

LAPORAN TUGAS BESAR 2

IF2211 STRATEGI ALGORITMA

*Pengaplikasian Algoritma BFS dan DFS
dalam Implementasi Folder Crawling*



Kelompok SearchLikeWereDying

Nadia Mareta Putri Leiden	13520007
Adelline Kania Setiyawan	13520084
Angelica Winasta Sinisuka	13520097

Program Studi Sarjana Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2022

I. Deskripsi Tugas

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.

Meskipun demikian, kita tidak perlu cemas dalam menghadapi persoalan tersebut sekarang. Pasalnya, hampir seluruh sistem operasi sudah menyediakan fitur *search* yang dapat digunakan untuk mencari file yang kita inginkan. Kita cukup memasukkan *query* atau kata kunci pada kotak pencarian, dan komputer akan mencarikan seluruh file pada suatu *starting directory* (hingga seluruh *children*-nya) yang berkorespondensi terhadap *query* yang kita masukkan. Fitur ini diimplementasikan dengan teknik *folder crawling*, di mana mesin komputer akan mulai mencari file yang sesuai dengan *query* mulai dari *starting directory* hingga seluruh *children* dari *starting directory* tersebut sampai satu file pertama/seluruh file ditemukan atau tidak ada file yang ditemukan. Algoritma yang dapat dipilih untuk melakukan *crawling* tersebut pun dapat bermacam-macam dan setiap algoritma akan memiliki teknik dan konsekuensinya sendiri. Oleh karena itu, penting agar komputer memilih algoritma yang tepat sehingga hasil yang diinginkan dapat ditemukan dalam waktu yang singkat.

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*.

II. Landasan Teori

A. Dasar Teori

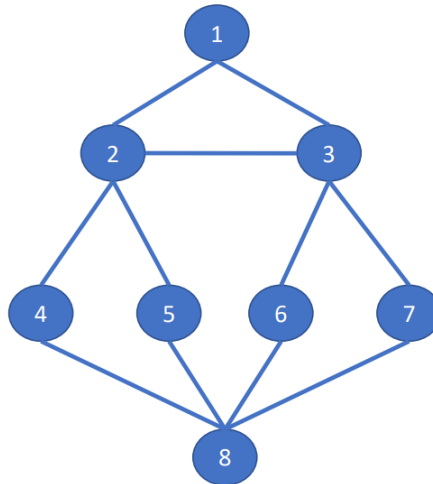
Traversal Graf adalah sebuah algoritma dalam ilmu komputer yang mengunjungi simpul-simpul pada suatu graf secara sistematis. Algoritma traversal graf bertujuan untuk melakukan pencarian solusi dari suatu permasalahan yang direpresentasikan dalam graf. Algoritma pencarian solusi ini dapat melakukan pencarian pada persoalan yang bersifat tanpa informasi tambahan (*uninformed/blind search*), seperti DFS, BFS, *Depth Limited Search*, *Iterative Deepening Search*, *Uniform Cost Search*. Selain itu, algoritma pencarian solusi ini juga dapat melakukan pencarian pada persoalan yang bersifat dengan informasi atau berbasis heuristik (*informed search*), yaitu pencarian solusi yang mengetahui *non-goal state* yang lebih menjanjikan daripada yang lain, seperti *Best First Search*.

Terdapat dua pendekatan dalam proses pencarian solusi menggunakan traversal graf, yaitu graf statis dan dinamis. Pada graf statis, graf sudah terbentuk sebelum proses pencarian dilakukan sehingga graf direpresentasikan sebagai struktur data, sedangkan pada graf dinamis, graf akan terbentuk saat proses pencarian dilakukan atau dalam kata lain graf tidak tersedia sebelum proses pencarian sehingga graf akan dibangun selama pencarian solusi.

Graf statis memiliki jumlah simpul dan sisi yang dapat ditetapkan. Biasanya parameter dari tipe data graf statik adalah kategori, yaitu jenis dari graf. Kemudian node dan sisi yang bersifat opsional untuk dimasukkan. Algoritma yang dapat menelusuri jenis graf ini adalah sebagai berikut.

1. *BFS*

BFS adalah singkatan dari *Breadth First Search* yang artinya adalah pencarian yang sifatnya melebar. Dalam kasus Graf Statis, berarti Graf tersebut sudah terbentuk dari awal, bukan terbentuk saat pencarian berlangsung sehingga pada konteks ini adalah melakukan penelusuran secara melebar pada suatu graf yang sudah tersedia. Menggunakan ilustrasi di bawah ini:



Gambar 2.1 sumber: DFS dan BFS, Rinaldi Munir

Langkah Algoritma yang dilakukan adalah sebagai berikut:

1. Mengunjungi simpul paling atas atau paling utama
2. Setelah itu pencarian dilakukan dengan cara melakukan ke tetangganya terlebih dahulu sebelum melakukan pencarian ke simpul anak

Sehingga secara sederhana urutan penelusuran dilakukan secara berurut dengan urutan: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8

2. *DFS*

DFS merupakan algoritma traversal graf yang mengunjungi setiap simpul secara sistematis. Sifat dari pencarian ini mendalam, berbeda dengan BFS yang pencarian dilakukan melebar atau *per-level*. DFS disebut juga dengan *depth first search*. Algoritma ini diimplementasikan secara rekursif sehingga memiliki kekurangan untuk kena batas *stack size* untuk graf yang besar.

Langkah-langkah pada algoritma DFS adalah sebagai berikut:

1. Kunjungi simpul pertama misalnya x
2. Mengunjungi simpul tetangganya misalnya w
3. Dilakukan DFS dari simpul w

4. Saat mencapai simpul paling ujung yang tidak memiliki tetangga yang belum dikunjungi, akan dilakukan back-tracking ke simpul terakhir yang dikunjungi terakhir sampai ketemu dengan simpul tetangga yang belum dikunjungi.
5. Pencarian berakhir apabila semua simpul telah dikunjungi.

Pada gambar 2.1, urutan penelusuran dilakukan dengan urutan 1-2-3-6-8-4-5-7 ini merupakan penelusuran tanpa backtrackingnya. Jika ditambah dengan backtrackingnya, maka urutan sebagai berikut 1-2-3-6-8-4-8-5-8-7.

B. Penjelasan Singkat Mengenai C# *Desktop Application Development*

Untuk menyelesaikan persoalan pada tugas besar ini, program akan dibuat dalam bentuk Windows Form Application dengan bahasa C#. Kakas yang digunakan untuk tugas besar ini adalah Visual Studio menggunakan .NET Framework.

III. Analisis Pemecahan Masalah

A. Langkah-Langkah Pemecahan Masalah

Pemecahan masalah dimulai dengan merepresentasikan input *path* folder beserta isinya dalam sebuah struktur data *tree*. Hal ini dilakukan untuk mempermudah proses pencarian menggunakan algoritma BFS dan DFS karena diperlukan sebuah graf yang statis.

Pada langkah selanjutnya, perlu ditetapkan goal state sebelum penelusuran. Goal state yang kami ambil ialah file yang ingin dicari oleh pengguna. Penelusuran dilakukan dari tree yang dibentuk di awal. Jenis penelusuran yang diambil tergantung dari pengguna. Jika pengguna memilih penelusuran dengan metode DFS, diperlukan tanda atau setiap node yang sudah dikunjungi. Sehingga kami memasang atribut *visited* pada setiap node untuk menentukan jika sudah dikunjungi atau belum. Penelusuran pertama kami tentukan dari *directory* yang dipilih oleh pengguna. Dalam kasus Find All Occurrence, pencarian dilakukan buat seluruh node. Pendekatan dilakukan dengan mengunjungi semua *node child* sampai tidak ada lagi kemudian melakukan *backtracking* yang akan mencari child node yang belum –di-visit. Jika sudah, maka dilakukan pembangkitan pohon berdasarkan solusi yang didapat. Pada kasus Non – Find Occurrence, pendekatan yang dilakukan sama seperti kasus All Occurrence, tetapi akan berhenti ketika mendapatkan solusi dan mereturn

sesuatu. Kami menggunakan *string* untuk mereturn hasilnya jika ketemu dan null jika tidak ketemu.

BFS dilakukan dengan cara menyimpan seluruh simpul anak dalam bentuk *List of Directory Tree*, yang akan ditelusuri secara berurut. Dalam kasus *Find All Occurrence*, maka pencarian hanya akan berhenti jika jumlah *List of Directory Tree* kosong atau tidak ada lagi simpul anak yang bisa dibangkitkan dari setiap simpul. Sedangkan pada kasus *Non – Find All Occurrence*, pencarian akan berhenti jika sudah ditemukan *path* terdekat yang mungkin.

Setiap melakukan penelusuran hingga mencapai suatu daun pada pohon, daun hingga akar pohon tersebut akan dibuatkan struktur data baru berupa graf yang memanfaatkan MSAGL untuk divisualisasikan. Graf tersebut akan diberi warna sesuai dengan informasi penelusuran simpul-simpulnya, yaitu apakah folder/file sudah diperiksa, folder/file sudah masuk antrian tetapi belum diperiksa, dan folder serta file merupakan rute hasil.

B. Proses *Mapping* Persoalan Menjadi Elemen-Elemen Algoritma BFS dan DFS

Algoritma DFS dan BFS merupakan algoritma pencarian pada graf. Persoalan di atas dapat dikelompokkan menjadi graf statis karena graf terbentuk sebelum proses pencarian di dalam *directory* dilakukan. Elemen yang dimiliki graf adalah simpul dan sisi. Untuk persoalan ini, simpul merupakan nama path setiap file dan memiliki atribut visited berupa tipe data boolean yang digunakan untuk pencarian pada algoritma DFS. Sedangkan algoritma BFS menggunakan tipe data list yang berisi elemen tipe data *tree* untuk queue penelusuran bfs dalam graf. Sisi graf merupakan hubungan dari 2 simpul. Pada kasus ini, sisi merupakan hubungan folder dengan folder atau folder dengan file lain.

C. Contoh Ilustrasi Kasus Lain

Kasus lain yang dapat menggunakan algoritma DFS dan BFS adalah penjelajahan web. Namun berbeda dengan kasus kami, graf yang digunakan adalah graf berarah. Setiap simpul menyatakan halaman *halaman webpage*. Kemudian sisi menyatakan link ke halaman web. Penelusuran web dimulai dari web page awal yang kemudian setiap link

ditelusuri menggunakan algoritma DFS atau BFS sampai setiap webpage tidak mengandung link.

IV. Implementasi dan Pengujian

A. Implementasi Program

1. Main Program

```
string startingPath <- input direktori file dari pengguna
string fileName <- input query nama file yang akan dicari
boolean isAllOccurence <- input apakah mencari semua file query yang ada
boolean isBFS <- input apakah metode pencarian BFS atau DFS

{ Menginisialisasi struktur data tree dengan root berupa startingPath }
DirectoryTree directoryTree = new DirectoryTree(startingPath)

{ Mengakses semua subdirektori dari root }
getAllDirs(directoryTree)

{ Menginisialisasi graph dan graph viewer }
Microsoft.Msagl.GraphViewerGdi.GViewer graphViewer
    = new Microsoft.Msagl.GraphViewerGdi.GViewer()
Microsoft.Msagl.Drawing.Graph graph = Microsoft.Msagl.Drawing.Graph("Folder Crawling")

{ Menginisialisasi dan memulai stopwatch untuk menghitung waktu eksekusi }
System.Diagnostics.Stopwatch watch = System.Diagnostics.Stopwatch.StartNew()
watch.Start()

{ Melakukan pencarian query file }
{ BFS }
if isBFS then
    { Inisialisasi treeList untuk BFS }
    List<DirectoryTree> tree_list = new List<DirectoryTree>()
    if isAllOccurence then
        { Menggunakan prosedur BFS dengan mencari semua kemunculan file }
        useBFSAllOccur(directoryTree, graph, fileName, comboBoxFile,
            textFolderRoute)
    else
```

```

        { Menggunakan prosedur BFS dengan mencari hanya satu kemunculan file }
        useBFS(directoryTree, graph, fileName, comboxFile, textFolderRoute)

    { DFS }

    else

        if isAllOccurence then
            { Menggunakan prosedur DFS dengan mencari semua kemunculan file }
            useDFSAllOccur(directoryTree,      graph,      fileName,      comboxFile,
            textFolderRoute)

        else

            { Menggunakan prosedur DFS dengan mencari hanya satu kemunculan file }
            useDFS(directoryTree, graph, fileName, comboxFile, textFolderRoute)

    { Memberhentikan waktu stopwatch }

    watch.Stop()

    { Menampilkan output graph dan waktu eksekusi }

    output(watch.ElapsedMilliseconds)

    output(graph)

```

2. BFS

```

public void useBFSAllOccur(tree: DirectoryTree, graph: Microsoft.Msagl.Drawing.Graph,
searchDir: string, comboBoxFile: ComboBox, textFolderRoute: RichTextBox)

```

```

    List<DirectoryTree> tree_list = new List<DirectoryTree>()

    for(i sebanyak jumlah simpul anak)

        tambahkan simpul anak ke tree_list

    while(simpul anak tidak nol) do

        searchTreeModified(tree_list, searchDir, graph, comboBoxFile, textFolderRoute)

        tree_list = arrayDir(tree_list)

```



```
public void useBFS(tree: DirectoryTree, graph: Microsoft.Msagl.Drawing.Graph, searchDir:
string, comboBoxFile: comboBox, textFolderRoute: RichTextBox)
```

```
List<DirectoryTree> tree_list = new List<DirectoryTree>()

for(i sebanyak jumlah simpul anak)

    tambahkan simpul anak ke tree_list

while(simpul anak tidak nol dan file belum ketemu)do

    if (searchTreeList(tree_list, searchDir, graph, comboBoxFile, textFolderRoute))

        isFound = true

    else

        tree_list = arrayDir(tree_list)
```

```
public List<DirectoryTree> arrayDir (treeArray: List<DirectoryTree>)
```

```
int total_length = 0

int idx = 0

for (i sebanyak simpul anak treeArray)

    menghitung seluruh simpul anak dari List Directory Tree

    menyimpan dalam total_length

List<DirectoryTree> array = new List<DirectoryTree>()

for(i sebanyak simpul anak treeArray)

    DirectoryTree childTree = treeArray[i]

    for(j sebanyak simpul anak dari TreeArray)

        menyimpan simpul anak dalam array DirectoryTree

-> array
```

```
public bool searchTreeList (treeArray: List<DirectoryTree> treeArray, nameDir: string,
graph: Microsoft.Msagl.Drawing.Graph, comboBoxFile: ComboBox, textFolderRoute: RichTextBox)
```

```

bool found = false

int idx = 0

for(i sejumlah simpul anak treeArray)

    string namefile = nama file dari treeArray

    string save_tree_data = alamat treeArray[i]

    mengubah data treeArray[i] menjadi "namefile (countNode)"

    countNode++

    if(not isFound) then

        if(namefile == nameDir) then

            comboBoxFile.Items.Add(save_tree_data)

            mewarnai path dengan warna biru

            isFound = true

            found = true

            idx = i

        else

            output.displayTreeDirs(treeArray[i], graph, "red")

            output.printFolderRoute(namefile, textFolderRoute)

            output.printFolderRoute("\n", textFolderRoute)

        else

            output.displayTreeDirs(treeArray[i], graph, "black")

-> found

public void searchTreeListModified (treeArray: List<DirectoryTree> treeArray, nameDir:
string, graph: Microsoft.Msagl.Drawing.Graph, comboBoxFile: ComboBox, textFolderRoute:
RichTextBox)

    int idx = 0

    for(i sejumlah simpul anak treeArray)

```

```

        string namefile = nama file dari treeArray

        string save_tree_data = alamat treeArray[i]

        mengubah data treeArray[i] menjadi "namefile (countNode)"

        countNode++

        if(namefile == nameDir) then

            comboBoxFile.Items.Add(save_tree_data)

idx = i

        mewarnai path dengan warna biru

    else

        output.displayTreeDirs(treeArray[i], graph, "red")

        output.printFolderRoute(namefile, textFolderRoute)

        output.printFolderRoute("\n", textFolderRoute)

-> found

```

3. DFS

```

{Fungsi untuk memanggil DFS dan kemudian di display}

{global variable}

countNode: integer

got_it : boolean

procedure useDFS (tree : DirectoryTree, graph: Mircrosoft.Msagl.Drawing.Graph,
searchDir:string , comboBoxFile: ComboBox, textFolderRoute: RichTextBox, searchFileName:
string)

{KAMUS LOKAL}

path: string,filename,result

{ALGORITMA}

path ← searchDir

```

```
filename ← searchFileName
```

```
result ← dfsnotall2(tree, path, filename, textFolderRoute, comboBoxFile)
```

```
printTreesNotAll (tree.graph)
```

```
{Fungsi untuk mencari semua occurrences nama file tersebut}
```

```
Procedure dfsini (pohon: DirectoryTree, path: string, filename:string, textFolderRoute:  
RichTextBox, comboBoxFile: ComboBox)
```

```
{KAMUS LOKAL}
```

```
    nameFile, temp:string
```

```
{ALGORITMA}
```

```
    Namefile ← Path.GetFileName(pohon.Data);
```

```
    output.printFolderRoute(namefile, textFolderRoute);
```

```
    output.printFolderRoute("\n", textFolderRoute);
```

```
    if (pohon.visited) then
```

```
        →
```

```
    if (pohon.Data = path and pohon.Level != 0) then
```

```
        solution.Add(pohon)
```

```
        comboBoxFile.Items.Add(pohon.Data)
```

```
    if (pohon.Count > 0 ) {Node yang memiliki child dikunjungi dulu}
```

```
        temp ← pohon.Data + "\" + filename;
```

```
        for j←pohon.Count-1 to j=0 do
```

```
            if (not pohon[j].Visited) then
```

```
                dfsini(pohon[j],temp,filename,textFolderRoute,comboBoxFile)
```

{ mencari satu solusi, masuk ke solution string, if dfsnotall returns true dalam bentuk string maka ada jawaban di solution.}

{jika dfs returns false artinya ngga ada jawaban di solutionnya.}

function dfsnotall2(pohon: DirectoryTree, path: string, filename: string, textFolderRoute: RichTextBox, comboBoxFile: Combobox)

{KAMUS LOKAL}

temp, namefile, found: string

{ALGORITMA}

pohon.SetVisited(true);

temp ← pohon.Data;

namefile ← Path.GetFileName(pohon.Data) {mengambil nama file}

output.printFolderRoute(namefile, textFolderRoute);

output.printFolderRoute("\n", textFolderRoute);

if (pohon.Data = path and pohon.Level != 0) then

 solution.Add(pohon);

 comboBoxFile.Items.Add(pohon.Data);

 ➔ temp;

else

 if (pohon.Count > 0)

 temp ← pohon.Data + "\\ " + filename

 for i←pohon.Count-1 to i=0 do

 if (not pohon[i].Visited) then

 found = dfsnotall2(pohon[i], temp, filename,

 textFolderRoute, comboBoxFile);

 if (not string.IsNullOrEmpty(found)) then

 ➔ found

 ➔ null;

{Mengeluarkan display tree setelah semua simpul dikunjungi}

Procedure printTrees(DirectoryTree pohon, Microsoft.Msagl.Drawing.Graph graph)

{KAMUS LOKAL}

count: integer

{ALGORITMA}

count \leftarrow pohon.Count

countNode \leftarrow countNode + 1

if (count > 0) then

 printingTrees(pohon,graph); {memanggil procedure untuk printing}

 for i \leftarrow pohon.Count-1 to i=0 do

 printTrees(pohon[i], graph);

else

 printingTrees(pohon,graph);

{Mengeluarkan display tree untuk kasus pencarian berhenti setelah mendapat 1 solusi}

Procedure printTreesNotAll(DirectoryTree pohon, Microsoft.Msagl.Drawing.Graph graph)

{KAMUS LOKAL}

count: integer

{ALGORITMA}

if (solution.Count != 0) then

 int count \leftarrow pohon.Count

 countNode \leftarrow countNode+1

 if (count > 0 && not got_it)

 printingTrees(pohon, graph); {mengprint node yg punya child}

 for i \leftarrow pohon.Count-1 to i=0 do

 printTreesNotAll(pohon[i],graph)

 else if (not got_it)

```

        printingTrees(pohon, graph);
    else
        {tidak ketemu solusi, mengeprint setiap node}
        printTrees(pohon,graph)

    {Mendisplay grafik node ujung}

```

Procedure printingTrees (pohon: DirectoryTree, graph: Microsoft.Msagl.Drawing.Graph)

{KAMUS LOKAL}

found : bool

namefile: string

{ALGORITMA}

found \leftarrow false

foreach(var s in solution) do {transverse solusi yang didapat dari penelusuran dfs}

if (pohon.Data = s.Data)

found \leftarrow true;

got_it \leftarrow true;

namafile \leftarrow Path.GetFileName(pohon.Data);

if (countNode != 0) then

pohon.changeData(namafile + " (" + countNode + ")") {mengubah nama node}

if (found) then

output.displayTreeDirs(pohon, graph, "blue")

else if (pohon.Visited) then

output.displayTreeDirs(pohon, graph, "red");

else

output.displayTreeDirs(pohon, graph, "black");

B. Struktur Data dan Spesifikasi Program

Struktur data yang digunakan untuk menyelesaikan persoalan ini adalah struktur data pohon atau *tree* yang merepresentasikan lokasi folder atau file yang direpresentasikan dalam simpul yang merupakan bagian dari suatu lokasi folder utama atau *root*-nya. Setelah dilakukan pencarian menggunakan BFS ataupun DFS, pohon ini akan ditransformasikan menjadi suatu graf berbentuk pohon dengan memanfaatkan kaskas visualisasi MSAGL.

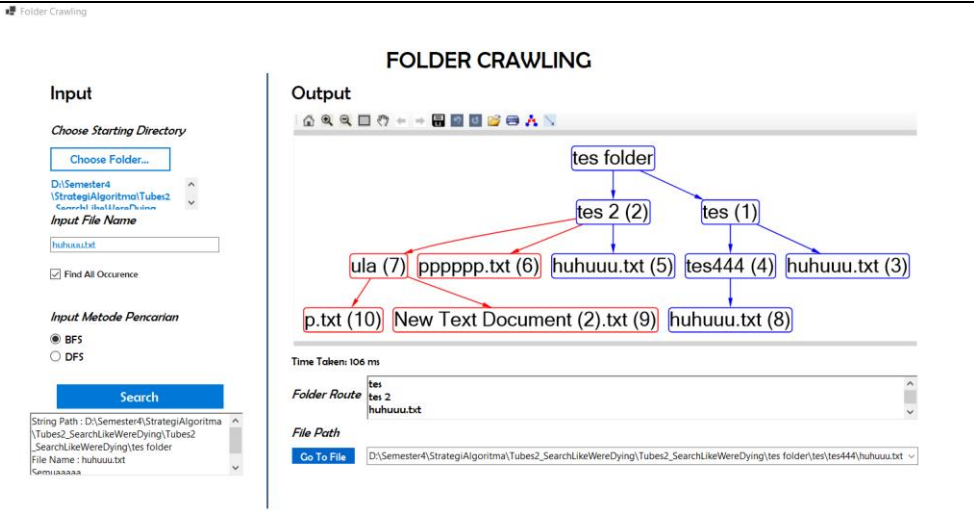
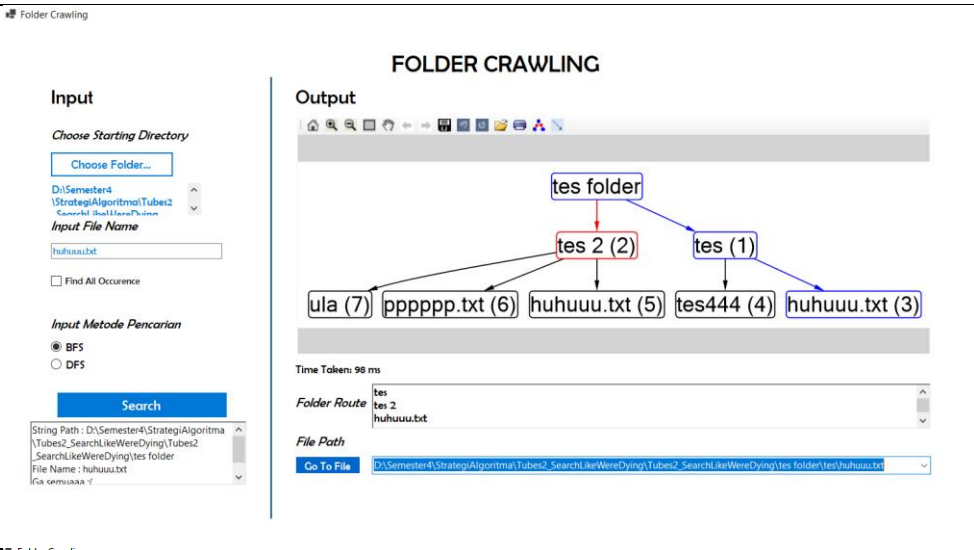
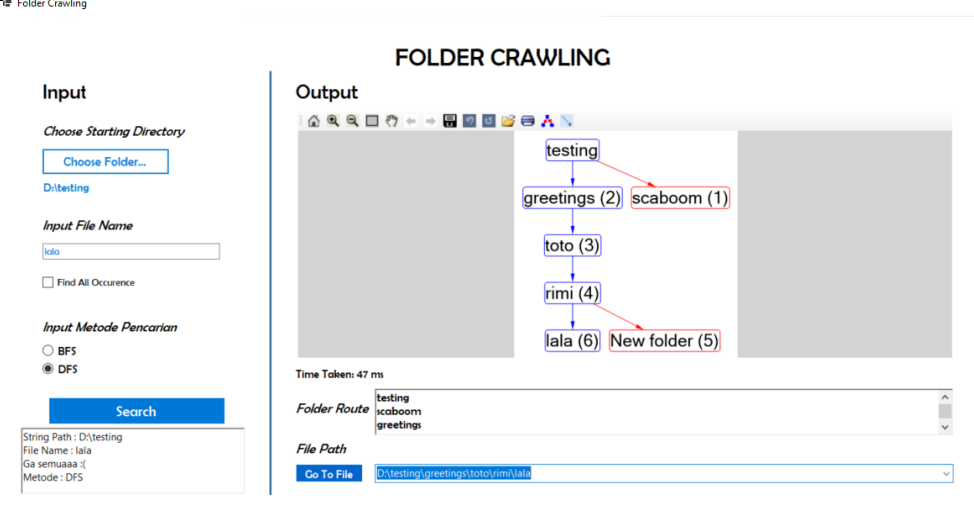
Program dapat menerima input berupa *path* folder dan query berupa nama file. Program juga dapat menerima input berupa metode pencarian apa yang ingin digunakan, BFS atau DFS, serta input pilihan hasil yang ingin ditampilkan, satu hasil saja atau semua hasil. Kemudian, program dapat menampilkan pohon hasil pencarian file tersebut dengan memberikan keterangan folder/file yang sudah diperiksa, folder/file yang sudah masuk antrian tetapi belum diperiksa, dan rute folder serta file yang merupakan rute hasil pertemuan. Terakhir, program dapat menampilkan hasil pencarian berupa rute/path serta durasi waktu algoritma.

C. Tata Cara Penggunaan Program

Penggunaan program dapat dilakukan melalui proses berikut:

1. Membuka file FolderCrawling.sln melalui Visual Studio dan mulai debugging
2. Setelah itu pada starting directory, silahkan memilih directory yang diinginkan
3. Input file name pada kotak kosong di bawahnya sesuai dengan keinginan
4. Memilih input metode pencarian, DFS atau BFS, beserta tentukan apakah ingin melakukan dalam mode “Find All Occurrence” atau tidak
5. Jika folder ditemukan, maka rute menuju *Path* akan ditampilkan pada Folder Route
6. Untuk mengunjungi alamat dimana file disimpan klik “Go To File”, pilih alamat yang diinginkan

D. Hasil Pengujian

Hasil	Keterangan
 <p>The screenshot shows the 'Folder Crawling' application interface. On the left, the 'Input' section includes a 'Choose Starting Directory' button, a file path 'D:\Semester4\StrategiAlgoritma\Tubes2', an 'Input File Name' field with 'huhuuu.txt', a checked 'Find All Occurrence' checkbox, and radio buttons for 'BFS' (selected) and 'DFS'. A 'Search' button is at the bottom. The 'Output' section displays a tree diagram starting from 'tes folder'. 'tes folder' branches into 'tes 2 (2)' and 'tes (1)'. 'tes 2 (2)' branches into 'ula (7)', 'pppppp.txt (6)', and 'huhuuu.txt (5)'. 'tes (1)' branches into 'tes444 (4)' and 'huhuuu.txt (3)'. Further sub-branches are shown for 'ula (7)' and 'pppppp.txt (6)'. Below the diagram, it says 'Time Taken: 106 ms'. The 'Folder Route' field shows 'tes', 'tes 2', and 'huhuuu.txt'. The 'File Path' field shows the full path to 'huhuuu.txt'.</p>	<p>Kasus BFS All Occurrence untuk file berjudul “huhuuu.txt”, seluruh path yang menuju ke file “huhuhu.txt” berhasil diwarnai dengan warna biru. Folder Route menunjukkan seluruh alamat yang mungkin dituju.</p>
 <p>This screenshot is similar to the first one, but the 'Folder Route' field now shows 'tes', 'tes 2', and 'huhuuu.txt'. The tree diagram is the same, but the path from 'tes folder' to 'tes 2 (2)' to 'huhuuu.txt (5)' is highlighted in red, indicating a single route found by BFS.</p>	<p>Kasus BFS Not All Occurrence dapat dilihat bahwa pada folder route muncul satu rute yang dapat dipilih dan menuju ke alamat file yang ditemukan melalui metode BFS.</p>
 <p>The screenshot shows the 'Folder Crawling' application with 'D:\testing' as the starting directory and 'lala' as the input file name. The 'BFS' radio button is selected. The 'Output' section shows a tree diagram starting from 'testing'. 'testing' branches into 'greetings (2)' and 'scaboom (1)'. 'greetings (2)' branches into 'toto (3)', which then branches into 'rimi (4)', which finally branches into 'lala (6)' and 'New folder (5)'. Below the diagram, it says 'Time Taken: 47 ms'. The 'Folder Route' field shows 'testing', 'scaboom', and 'greetings'. The 'File Path' field shows the full path to 'lala'.</p>	<p>Kasus DFS Not All Occurrence untuk folder atau file bernama lala. Pencarian DFS not all occurrence Kembali jika menemukan solusi</p>

Folder Crawling

Input

Choose Starting Directory

Choose Folder...

D:\testing

Input File Name

New folder

Find All Occurrence

☒

Input Metode Pencarian

☐ BFS

☒ DFS

Search

String Path : D:\testing

File Name : New folder

Semuaaaaa

Metode : DFS

FOLDER CRAWLING

Output

```
graph TD
    testing --> biabia_1[biabia (15)]
    testing --> greetings[greetings (2)]
    testing --> scaboom[scaboom (1)]
    greetings --> foto[foto (3)]
    foto --> do[do (17)]
    foto --> new_folder_1[New folder (16)]
    foto --> rrm[rrm (4)]
    do --> lala_1[lala (6)]
    do --> new_folder_2[New folder (5)]
    lala_1 --> lala_2[lala (10)]
    lala_2 --> po[po (7)]
    po --> tinkiwiki_1[tinkiwiki (11)]
    po --> new_folder_3[New folder (9)]
    po --> new_folder_4[New folder (2) (5)]
    tinkiwiki_1 --> tinkiwiki_2[tinkiwiki (12)]
    tinkiwiki_2 --> biabia_2[biabia (15)]
    tinkiwiki_2 --> new_folder_5[New folder (14)]
    tinkiwiki_2 --> tinkiwiki_3[tinkiwiki (13)]
```

Time Taken: 65 ms

Folder Route

testing
scaboom
greetings

File Path

Go To File

D:\testing\greetings\foto\rrm\lala\lala\tinkiwiki\tinkiwiki\New folder

Kasus DFS All Occurrence untuk folder atau file bernama “New folder”. Dalam kasus ini, dapat menelusuri file yang bernama sama tapi level yang berbeda.

Input

Choose Starting Directory

Choose Folder...

D:\Angelica\semester4\stigma\tube2_angel\Tube2_SearchLikeWereDying

Input File Name

e.g. word.pdf

Find All Occurrence

☒

Input Metode Pencarian

☒ BFS

DFS

Search

String Path : D:\Angelica\semester4\stigma\tube2_angel\Tube2_SearchLikeWereDying

File Name : e.g. word.pdf

Semuaaaaa

FOLDER CRAWLING

Output

```
graph TD
    tes_folder[tes folder] --> tes_2[tes 2 (2)]
    tes_folder --> tes_1[tes (1)]
    tes_2 --> ula[ula (7)]
    tes_2 --> pppppp[pppppp.txt (6)]
    tes_2 --> huhuuu_1[huhuuu.txt (5)]
    tes_1 --> tes444[tes444 (4)]
    tes_1 --> huhuuu_2[huhuuu.txt (3)]
    ula --> p_txt[p.txt (10)]
    ula --> new_text_document[New Text Document (2).txt (9)]
    huhuuu_1 --> huhuuu_3[huhuuu.txt (8)]
```

Time Taken: 35 ms

Folder Route

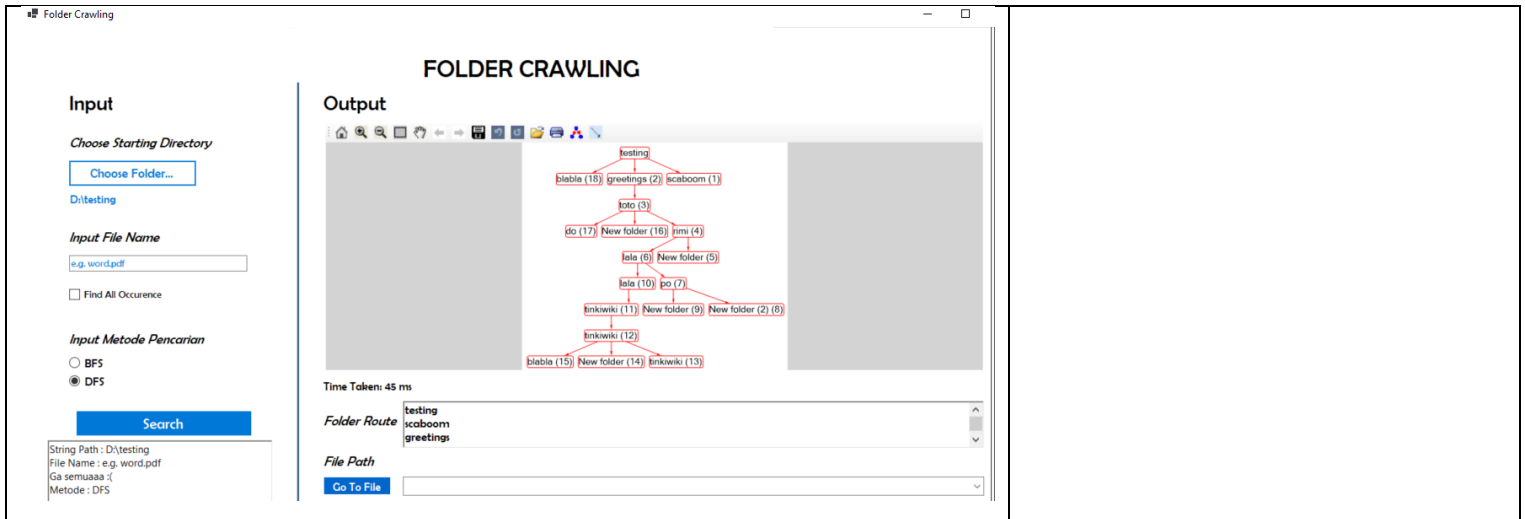
tes
tes 2
huhuuu.txt

File Path

Go To File

Kasus BFS tidak menemukan solusi untuk file Bernama “Iniapa”, tidak ditemukan. Sehingga folder route kosong dan file path kosong.

Kasus DFS tidak menemukan solusi untuk file bernama “e.g word.pdf”



E. Analisis Desain Solusi

Penelusuran mencari file dapat dilakukan dengan menggunakan 2 algoritma, yaitu DFS dan BFS, berdasarkan implementasi test kasus mencari “New folder”, langkah yang perlu diambil jika menggunakan DFS adalah 14, sedangkan pada BFS adalah 17. Sedangkan pada kasus kedua dimana pada folder tes, pencarian “huhuuu.txt” menggunakan BFS memerlukan sebanyak 3 langkah akan tetapi memakan lebih banyak *space memory*, jika dicari menggunakan DFS memerlukan sebanyak 6 langkah dengan penggunaan *space memory* yang lebih sedikit.

Dapat disimpulkan untuk kasus pencarian yang mendalam lebih baik menggunakan algoritma DFS. Namun apabila directory melebar, lebih baik menggunakan strategi BFS karena pencarian file dengan DFS dilakukan melalui tetangga simpul terlebih dahulu. Sehingga dapat disimpulkan kalau untuk kasus yang *target* lebih dekat ke *starting path*, maka BFS adalah strategi algoritma lebih baik, tetapi apabila *target* lebih jauh dari starting path, lebih baik menggunakan strategi algoritma DFS. Dalam implementasinya, BFS menggunakan queue sedangkan DFS menggunakan stack untuk penelusurannya, sehingga untuk algoritma DFS dibatasi oleh *stack size*, sedangkan BFS mengambil *space* yang besar. Kelebihan dari DFS adalah pada kedua kasus, membutuhkan langkah yang lebih banyak dalam mencari file target, sedangkan BFS memerlukan langkah yang lebih sedikit untuk menemukan objek.

V. Kesimpulan dan Saran

Algoritma pencarian graf DFS dan BFS tepat digunakan untuk mencari file dengan kelebihan dan kekurangan masing-masing. Jika efisiensi kapasitas *memory* bukan suatu prioritas, maka dapat digunakan BFS sebagai metode pencarian file. Akan tetapi, jika lebih memprioritaskan efisiensi kapasitas *memory* DFS akan lebih cocok untuk digunakan dalam proses *folder crawling*.

VI. Daftar Pustaka

[1] Slide Kuliah IF2211 Strategi Algoritma – Algoritma BFS-DFS Bagian 1

[2] Slide Kuliah IF2211 Strategi Algoritma – Algoritma BFS-DFS Bagian 2

VII. Lampiran

Link Repository Github: https://github.com/adellinekania/Tubes2_SearchLikeWereDying

Link Video Youtube: <https://youtu.be/2RhH1dsxIFc>