PENGAPLIKASIAN ALGORITMA BFS DAN DFS DALAM IMPLEMENTASI *FOLDER CRAWLING*

LAPORAN TUGAS BESAR II

Diajukan untuk memenuhi Tugas Besar II
IF2211 Strategi Algoritma

Oleh

Damianus Clairvoyance Diva Putra	13520035
Nicholas Budiono	13520121
Fikri Ihsan Fadhiilah	13520148

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

2022

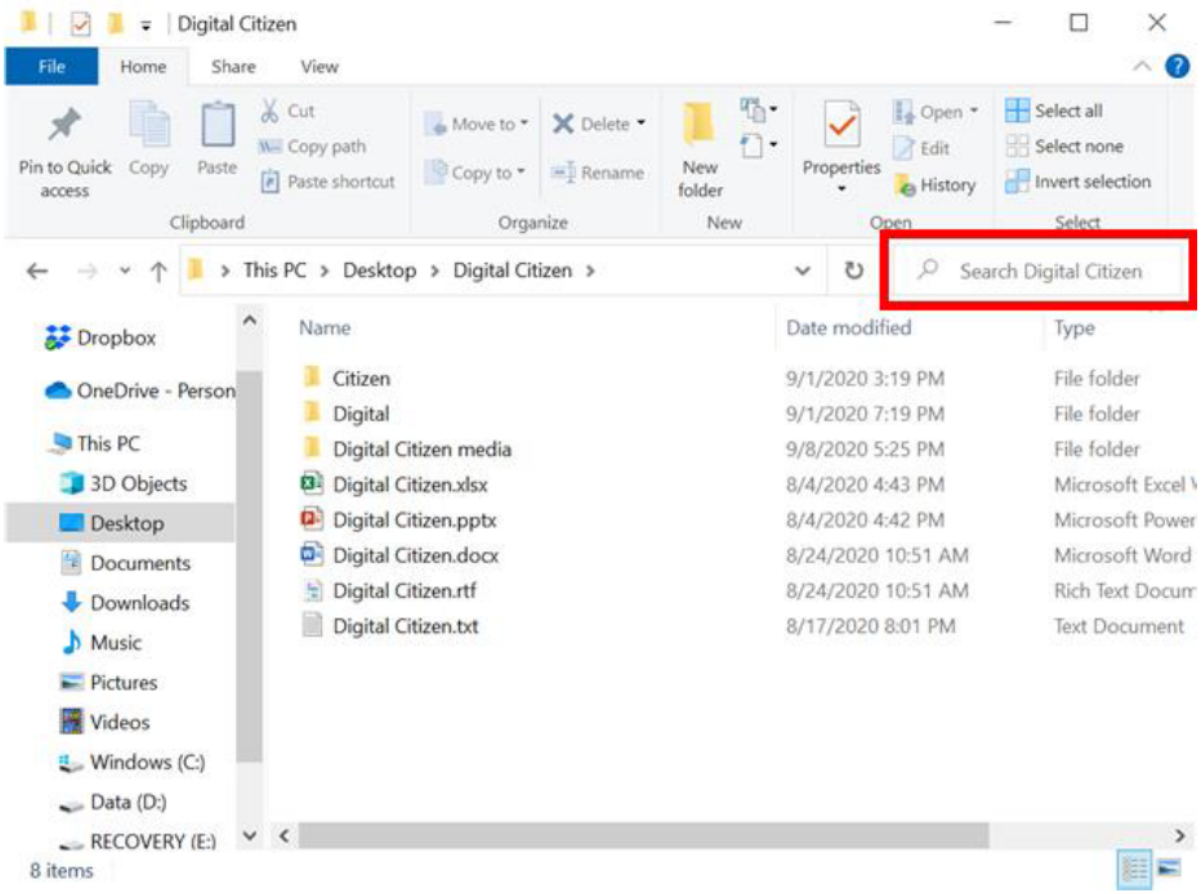
DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Dalam tugas besar ini, kami 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), kami dapat menelusuri folder-folder yang ada pada direktori untuk mendapatkan direktori yang kami inginkan. Kami juga diminta untuk memvisualisasikan hasil dari pencarian folder tersebut dalam bentuk pohon.

Selain pohon, kami 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*.

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.

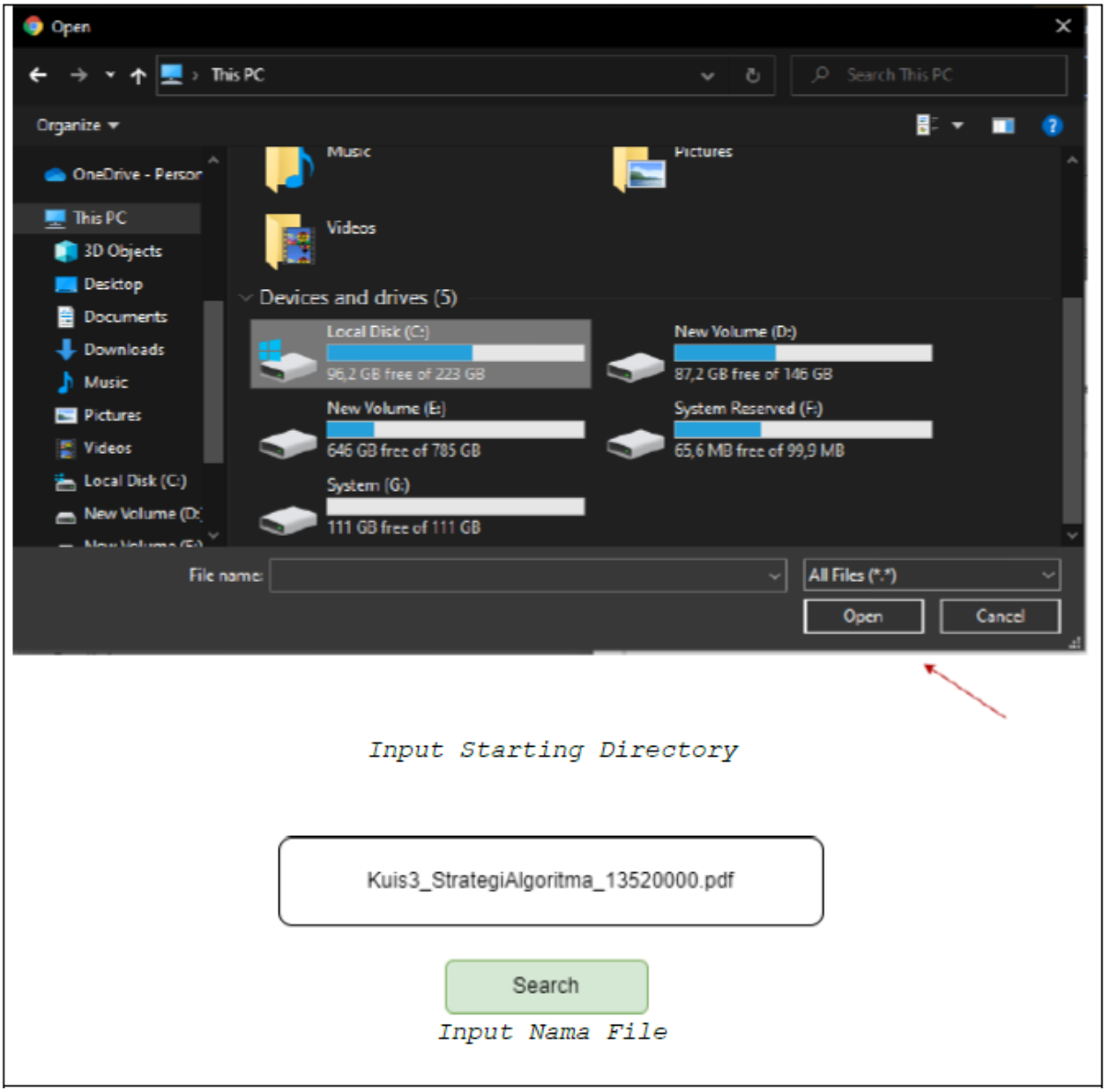


Gambar 1. Fitur Search pada Windows 10 File Explorer

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.

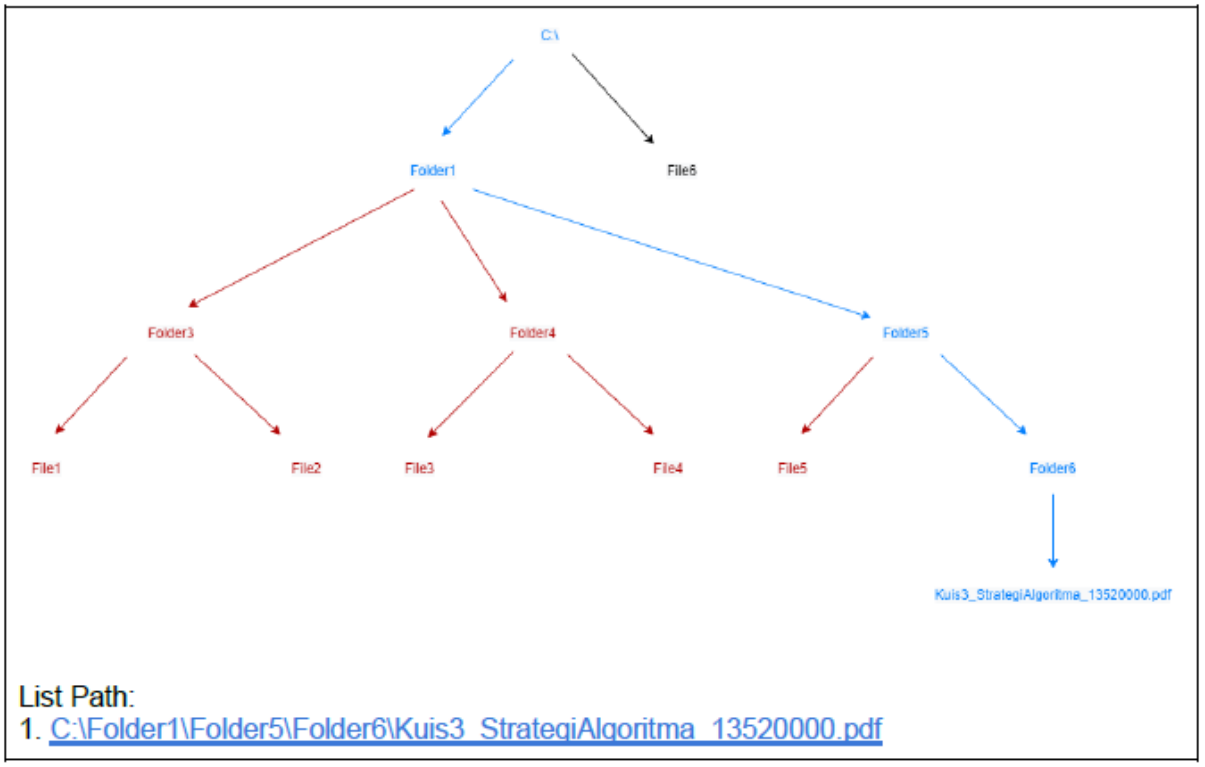
Fitur ini diimplementasikan dengan teknik folder crawling, di mana mesin komputer akan mulai mencari file yang sesuai dengan query mulai dari starting directory hingga seluruh children dari starting directory tersebut sampai satu file pertama/seluruh file ditemukan atau tidak ada file yang ditemukan. 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 2. Contoh input program

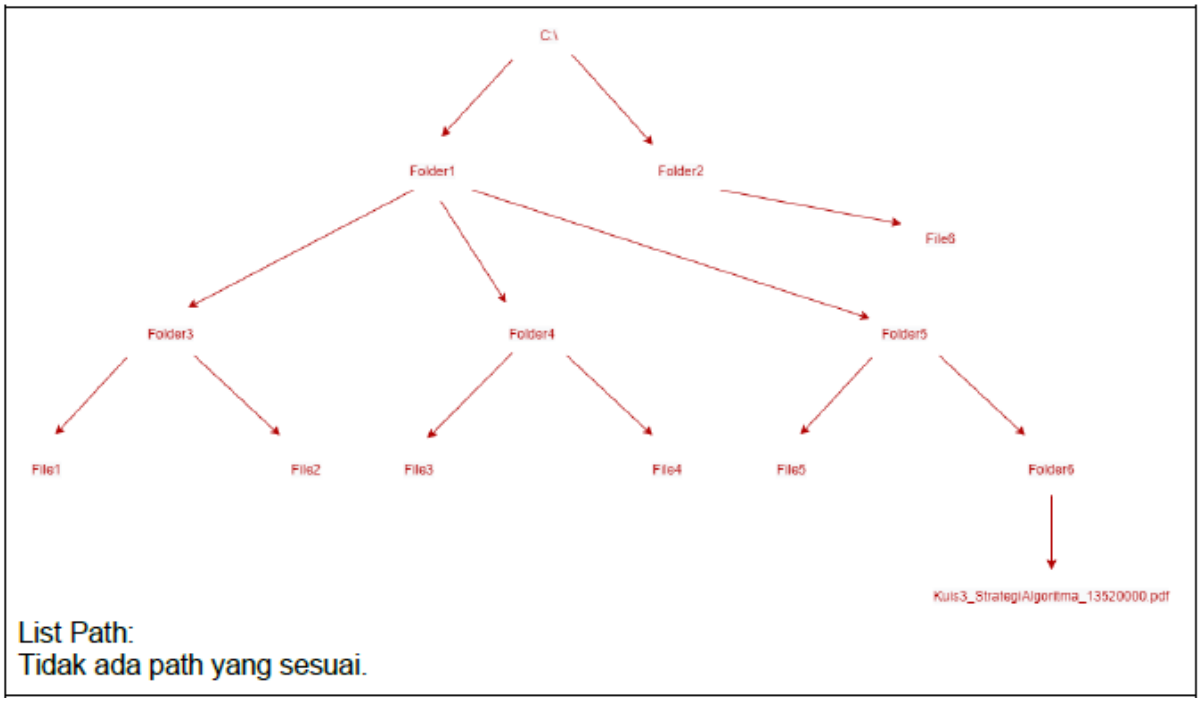
Contoh output aplikasi:



Gambar 3. Contoh output program

Misalnya pengguna ingin mengetahui langkah folder crawling untuk menemukan file Kuis3_StrategiAlgoritma_13520000.pdf. Maka, path pencarian DFS adalah sebagai berikut. C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3_StrategiAlgoritma_13520000.pdf.

Pada gambar di atas, rute yang dilewati pada pencarian DFS diwarnai dengan warna merah. Sedangkan, rute untuk menuju tempat file berada diberi warna biru. Rute yang masuk antrian tapi belum diperiksa diberi warna hitam. Anda bebas menentukan warnanya asalkan dibedakan antara ketiga hal tersebut.

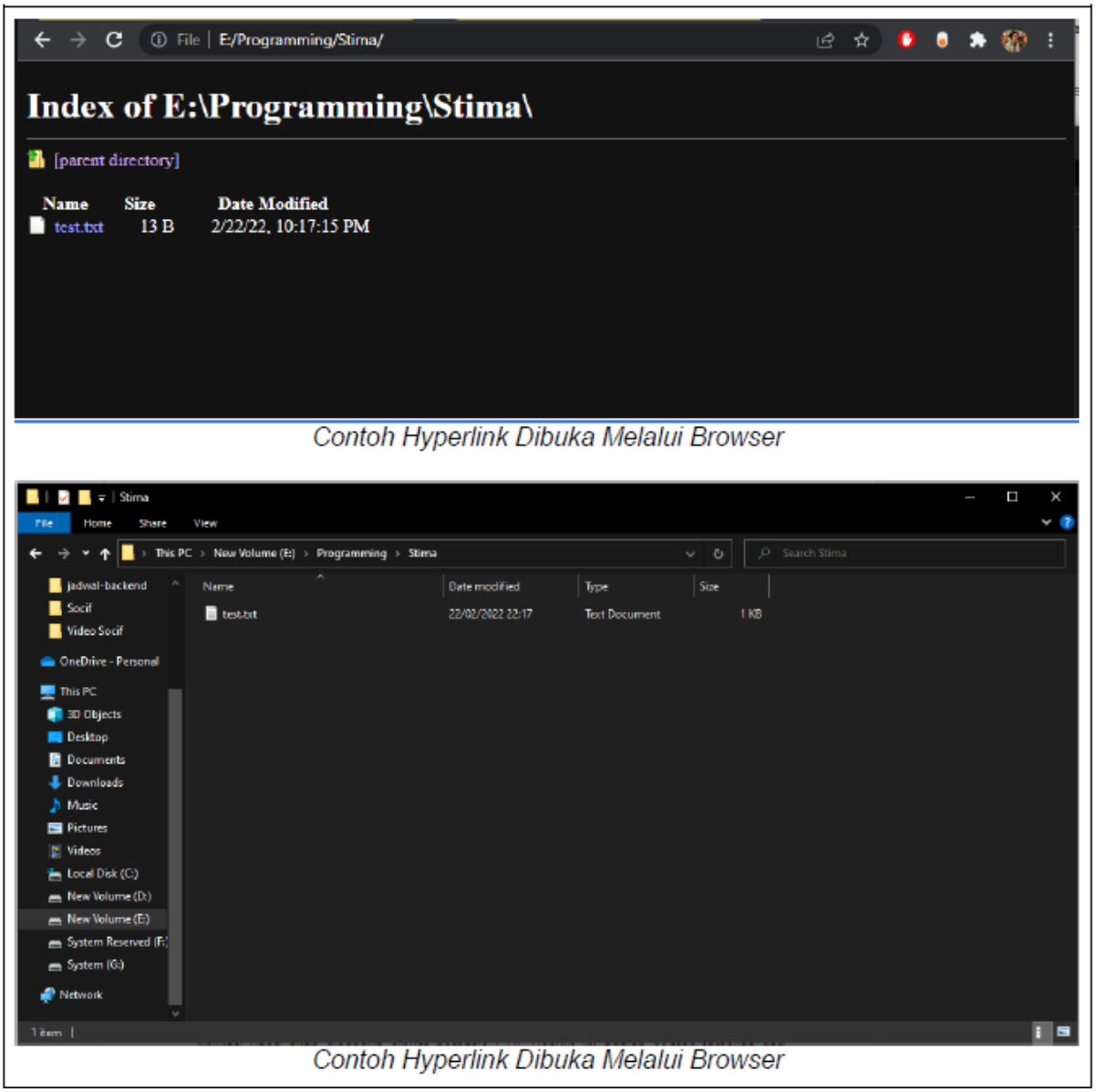


Gambar 4. Contoh output program jika file tidak ditemukan

Jika file yang ingin dicari pengguna tidak ada pada direktori file, misalnya saat pengguna mencari Kuis3Probstat.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 5. Contoh ketika hyperlink di-klik

BAB II
LANDASAN TEORI

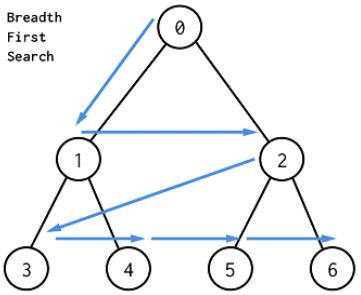
1. Dasar Teori (Graf Traversal, BFS, dan DFS) Secara Umum

- Graf Traversal

  Bila dilihat dari definisinya, Graf Traversal mengartikan graf yang diakses secara satu-satu secara sistematis, yang mengartikan dengan caranya tertentu. Cara tersebut dibagi menjadi dua, yaitu BFS (Breadth First Search) dan DFS (Depth First Search).

- BFS(Breadth First Search)

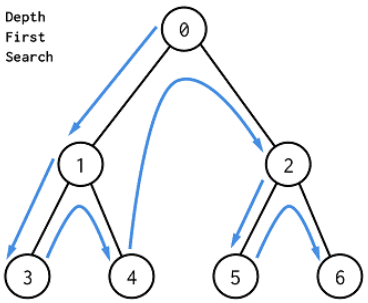
  Algoritma BFS mendeskripsikan cara atau langkah-langkah sebuah algoritma untuk melakukan pengaksesan simpul dari sebuah graf dengan cara mengecek keseluruhan dari anak terlebih dahulu, setelah sudah, maka akan dicari anak dari anak itu lagi, sehingga, mengecek yang selevel dulu baru kebawah.



Gambar 6. BFS

- DFS(Depth First Search)

  Algoritma DFS mendeskripsikan cara atau langkah-langkah sebuah algoritma untuk melakukan pengaksesan simpul dari sebuah graf dengan cara mengakses langsung anak dari sebuah simpul jika ada, diteruskan hingga tidak memiliki anak lagi, jika tidak memiliki anak, akan backtrack ke parent dan mencari ke anak lagi yang belum pernah dicari. Langkah ini Diteruskan hingga tidak ada lagi yang bisa diakses.



Gambar 7. DFS

2. Penjelasan Singkat C# Desk*top Application Development*

Program yang dibuat sebagai pengerjaan dari tugas besar ini menggunakan Visual Studio dengan bahasa C#. Dengan menggunakan MSAGL sebagai *library* yang berguna untuk memvisualisasikan graf pohon sebagai direktori file yang didapatkan.

Framework yang digunakan adalah .NET -pengembang menggunakan versi 4.7.2-, tetapi seharusnya masih bisa dijalankan di .NET CORE (.NET versi lama). *Framework* ini dipilih karena mampu mengintegrasikan semua kebutuhan dalam tugas besar ini dengan alur penggunaan yang paling sederhana.

Sementara itu, untuk *user interface*, pengembang menggunakan Windows Forms. Selain kelemahan berupa tidak mampunya program kompatibel secara *cross-platform* (pada sistem operasi selain Windows), Windows Forms sangat memenuhi kebutuhan dalam tugas besar ini, lagi-lagi dengan alur penggunaan yang paling sederhana, tidak seperti pilihan lain seperti WPF yang mengharuskan pengembang memahami XAML lebih dulu.

Sedikit informasi tambahan, pengguna sempat melakukan eksplorasi mengenai kakas *user interface* seperti Avalonia UI dan Gtk. Kendala utama dari Avalonia UI ialah -setidaknya ketika pengembang mencobanya- ialah belum *native*-nya kakas pada Visual Studio 2019 untuk sistem operasi MacOS, sehingga pada akhirnya tidak jauh berbeda dengan Windows Forms, sedangkan kendala utama dari Gtk ialah sulitnya visualisasi graf dengan MSAGL, terlebih karena pengembang harus memahami lebih dulu kakas yang di-*wrap* oleh Gtk untuk *Drawing*, seperti Cairo, serta minimnya dokumentasi dari MSAGL.

BAB III
ANALISIS PEMECAHAN MASALAH

Masalah yang diberikan pada tubes kali ini adalah pencarian suatu file dengan nama tertentu pada folder, sistem yang bekerja dengan fungsi search yang ada pada File Explorer. Permasalahan ini memiliki spesifikasi seperti graf yang statik(tidak berubah-ubah). Dengan spesifikasi tersebut, solusi yang bisa digunakan adalah BFS dan/atau DFS yang berfungsi untuk menelusuri isi dari folder sehingga didapatkannya sebuah solusi, baik dengan ditemukannya file yang ingin dicari atau tidak menemukannya sama sekali.

Dengan menggunakan list of string untuk mendapatkan semua direktori atau lokasi dari suatu file, akan di graf-kan data tersebut pada graf pohon dengan cara-cara tertentu. Jika sudah melakukan pengecekkan, maka panah akan berubah warna, merah jika tidak ditemukannya file dalam direktori tersebut, hijau jika ditemukan file yang dicari dalam folder tersebut, abu-abu jika belum dilakukannya pencarian pada direktori tersebut (terjadi karena tidak all-occurrence).

BAB IV
IMPLEMENTASI DAN PENGUJIAN

1. Implementasi Program (Pseudocode Program Utama)

BFS.cs

```
using Folder;

namespace BFS_find
{
    class BFS
    {
        public string FileName { get; set; } = string.Empty;
        public bool Found;
        public bool Occur;
        public Queue<folder> BFS_show = new Queue<folder>();

        public BFS(string nama, bool Occur)
        {
            FileName = nama;
            Found = false;
            this.Occur = Occur;
        }

        public void BFS_search(string path)
        {
            // Queue buat alur BFS
            Queue<string> Visited = new Queue<string>();

            string[] folder = Directory.GetDirectories(path, "*",
SearchOption.TopDirectoryOnly);

            BFS_show.Enqueue(new folder("", path));

            foreach (string fold in folder)
            {
                Visited.Enqueue(fold);
                BFS_show.Enqueue(new folder(path, fold));
            }

            string[] files = Directory.GetFiles(path, "*.*",
SearchOption.TopDirectoryOnly);
            foreach (string file in files)
            {
                Visited.Enqueue(file);
                BFS_show.Enqueue(new folder(path, file));
            }

            while (Visited.Count > 0)
            {
                string now = Visited.Dequeue();
```

```
                FileAttributes trib = File.GetAttributes(now);

                //Folder
                if (trib.HasFlag(FileAttributes.Directory))
                {
                    string[] fold = Directory.GetDirectories(now, "*",
SearchOption.TopDirectoryOnly);
                    foreach (string fd in fold)
                    {
                        Visited.Enqueue(fd);
                        BFS_show.Enqueue(new folder(now, fd));
                    }
                    string[] files_child = Directory.GetFiles(now, "*.*",
SearchOption.TopDirectoryOnly);
                    foreach (string file in files_child)
                    {
                        Visited.Enqueue(file);
                        BFS_show.Enqueue(new folder(now, file));
                    }
                }
                //File
                else
                {
                    if (Found == false)
                    {
                        if (Path.GetFileName(now) == FileName)
                        {
                            //Jika All_Occurance False, Pragram akan jalan
terus
                            if (Occur == false)
                            {
                                System.Environment.Exit(0);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

DFS.cs

```
using Folder;

namespace DFS_find
{
    public class DFS
    {
        public string FileName { get; set; } = string.Empty;
        public bool found;
        public bool Occur;
        public Queue<folder> urutan = new Queue<folder>();
```

```
        public DFS(string nama, bool Occur)
        {
            FileName = nama;
            found = false;
            this.Occur = Occur;
        }

        public void DFS_start(string Path)
        {
            this.DFS_Search(new DirectoryInfo(Path));
        }
        public void DFS_Search(DirectoryInfo directory)
        {
            string[] Files = Directory.GetFiles(directory.FullName, "*.*");
            if (urutan.Count == 0)
                urutan.Enqueue(new folder("", directory.FullName));

            foreach (string file in Files)
            {
                urutan.Enqueue(new folder(directory.FullName, file));
                if (found == false)
                {
                    if (Path.GetFileName(file) == FileName)
                    {
                        //Jika All_Occurance False, Pragram akan jalan terus
                        if (Occur == false)
                        {
                            System.Environment.Exit(0);
                        }
                    }
                }
            }

            DirectoryInfo[] children = directory.GetDirectories();
            if (found == false)
            {
                foreach (DirectoryInfo child in children)
                {
                    urutan.Enqueue(new folder(directory.FullName,
child.FullName));
                    this.DFS_Search(child);
                }
            }
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Windows.Forms;
using Microsoft.WindowsAPICodePack.Dialogs;
using System.Diagnostics;

namespace MariKitaCari
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            //create a viewer object
            Microsoft.Msagl.GraphViewerGdi.GViewer viewer = new
Microsoft.Msagl.GraphViewerGdi.GViewer();
            //create a graph object
            Microsoft.Msagl.Drawing.Graph graph = new
Microsoft.Msagl.Drawing.Graph("graph");
            //create the graph content
            // do nothing (blank)
            //bind the graph to the viewer
            viewer.Graph = graph;
            //associate the viewer with the form
            panel1.SuspendLayout();
            viewer.Dock = System.Windows.Forms.DockStyle.Fill;
            panel1.Controls.Add(viewer);
            panel1.ResumeLayout();
            //ScrollBar vScrollBar1 = new VScrollBar();
            //vScrollBar1.Dock = DockStyle.Right;
            //vScrollBar1.Scroll += (sender, eScroll) => {
flowLayoutPanel1.VerticalScroll.Value = vScrollBar1.Value; };
            //flowLayoutPanel1.Controls.Add(vScrollBar1);
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            labelPath.Text = "";
            labelOutputTimeSpent.Text = "";
            linkLabelList.Text = "";
        }

        private void ButtonChangeFolder_Click(object sender, EventArgs e)
        {
            /*
            // select and acquire starting folder path (Windows 7 below but
better UI)
            var dialog = new CommonOpenFileDialog();
            dialog.IsFolderPicker = true;
            CommonFileDialogResult result = dialog.ShowDialog();
            labelPath.Text = dialog.FileName;
```

```
            */

            // select and acquire starting folder path (Windows 7 above but
worse UI)
            using (var folderbdialog = new FolderBrowserDialog()){
                DialogResult dialogresult = folderbdialog.ShowDialog();
                labelPath.Text = folderbdialog.SelectedPath;
            }
        }

        private void ButtonSearch_Click(object sender, EventArgs e)
        {
            // add timer
            Stopwatch Timer = new Stopwatch();
            if (labelPath.Text == "") // IsNullOrEmpty()
            {
                MessageBox.Show("Pilih starting folder, mas!", "Gagal Ngab!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            else if (textBoxFileName.Text == "") // IsNullOrEmpty()
            {
                MessageBox.Show("Isi file name, sob!", "Gagal Ngab!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            else
            {
                string root_folder = labelPath.Text;

                // clear list to reuse
                List<string> listLink = new List<string>();
                List<string> listPath = new List<string>();
                listLink.Clear();
                listPath.Clear();
                linkLabelList.Text = "";
                linkLabelList.Links.Clear();

                if (radiobtnBFS.Checked)
                {
                    //create a viewer object
                    Microsoft.Msagl.GraphViewerGdi.GViewer viewer = new
Microsoft.Msagl.GraphViewerGdi.GViewer();
                    //create a graph object
                    Microsoft.Msagl.Drawing.Graph graph = new
Microsoft.Msagl.Drawing.Graph("graph");
                    viewer.Graph = null;
                    panel1.Controls.Clear();
                    Timer.Start();
                    BFS bfs_search = new BFS(textBoxFileName.Text,
checkBoxOccurence.Checked);
                    // Queue<string> solution = new Queue<string>(); //Queue
of solution
                    bool temu = false; // handle all occur atau engga
```

```
                    bfs_search.BFS_Search(root_folder);
                    int i = 0;
                    foreach (Folder anak in bfs_search.bfs_show)
                    {
                        foreach (Folder ortu in bfs_search.bfs_show)
                        {
                            if (anak.parent == ortu.direct)
                            {
                                if (bfs_search.jalur.Contains(anak.direct))
                                {
                                    graph.AddEdge(ortu.direct,
anak.direct).Attr.Color = Microsoft.Msagl.Drawing.Color.Green;
                                }
                                else if (temu == true && bfs_search.all_occur
== false)
                                {
                                    graph.AddEdge(ortu.direct,
anak.direct).Attr.Color = Microsoft.Msagl.Drawing.Color.Gray;
                                }
                                else
                                {
                                    graph.AddEdge(ortu.direct,
anak.direct).Attr.Color = Microsoft.Msagl.Drawing.Color.Red;
                                }
                                graph.FindNode(anak.parent).Label.Text = new
DirectoryInfo(anak.parent).Name;
                                graph.FindNode(anak.direct).Label.Text = new
DirectoryInfo(anak.direct).Name;
                                break;
                            }
                        }
                        if (String.Compare(Path.GetFileName(anak.direct),
bfs_search.Namafile) == 0 && i < bfs_search.solution.Count())
                        {
                            graph.FindNode(anak.direct).Attr.FillColor =
Microsoft.Msagl.Drawing.Color.Green;
                            temu = true;
                            i++;
                            listPath.Add(anak.direct);
                            listLink.Add(anak.parent);
                        }
                    }

                    int charCounter = 0;
                    int countLink = 0;
                    string[] arrayLink = listLink.ToArray();
                    string[] arrayPath = listPath.ToArray();
                    foreach (var link in arrayLink)
                    {
                        linkLabelList.Text += arrayPath[countLink];
                        linkLabelList.Text += "\n";
                        linkLabelList.Links.Add(charCounter,
arrayPath[countLink].Length, arrayLink[countLink]);
```

```
                        charCounter += (arrayPath[countLink].Length + 1);
                        countLink++;
                    }
                    linkLabelList.LinkClicked += (s, eLink) =>
Process.Start((string)eLink.Link.LinkData);

                    Timer.Stop();
                    labelOutputTimeSpent.Text =
Timer.Elapsed.TotalSeconds.ToString() + " detik";
                    //bind the graph to the viewer
                    viewer.Graph = graph;
                    //associate the viewer with the form
                    panel1.SuspendLayout();
                    viewer.Dock = System.Windows.Forms.DockStyle.Fill;
                    panel1.Controls.Add(viewer);
                    panel1.ResumeLayout();
                }
                else if (radiobtnDFS.Checked)
                {
                    //create a viewer object
                    Microsoft.Msagl.GraphViewerGdi.GViewer viewer = new
Microsoft.Msagl.GraphViewerGdi.GViewer();
                    //create a graph object
                    Microsoft.Msagl.Drawing.Graph graph = new
Microsoft.Msagl.Drawing.Graph("graph");
                    viewer.Graph = null;
                    panel1.Controls.Clear();
                    Timer.Start();
                    DFS dfs_search = new DFS(textBoxFileName.Text,
checkBoxOccurence.Checked);
                    Queue<string> solution = new Queue<string>(); //Queue of
solution
                    bool temu = false; // handle all occur atau engga

                    // create graph content
                    dfs_search.DFS_Start(root_folder);
                    int i = 0;
                    foreach (Folder anak in dfs_search.urutan)
                    {
                        foreach (Folder ortu in dfs_search.urutan)
                        {
                            if (anak.parent == ortu.direct)
                            {
                                if (dfs_search.jalur.Contains(anak.direct))
                                {
                                    graph.AddEdge(ortu.direct,
anak.direct).Attr.Color = Microsoft.Msagl.Drawing.Color.Green;
                                }
                                else if (temu == true && dfs_search.all_occur
== false)
                                {
                                    graph.AddEdge(ortu.direct,
anak.direct).Attr.Color = Microsoft.Msagl.Drawing.Color.Gray;
```

```
                                }
                                else
                                {
                                    graph.AddEdge(ortu.direct,
anak.direct).Attr.Color = Microsoft.Msagl.Drawing.Color.Red;
                                }
                                graph.FindNode(anak.parent).Label.Text = new
DirectoryInfo(anak.parent).Name;
                                graph.FindNode(anak.direct).Label.Text = new
DirectoryInfo(anak.direct).Name;
                                break;
                            }
                        }
                        if (String.Compare(Path.GetFileName(anak.direct),
dfs_search.Namafile) == 0 && i < dfs_search.solution.Count())
                        {
                            graph.FindNode(anak.direct).Attr.FillColor =
Microsoft.Msagl.Drawing.Color.Green;
                            temu = true;
                            i++;
                            listPath.Add(anak.direct);
                            listLink.Add(anak.parent);
                        }
                    }

                    int charCounter = 0;
                    int countLink = 0;
                    string[] arrayLink = listLink.ToArray();
                    string[] arrayPath = listPath.ToArray();
                    foreach (var link in arrayLink)
                    {
                        linkLabelList.Text += arrayPath[countLink];
                        linkLabelList.Text += "\n";
                        linkLabelList.Links.Add(charCounter,
arrayPath[countLink].Length, arrayLink[countLink]);
                        charCounter += (arrayPath[countLink].Length + 1);
                        countLink++;
                    }
                    linkLabelList.LinkClicked += (s, eLink) =>
Process.Start((string)eLink.Link.LinkData);

                    Timer.Stop();
                    labelOutputTimeSpent.Text =
Timer.Elapsed.TotalSeconds.ToString() + " detik";
                    //bind the graph to the viewer
                    viewer.Graph = graph;
                    //associate the viewer with the form
                    panel1.SuspendLayout();
                    viewer.Dock = System.Windows.Forms.DockStyle.Fill;
                    panel1.Controls.Add(viewer);
                    panel1.ResumeLayout();
                }
                else
```

```
                {
                    MessageBox.Show("Pilih salah satu metode, gan!", "Gagal
Ngab!", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }
    }
}
```

2. Penjelasan Struktur Data dan Spesifikasi Program

BFS dan DFS

Program ini terdiri dari beberapa file dimana pemrosesan BFS dan DFS dipisah menjadi 2 file untuk memodularkan program. Pada file BFS dan DFS kita membuat class dengan beberapa atribut dan method.

```
class BFS
{
    public string Namafile { get; set; } = string.Empty;
    public bool found;
    public bool all_occur;
    public Queue<Folder> bfs_show = new Queue<Folder>();
    public Queue<string> solution = new Queue<string>();
    public Queue<string> jalur = new Queue<string>();

    public BFS(string nama, bool all_occur)
    {
        Namafile = nama;
        found = false;
        this.all_occur = all_occur;
    }

    public int substring(string sub, string main)
    {
        int M = sub.Length;
```

```
public class DFS
{
    public string Namafile { get; set; } = string.Empty;
    public bool found;
    public bool all_occur;
    public Queue<Folder> urutan = new Queue<Folder>();
    public Queue<string> solution = new Queue<string>();
    public Queue<string> jalur = new Queue<string>();

    public DFS(string nama, bool all_occur)
    {
        Namafile = nama;
        found = false;
        this.all_occur = all_occur;
    }
    public int substring(string sub, string main)
    {
        int M = sub.Length;
        int N = main.Length;
```

Pada dasarnya, kedua metode ini memiliki atribut dan class method yang sama. Kedua class ini bertujuan untuk menghasilkan sebuah urutan pencarian berdasarkan kondisi kondisi yang ada seperti semua kemunculan file yang dicari atau tidak. Ada 3 buah variabel *Queue* di atribut class, *Queue* bfs_show dan urutan berguna untuk menyimpan urutan pencarian.

```
public void DFS_Search(DirectoryInfo dir)
{
    string[] files = Directory.GetFiles(dir.FullName, "*.*");
    if (urutan.Count == 0)
        urutan.Enqueue(new Folder("", dir.FullName));

    foreach (string file in files)
    {
        urutan.Enqueue(new Folder(dir.FullName, file));
        if (string.Compare(Path.GetFileName(file), Namafile) =
        {
            if (this.all_occur == false)
            {
                this.found = true;
            }
            solution.Enqueue(file);
        }
    }
```

```
string[] fold = Directory.GetDirectories(now, "*",
foreach (string fd in fold)
{
    dir_visited.Enqueue(fd);
    bfs_show.Enqueue(new Folder(now, fd));
}
string[] files_child = Directory.GetFiles(now, "*.
foreach (string file in files_child)
{
    dir_visited.Enqueue(file);
    bfs_show.Enqueue(new Folder(now, file));
}
```

Saat mencari rute pencarian tersebut, kita mempertimbangkan kondisi pencarian untuk semua kemunculan atau tidak

```
if (string.Compare(Path.GetFileName(now), Namafile) == 0 && this.found == false)
{
    if (this.all_occur == false)
    {
        this.found = true;
    }
    solution.Enqueue(now);
}
```

Hal itu berlaku untuk kedua metode baik DFS maupun BFS. Setelah menemukan file yang dicari, maka dimulai pencarian jalur mana saja untuk menuju ke file hasil dari folder akar dengan memanfaatkan pengecekan substring.

```
public int substring(string sub, string main)
{
    int M = sub.Length;
    int N = main.Length;

    /* A loop to slide pat[] one by one */
    for (int i = 0; i <= N - M; i++)
    {
        int j;

        /* For current index i, check for
        pattern match */
        for (j = 0; j < M; j++)
            if (main[i + j] != sub[j])
                break;

        if (j == M)
            return i;
    }

    return -1;
}
```

```
foreach (string sol in this.solution)
{
    foreach (Folder ok in this.bfs_show)
    {
        int coba = this.substring(ok.direct, sol);
        if (coba != -1)
        {
            this.jalur.Enqueue(ok.direct);
        }
    }
}
```

Hal tersebut berlaku untuk kedua metode guna untuk mewarnai jalur pada visualisasi graph.

Visualisasi Graph

Pada program visualisasi graph, kita hanya berfokus pada pembuatan jalur dengan mewarnai setiap edge yang bersangkutan.

- Edge Hijau = Jalur menuju file yang dicari
- Edge Merah = Jalur yang sudah ditelusuri namun tidak menemukan file yang dicari

- Edge Abu-Abu = Jalur yang belum ditelusuri karena sudah menemukan hasil dan kondisi pencarian yaitu hanya mencari satu kemunculan file.

```
foreach (Folder anak in bfs_search.bfs_show)
{
    foreach (Folder ortu in bfs_search.bfs_show)
    {
        if (anak.parent == ortu.direct)
        {
            if (bfs_search.jalur.Contains(anak.direct))
            {
                graph.AddEdge(ortu.direct, anak.direct).Attr.Color = Microsoft.Msagl.Drawing.Color.Green;
            }
            else if (temu == true && bfs_search.all_occur == false)
            {
                graph.AddEdge(ortu.direct, anak.direct).Attr.Color = Microsoft.Msagl.Drawing.Color.Gray;
            }
            else
            {
                graph.AddEdge(ortu.direct, anak.direct).Attr.Color = Microsoft.Msagl.Drawing.Color.Red;
            }
            graph.FindNode(anak.parent).Label.Text = new DirectoryInfo(anak.parent).Name;
            graph.FindNode(anak.direct).Label.Text = new DirectoryInfo(anak.direct).Name;
            break;
        }
    }
    if (String.Compare(Path.GetFileName(anak.direct), bfs_search.Namafile) == 0 && i < bfs_search.solution.Count())
    {
        graph.FindNode(anak.direct).Attr.FillColor = Microsoft.Msagl.Drawing.Color.Green;
        temu = true;
        i++;
        listPath.Add(anak.direct);
        listLink.Add(anak.parent);
    }
}
```

Untuk mewarnai jalur pencarian yang benar, kita cek apakah path node yang akan kita buat tersebut, dimiliki juga oleh *Queue* jalur, jika iya maka edge yang bersangkutan diberi warna hijau. Untuk kondisi pencarian hanya satu kemunculan, maka kita cek dengan variabel “temu” untuk indikasi apakah sudah menemukan file atau belum, jika sudah maka selanjutnya setiap edge akan diberi warna abu-abu. Untuk kondisi sisanya adalah untuk mewarna edge bersangkutan dengan warna merah.

Untuk *path node* yang mengandung file yang dicari, maka akan diberi *fill color* berwarna hijau.

GUI

Secara garis besar, ketika *build* program, file yang pertama kali dijalankan adalah Program.cs yang akan memanggil Form1.cs. Pada Form1.cs, terdapat konstruktor untuk membangkitkan setiap objek pada GUI, seperti *button*, untuk pertama kalinya. Terdapat pula fungsi *load* untuk menginisialisasi label dengan nilai default. Selain itu, terdapat fungsi untuk setiap *event* yang perlu di-*handle*, misalnya *event* ketika *button* ditekan oleh pengguna program. *Event* yang di-handle pada program ini hanyalah ketika *button* “Choose Folder...” dan “Search” ditekan.

Struktur data yang dipakai hanyalah list of string untuk menyimpan path dari file sekaligus *link parent directory* dari file yang telah ditemukan (nama file sama dengan yang diisikan pengguna). Namun, terdapat banyak *predefined* objek yang digunakan, yaitu sebagai berikut.

- Graph dan GViewer dari kakas MSAGL, untuk memvisualisasikan graf.
- FolderBrowserDialog, untuk membuka direktori dan memilih *starting folder*.
- StopWatch, untuk mencatat waktu proses algoritma.

- Button, RadioButton, CheckBox, TableLayoutPanel, FlowLayoutPanel, label, labelLink, TextBox, dan sebagainya, sebagai komponen utama dari GUI. Semuanya merupakan bawaan Windows Forms pada .NET Framework.

Keterangan lebih lengkapnya dapat dilihat pada Bab IV Nomor 1. Adapun seluruh source code untuk menentukan tata letak dan *properties* dapat tercatat pada file Form1.Designer.cs.

3. Penjelasan Tata Cara Penggunaan Program (Interface dan Fitur)

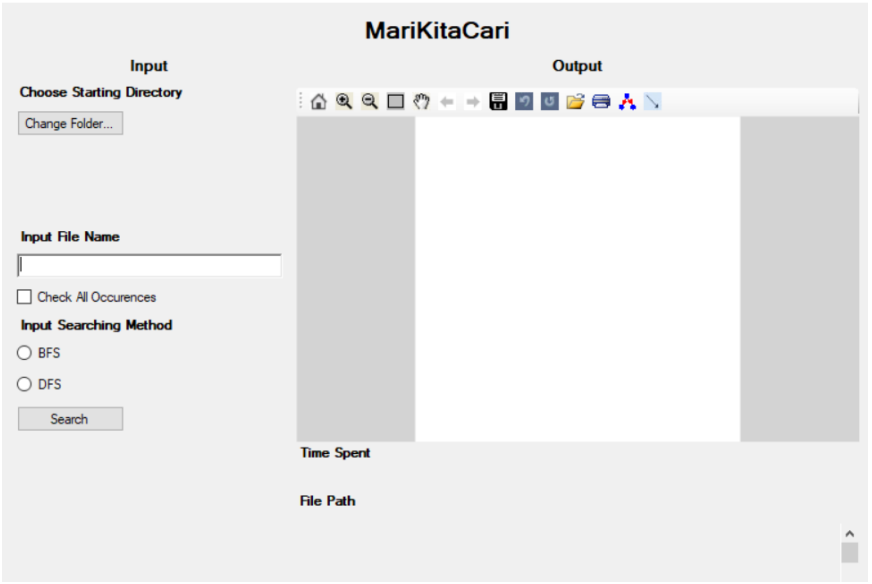
Untuk menggunakan program MariKitaCari, pengguna tidak perlu membuild ulang program, tetapi cukup me-run file executable MariKitaCari.exe yang terletak pada directory src > bin > Debug > MariKitaCari.exe. Akan tetapi, karena mengandung *dependencies* berupa *package* C#, program ini belum dapat dijalankan di luar *directory*.

Pengguna akan disambut dengan tampilan awal MariKitaCari (lihat Bab IV Nomor 4). Lalu, pengguna wajib memilih *starting folder* pencarian melalui *button* "Choose Folder..." (atau *key* 'C'), mengisi nama file yang ingin dicari melalui *text box*, dan memilih metode pencarian, yaitu antara BFS atau DFS, melalui *radio button*. Apabila pengguna lupa mengisi salah satu dari tiga kewajiban tersebut, akan terdapat pesan *error* yang dilengkapi penjelasan (lihat Bab IV Nomor 4).

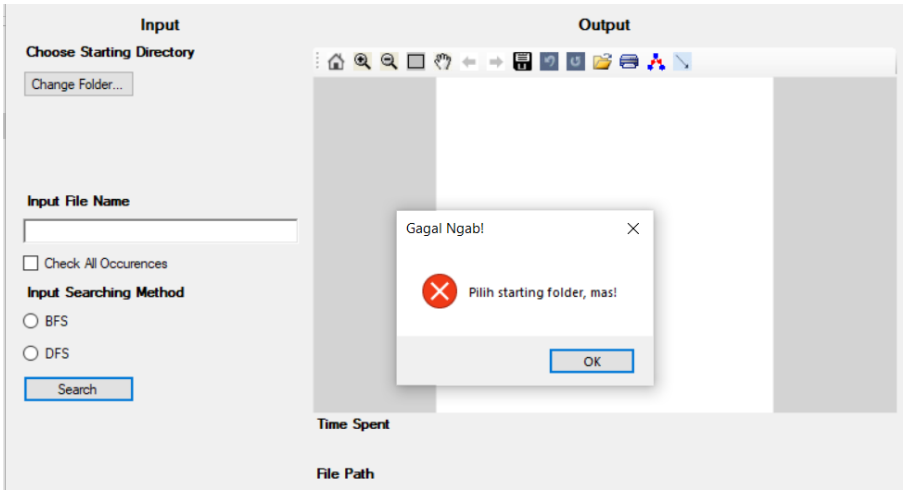
Pengguna dapat mengecheck kotak "Check All Occurrence" apabila ingin mencari semua file dengan nama file yang telah diisi, atau sebaliknya, tidak mengecheck kotak apabila hanya ingin mencari satu file pertama yang ditemukan oleh algoritma.

Pengguna dapat langsung menekan *button* "Search" (atau *key* 'S') untuk mulai mencari file. Program akan berjalan, lalu langsung menampilkan visualisasi graf (atau tepatnya *tree*) dari *directory*, dengan root berupa *starting folder* yang dipilih, dan keterangan warna sesuai Bab IV Nomor 2 di atas.

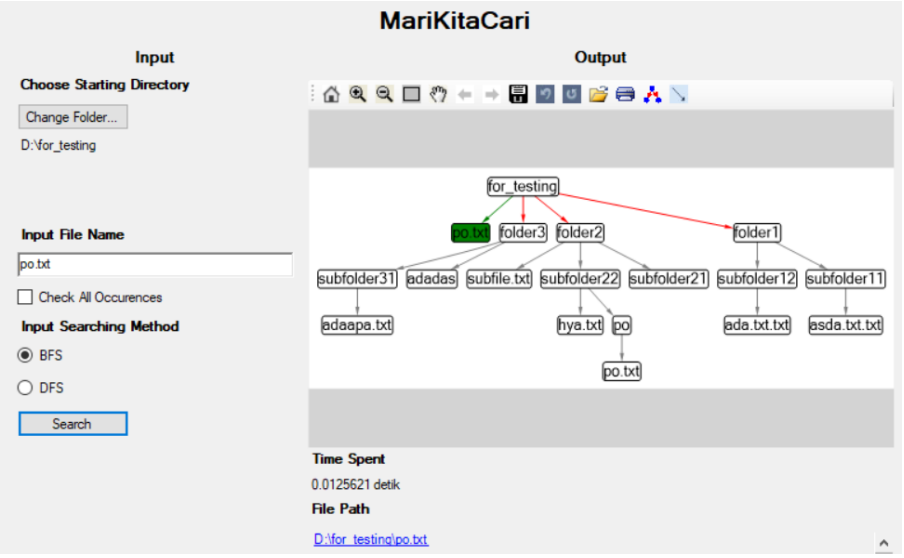
4. Hasil Pengujian



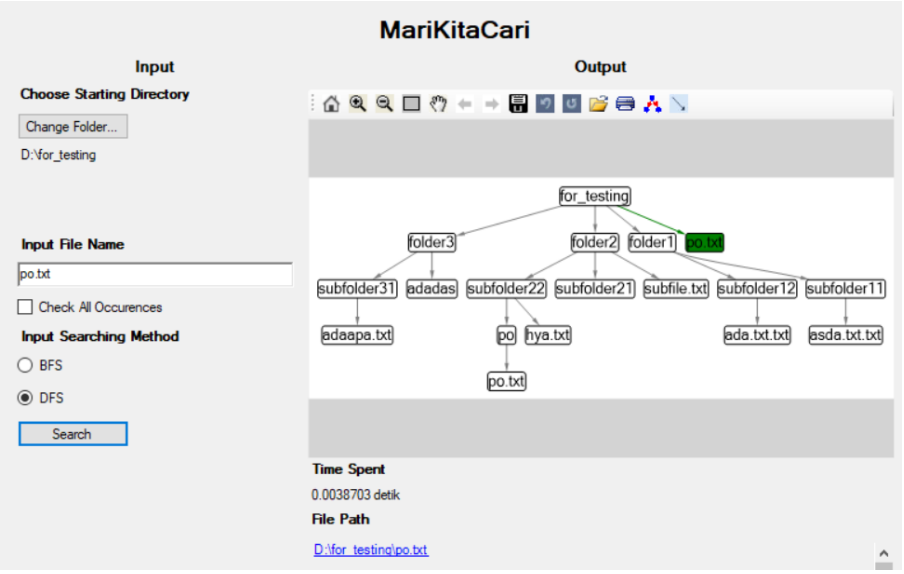
Kasus nol, tampilan awal program MariKitaCari.



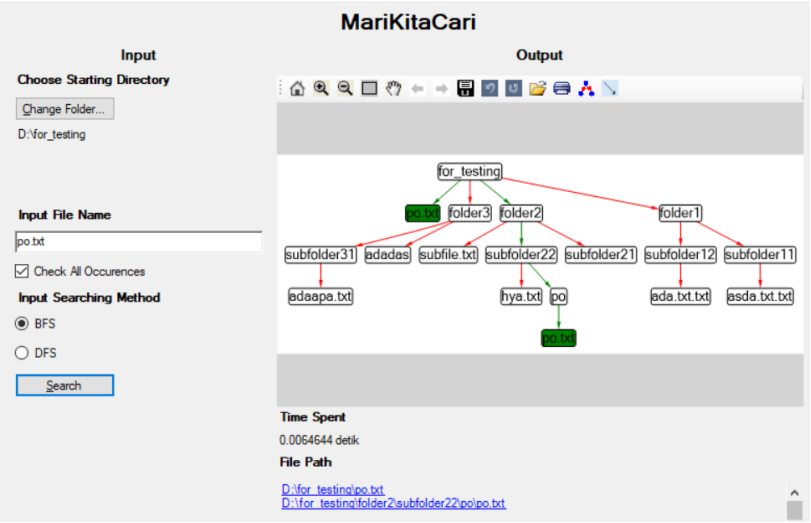
Kasus pertama, pengguna lupa mengisi salah satu dari ketiga input wajib (starting folder, nama file, dan metode pencarian). Akan ditampilkan pesan error seperti pada gambar berikut.



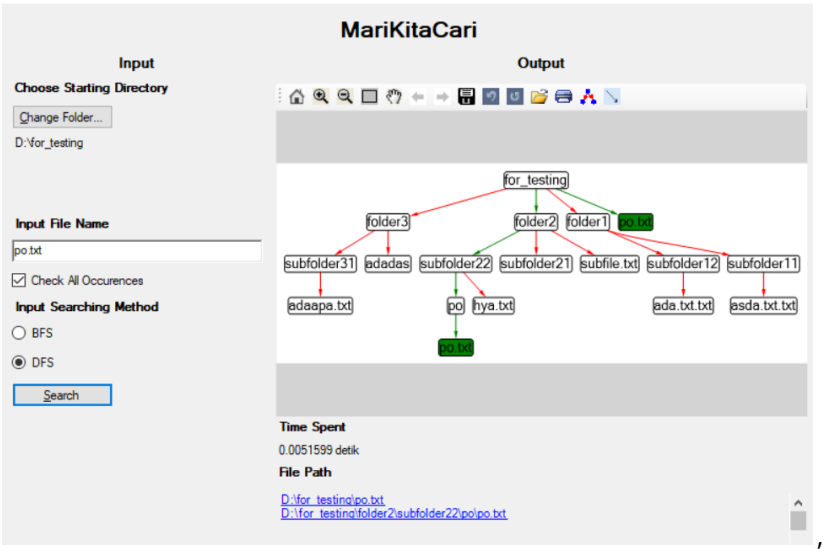
Kasus kedua, pengguna mencari sebuah file dengan metode bfs dan untuk hanya satu kemunculan. Pada kasus ini terdapat edge dengan warna abu abu dimana file dan folder yang bersangkutan belum masuk ke penelusuran karena file yang dicari sudah ditemukan.



Kasus ketiga, pengguna mencari sebuah file dengan metode dfs dan untuk pencarian satu kemunculan. Pada program DFS, pengecekan file didahulukan dari pada folder, maka dari itu pada contoh kasus diatas, hanya satu edge saja yang berwarna hijau karena sudah menemukan file yang dicari.

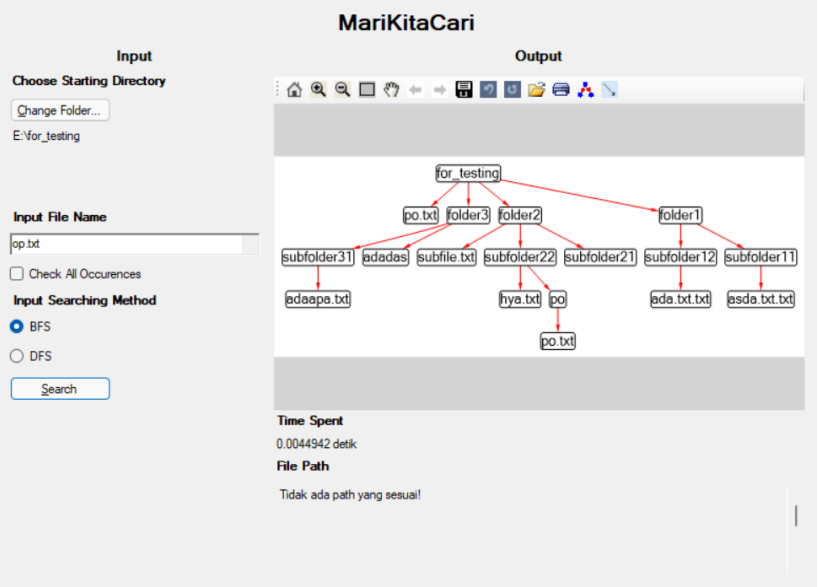


Kasus keempat, pengguna mencari file dengan metode bfs dan pencarian untuk semua kemunculan. Pada kasus all Occurance, tidak akan ada edge dengan warna abu abu karena pada dasarnya, all occurrence ini memaksa untuk mencari ke semua file dan folder didalam folder akar.



,

Kasus kelima, pengguna mencari file dengan metode dfs dan pencarian untuk semua kemunculan. Kasus kelima ini sama seperti kasus keempat, mencari keseluruhan file dan folder.



Kasus keenam, pengguna mencari file dengan metode apa pun, tetapi file tersebut tidak tersedia pada *directory* maupun setiap *child directory* dari starting folder yang telah dipilih.

5. Analisis dan Desain dari Solusi Algoritma BFS dan DFS Tiap Pengujian

Pada tubes ini, terdapat dua cara atau pendekatan dari masalah yang diberikan, yaitu BFS dan DFS. Dua cara ini memiliki kelebihan dan kekurangannya masing-masing. Dilihat dari cara kerjanya, dapat dengan jelas bahwa BFS itu lebih baik jika file yang dicari tidak terlalu dalam (folder dalam folder dst) karena BFS harus mencari semua yang ada dalam satu level, baru lanjut, jika file yang terdapat pada kedalaman terdalam, maka algoritma harus mencarinya secara brute force. Sedangkan, DFS baik untuk file yang memiliki kedalaman yang dalam, jika file yang dangkal dicari dengan cara DFS, maka tidak akan seefisien BFS karena DFS harus masuk ke dalam folder yang ada dulu baru bisa lanjut, dan karena hal ini, akan jadi lebih lama dibandingkan BFS.

BAB V
KESIMPULAN DAN SARAN

1. Kesimpulan

Algoritma BFS dan DFS merupakan algoritma yang efektif dalam pencarian sebuah file. Kedua algoritma ini akan memiliki efektifitas yang semakin tinggi dengan menangani kasus pencarian yang tepat. Untuk BFS akan semakin efektif apabila file yang dicari memiliki kedalaman file yang relatif kecil dan DFS akan semakin efektif jika mencari file dengan kedalaman yang relatif besar. Untuk kasus umum, algoritma BFS masih menghasilkan pencarian yang lebih efektif dibandingkan dengan DFS.

2. Saran

Pengerjaan tugas besar ini akan jauh lebih mudah dan cepat dengan menggunakan lingkungan pengembangan yang ideal. Dari pengalaman penulis, dengan melihat keperluan beberapa *package* tambahan dan kerangka Windows Form yang hanya bekerja pada sistem operasi tertentu, rupanya lingkungan ideal tersebut adalah sistem operasi Windows 10.

Alasannya ialah penggunaan Windows Form yang sangat sederhana karena pengembang dapat hanya langsung menempatkan objek dengan *drag-and-drop* dan meng-*handle event* dalam pengaturan properti tiap objeknya. Selain itu, dokumentasi MSAGL selain Windows Forms juga kurang lengkap. Misalnya, untuk kakas (atau *framework*) Gtk, pengembang harus terlebih dahulu memahami kakas lain yang di-*wrap*, seperti Cairo untuk penggambaran dan sebagainya. Dengan kondisi banyaknya tugas besar dalam waktu bersamaan, sepertinya hal tersebut kurang cocok.

BAB VI
TAUTAN VIDEO DEMO DAN *REPOSITORY*

Video demo dapat diakses melalui tautan berikut.
https://youtu.be/vJ3DMrPoaNw
Repository GitHub dapat diakses melalui tautan berikut.
https://github.com/nicholass25/TubesStima2_MariKitaCari

DAFTAR PUSTAKA

https://www.ilmuskripsi.com/2016/05/mengenal-algoritma-traversal-di-dalam.html

https://www.freelancinggig.com/blog/2019/02/06/what-is-the-difference-between-bfs-and-dfs-algorithms/