Laporan
Tugas Besar II IF2211 Strategi Algoritma
Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling

Disusun Oleh :

Fransiskus Davin Anwari – 13520025
Muhammad Risqi Firdaus – 13520043
Farnas Rozaan Iraqee – 13520067

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

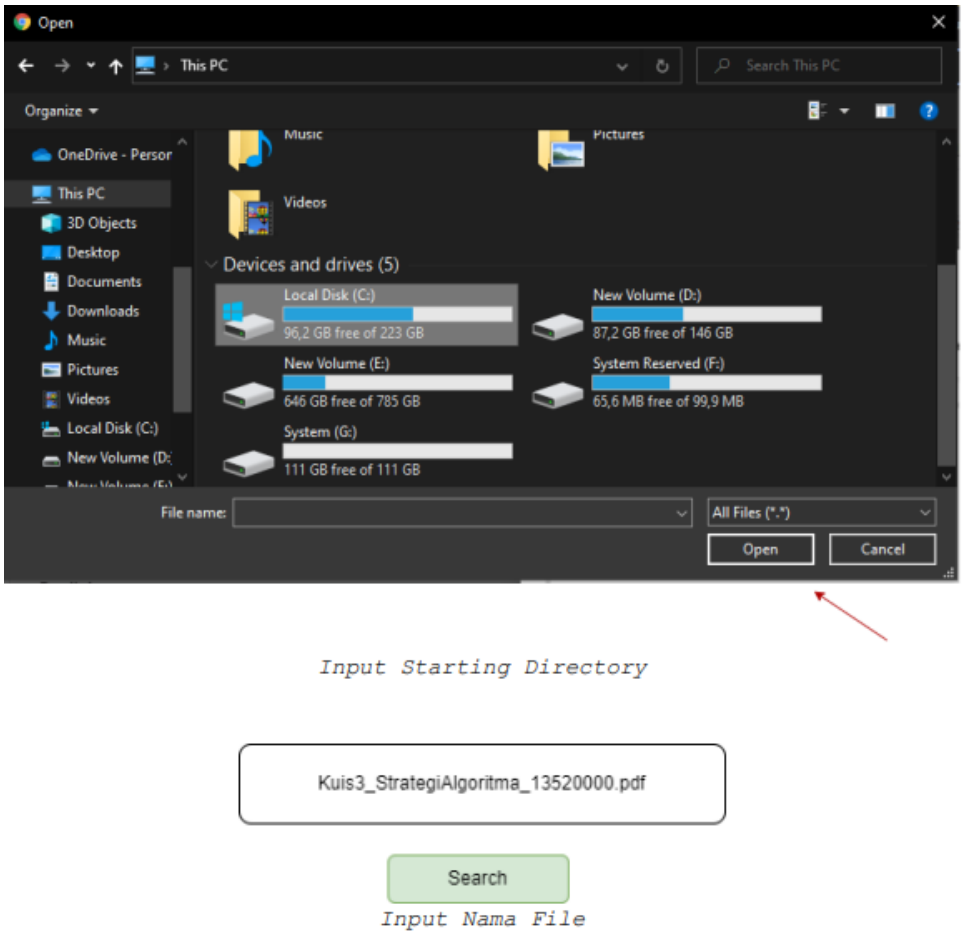
BAB 1
DESKRIPSI TUGAS

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.

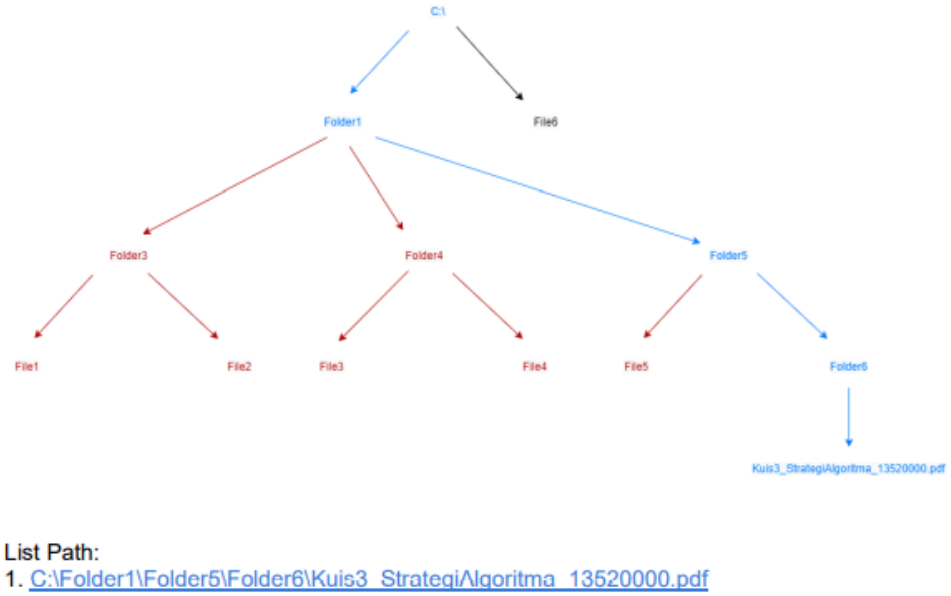
Contoh Input dan Output Program

Contoh masukan aplikasi :



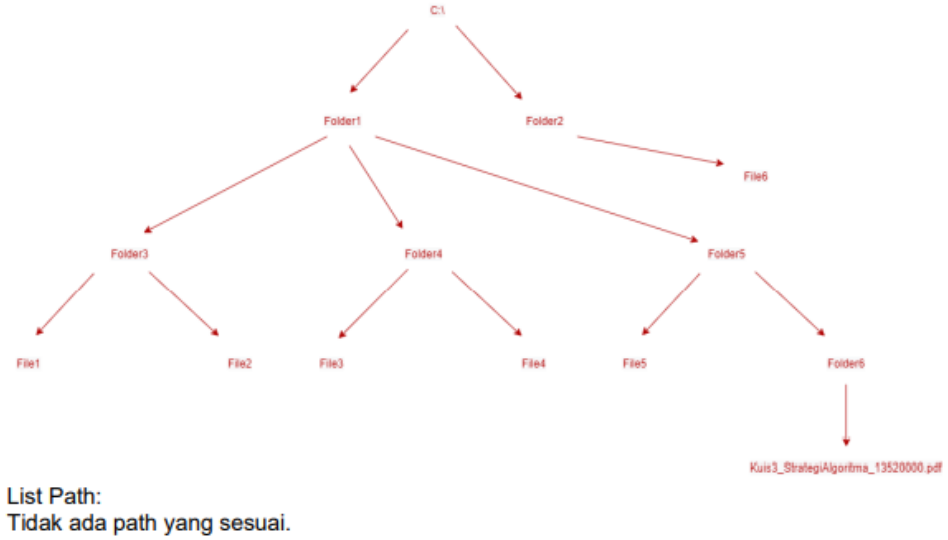
Gambar 1. Contoh Input Program

Contoh Keluaran aplikasi:



Gambar 2. Contoh Output Program

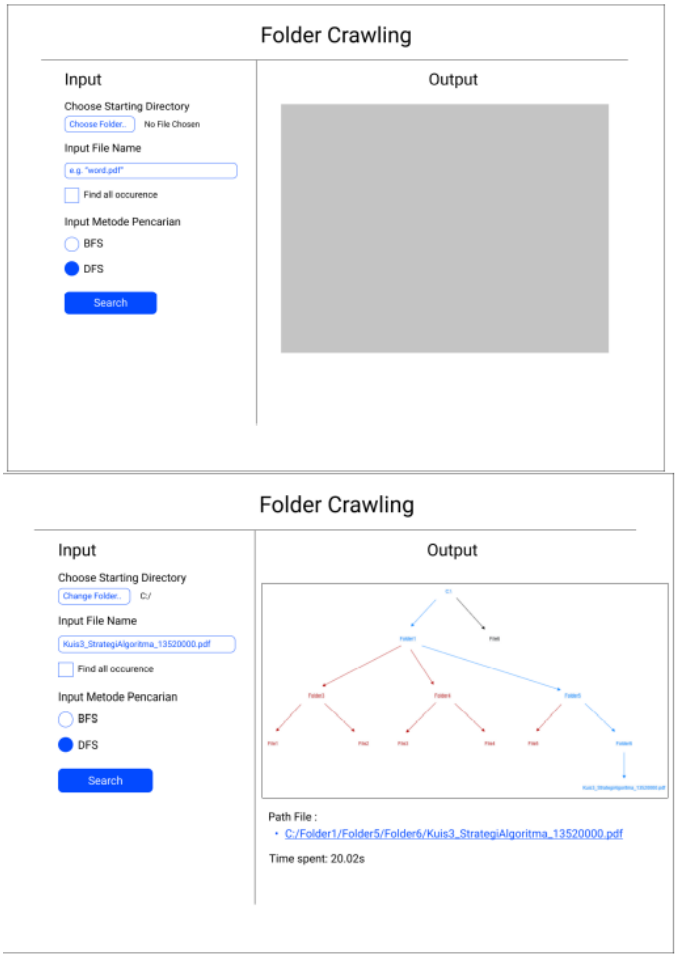
Misalnya pengguna ingin mengetahui langkah folder crawling untuk menemukan file Kuis3_StrategiAlgoritma_13520000.pdf. Maka, path pencarian DFS adalah sebagai berikut. C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3_StrategiAlgoritma_13520000.pdf. Pada gambar di atas, rute yang dilewati pada pencarian DFS diwarnai dengan warna merah. Sedangkan, rute untuk menuju tempat file berada diberi warna biru. Rute yang masuk antrian tapi belum diperiksa diberi warna hitam. Pemberian warna dibebaskan asalkan dibedakan antara ketiga hal tersebut.



Gambar 3. 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.

Aplikasi yang dibangun akan berbasis GUI. Berikut adalah contoh tampilan dari aplikasi GUI yang dibangun.



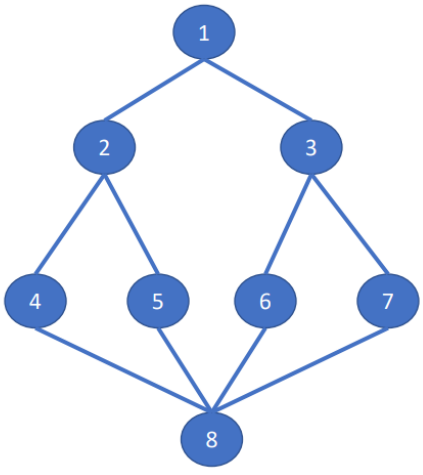
Gambar 4. Contoh Aplikasi Folder Crawling berbasis GUI

BAB 2
LANDASAN TEORI

Pada penelusuran suatu graf, dapat dimanfaatkan berbagai macam algoritma untuk membantu pada pengunjungan tiap simpul graf secara sistematik. Algoritma pencarian solusi pada graf dapat terbagi menjadi 2 yaitu Tanpa Informasi dan Dengan Informasi, pada kasus penggunaan algoritma Breadth First Seacrh (BFS) dan Depth First Search (DFS), kedua algoritma tersebut tergolong ke algoritma pencarian graf Tanpa Informasi. Selain terbagi menurut ketersediaan informasi, penelusuran graf juga tergantung dari jenis grafnya baik statis maupun tidak statis, dimana Graf Statis adalah graf yang sudah terbentuk sebelum penelusurannya, sedangkan Tidak Statis adalah graf yang terbentuk selama penelusurannya. BFS dan DFS keduanya merupakan penelusuran graf statis.

1. BREADTH FIRST SEARCH

Breadth First Search atau BFS adalah algoritma penelusuran graf yang berjalan traversalh secara melebar. BFS memanfaatkan struktur data queue sebagai urutan simpul yang dikunjungkan. Secara garis besar, jika traversal dimulai dari simpul v, maka algoritma akan mengunjungi simpul v dan dilanjutkan dengan menambahkan semua tetangga simpul v ke queue yang sudah ada. Setelah itu algoritma akan mengunjungi setiap tetangga simpul v yang sudah dimasukkan kedalam queue, dan ketika simpul yang dikunjungi memiliki tetangga yang belum dikunjungi dan tidak ada di queue, maka tetangga tersebut dimasukan ke dalam queue. Algoritma akan diulang terus menerus hingga sampai ke simpul yang dicari, atau semua simpul sudah dikunjungi sehingga queue kosong. Contoh penerapan algoritma BFS dapat dilihat sebagai berikut.



Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

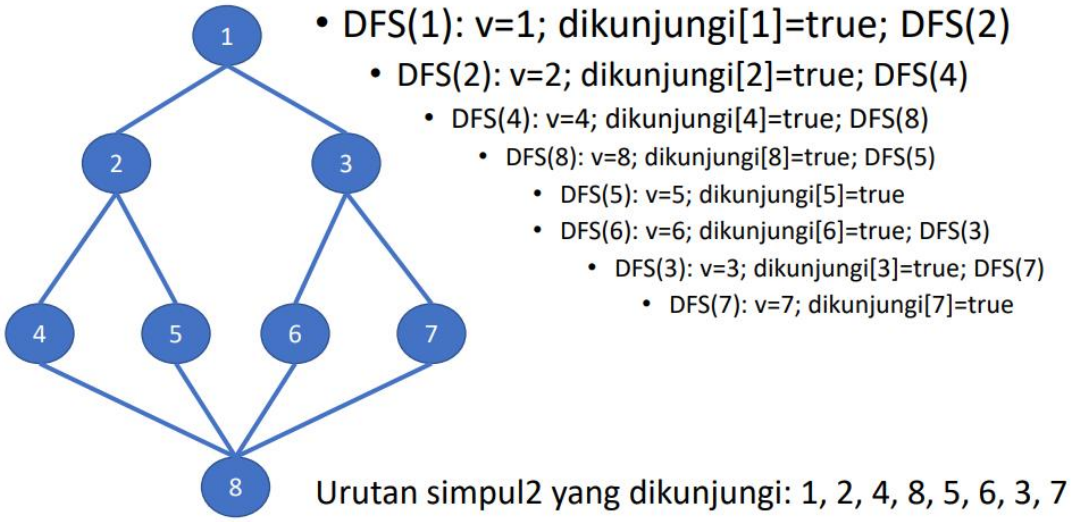
Urutan simpul2 yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

Gambar 5. Contoh Penerapan Algoritma BFS

Pada contoh penerapan tersebut bisa dilihat setiap iterasi penulusuran ditambah ke queue simpul tetangga dari simpul yang dikunjungi, dan dikeluarkan dari queue simpul yang dikunjungi. Setelah semua simpul telah dikunjungi maka dapat dilihat queue hanya akan terus berkurang dan tidak dilakukan penambahan simpul baru lagi. Setelah queue kosong maka algoritma penelusuran selesai dan didapatkan urutan simpul yang dikunjungi menurut algoritma BFS.

2. DEPTH FIRST SEARCH

Depth First Search atau DFS adalah algoritma penelusuran graf yang berjalan traversal secara mendalam. Perbedaannya dengan BFS adalah algoritma DFS tidak menggunakan struktur data apapun untuk menyimpan urutan simpul yang akan dikunjungi, melainkan algoritma selalu menentukan simpul selanjutnya yang dikunjungi tepat pada saatnya saja. Cara kerjanya algoritma DFS secara garis besar dimulai dengan inisialisasi pada simpul yang ditentukan seperti BFS, tetapi algoritma tidak perlu mencatat semua tetangga simpul tersebut, melainkan algoritma mencatat tiap simpul yang dikunjungi, algoritma lalu langsung berpindah ke tetangga simpul yang pertama ditemukan dan juga mencatatnya sebagai simpul yang telah dikunjungi. Hal ini diulang berkali-kali hingga simpul yang dikunjungi tidak memiliki tetangga lagi, maka algoritma akan backtrack atau berpindah ke simpul yang sebelumnya dikunjungi dan mengecek jika simpul sebelumnya memiliki tetangga yang belum dikunjungi. Backtracking ini dapat berulang hingga sampai simpul awal dan terus berjalan hingga semua simpul telah dikunjungi. Contoh penerapan algoritma DFS dapat dilihat sebagai berikut.



Gambar 6. Penerapan Algoritma DFS

Pada contoh penerapan diatas dapat dilihat bahwa simpul mengunjungi tetangganya berurutan sesuai nomor simpul yang paling kecil. Jika nama simpul berbentuk karakter, maka pemilihan tetangga akan sesuai abjad. Pada saat DFS(5), dapat dilihat bahwa simpul 5 tidak memiliki tetangga yang belum dikunjungi, sebab tetangganya yaitu simpul 2 dan simpul 8 telah dikunjungi, maka algoritma melakukan backtracking dan kembali ke simpul 8 dan mencari

tetangga simpul 8 yang belum dikunjungi. Algoritma pun akan selesai pada simpul 7 ketika semua simpul telah dikunjungi.

Dapat dilihat bahwa penelusuran graf yang sama oleh BFS dan DFS dapat menghasilkan urutan simpul yang berbeda. Karena itu pencarian suatu simpul yang khusus tidak tentu lebih efisien BFS atau DFS karena terdapata ketergantungan bagaimana graf terstruktur dan letak simpul yang dicari.

3. C# Desktop Application Development

Perangkat lunak dibuat dengan kakas Visual Studio dan Bahasa C#. Dalam pengembagnannya dibantu dengan library gagasan Microsoft MSGAL. Pengembangannya menggunakan template Form Desktop Application yang disedikan oleh Visual Studio.

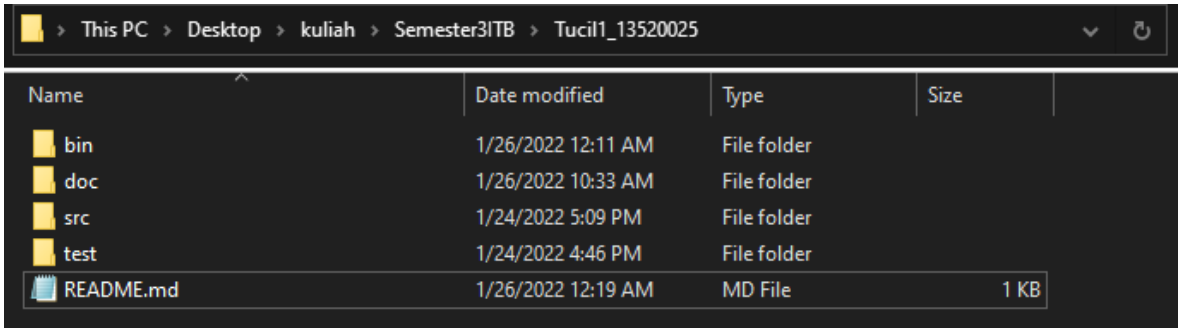
Pengembangan aplikasi desktop ini menggunakan partial class Form1 yang merupakan turunan dari kelas Form. Program utama berjalan dalam sebuah fungsi button2_clicked. Program di-compile menggunakan tools Visual Studio. Design pengembangan perangkat lunak juga dilakukan dengan Visual Studio Design.

Dalam pembuatan Graf digunakan Library milik Microsoft yakni MSGAL. MSGAL merupakan library buatan Microsoft yang diinstall merlalui Package Manager CLI. Untuk pengembangan kali ini digunakan MSGAL versi 1.1.11.

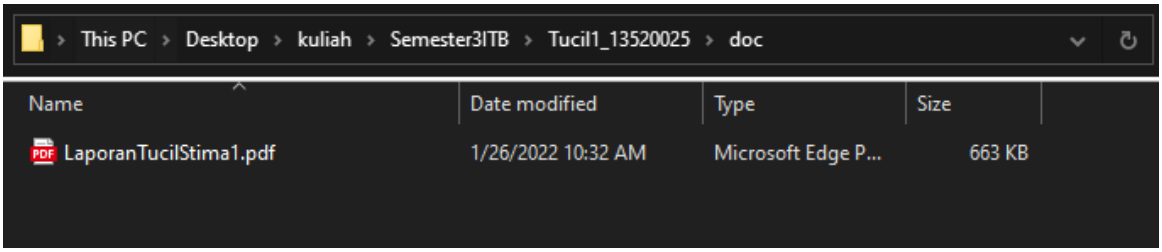
BAB 3
ANALISIS PEMECAHAN MASALAH

Pencarian suatu file pada suatu direktori merupakan salah satu jenis penelusuran graf statik. Karena itu maka dapat dimanfaatkan algoritma DFS dan BFS untuk menelusuri direktori tersebut. Menggunakan algoritma DFS dan BFS, direktori akan ditelusuri hingga ditemukan file yang ditujukan, hanya ketika file ditujukan maka algoritma akan berhenti, selain itu algoritma akan terus berjalan selama masih ada file/folder yang dapat ditelusuri. Agar dapat menghasilkan graf berwarna, maka algoritma dapat mencatat tiap file dan folder apakah merupakan file tujuan, jika bukan maka akan ditambah ke array berisi path salah, sedangkan jika benar maka file dan semua folder induknya dimasukkan ke dalam array berisi path solusi. Array berisi path solusi dan path salah bisa digunakan di fungsi lain untuk digambarkan menjadi graf.

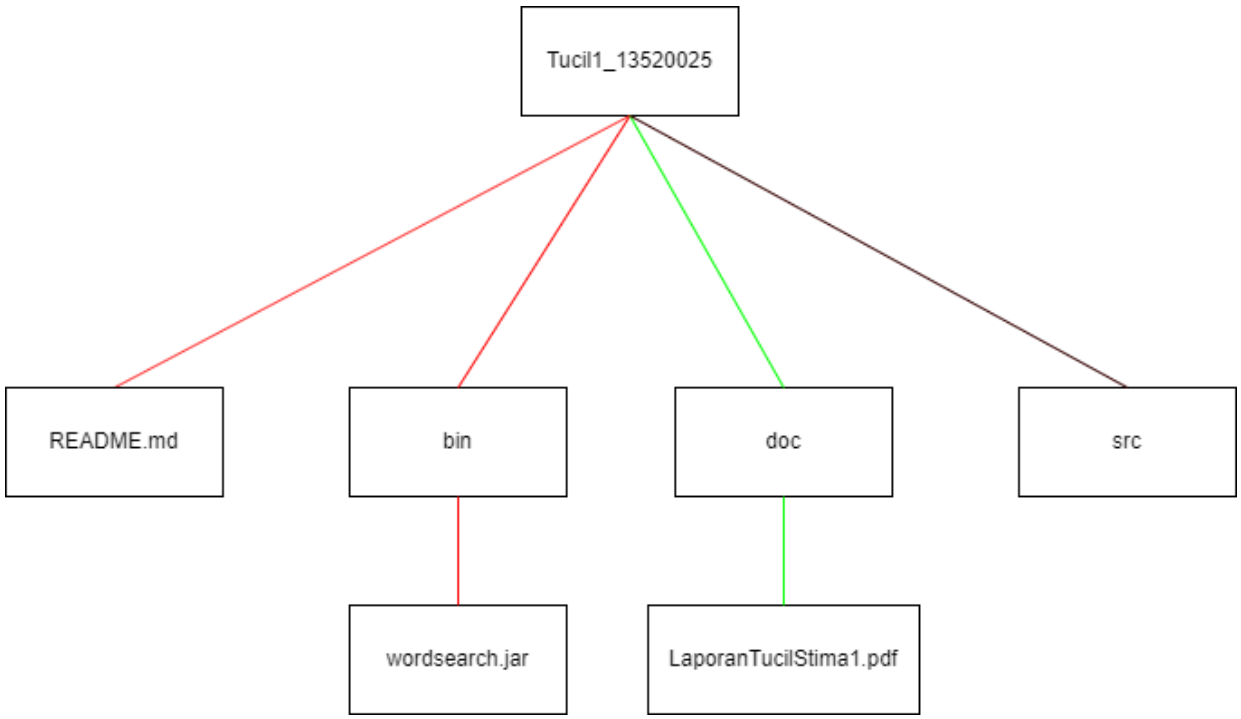
Jika folder awal penelusuran dianggap sebagai simpul awal graf, maka semua folder dan file di dalam folder awal tersebut dapat dianggap sebagai tetangga simpul awal. Karena file tidak dapat mencakup file didalamnya, maka file merupakan suatu simpul yang hanya mempunyai 1 tetangga yaitu folder induknya. Sedangkan suatu folder memiliki tetangga folder induknya dan juga isi dari folder tersebut. Misalkan untuk mencari file LaporanTucilStima1.pdf dari folder Tucil1_13520025 akan menghasilkan hasil graf sebagai berikut.



Gambar 7. Direktori Awal Pencarian File



Gambar 8. File Tujuan



Gambar 9. Hasil Penelusuran Mencari LaporanTucilStima1.pdf

Garis merah merupakan simpul-simpul salah, garis hijau menuju simpul tujuan, dan simpul hitam untuk simpul yang belum sempat ditelusuri. Ini merupakan contoh penelusuran jika menggunakan pendekatan BFS, dimana algoritma berhenti ketika sampai pada simpul tujuan sehingga terdapat simpul yang belum sempat diperiksa.

BAB 4
IMPLEMENTASI DAN PENGUJIAN

1. Implementasi

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;

namespace FolderCrawler
{

    class FileDestination {
        private string file_name;
        private string startFullPath;
        private bool found;
        private bool allOccurance;
        private List<string> redArr;
        private List<string> greenArr;
        private List<string> blackArr;
        protected List<string> answer = new List<string>();
        private Microsoft.Msagl.Drawing.Graph graph;

        public List<string> getAnswer()
        {
            return answer;
        }

        public FileDestination(string nama, bool semua) {
            file_name = nama;
            found = false;
            allOccurance = semua;
        }

        public Microsoft.Msagl.Drawing.Graph DFS(string dirpath)
        {
            this.graph = new Microsoft.Msagl.Drawing.Graph("graph");
            this.startFullPath = dirpath;
            this.graph.AddNode(getFolderOfPath(dirpath));
            this.greenArr = new List<string>();
            this.redArr = new List<string>();
            this.blackArr = new List<string>();

            recDFS(dirpath);
```

```
    return this.printGraph();
}
public void recDFS(string dirpath)
{

    DirectoryInfo dir = new DirectoryInfo(dirpath);
    //Console.WriteLine(dir.FullName);
    string[] filePaths = Directory.GetFiles(dir.FullName, "*");

    foreach (string file in filePaths)
    {
        if (this.found != true)
        {
            if (Path.GetFileName(file) == this.file_name)
            {
                this.answer.Add(file);
                this.greenArr.Add(file);
                this.getGreenNode(dir.FullName);


                if (!this.allOccurance)
                {
                    this.found = true;
                }
            }
            else
            {

                this.redArr.Add(file);
            }

            if(this.found != true)
            {
                this.getRedNode(dir.FullName);
            }
        }
    }
    string[] children = Directory.GetDirectories(dir.FullName, "*", SearchOption.TopDirectoryOnly);
    if (this.found != true)
    {
        foreach (string child in children)
        {

            this.blackArr.Add(child);
            this.recDFS(child);
        }

    }
```

```
}

public Microsoft.Msagl.Drawing.Graph BFS(string dirpath)
{
   this.graph = new Microsoft.Msagl.Drawing.Graph("graph");
   this.startFullPath = dirpath;
   this.graph.AddNode(getFolderOfPath(dirpath));
   this.greenArr = new List<string>();
   this.redArr = new List<string>();
   this.blackArr = new List<string>();

   solveBFS(dirpath);

   return this.printGraph();
}
public void solveBFS(string dirpath)
{
   Queue<string> queue = new Queue<string>();

   queue.Enqueue(dirpath);

   do
   {
      string current_dir = queue.Dequeue();
      DirectoryInfo dir = new DirectoryInfo(current_dir);
      string[] filePaths = Directory.GetFiles(dir.FullName, "*");

      foreach (string file in filePaths)
      {
         if (this.found != true)
         {
            if (Path.GetFileName(file) == this.file_name)
            {
               this.answer.Add(file);
               this.greenArr.Add(file);
               this.getGreenNode(dir.FullName);

               if (!this.allOccurance)
               {
                  this.found = true;
               }
            }
            else
            {
               this.redArr.Add(file);
            }

            if(this.found != true)
            {
```

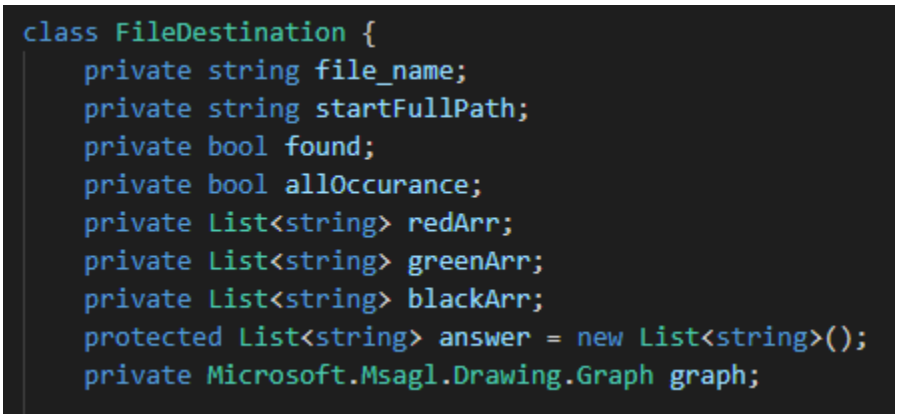
```
                this.getRedNode(dir.FullName);
            }
        }
    }
    string[] children = Directory.GetDirectories(@dir.FullName, "*", SearchOption.TopDirectoryOnly);
    foreach (string child in children)
    {
        this.blackArr.Add(child);
        queue.Enqueue(child);
    }
} while (queue.Count() != 0 && !this.found)

if (queue.Count() > 0)
{
    foreach (string path in queue)
    {
        DirectoryInfo dir = new DirectoryInfo(path);
        this.getBlackNode(dir.FullName);
    }
}
}

private string getFolderOfPath(string Path)
{
    string dir = new DirectoryInfo(@Path).Name;
    return dir;
}

public void getBlackNode(string path)
{

    while (!(this.greenArr.Contains(path) || this.blackArr.Contains(path)) && path!=this.startFullPath){
        this.blackArr.Add(path);
        DirectoryInfo p = new DirectoryInfo(@path);
        path = p.Parent.FullName;
    }
}

public void getRedNode(string path)
{

    while (!(this.greenArr.Contains(path) || this.redArr.Contains(path)) && path != this.startFullPath)
    {
        if (this.blackArr.Contains(path))
        {
            this.blackArr.Remove(path);
        }
        this.redArr.Add(path);
        DirectoryInfo pr = new DirectoryInfo(@path);
```

```
    }
}

public void getGreenNode(string path)
{
    while(!this.greenArr.Contains(path)&& this.startFullPath!= path)
    {
        if (this.redArr.Contains(path))
        {
            this.redArr.Remove(path);
        }
        if (this.blackArr.Contains(path))
        {
            this.blackArr.Remove(path);
        }
        this.greenArr.Add(path);
        DirectoryInfo p = new DirectoryInfo(@path);
        path = p.Parent.FullName;
    }
}

public Microsoft.Msagl.Drawing.Graph printGraph()
{
    DirectoryInfo p = new DirectoryInfo((this.startFullPath));
    if (this.greenArr.Count != 0)
    {
        this.graph.AddNode(p.Name).Attr.FillColor = Microsoft.Msagl.Drawing.Color.Green;
    }
    this.greenArr.ForEach(p =>
    {

        DirectoryInfo s = new DirectoryInfo(@p);
        this.graph.AddEdge(s.Parent.Name, s.Name).Attr.Color = Microsoft.Msagl.Drawing.Color.Green;
        this.graph.FindNode(s.Name).Attr.FillColor = Microsoft.Msagl.Drawing.Color.Green;
    });

    this.redArr.ForEach(p =>
    {
        DirectoryInfo s = new DirectoryInfo(p);
        this.graph.AddEdge(s.Parent.Name, s.Name).Attr.Color = Microsoft.Msagl.Drawing.Color.Red;
    });

    this.blackArr.ForEach(p =>
    {
        DirectoryInfo s = new DirectoryInfo(p);
        this.graph.AddEdge(s.Parent.Name, s.Name).Attr.Color = Microsoft.Msagl.Drawing.Color.Black;
    });
    return this.graph;
```

```
    }

  }
}
```

2. Penjelasan Program

Algoritma pencarian file dibantu dengan kelas FileDestination yang nantinya berfungsi untuk membantu mengubah hasil pencarian menjadi bentuk graf dengan pewarnaan solusi. Kelas FileDestination sebagai berikut.



```
class FileDestination {
    private string file_name;
    private string startFullPath;
    private bool found;
    private bool allOccurance;
    private List<string> redArr;
    private List<string> greenArr;
    private List<string> blackArr;
    protected List<string> answer = new List<string>();
    private Microsoft.Msagl.Drawing.Graph graph;
```

Gambar 10. Kelas FileDestination

Atribut startFullPath dan file_name masing-masing merepresentasikan simpul awal DFS dimulai dan simpul tujuan. Atribut found dan allOccurance berfungsi untuk implementasi pencarian file lebih dari 1 dengan nama yang sama. Atribut redArr, greenArr, dan blackArr menyimpan string dari path-path folder atau file yang ditelusuri, redArr untuk jalan yang salah, blackArr untuk jalan yang belum ditelusuri, dan greenArr untuk jalan menuju simpul tujuan. Ketiga array itu akan membantu pembentukan graf akhir. Atribut answer berupa kumpulan path solusi dan graph sebagai gambaran graf hasil penelusuran.

Setelah didapatkan himpunan solusi dari proses pencarian file dalam bentuk list answer, maka dijalankan method printGraph yang akan menuliskan hasil penelusuran BFS/DFS dari array-array berwarna menjadi suatu graf berwarna.

Method-method yang ada dalam kelas ini adalah sebagai berikut:

a. getAnswer :mengembalikan array answer yang berisi path ke file tujuan
b. DFS :mengembalikan graf hasil penelusuran secara DFS
c. recDFS :penelusuran file secara DFS
d. BFS : mengembalikan graf hasil penelusuran secara BFS
e. solveBFS : penelusuran file secara BFS
f. getFolderOfPath :mengembalikan nama dari direktori dari path tertentu
g. getBlackNode :mewarnai node dengan warna hitam

h. getRedNode :mewarnai node dengan warna merah
i. getGreenNode :mewarnai node dengan warna hijau
j. printGraph :print graf hasil penelusuran
k. getTime :mengembalikan waktu yang digunakan untuk melakukan searching

2.1. BREADTH FIRST SEARCH

Untuk BFS, hal yang pertama dilakukan adalah membuat queue dan meng-enqueue direktori awal seperti yang ditunjukkan pada gambar di bawah ini.

```
Queue<string> queue = new Queue<string>();

queue.Enqueue(dirpath);
```

Gambar 11. Enqueue Direktori Awal

Kemudian dilakukan dequeue terhadap queue yang hasilnya akan digunakan untuk melakukan proses pencarian file. Proses pencarian dilakukan mulai dari file dan folder yang ada di direktori tersebut. File yang terdapat dalam direktori yang sedang ditelusuri akan dicek satu-persatu apakah sama dengan file yang hendak dicari. Sedangkan untuk setiap folder yang ditemukan dalam direktori yang ditelusuri akan di-enqueue ke dalam queue. Proses ini terus dilakukan sampai file yang hendak dicari ditemukan atau queue sudah kosong yang dalam hal ini proses pencarian file gagal.

```
do
{
    string current_dir = queue.Dequeue();
    DirectoryInfo dir = new DirectoryInfo(current_dir);
    string[] filePaths = Directory.GetFiles(dir.FullName, "*");

    foreach (string file in filePaths)
    {
        if (this.found != true)
        {
            if (Path.GetFileName(file) == this.file_name)
            {
                this.answer.Add(file);
                this.greenArr.Add(file);
                this.getGreenNode(dir.FullName);

                if (!this.allOccurance)
                {
                    this.found = true;
                }
            }
            else
            {
                this.redArr.Add(file);
            }

            if(this.found != true)
            {
                this.getRedNode(dir.FullName);
            }
        }
    }
    string[] children = Directory.GetDirectories(dir.FullName, "*", SearchOption.TopDirectoryOnly);
    foreach (string child in children)
    {
        this.blackArr.Add(child);
        queue.Enqueue(child);
    }
} while (queue.Count() != 0 && !this.found)
```

Gambar 12. Proses Pencarian File secara BFS

Apabila setelah proses pengulangan tersebut queue masih ada isinya berarti folder yang ada dalam queue tersebut tidak ditelusuri karena file yang hendak dicari sudah ditemukan.

```
if (queue.Count() > 0)
{
    foreach (string path in queue)
    {
        DirectoryInfo dir = new DirectoryInfo(path);
        this.getBlackNode(dir.FullName);
    }
}
```

Gambar 13. Pengecekan Folder yang Belum Ditelusuri

2.2. DEPTH FIRST SEARCH

Untuk algoritma DFS yang pertama kali dilakukan adalah menelusuri simpul tetangga dari simpul awal yang dalam hal ini adalah direktori awal penelusuran akan dimulai. Simpul tetangga tersebut dapat berupa file ataupun folder. Sama seperti BFS karena file merupakan simpul daun maka akan dicek file-file yang ada dalam suatu direktori apakah file tersebut merupakan file yang hendak dicari atau bukan. Sedangkan

apabila ditemukan folder, berbeda dengan BFS yang melakukan enqueue folder ke dalam queue, DFS akan memanggil fungsi pencarian file tersebut secara rekursif dengan folder menjadi direktori awalnya. Hal ini dilakukan secara terus menerus sampai file yang hendak dicari ditemukan atau proses pencarian file gagal.

```
DirectoryInfo dir = new DirectoryInfo(dirpath);
string[] filePaths = Directory.GetFiles(dir.FullName, "*");

foreach (string file in filePaths)
{
    if (this.found != true)
    {
        if (Path.GetFileName(file) == this.file_name)
        {
            this.answer.Add(file);
            this.greenArr.Add(file);
            this.getGreenNode(dir.FullName);


            if (!this.allOccurance)
            {
                this.found = true;
            }
        }
        else
        {

            this.redArr.Add(file);
        }

        if(this.found != true)
        {
            this.getRedNode(dir.FullName);
        }
    }
}
```

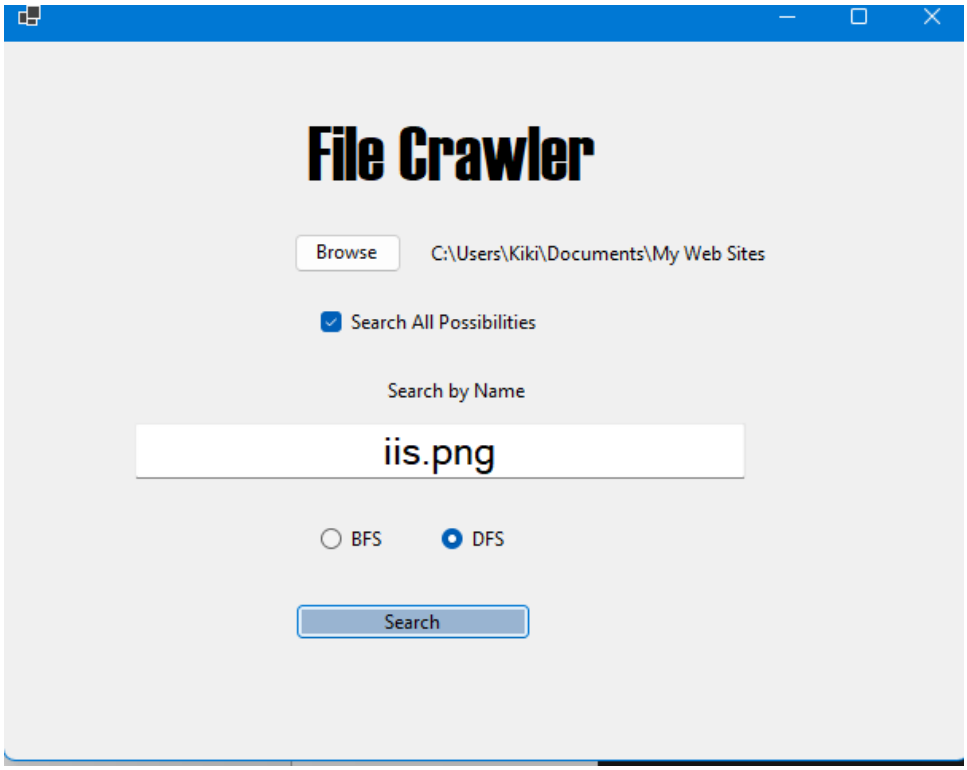
Gambar 14. Penelusuran File pada DFS

```
string[] children = Directory.GetDirectories(dir.FullName, "*", SearchOption.TopDirectoryOnly);
if (this.found != true)
{
    foreach (string child in children)
    {

        this.blackArr.Add(child);
        this.recDFS(child);
    }

}
```

Gambar 15. Penelusuran Folder dan Pemanggilan DFS secara Rekursii

2.3. GUI dengan MSGAL

GUI ditulis pada program utama form1 dalam sebuah partial class Form1 yang merupakan implementasi dari class Form. Program ini menghasilkan interface utama pada GUI. Untuk setiap pencarian akan dibuat sebuah windows baru berisi graph serta linklabel jika ditemukan dan waktu yang dibutuhkan untuk melakukan komputasi.



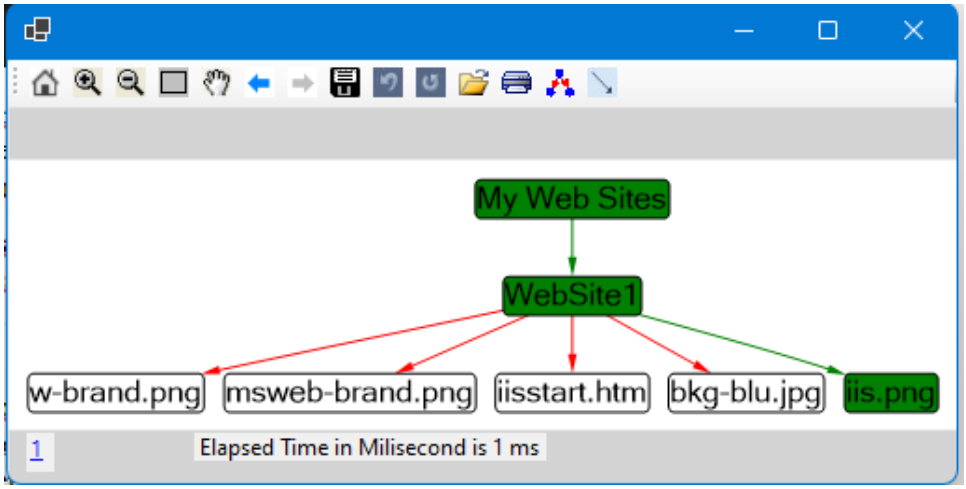

```
public partial class Form1 : Form
{
    private LinkLabel LinkLabel1;
    private string links;
    1 reference
    public Form1()
    {
        InitializeComponent();
    }
```

Fungsi utama dari class partial Form1 ada pada void button2_Click. Fungsi ini merupakan fungsi yang berjalan ketika tombol pencarian dilakukan. Ketika pencarian dilakukan, tetapi path folder serta jenis pencarian belum ditentukan maka program tidak akan melakukan apapun.

```
private void button2_Click(object sender, EventArgs e)
{
    Form graphWin = new Form();
```

Ketika dilakukan pencarian dan berhasil, program akan membentuk form (windows baru) yang berisi graf, linklabel dan label waktu. Ketika Form hasil dihapus maka program utama akan masi berjalan dan dapat melakukan pencarian lagi.

Ketika file ditemukan vertex yang menuju node jawaban akan diberi warna hijau. Untuk tiap vertex yang belum dicek akan diberi warna hitam, sedangkan vertex yang sudah merupakan ujung dari sebuah folder dan tidak memiliki jawaban di dalamnya akan diberi warna merah.

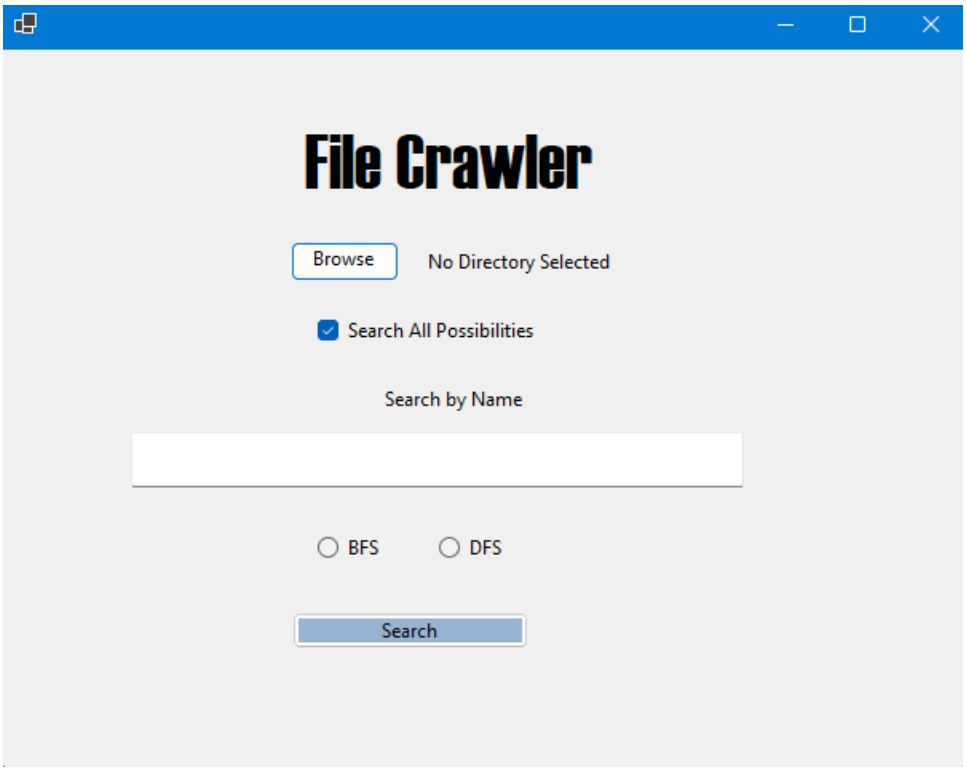


LinkLabel yang dihasilkan program akan bergantung berapa banyak file ditemukan. Oleh karena itu, linklabel pada program ini disimpan dalam List. Ketika ditekan, link label akan otomatis membuka file explorer, folder tempat file ditemukan.

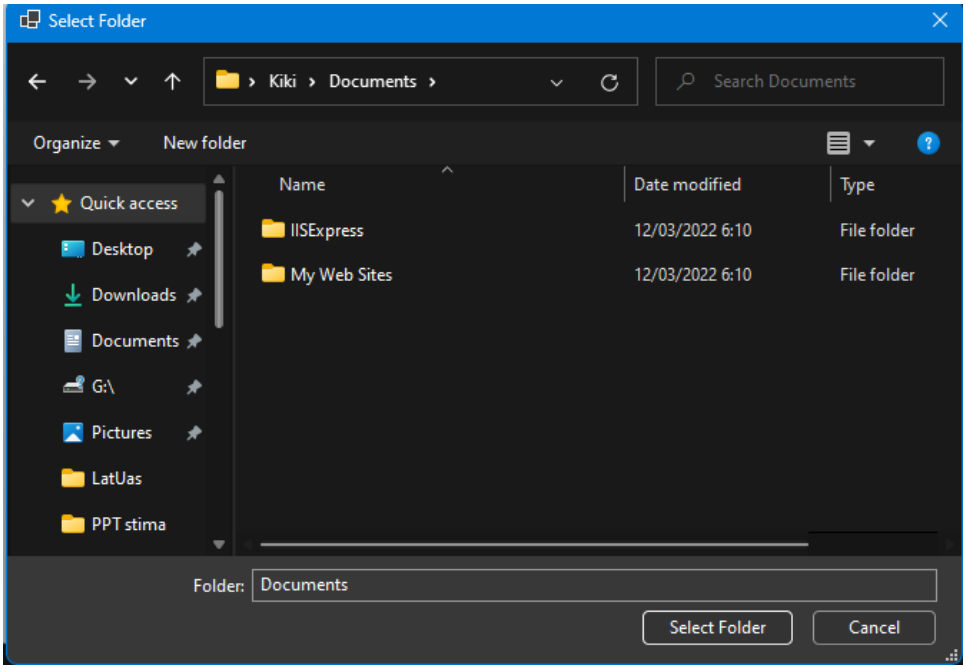
```
filed.getAnswer().ForEach(x =>
{
    DirectoryInfo res = new DirectoryInfo(x);


    count++;
    this.LinkLabel1.Text += count + " ";
    links.Add(new LinkLabel.Link(2*(count-1), 2*(count-1)+1, res.Parent.FullName));

});
foreach(var link in links)
{
    this.LinkLabel1.Links.Add(link);
}
this.LinkLabel1.Location = new Point(10, graphWin.Height -100);
this.LinkLabel1.LinkClicked += (s, e) => {
    System.Diagnostics.Process.Start("explorer.exe", (string)e.Link.LinkData);
};

timeEst = filed.getTime();

graphWin.Controls.Add(this.LinkLabel1);
```

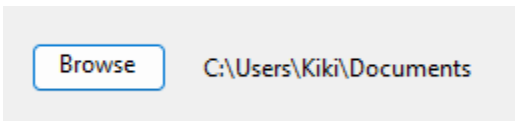
2.4. Tata Cara Penggunaan

Setalah program berjalan, program akan menampilkan windows form berisi parameter pencarian. Parameter tersebut di antaranya, folder input, checkbox keseluruhan pencarian, textbox nama file serta radio metode pencarian, dan tombol search.

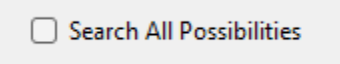


Pertama buka folder pencarian, dan tentukan folder dasar tempat dilakukannya pencarian file.

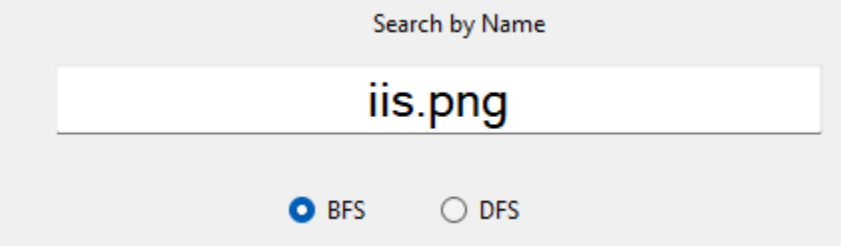




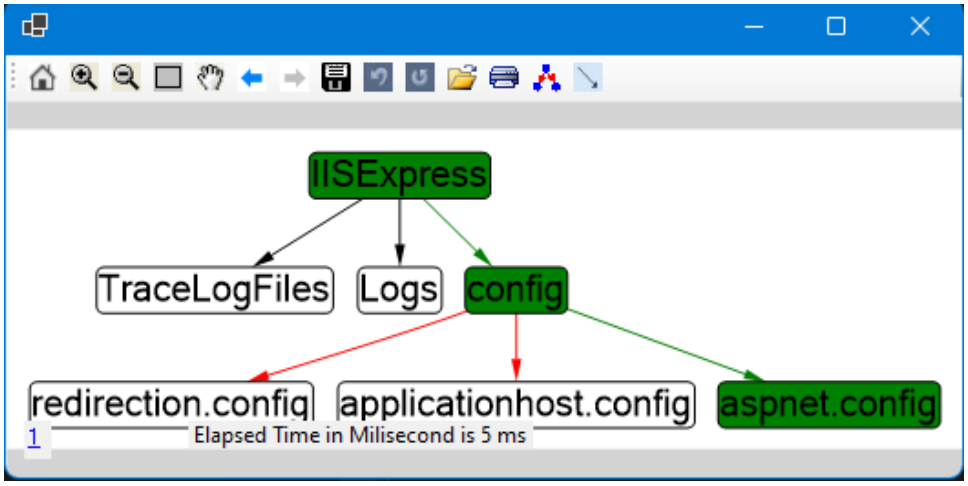
Kemudian tentukan, apakah ingin menemukan semua file dengan nama yang dicari atau tidak dengan checkbox.



Kemudian, masukkan nama file yang ingin dicari di textbox dan tentukan metode apa yang ingin digunakan.



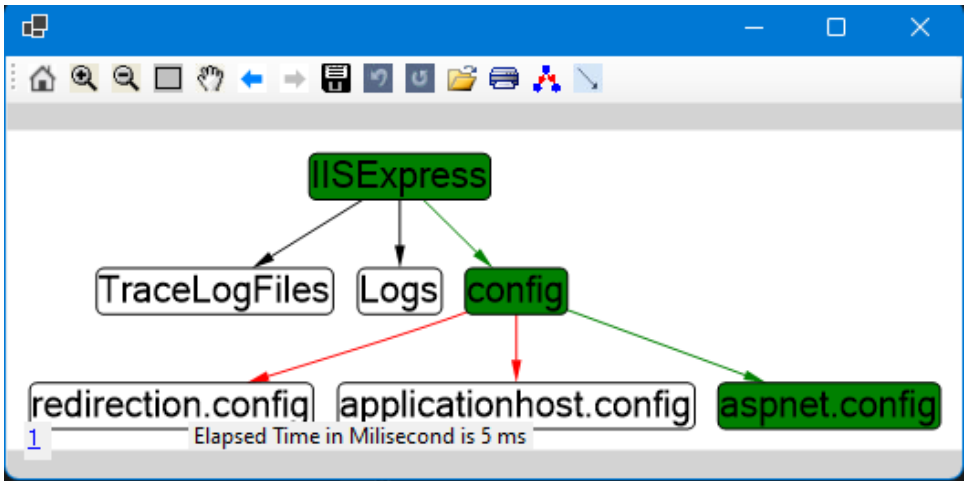
Terakhir, tekan tombol search, dan akan muncul window berisi graf, linklabel menuju folder tersebut, serta waktu lama pencarian.



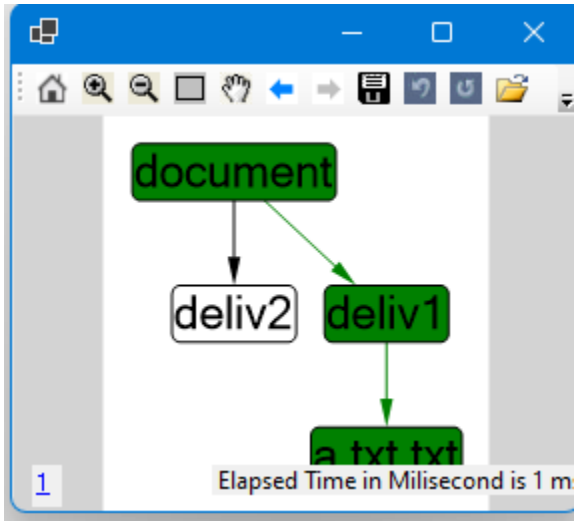
Dengan menekan tombol X, pengguna dapat kembali menggunakan aplikasi untuk melakukan pencarian.

2.5. Hasil Pengujian

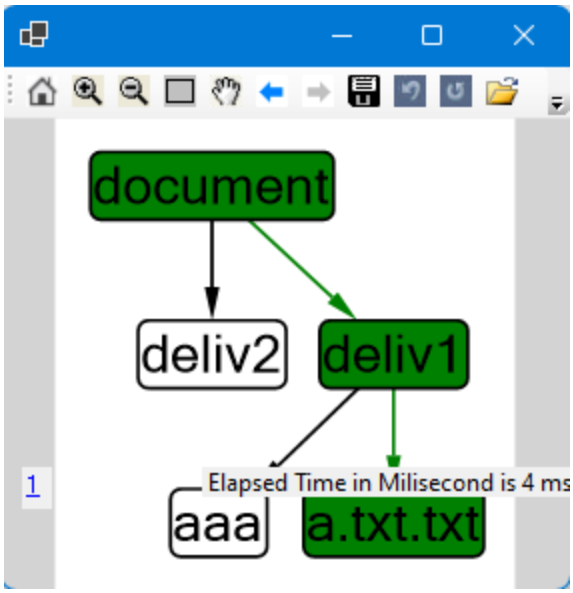
- Mencari file aspnet.config dari Folder IISExpress tanpa AllOccurence



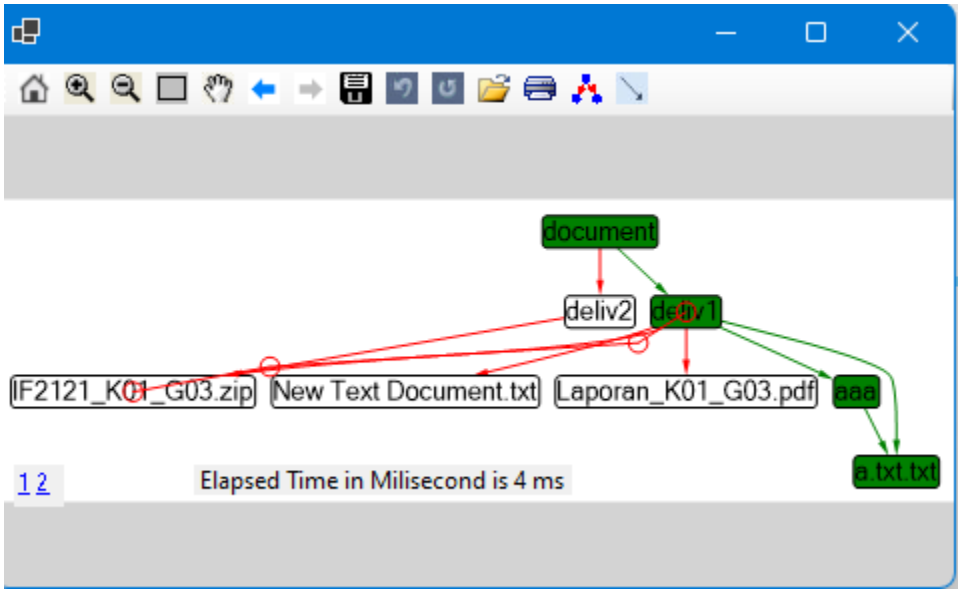
- Mencari file a.txt.txt dari Folder LegoLogkom/document tanpa AllOccurence DFS



- Mencari file a.txt.txt dari Folder LegoLogkom/document tanpa AllOccurence BFS



- Mencari file a.txt.txt dari Folder LegoLogkom/document dengan AllOccurence

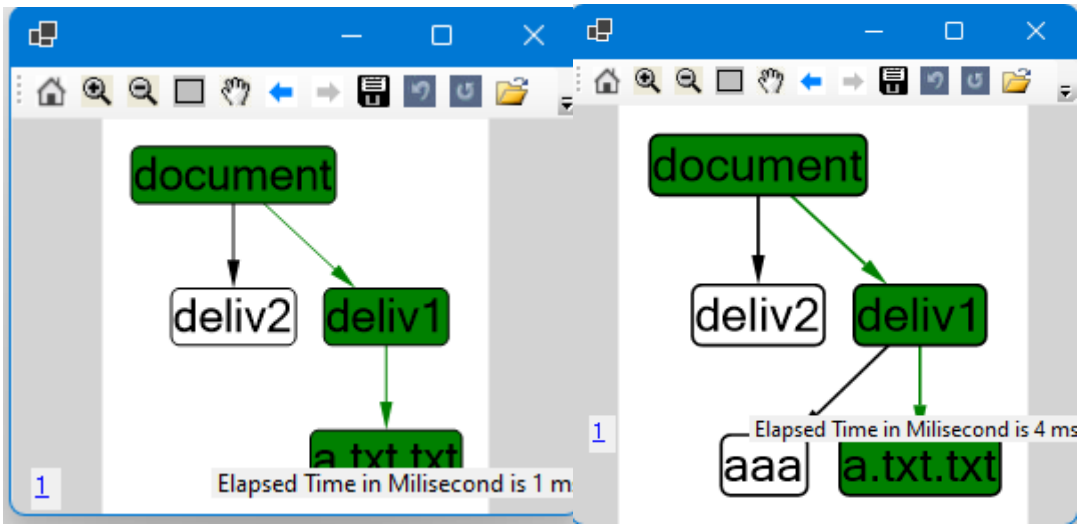


2.6. Analisis Desain Solusi

Algoritma BFS dan DFS dapat berkerja lebih baik satu sama lain tergantung dengan kasus yang diberikan. BFS merupakan penelusuran yang melebar, dimana program mengutamakan penelusuran pada level atau kedalaman yang sama terlebih dahulu hingga semua simpul pada kedalaman tersebut sudah dikunjungi. Karena itu BFS akan lebih optimal untuk mencari file yang tidak terdapat pada kedalaman yang tinggi pada direktori tersebut. DFS kurang optimal karena DFS mengutamakan untuk menelusuri secara mendalam hingga jika file terdapat pada kedalaman sedikit seperti 3 folder, sedangkan direktori bisa ditelusuri hingga kedalaman 8

folder, terdapat kemungkinan DFS akan menelusuri folder dengan kedalaman 8 terlebih dahulu sebelum menelusuri folder kedalam 3 yang lain.

Algoritma DFS lebih optimal untuk mencari file yang memiliki kedalaman cukup besar. Sebab DFS akan mengutamakan untuk menyelesaikan penelusuran suatu folder hingga bagian paling dalam direktori folder tersebut. Tetapi dibandingkan dengan BFS, DFS juga memiliki faktor lain yang berpengaruh kepada optimalisasi algoritma. Karena DFS akan menelusuri folder dengan mengutamakan abjad yang lebih rendah terlebih dahulu, jika file yang dicari terdapat pada folder dengan nama yang diawali huruf abjad tinggi contohnya “Zebra” maka DFS kemungkinan akan sama atau kurang optimal dibandingkan dengan DFS, karena algoritma akan mengutamakan untuk menelusuri folder yang diawali huruf abjad rendah seperti “Apel”. Untuk visualisasi perbandingan dapat dilihat dari gambar-gambar berikut.



Gambar sebelah kiri merupakan pencarian menggunakan algoritma DFS dan sebelah kanan merupakan menggunakan algoritma BFS. Pada penggunaan algoritma DFS waktu jalan program lebih cepat dibandingkan dengan BFS, hal ini disebabkan karena pada DFS program yang kami buat mengutamakan untuk menelusuri simpul bentuk file terlebih dahulu. Sedangkan pada algoritma BFS program melewati folder ‘aaa’ terlebih dahulu sebelum mengecek file yang a.txt.txt sehingga waktu penelusuran lebih lama. Selain itu algoritma BFS juga memakan waktu lama karena terdapat memasukan semua file dan folder dalam bentuk queue terlebih dahulu sebelum mengecek simpul 1 per 1.

Karena itu jika ingin mendapatkan hasil yang konsisten baik selalu cepat atau selalu lambat, maka lebih optimal menggunakan BFS, sedangkan DFS memiliki ketergantungan besar atas faktor penamaan folder dan file sehingga hasil penelusuran kecepatannya tidak selalu konsisten.

BAB 5
SIMPULAN DAN SARAN

5.1. Simpulan

Algoritma BFS dan DFS merupakan algoritma yang efektif untuk melakukan pencarian. Efektifitas keduanya dapat ditingkatkan dengan melakuka pemilihan algoritma pada kasus yang tepat. Pada kasus keberadaan yang pasti pada keadalaman tinggi, algoritma DFS dinilai lebih efektif, sedangkan pada kasus yang posisi file yang masih belum jelas, algoritma BFS akan menghasilkan pencarian yang lebih baik. Ketika all occurrence dicentang, maka algoritma akan mencari semua kemungkinan, sehingga bisa disebut akan menghasilkan worst case pada tiap kasus.

MSGAL merupakan library yang baik untuk membuat grafik Degnan C#. Namun, dalam keberjalanannya, library ini sedikit kurang stabil terkai version control dan peng-instalan. Selain itu, pengembangan dengan C# dan Form memungkinkan terjadinya deadlock akibat penggunaan thread yang saling tunggu. Selain itu, pencarian masi terbatas pada folder yang terbuka, pada folder read-only atau restricted, pencarian akan berhenti dengan sendirinya, atau bahkan tak dapat dilakukan sama sekali.

5.2. Saran

Dalam Pengembangannya, aplikasi ini masi memiliki banhyak kekurangan dan membutuhkan pengembangan lebih jauh. Pengembangan pertama yang dapat dilakukan ialah memperbaiki UI-UX sehingga lebih nyaman digunakan oleh pengguna. Selanjutnya, akibat aplikasi yang belum bisa menghandle folder yang diberi restriction oleh windows perlu diimplementasikan Application Pool Identity. Selain itu, aplikasi belum dapat megakses folder yang diakses oleh program lain, sehingga perlu implentasi mutex (wait and signaling) agar tidak terjadi deadlock.

Dalam segi pengembangan, perlu adanya kelompok yang lebih baik dalam hal koordinasi serta kebersamaan. Pun, sebaiknya perlu diadakan analisis, apakah ada mahasiswa yang terkendala dalam pengerjaan tugas akibat ketidakmampuan prasarana seperti computer pribadi. Visual Studio merupakan aplikasi yang cukup berat sehingga tidak semua computer dapat menjalankannya. Pada kelompok ini saja, hanya satu orang yang dapat menginstal Visual Studio hingga kesannya seperti membebani satu orang. Diharapkan pemberian tugas dapat merangkul seluruh elemen dari mahasiswa IF.

DAFTAR PUSTAKA

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/file-system/how-to-iterate-through-a-directory-tree

https://www.techiedelight.com/measure-execution-time-csharp/

https://docs.microsoft.com/en-us/dotnet/api/system.io?view=net-6.0

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf

LAMPIRAN

Repository GitHub: https://github.com/OjaanIr/BingungFirstSearch

Video Demo: https://www.youtube.com/watch?v=aPQZ0sCv-Gw