuran BFS, penelusuran secara melebar pada suatu directory sesuai dengan Breadth First Search, sampai ditemukan satu file saja, namun jika tidak ditemukan, maka pasti seluruh node ditelusuri. Dengan metode BFS, adapun penelusurannya sebagai berikut.

Root -> A -> hasil -> Ini_apa -> B -> hasil.pdf (ditemukan).

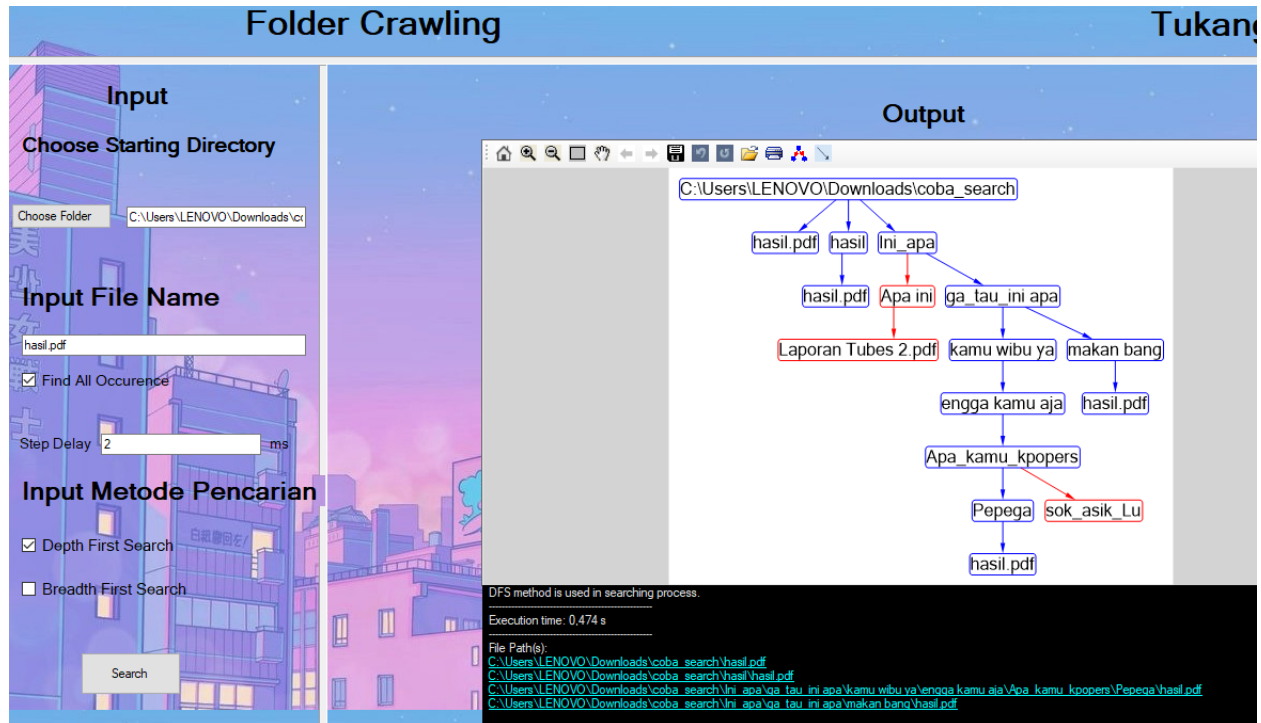
Input : C:\Users\LENOVO\Downloads\coba_search

File yang ingin dicari : hasil.pdf

Step Delay : 2 ms

Find All Occurrence : Yes

Metode Pencarian : DFS



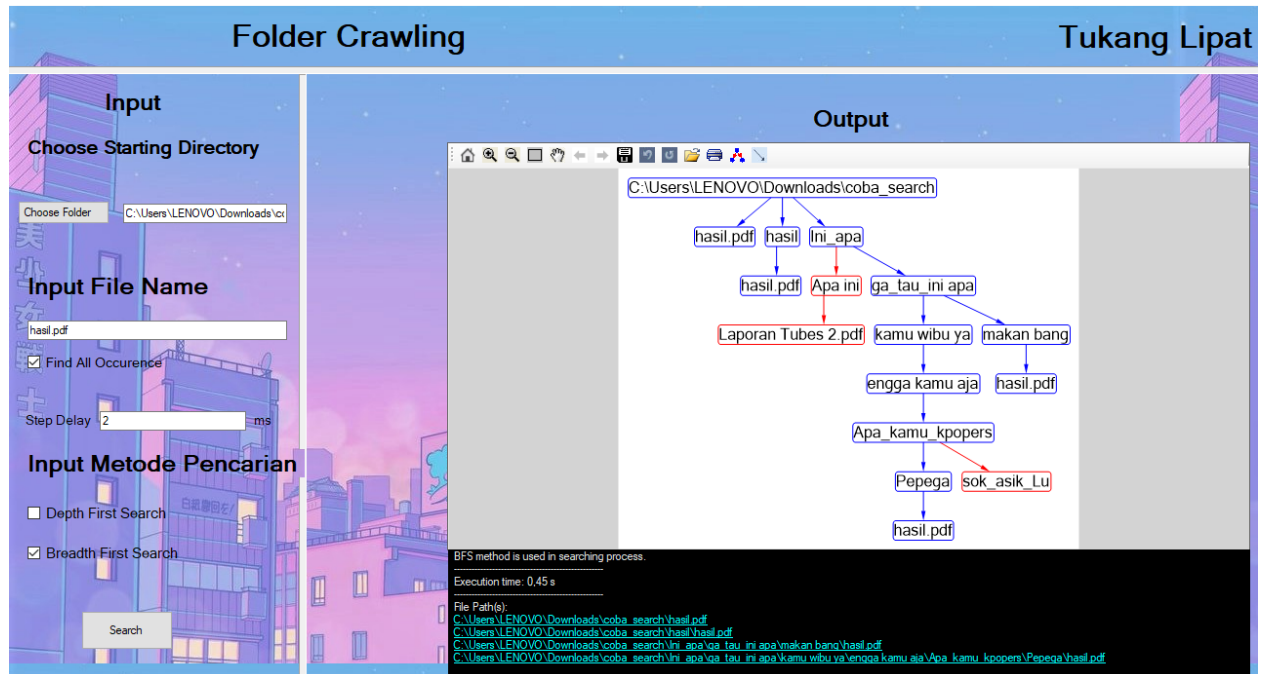
Gambar 4.3 Hasil Uji 3

Analisis :

Dari visualisasi graf di atas, untuk mencari file .pdf dengan penelusuran DFS, penelusuran secara mendalam pada suatu directory (sampai tidak ditemukan directory lagi pada masing-masing subdirectory) sesuai dengan Depth First Search, sampai semua node ditelusuri. Adapun urutan penelusurannya itu sebagai berikut.

Root -> hasil.pdf (ditemukan) -> hasil -> hasil.pdf (ditemukan) -> Ini_apa -> Apa ini -> Laporan Tubes 2.pdf (tidak ditemukan) -> ga_tau_ini apa -> kamu wibu ya -> engga kamu aja -> Apa_kamu_kpopers -> Pepega -> hasil.pdf (ditemukan) -> sok_asik_Lu (tidak ditemukan) -> makan bang -> hasil.pdf (ditemukan).

Input : C:\Users\LENOVO\Downloads\coba_search
File yang ingin dicari : hasil.pdf
Step Delay : 2 ms
Find All Occurence : Yes
Metode Pencarian : BFS



Gambar 4.4 Hasil Uji 4

Analisis :

Dari visualisasi graf di atas, untuk mencari file .pdf dengan penelusuran BFS, penelusuran secara melebar pada suatu directory sesuai dengan Breadth First Search, sampai semua node ditelusuri. Adapun urutan penelusurannya itu sebagai berikut.

Root -> hasil.pdf (ditemukan) -> hasil -> Ini_apa -> hasil.pdf (ditemukan) -> Apa ini -> ga_tau ini apa -> Laporan Tubes 2.pdf -> kamu wibu ya -> makan bang -> engga kamu aja -> hasil.pdf (ditemukan) -> Apa_kamu_kpopers -> Pepega -> sok_asik_Lu -> hasil.pdf (ditemukan).

5. Analisis Desain DFS dan BFS

1. BFS

Kelebihan BFS : Tidak akan menemukan jalan buntu. Jika ada satu solusi maka BFS akan menemukannya. Jika ada lebih dari satu solusi, maka solusi minimum akan ditemukan (solusi terbaik).

Kelemahan BFS : Membutuhkan memori cukup besar, karena menyimpan semua node dalam satu pohon. Selain itu BFS membutuhkan waktu yang cukup lama, karena akan menguji tiap n level untuk menemukan atau mendapatkan solusi pada level yang ke-(n-1)

2. DFS

Kelebihan DFS : Tidak membutuhkan memori yang cukup besar, karena node yang pernah dibangkitkan tidak perlu disimpan dalam queue seperti BFS. Hal ini diakibatkan DFS yang

Laporan Tugas Besar 2

IF2211–Kelompok Tukang Lipat

biasanya diimplementasikan secara rekursif. Algoritma DFS juga lebih baik apabila file yang dicari terdapat di bawah folder bagian kiri (yang pertama dibangkitkan).

Kelemahan DFS : Apabila solusi terdapat di bawah folder yang paling kanan(dibangkitkan terakhir), algoritma DFS akan mencapai solusi secara lebih lama karena harus menelusuri seluruh folder bagian kiri terlebih dahulu. Algoritma ini juga tidak cocok untuk menemukan file terdekat dari root folder.

BAB V

KESIMPULAN DAN SARAN

1. Kesimpulan

Dari tugas besar IF211 Strategi Algoritma ini, kami berhasil membuat *desktop application* yang dikembangkan menggunakan bahasa C# dengan fitur untuk mencari suatu file yang berada di dalam *root folder* tertentu. Folder ditelusuri dengan memanfaatkan algoritma BFS dan DFS. Selain mengerjakan tugas besar ini juga menambah pemahaman kami terhadap materi di kelas dan pengaplikasiannya di dunia nyata.

2. Saran

Saran yang dapat kami berikan untuk tugas besar 2 IF2211 Strategi Algoritma yang mendatang adalah sebagai berikut.

1. Konsep DFS dan BFS yang kami digunakan masih dapat dikembangkan dan digunakan secara lebih kreatif agar algoritma semakin efisien.
2. Menggunakan *tools* dan *stack* yang lebih universal dan bisa berjalan di berbagai sistem operasi. Hal ini cukup memberatkan salah satu anggota kelompok kami yang menggunakan sistem operasi macOS.

Daftar Pustaka

- Munir, Rinaldi dan Maulidevi, Nur Ulfa. (2021), Breadth/Depth First Search (BFS/DFS) Bagian 1.
Diakses online dari
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf> pada 17
Maret 2021.
- Munir, Rinaldi dan Maulidevi, Nur Ulfa. (2021), Breadth/Depth First Search (BFS/DFS) Bagian 2.
Diakses online dari
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf> pada 17
Maret 2021.

Lampiran

Link Repo : [geraldakbar/Tubes2_13520060 \(github.com\)](https://github.com/geraldakbar/Tubes2_13520060)

Link Video : <https://youtu.be/S5kv2wGVVmo>