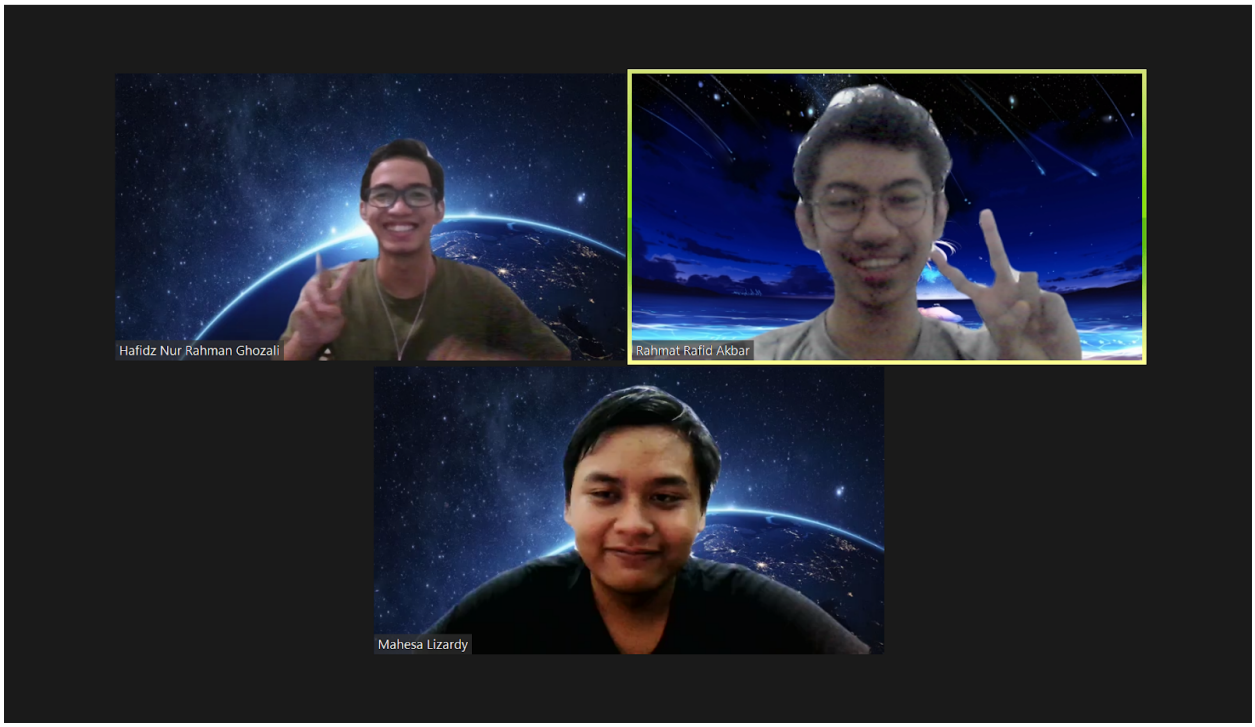
Laporan Tugas Besar II
IF2211 Strategi Algoritma
Semester II Tahun 2021/2022
Folder Crawling

Anggota:

Rahmat Rafid Akbar	13520090
Mahesa Lizardy	13520116
Hafidz Nur Rahman Ghozali	13520117



Institut Teknologi Bandung
2022

Daftar Isi

Daftar Isi **1**

Bab I
Deskripsi Tugas **3**

Bab II
Landasan Teori **6**
2.1 Traversal Graf 6
2.2 BFS 6
2.3 DFS 7
2.4 C# Desktop Application Development 7

Bab III
Analisis Pemecahan Masalah **8**
3.1 Pemetaan Masalah 8
3.2 Langkah-langkah Penyelesaian Masalah 8
3.2.1 Penyelesaian Masalah dengan Algoritma BFS 8
3.2.2 Penyelesaian Masalah dengan Algoritma DFS 8
3.3 Ilustrasi Kasus Lain dalam Folder Crawling 9

Bab IV
Implementasi dan Pengujian **11**
4.1 Implementasi Algoritma pada Folder Crawling 11
4.1.1 Algoritma pada BFS 11
4.1.1.1 Fungsi Solve 11
4.1.2 Algoritma pada DFS 12
4.1.2.1 Fungsi DFSearch 12
4.1.3 Algoritma pada Form 13
4.1.3.1 Fungsi Form1 13
4.1.3.2 Fungsi buttonChooseFolder_Click 13
4.1.3.3 Fungsi buttonSearch_Click 14
4.1.3.4 Fungsi wait 17
4.1.3.5 Fungsi listLinkPath_SelectedIndexChanged 17
4.1.4 Algoritma pada Form.designer 18
4.1.4.1 Fungsi InitializeComponent 18
4.2 Struktur Data pada Program 19
4.2.1 Form1.designer.cs 20
4.2.2 Form1.cs 20
4.2.3 BFS.cs 20
4.2.4 DFS.cs 21
4.3 Penggunaan Program 21

Gambar 4.3 Tampilan GUI dan fitur-fiturnya 21
4.4 Analisis Hasil Pengujian 22
4.4.1 DFS 22
4.4.2 DFS find all occurence 22
4.4.3 BFS 23
4.4.4 BFS find all occurence 24
4.4.5 Pencarian file tidak ditemukan 24
4.5 Analisis Solusi Program 25

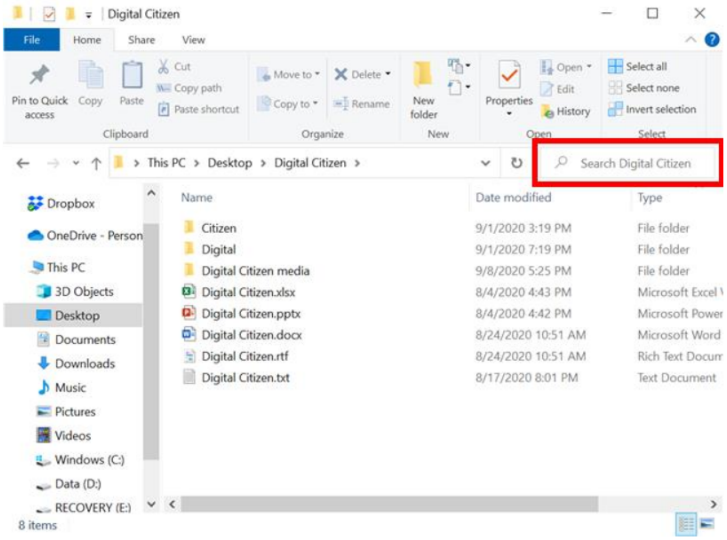
Bab V
Kesimpulan dan Saran **26**
5.1 Kesimpulan 26
5.2 Saran 26

Daftar Pustaka **27**

Link Repository Github
Kelompok Mesin pencari **27**

Bab 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

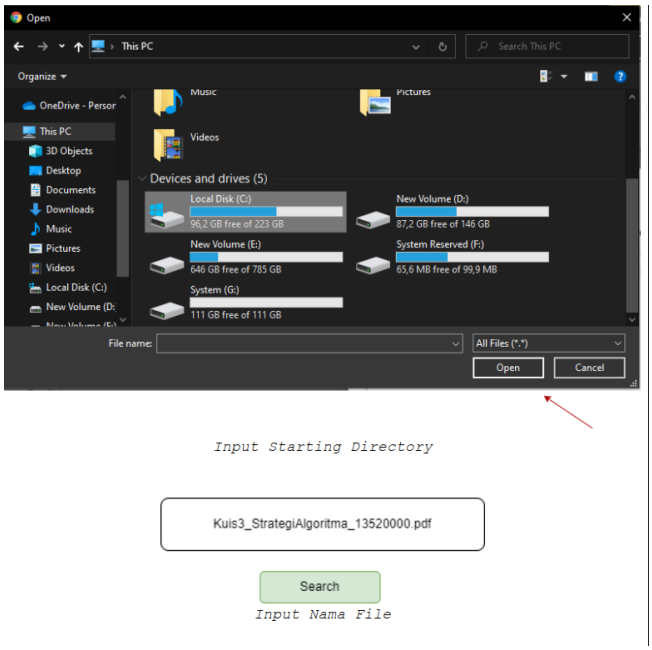


Gambar 1.1 Fitur Search pada Windows 10 File Explorer
(Sumber:https://www.digitalcitizen.life/wp-content/uploads/2020/10/explorer_search_10.png)

Meskipun demikian, kita tidak perlu cemas dalam menghadapi persoalan tersebut sekarang. Pasalnya, hampir seluruh sistem operasi sudah menyediakan fitur search yang dapat digunakan untuk mencari file yang kita inginkan. Kita cukup memasukkan query atau kata kunci pada kotak pencarian, dan komputer akan 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

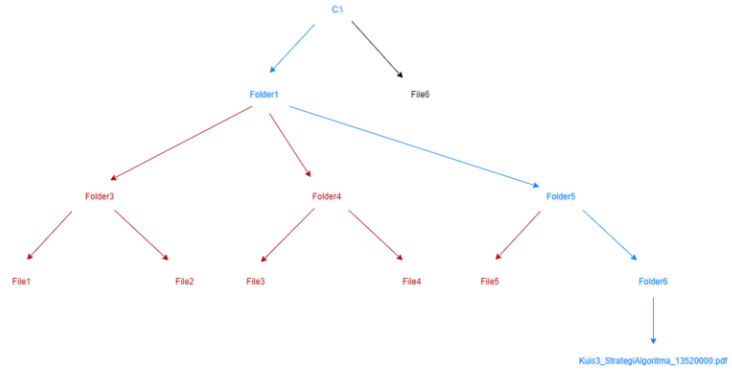
Deskripsi tugas:
Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari file explorer pada sistem operasi, yang pada tugas ini disebut dengan Folder Crawling. Dengan memanfaatkan algoritma Breadth First Search (BFS) dan

Depth First Search (DFS), Anda dapat menelusuri folder-folder yang ada pada direktori untuk mendapatkan direktori yang Anda inginkan. Anda juga diminta untuk memvisualisasikan hasil dari pencarian folder tersebut dalam bentuk pohon. Selain pohon, Anda diminta juga menampilkan list path dari daun-daun yang bersesuaian dengan hasil pencarian. Path tersebut diharuskan memiliki hyperlink menuju folder parent dari file yang dicari, agar file langsung dapat diakses melalui browser atau file explorer. Contoh hal-hal yang dimaksud akan dijelaskan di bawah ini



Gambar 1.2 Contoh input Program

Contoh output aplikasi:



List Path:
1. C:\Folder1\Folder5\Folder6\Kuis3_StrategiAlgoritma_13520000.pdf

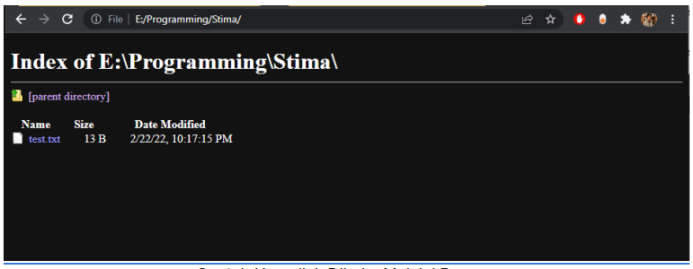
Gambar 1.4. Contoh output program jika ditemukan

Misalnya pengguna ingin mengetahui langkah folder crawling untuk menemukan file Kuis3_StrategiAlgoritma_13520000.pdf. Maka, path pencarian DFS adalah sebagai berikut.

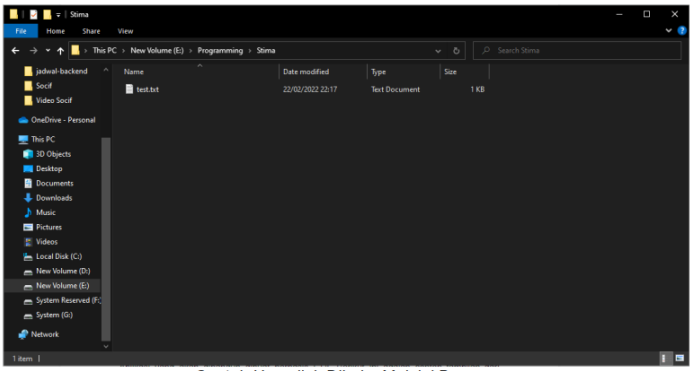
C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3_StrategiAlgoritma_13520000.pdf.

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

Contoh Hyperlink Pada Path:



Contoh Hyperlink Dibuka Melalui Browser



Contoh Hyperlink Dibuka Melalui Browser

Gambar 1.5. Contoh ketika hyperlink di klik

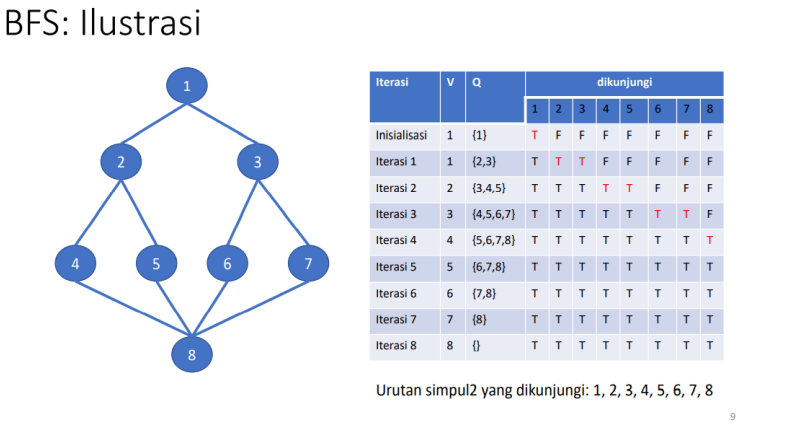
Bab II
Landasan Teori

2.1 Traversal Graf

Graph adalah kumpulan titik/simpul (node) dan garis (edge) dimana pasangan simpul (node) tersebut dihubungkan oleh segmen garis (edge). Algoritma traversal Graph merupakan algoritma mengunjungi simpul (node) secara sistematik untuk mencari solusi. Terdapat beberapa metode traversal graph yaitu breadth first search (BFS) dan depth first search (DFS) dengan asumsi bahwa setiap simpul saling terhubung dengan simpul lainya. Dalam pencarian solusi terdapat 2 pendekatan yang dilakukan yaitu graf statis dan graf dinamis. Graf statis merupakan graf yang sudah terbentuk sebelum proses pencarian dilakukan (graf direpresentasikan dalam struktur data). Sedangkan graf dinamis merupakan graf yang terbentuk saat proses pencarian (graf dibangun selama pencarian solusi).

2.2 BFS

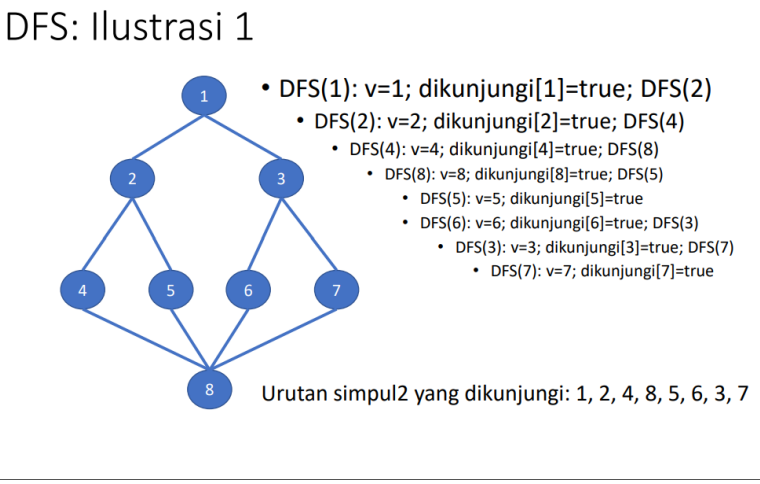
BFS atau *Breadth First Search* merupakan algoritma yang melakukan pencarian simpul pada graph secara melebar. Hal ini berarti BFS akan mengunjungi suatu simpul kemudian mengunjungi simpul lain yang bertetangga dengan simpul tersebut terlebih dahulu. Algoritma BFS memerlukan suatu antrian untuk menyimpan simpul-simpul yang telah dikunjungi. Algoritma ini juga memerlukan suatu tabel boolean *“dikunjungi”* untuk menandakan suatu simpul sudah dikunjungi atau belum sehingga tidak ada simpul yang akan dikunjungi lebih dari sekali.



Gambar 2.2.1 Ilustrasi BFS

2.3 DFS

DFS atau *Depth First Search* merupakan algoritma yang melakukan pencarian simpul pada graf secara kedalaman. Hal ini berarti DFS akan mengunjungi simpul kemudian mengunjungi simpul anaknya berdasarkan prioritas tertentu hingga mencapai level terdalam(tidak punya anak lagi). Jika sudah mencapai level terdalam akan dilakukan *backtracking* (kembali ke simpul sebelumnya) untuk memeriksa simpul lainnya dengan metode yang sama seperti sebelumnya.



Gambar 2.3.1 Ilustrasi DFS

2.4 C# Desktop Application Development

Bahasa c# merupakan salah satu pilihan untuk membangun gui berbasis aplikasi desktop. Windows form application merupakan salah satu gui framework untuk membangun aplikasi windows berbasis desktop berdasarkan desainer visual yang disediakan di Visual Studio. Banyak sekali fitur yang disediakan oleh Windows Form di Visual Studio, seperti label, panel dan beberapa fitur lainya yang menunjang developer dalam UI/UX aplikasi. Fungsionalitas seperti penempatan drag-and-drop kontrol visual memudahkan untuk membangun aplikasi desktop. Windows form juga dapat mengakses perangkat keras lokal dan sistem file komputer tempat aplikasi berjalan. Winforms atau Windows form application tergabung dengan framework .Net. Selain windows form terdapat juga framework lain yang menggunakan bahasa c# yaitu WPF (Windows Presentation Foundation)

Bab III
Analisis Pemecahan Masalah

3.1 Pemetaan Masalah

Dalam permasalahan Folder Crawling, setiap folder atau file merupakan sebuah simpul di dalam graph. Setiap folder atau file yang berada di dalam folder tertentu memiliki hubungan ketetanggaan dengan simpul folder tersebut. Simpul folder yang dimasukkan adalah simpul awal pencarian. Simpul file yang dicari merupakan simpul tujuan pencarian. Permasalahan pencarian file dapat dicari menggunakan 2 algoritma, yaitu BFS dan DFS.

3.2 Langkah-langkah Penyelesaian Masalah

3.2.1 Penyelesaian Masalah dengan Algoritma BFS

Langkah-langkah penyelesaian masalah dengan algoritma BFS:

1. Simpul folder awal ditandai sudah dikunjungi dan dimasukkan ke dalam antrian pencarian
2. Keluarkan simpul pertama di dalam antrian
3. Cek apakah simpul tersebut merupakan simpul tujuan yang dicari.
4. Jika simpul tersebut adalah simpul tujuan, masukkan path file ke dalam himpunan solusi. Apabila tidak memilih untuk mencari seluruh kemunculan file, hentikan pencarian. Jika simpul tersebut buka simpul tujuan, masukkan semua simpul yang belum dikunjungi dan bertetangga dengan simpul tersebut ke dalam antrian pencarian dan ditandai sudah dikunjungi
5. Ulangi langkah ke-5 dan ke-6 hingga ditemukan simpul tujuan apabila tidak memilih untuk mencari seluruh kemunculan di folder. Apabila memilih untuk mencari seluruh kemunculan, pemrosesan dilakukan hingga antrian pencarian sudah habis
6. Apabila tidak menemukan file yang dicari, himpunan solusi akan bernilai kosong. Apabila ditemukan file yang dicari, himpunan solusi akan berisi beberapa path menuju file yang dicari
7. Untuk setiap path dalam himpunan solusi, beri warna biru pada simpul-simpul dan sisi-sisi yang dilalui dari simpul folder hingga simpul file. Selain itu, warna default pada simpul dan sisi yang lain adalah merah. Apabila tidak memilih untuk mencari seluruh kemunculan file, simpul yang belum dikunjungi akan diberi warna hitam

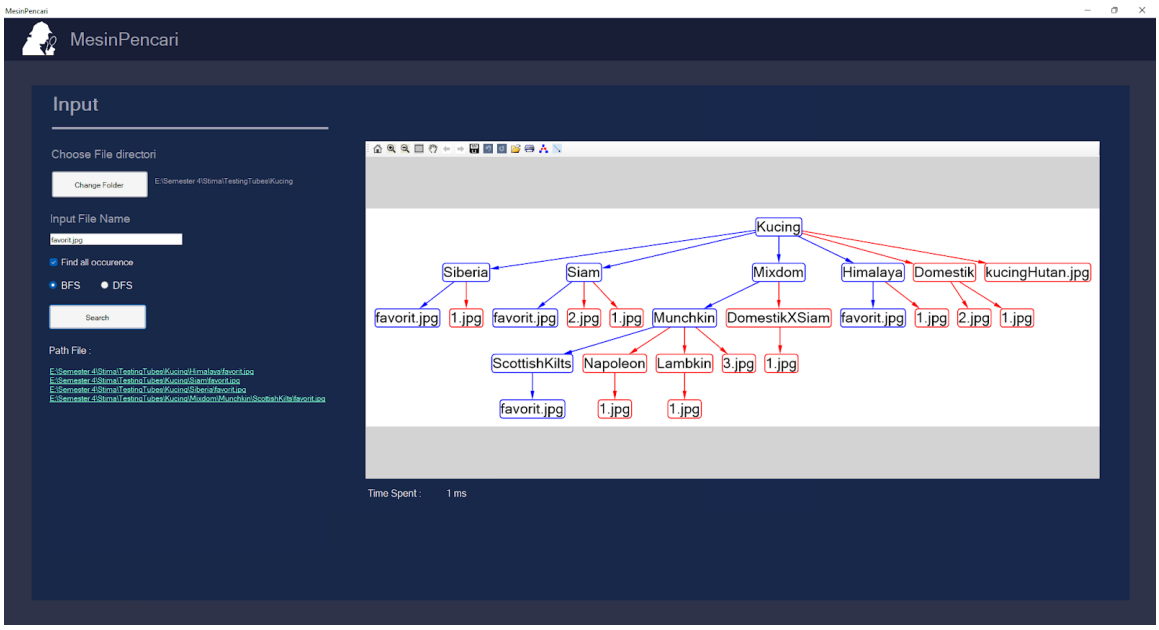
3.2.2 Penyelesaian Masalah dengan Algoritma DFS

Langkah-langkah penyelesaian masalah dengan algoritma DFS:

1. Simpul folder awal ditandai sudah dikunjungi dan merupakan *currentSimpul*

2. Pada setiap iterasi, cek semua file yang ada pada *currentSimpul* yang merupakan simpul daun. Akan terdapat 2 buah kemungkinan
   - Jika simpul daun tersebut adalah file yang dicari, masukkan path file tersebut ke himpunan solusi. Apabila **tidak** melakukan pengecekan secara **menyeluruh**, hentikan pencarian setelah penemuan pertama. Sedangkan jika dilakukan pengecekan secara **menyeluruh**, lanjutkan pengecekan terhadap semua simpul daun pada *currentSimpul* dengan mengulangi langkah 2 dan setelah semuanya dicek lanjut ke langkah 3.
   - Jika simpul daun tersebut bukan simpul tujuan, tandai simpul sebagai simpul yang telah dikunjungi dan lanjut ke langkah 3.
3. Masukkan semua simpul anakan (child) dari *currentSimpul* ke dalam antrian. Simpul anakan ini adalah folder/simpul yang akan dicek secara rekursif (dan ditandai telah dikunjungi) dengan mengulangi langkah 2 s.d. langkah 3 hingga ditemukan simpul pertama yang sesuai (**tidak menyeluruh**) atau semua simpul telah dicek (**menyeluruh**).
4. Apabila tidak menemukan file yang dicari, himpunan solusi akan bernilai kosong. Apabila ditemukan file yang dicari, himpunan solusi akan berisi beberapa path menuju file yang dicari
5. Untuk setiap path dalam himpunan solusi, beri warna biru pada simpul-simpul dan sisi-sisi yang dilalui dari simpul folder hingga simpul file. Selain itu, warna default pada simpul dan sisi yang lain adalah merah. Apabila tidak memilih untuk mencari seluruh kemunculan file, simpul yang belum dikunjungi akan diberi warna hitam

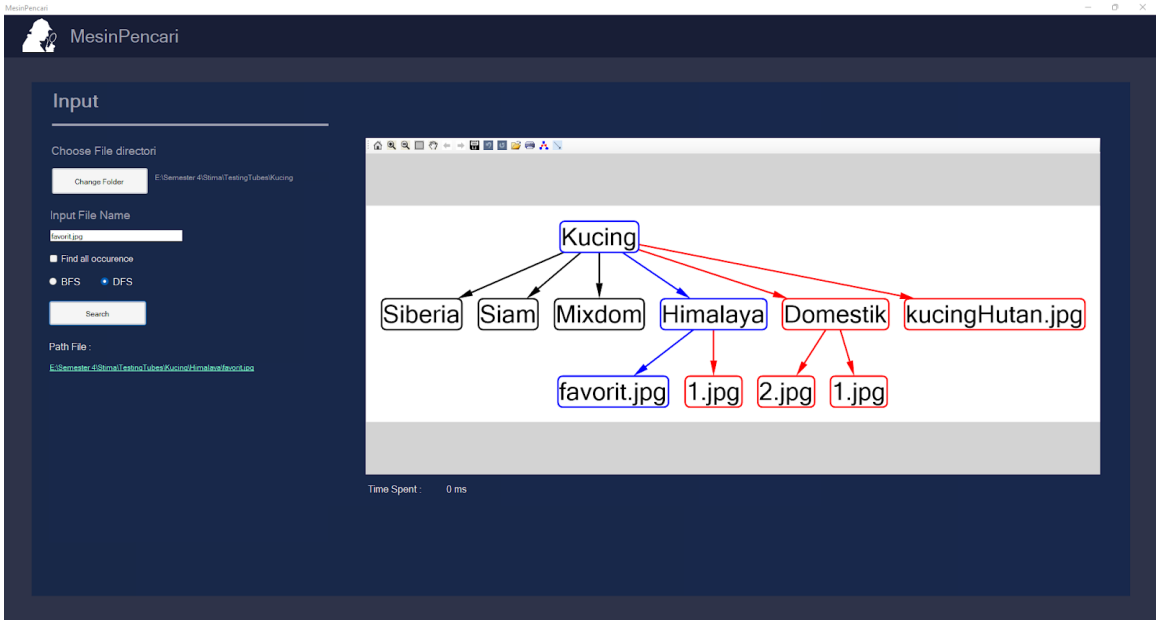
3.3 Ilustrasi Kasus Lain dalam Folder Crawling



Gambar 3.3.1 Pencarian All Occurrence dengan BFS

Pada kasus ini, folder yang dimasukkan merupakan folder bernama Kucing dan ingin mencari semua kemunculan file bernama favorit.jpg. Simpul Kucing akan dimasukkan ke dalam antrian. Selama antrian belum kosong, simpul paling depan akan diambil dan dicek. Pada pengecekan

simpul Kucing, program akan membangkitkan simpul-simpul yang bertetangga dengan simpul Kucing dan mengecek simpulnya. Saat ini, Antrian berisi kucingHutan.jpg, Domestik, Himalaya, Mixdom, Siam, dan Siberia. Simpul kucingHutan.jpg bukan merupakan simpul yang dicari sehingga pencarian tetap dilanjutkan. Saat memeriksa simpul Himalaya dan menambahkan simpul favorit.jpg, ini merupakan hasil pertama yang ditemukan. Pencarian berlanjut ke simpul Mixdom dan membangkitkan simpul Munchkin dan DomestikXSiam. Kemudian lanjut mengambil simpul Sial dan membangkitkan simpul 1.jpg, 2.jpg, dan favorit.jpg. Pada saat ini telah ditemukan kemunculan kedua. Kemudian akan mengambil simpul Siberia dan membangkitkan simpul 1.jpg dan favorit.jpg. Kemunculan ketiga telah ditambahkan. Saat ini antrian berisi simpul DomestikXSiam dan Munchkin. Kemudian mengambil simpul DomestikXSiam dan membangkitkan simpul 1.jpg. Setelah itu mengambil simpul Munchkin dan membangkitkan simpul 3.jpg, Lambkin, Napoleon, dan ScottishKitts. Saat ini antrian berisi simpul Lambkin, Napoleon, dan ScottishKitts. Setelah itu mengambil simpul Lambkin dan membangkitkan simpul 1.jpg. Kemudian mengambil simpul Napoleon dan membangkitkan simpul 1.jpg. Kemudian mengambil simpul ScottishKitts dan membangkitkan simpul favorit.jpg. Path yang ditemukan ini ditambahkan ke dalam hasilnya. Karena antrian sudah kosong, pencarian dihentikan dan mengeluarkan 4 path ditemukannya file favorit.jpg.



Gambar 3.3.2 Pencarian dengan DFS

Pada ilustrasi di atas, folder yang dimasukkan adalah folder Kucing dan file bernama favorit.jpg. Pencarian dilakukan dengan algoritma DFS. Pencarian tidak mencari seluruh kemunculan file, hanya satu file saja. Dalam pencarian ini, alur program adalah sebagai berikut

Kucing → kucingHutan.jpg → Kucing (backtrack) → Domestik → 1.jpg → Domestik (backtrack) → 2.jpg → Domestik (backtrack) → Kucing (backtrack) → Himalaya → 1.jpg → Himalaya (backtrack) → favorit.jpg. Saat ini sudah ditemukan file dengan nama favorit.jpg sehingga pencarian dihentikan. Namun terdapat 3 simpul yang masih belum dikunjungi dari simpul Kucing sehingga simpul Mixdom, Siam, dan Siberia masih berwarna hitam.

Bab IV
Implementasi dan Pengujian

4.1 Implementasi Algoritma pada Folder Crawling

4.1.1 Algoritma pada BFS

4.1.1.1 Fungsi Solve

Fungsi ini merupakan implementasi dari BFS untuk folder crawling. Fungsi ini menerima dua parameter, yaitu parameter pertama berupa path folder yang dicari dan parameter kedua merupakan nama file yang akan dicari. Fungsi ini mengembalikan list of string yang berisi path-path ditemukannya file dengan nama file yang bersesuaian

```
Function Solve(folder: string, filename: string) → list of string
Kamus
antrian : queue of string
pengecekan : list of string
result : list of string
first,  : string
dirs, files, splitFile : list of string

function getDirectories(folder : string) → List of string
{Fungsi yang mengembalikan list path folder yang ada di dalam folder}

function getFiles(folder : string) → List of string
{Fungsi yang mengembalikan list path file yang ada di dalam folder}

function split(sep : string, arr : string) → List of string
{Fungsi yang mengembalikan string hasil pemisahan arr dengan pemisah
sep}

function join(sep : string, arr : List of string) → string
{Fungsi yang mengembalikan string hasil penggabungan array arr dengan
penghubung sep}

Algoritma
antrian.enqueue(pathfile)
while (antrian.Count > 0) do
    first ← antrian.dequeue()
    files ← getFiles(first)
    foreach (file: string in files) do
        splitFile ← split("\", file)
        pengecekan.add(file)
        if (splitFile[splitFile.length()-1] = filename) then
```

```
            result.add(file)

    dirs ← getDirectories(first)
    foreach (dir : string in dirs) do
        pengecekan.add(dir)
        antrian.enqueue(dir)

    if (result.Count > 0 and not allOccur) then
        → result

→ result
```

4.1.2 Algoritma pada DFS

4.1.2.1 Fungsi DFSearch

Fungsi ini merupakan implementasi dari DFS untuk folder crawling

```
Procedure DFSearch(string folder, string filesearch, array of string
pengecekan, array of string result, boolean checkAll)

KAMUS
found_1 : boolean
currfiles : array of string
dirs : array of string

ALGORITMA
                    //* -> Bagian 1 : Basis *//

found_1 = false
{Cek semua file dalam folder sekarang,
 file dicek terlebih dahulu sebelum folder lainnya secara DFS.}
currfiles = Directory.GetFiles(folder)

foreach (file : string in currfiles)do
{File yang telah diperiksa ditambahkan ke list pengecekan}
    pengecekan.Add(file)

{Jika file yang dicek merupakan file yang dicari, tambahkan ke list
result}
    if (Path.GetFileName(file) == filesearch) then
        result.Add(file)
        found_1 = true

                    //* -> Bagian 2 : Rekursif *//

{Cek semua folder yang ada, lalu lakukan pengecekan secara rekursif
 untuk setiap folder}
dirs = Directory.GetDirectories(folder)
```

```
foreach (dir : string in dirs) do
    if (!checkAll) then
        pengecekan.Add(dir)
        if (result.Count == 0) then
            DFSearch(dir, filesearch, pengecekan, result, checkAll)

{Jika tidak perlu mengecek semua file dan telah ditemukan 1 yang
 cocok, maka keluar dari proses pengecekan.}
    else
        pengecekan.Add(dir)
        DFSearch(dir, filesearch, pengecekan, result, checkAll)

                    //* -> checkAll Constraint <- *\\
{Jika tidak perlu mengecek semua file dan telah ditemukan 1 yang
 cocok, maka keluar dari proses pengecekan.}
if (!checkAll && found_1) then
    return
```

4.1.3 Algoritma pada Form

4.1.3.1 Fungsi Form1

Fungsi ini untuk melakukan pemanggilan fungsi InitializeComponent

```
procedure Form1()
Algoritma
    InitializeComponent()
    { call procedure InitializeComponent()}
```

4.1.3.2 Fungsi buttonChooseFolder_Click

Fungsi ini akan membuka *file explorer* untuk user memilih *starting directory* ketika buttonChooseFolder di klik

```
procedure buttonChooseFolder_Click(object sender, EventArgs e)
Algoritma
{
    if (dirs!=null)then
        Array.Clear(dirs, 0, dirs.Length)//hapus setiap elemen pada dirs
    if (files!=null) then
        Array.Clear(files, 0, files.Length) //hapus semua elemen pada
dirs
    if (dialog.ShowDialog() == DialogResult.OK
        labelFolderPath.Text <- dialog.SelectedPath
```

```
        files <- Directory.GetFiles(dialog.SelectedPath) // get all file
in starting directory
        dirs <- Directory.GetDirectories(dialog.SelectedPath) // get all
directory starting directory// get file in starting directory
```

4.1.3.3 Fungsi buttonSearch_Click

Fungsi ini merupakan fungsi utama pada program ini yaitu dilakukan pencarian sesuai dengan input user, fungsi ini dipanggil ketika buttonSearch di klik.

```
procedure buttonSearch_Click(object sender, EventArgs e)
Algoritma
    var watch <- new System.Diagnostics.Stopwatch()
    folder <- labelFolderPath.Text
    fileSearch <- textBoxInputFileName.Text // file yang akan dicari

    Microsoft.Msagl.GraphViewerGdi.GViewer viewer <- new
Microsoft.Msagl.GraphViewerGdi.GViewer()
    Microsoft.Msagl.Drawing.Graph graph <- new
Microsoft.Msagl.Drawing.Graph("graph")

    if (fileSearch != null AND folder != "No File Choosen..") then
        panelOutput.Visible <- false
        // hapus semua elemen bekas pencarian sebelumnya
        testing.Items.Clear()
        listLinkPath.Items.Clear()
        BFS.pengecekan.Clear()
        BFS.antrian.Clear()

        List<string> result <- new List<string>()
        Dictionary<string, string> pathColor <- new
Dictionary<string, string>()
         string[] splitPath
         bool colorize <- true

         // Mengambil root node (folder pencarian)
         splitPath <- folder.Split("\\")
         string root <- splitPath[splitPath.Length - 1]
         string file, warna, parent, child, ujung
```

```
        // BFS
        if (radioButtonBfs.Checked) then
            watch.Start()
            result = BFS.Solve(folder, fileSearch,
checkBoxFindAll.Checked)
            watch.Stop()
            pathColor = BFS.warnaPath
        // DFS
        if (radioButtonDfs.Checked) then

        if (result.Count == 0)then
        else

        labelRuntime.Text <- $"{watch.ElapsedMilliseconds} ms";
        panelOutput.Visible <- true;

        if (result.Count = 0) then
            graph.AddNode(root).Attr.Color <-
Microsoft.Msagl.Drawing.Color.Red;
        else
            graph.AddNode(root).Attr.Color <-
Microsoft.Msagl.Drawing.Color.Blue;


        foreach (KeyValuePair<string, string> pair in pathColor) do
            file <- pair.Key;
            warna <- pair.Value;

            splitPath <- file.Split('\\');
            ujung <- splitPath[splitPath.Length - 1]
            splitPath <- splitPath[..^1]

            parent <- string.Join("\\", splitPath)
            child <- ujung;

            int i <- 0;
```

```
            var parentNode <- graph.FindNode(parent);
            while (parentNode = null AND parent != "") do
                parent <- string.Join("\\", splitPath.Skip(++i))
                parentNode <- graph.FindNode(parent)

            if (graph.FindNode(child) != null) then
                child <- parent + "\\" + child

                   // Pemberian warna pada node dan edge pada graph
            if (warna == "Red") {
                if (!checkBoxFindAll.Checked AND !colorize) then
                    graph.AddEdge(parent, child).Attr.Color <-
Microsoft.Msagl.Drawing.Color.Black
                    graph.FindNode(child).Attr.Color <-
Microsoft.Msagl.Drawing.Color.Black;
                else
                    graph.AddEdge(parent, child).Attr.Color <-
Microsoft.Msagl.Drawing.Color.Red;
                    graph.FindNode(child).Attr.Color <-
Microsoft.Msagl.Drawing.Color.Red;
            else
                graph.FindNode(parent).Attr.Color <-
Microsoft.Msagl.Drawing.Color.Blue;
                graph.AddEdge(parent, child).Attr.Color <-
Microsoft.Msagl.Drawing.Color.Blue;
                graph.FindNode(child).Attr.Color <-
Microsoft.Msagl.Drawing.Color.Blue;

            // Mengubah flag/tanda warna ketika tidak melakukan
pencarian menyeluruh dan telah ditemukan file yang dicari
            if (ujung.Equals(fileSearch) AND
!checkBoxFindAll.Checked) then
                colorize <- false;

            this.gViewer1.Graph = graph;
            wait(200);
```

4.1.3.4 Fungsi wait

Fungsi ini menghentikan proses selama input milliseconds

```
procedure wait(int milliseconds)
Algoritma
    var timer1 <- new System.Windows.Forms.Timer()
    if (milliseconds = 0 OR milliseconds < 0) then
        ->

    // start wait timer
    timer1.Interval <- milliseconds
    timer1.Enabled <- true
    timer1.Start()

    timer1.Tick += (s, e) =>
    {
        timer1.Enabled <- false
        timer1.Stop()
        // stop wait timer

    while (timer1.Enabled) do
        Application.DoEvents()
```

4.1.3.5 Fungsi listLinkPath_SelectedIndexChanged

Fungsi ini untuk membuka link path dari file yang ditemukan sesuai index yang dipilih oleh user pada interface aplikasi

```
procedure listLinkPath_SelectedIndexChanged(object sender, EventArgs e)
Algoritma
    int index <- 0;
    ListBox lb <- sender as ListBox
    if (lb != null) then
        indeks <- lb.SelectedIndex
    if (indeks < listLinkPath.Items.Count AND indeks >= 0) then
        ProcessStartInfo psi <- new ProcessStartInfo("Explorer.exe")
        psi.Arguments <- " /select," + testing.Items[indeks] //Set
arguments
```

```
        Process.Start(psi)
```

4.1.4 Algoritma pada Form.designer

4.1.4.1 Fungsi InitializeComponent

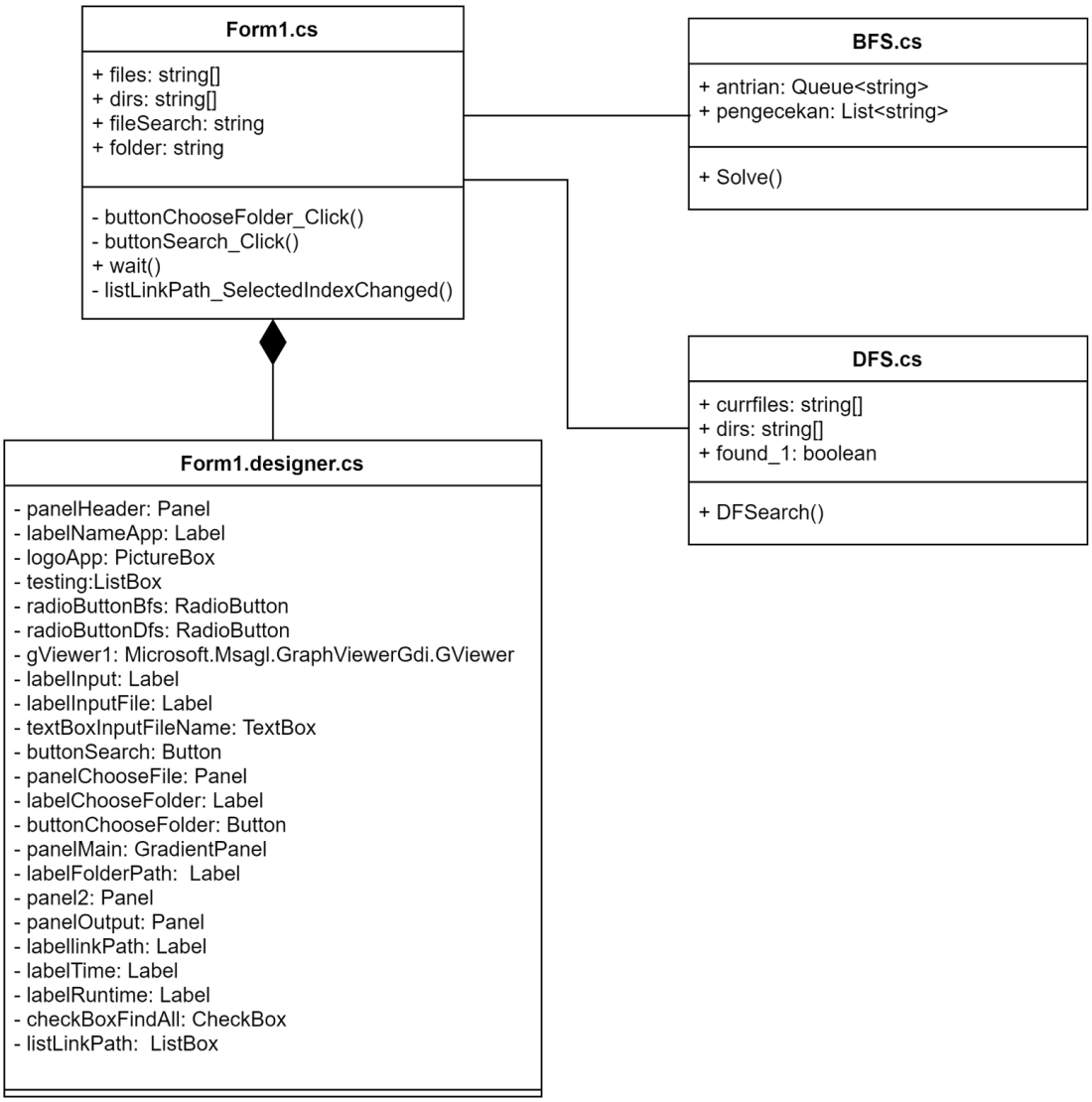
Fungsi ini untuk menginisialisasi komponen-komponen yang terdapat pada Interface aplikasi

```
procedure InitializeComponent()
Algoritma
    System.ComponentModel.ComponentResourceManager resources <- new
System.ComponentModel.ComponentResourceManager(typeof(Form1))
    this.panelHeader <- new System.Windows.Forms.Panel()
    this.labelNameApp <- new System.Windows.Forms.Label()
    this.logoApp <- new System.Windows.Forms.PictureBox()
    this.testing <- new System.Windows.Forms.ListBox()
    this.radioButtonBfs <- new System.Windows.Forms.RadioButton()
    this.radioButtonDfs <- new System.Windows.Forms.RadioButton()
    this.gViewer1 <- new Microsoft.Msagl.GraphViewerGdi.GViewer()
    this.labelInput <- new System.Windows.Forms.Label()
    this.labelInputFile <- new System.Windows.Forms.Label()
    this.textBoxInputFileName <- new System.Windows.Forms.TextBox()
    this.buttonSearch <- new System.Windows.Forms.Button()
    this.panelChooseFile <- new System.Windows.Forms.Panel()
    this.labelFolderPath <- new System.Windows.Forms.Label()
    this.labelChooseFolder <- new System.Windows.Forms.Label()
    this.buttonChooseFolder <- new System.Windows.Forms.Button()
    this.panelMain <- new src.GradientPanel()
    this.checkBoxFindAll <- new System.Windows.Forms.CheckBox()
    this.panelOutput <- new System.Windows.Forms.Panel()
    this.listLinkPath <- new System.Windows.Forms.ListBox()
    this.labelRuntime <- new System.Windows.Forms.Label()
    this.labellinkPath <- new System.Windows.Forms.Label()
    this.labelTime <- new System.Windows.Forms.Label()
    this.panel2 <- new System.Windows.Forms.Panel()
    this.panelHeader.SuspendLayout()

((System.ComponentModel.ISupportInitialize)(this.logoApp)).BeginInit();
```

```
    this.panelChooseFile.SuspendLayout();
    this.panelMain.SuspendLayout();
    this.panelOutput.SuspendLayout();
    this.SuspendLayout();

    {edit default all Component (setting position, default value,
anchor, color, etc.}
    ...
```

4.2 Struktur Data pada Program

Berikut Diagram Struktur data yang digunakan pada program yang telah kami buat.



Gambar 4.2 Pencarian dengan DFS

4.2.1 Form1.designer.cs

Pada bagian ini berisi komponen yang disediakan Windows Form yang merupakan Interface pada aplikasi

- Type Panel:
  - panelHeader
  - panelChooseFile
  - panelMain
  - panelOutput
- Type Label:
  - labelNameApp
  - labelInput
  - labelInputFile
  - labelChooseFolder
  - labelFolderPath
  - labelLinkPath
  - labelTime
  - labelRuntime
- Type PictureBox: logoApp
- Type ListBox:
  - Testing
  - listLinkPath
- Type RadioButton:
  - radioButtonBFS
  - radioButtonDFS
- Type Microsoft.Msagl.GraphViewerGdi.GViewer :gViewer1
- Type TextBox: textBoxInputFileName
- TypeCheckBox: checkBoxFindAll

4.2.2 Form1.cs

- files : Type String[]
  Menampung semua file yang terdapat pada starting directory
- dirs : Type String[]
  Menampung semua direktori yang terdapat pada starting directory
- Filesearch : Type String
  Menampung nama file yang akan dilakukan pencarian
- Folder : Type String
  Menampung nama *starting directory*

4.2.3 BFS.cs

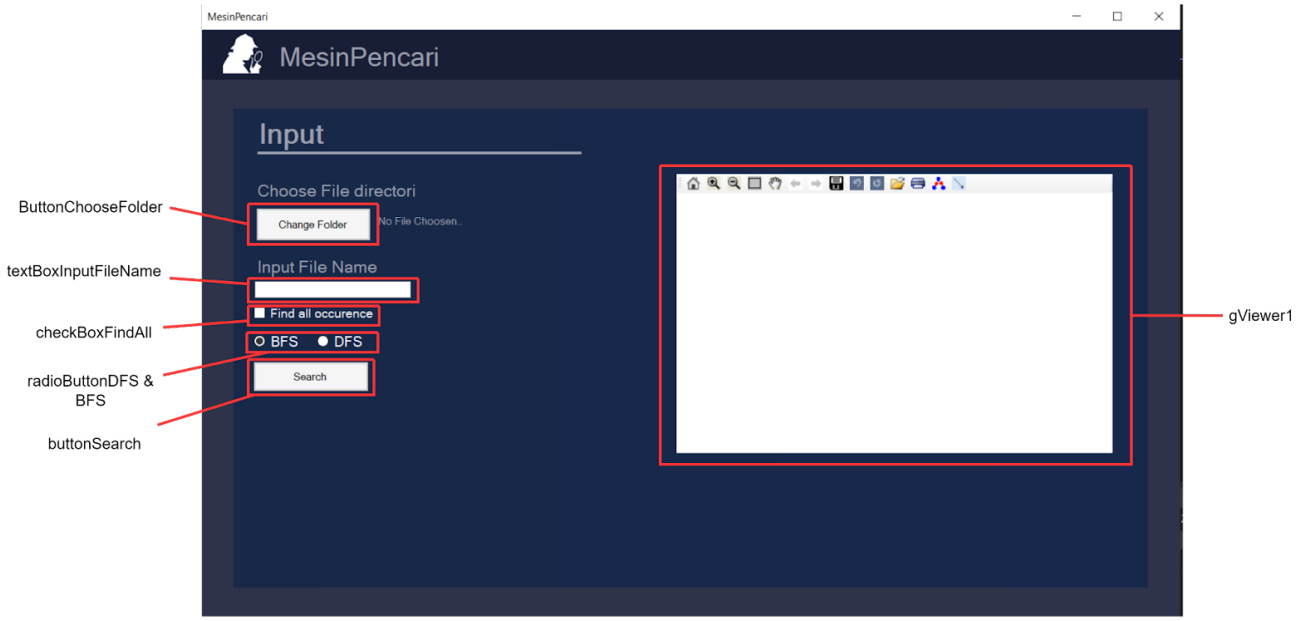
- antrian : Type Queue<string>
  Menampung antrian pengecekan node

- pengecekan : Type List<string>
  Menampung urutan pengecekan pada path file tertentu

4.2.4 DFS.cs

- currfiles : Type List<string>
  Menampung antrian pengecekan file yang berada pada folder sekarang
- dirs : Type List<string>
  Menampung antrian pengecekan folder turunan yang berada pada folder sekarang
- Found_1 : Type boolean
  Pembatas apabila pengecekan hanya dilakukan sampai penemuan pertama

4.3 Penggunaan Program



Gambar 4.3 Tampilan GUI dan fitur-fiturnya

1. **ButtonChooseFolder**
   Jika di klik akan membuka *file explorer*. User dapat memilih *starting directory* yang akan dilakukan pencarian / folder crawling

2. **textBoxInputFileName**
   Tempat untuk User memasukan nama file yang akan dilakukan pencarian pada *starting directory*

3. **checkBoxFindAll**
   Jika mencentang bagian ini. Program akan mencari semua kemunculan file pada folder root dan program akan menampilkan daftar semua rute file yang memiliki nama sama persis dengan input nama file

4. **radioButton BFS & DFS**
   User dapat memilih untuk melakukan pencarian file dengan menggunakan BFS atau DFS

5. **buttonSearch**
   Jika diklik maka pencarian akan dimulai dengan kondisi sebagai berikut:
   - User sudah memilih *starting directory*
   - User sudah memasukan nama file yang akan dicari
   - User sudah memilih skema pencarian BFS / DFS

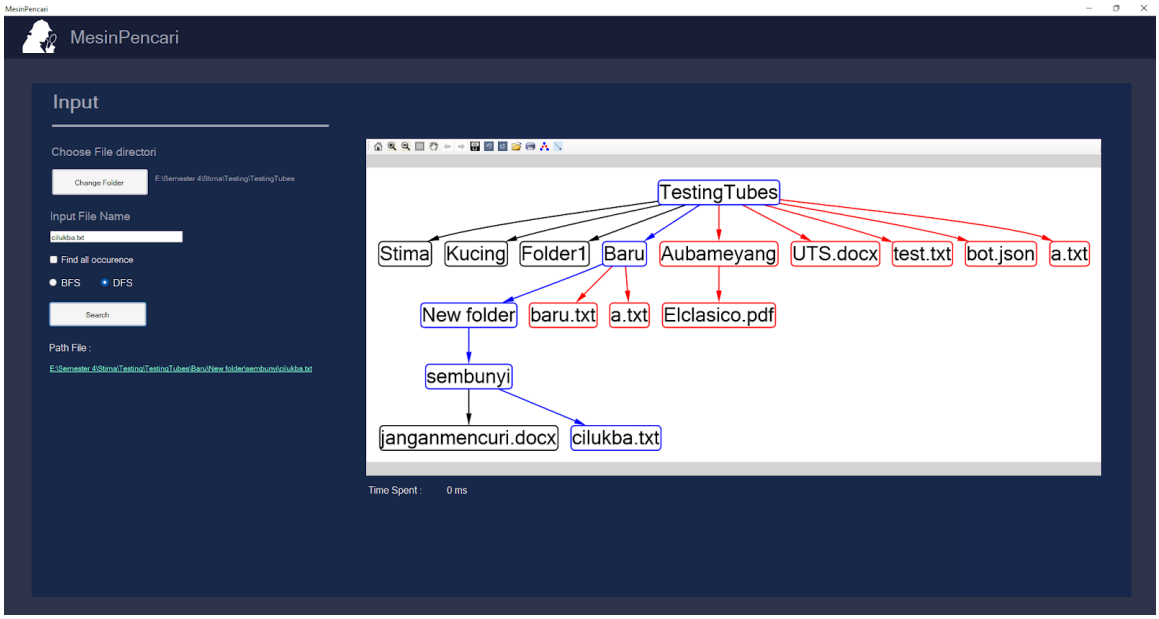
   Jika ketiga hal tersebut tidak dipenuhi maka pencarian tidak akan dilakukan

6. **gViewer1**
   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.

4.4 Analisis Hasil Pengujian

4.4.1 DFS

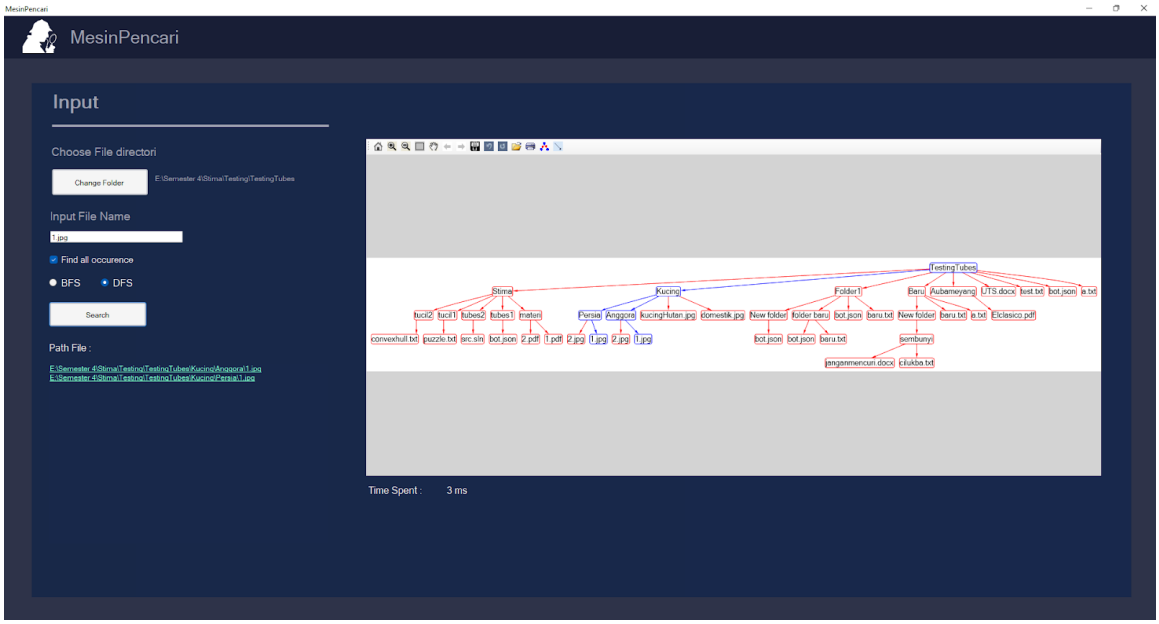
Pada pengujian ini, kami menggunakan algoritma DFS dan tidak mencari seluruh kemunculan file. File ditemukan pada kedalaman 4. Pada pencarian ini DFS memiliki performa yang cukup efektif. Hal ini karena file berada pada kedalaman yang cukup dalam.



Gambar 4.4.1 DFS

4.4.2 DFS find all occurence

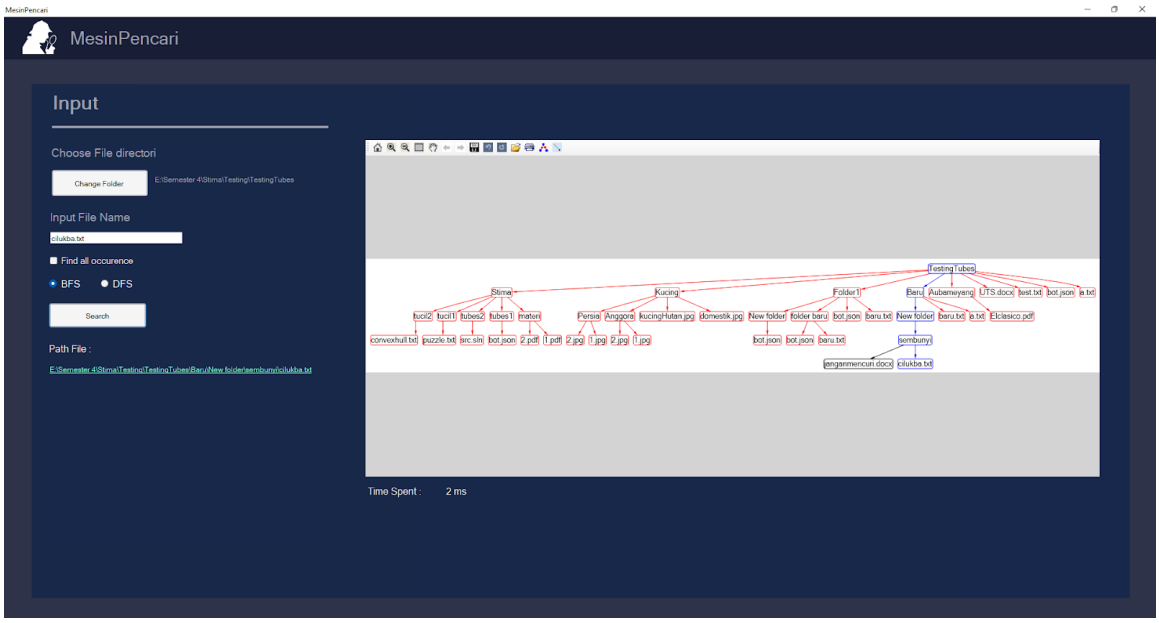
Pada pengujian ini, kami menggunakan algoritma DFS dan mencari seluruh kemunculan dari file 1.jpg. Kedua file ditemukan pada kedalaman 3.



Gambar 4.4.2 DFS find all occurence

4.4.3 BFS

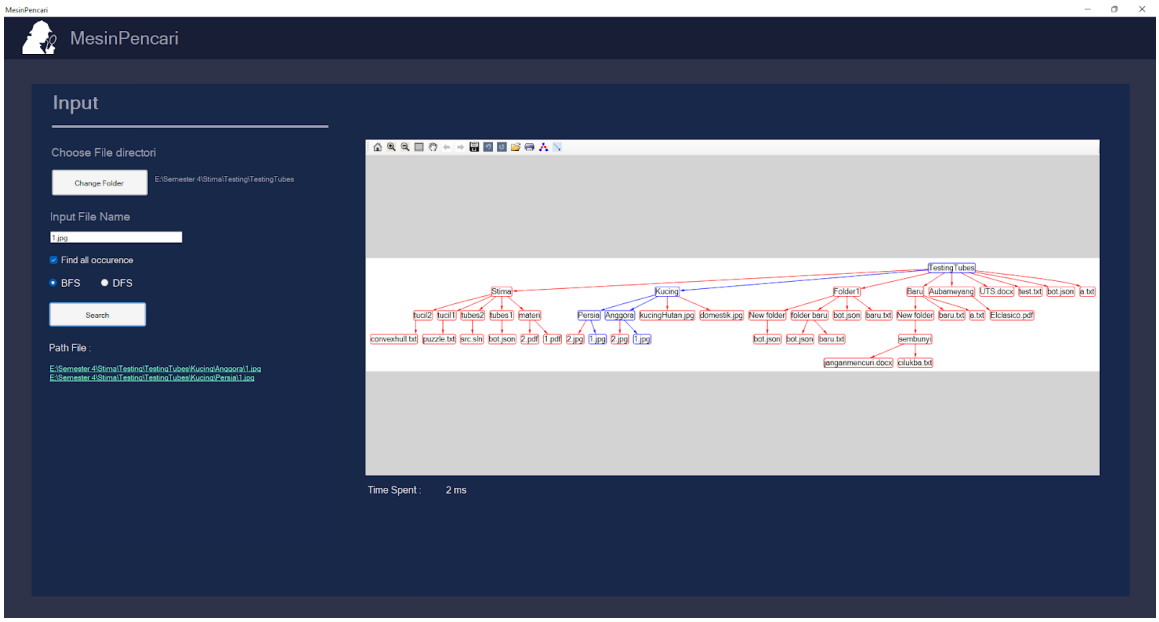
Pada pengujian ini, kami menggunakan algoritma BFS dan tidak mencari seluruh kemunculan file. Pada pengujian ini, algoritma BFS tidak memberikan performa yang baik jika dibandingkan dengan algoritma DFS pada pengujian pertama. Hal ini disebabkan oleh tipe folder yang dicari merupakan folder yang memiliki banyak isi dan file berada di kedalaman yang cukup dalam



Gambar 4.4.3 BFS

4.4.4 BFS find all occurence

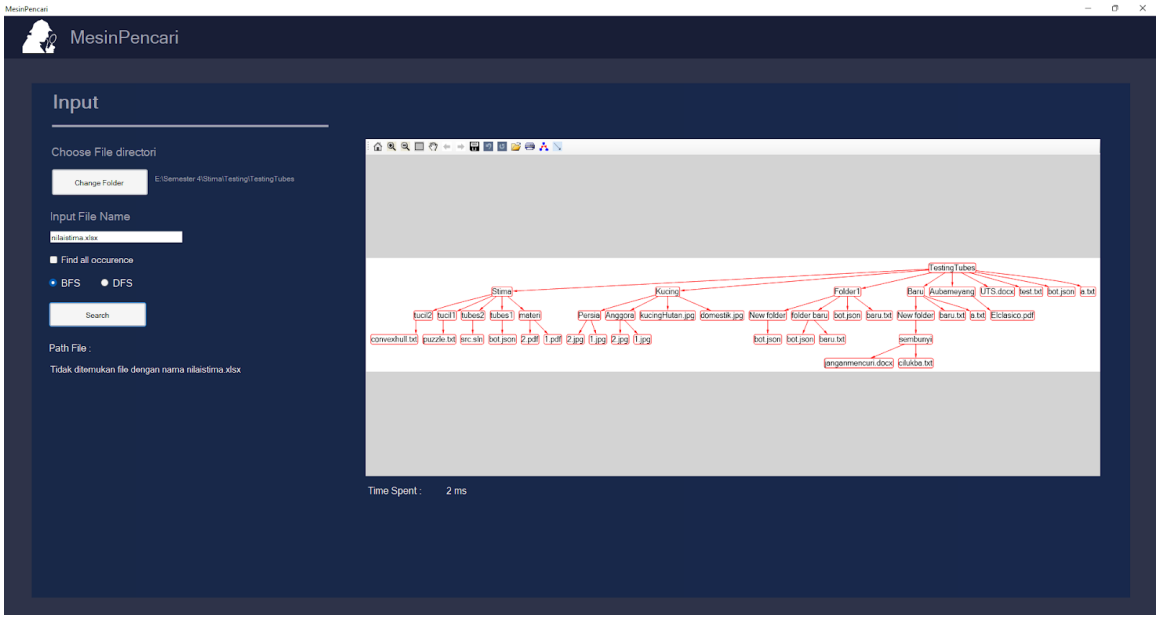
Pada pengujian ini, kami menggunakan algoritma BFS dengan mencari seluruh kemunculan file. Pencarian ini lebih efektif menggunakan algoritma BFS karena folder berisi banyak file/folder lagi di dalamnya.



Gambar 4.4.4 BFS find all occurence

4.4.5 Pencarian file tidak ditemukan

Pada pengujian ini, kami menggunakan algoritma BFS. Pada pencarian ini tidak ditemukan file dengan nama nilaistima.xlsx sehingga semua node berwarna merah yang berarti sudah dikunjungi semuanya



Gambar 4.4.5 BFS file tidak ditemukan

4.5 Analisis Solusi Program

Berdasarkan pengujian yang kami lakukan, BFS dan DFS memiliki kekurangan dan kelebihan masing-masing. Algoritma DFS lebih baik apabila folder yang dicari memiliki sedikit folder atau file di dalamnya namun memiliki kedalaman yang cukup dalam. Apabila folder memiliki banyak file/folder, algoritma BFS akan memiliki performa yang lebih baik. Keduanya dapat memberikan efektivitas yang baik dalam folder yang berukuran kecil. Apabila folder berukuran besar dan memiliki kedalaman yang cukup dalam, kedua algoritma memiliki performa yang kurang baik.

Bab V
Kesimpulan dan Saran

5.1 Kesimpulan

Pada Tugas Besar IF2211 Strategi Algoritma 2021/2022 berjudul "Pengaplikasian Algoritma BFS dan DFS dalam Implementasi *Folder Crawling*," kami sudah berhasil membuat sebuah aplikasi yang dapat mencari keberadaan suatu file di dalam folder tertentu. Program kami dapat menggunakan 2 algoritma pencarian, yaitu algoritma BFS (*Breadth First Search)* dan DFS *(Deep First Search)*. Kedua algoritma tersebut dapat menyelesaikan permasalahan ini dengan baik. Pencarian file juga dapat dilakukan untuk mencari satu file pertama saja atau seluruh kemunculan seluruh file di dalam folder dengan nama yang sama.

5.2 Saran

Beberapa saran yang dapat kami berikan untuk Tugas Besar II IF2211 Strategi Algoritma 2021/2022 ini adalah:

1. Ilustrasi graf pada aplikasi yang kami buat tidak cukup baik jika folder yang akan dicari mempunyai isi file dan folder yang terlalu banyak. Gambar graf yang terbentuk akan semakin kecil dan tidak kelihatan perlu di zoom agar dapat terlihat dengan jelas, oleh karena itu dapat menggunakan ilustrasi graf lain yang lebih baik
2. Perbanyak referensi mengenai bahasa pemrograman C# dan framework .NET untuk membangun sebuah aplikasi desktop

Daftar Pustaka

[1] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung. 2009.

Link Repository Github Kelompok Mesin pencari
github.com/lizardyy/Tubes2_13520090

Link Video Demonstrasi
https://youtu.be/KU90fCJi-pA