

Tugas Besar II IF2211 Strategi Algoritma

# Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling



Disusun oleh:

Fikri Khoiron Fadhlila - 13520056

Ubaidillah Ariq Pratama - 13520085

Marchotridyo - 13520119

**Institut Teknologi Bandung**

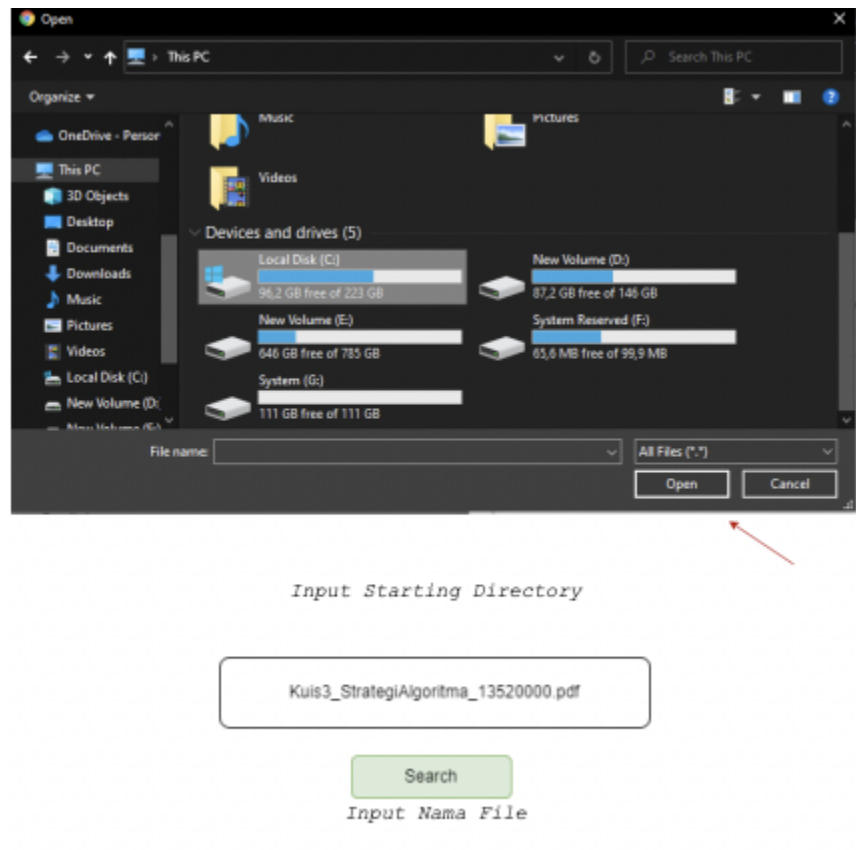
**2021/2022**



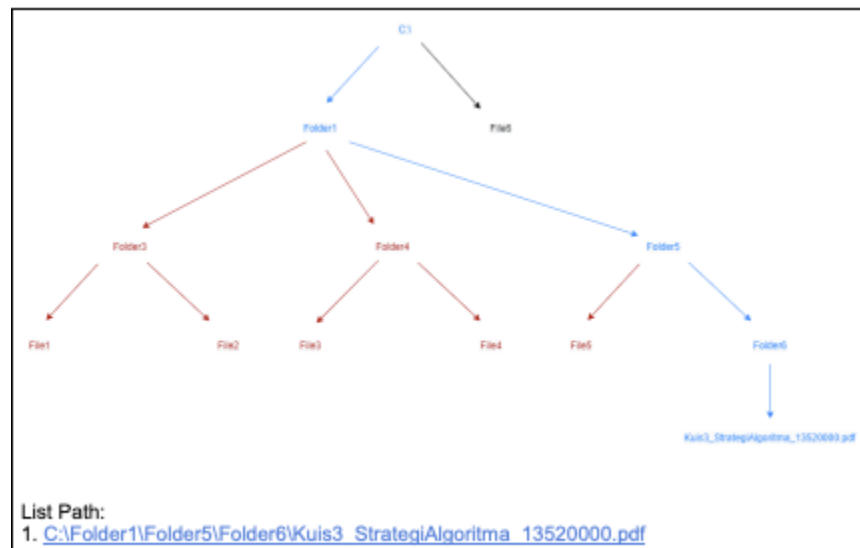
## Bab 1 - Deskripsi Tugas

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.


## Contoh Input dan Output Program



Gambar 2. Contoh input program



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.

## Bab 2 - Landasan Teori

### Graph Traversal

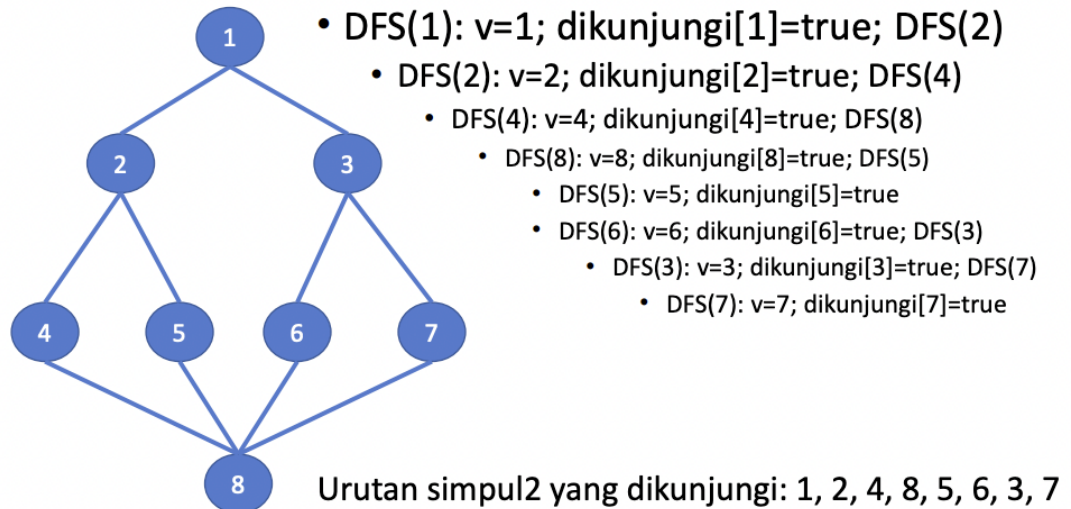
Graph traversal adalah sebuah proses dalam mengunjungi simpul - simpul dengan cara yang sistematis. Persoalan direpresentasikan dalam bentuk graf, dan untuk melakukan pencarian solusi dapat menggunakan graph traversal. Beberapa algoritma traversal graf yang terkenal seperti BFS (breadth first search) dan DFS (depth first search) dengan asumsi graf merupakan graf terhubung. Dalam proses pencarian solusi, terdapat dua pendekatan yaitu Graf Statis dan Graf Dinamis. Graf statis yaitu graf yang sudah terbentuk sebelum proses pencarian dilakukan, sedangkan graf dinamis yaitu graf yang terbentuk saat proses pencarian dilakukan.

### DFS

DFS (depth first search), berikut merupakan algoritma dari DFS dengan traversal dimulai dari simpul v

1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v.
3. Ulangi DFS mulai dari simpul w.
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi

## DFS: Ilustrasi 1



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

## BFS

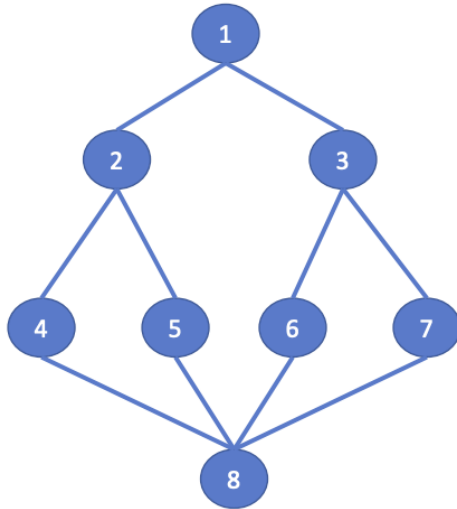
Berikut adalah algoritma BFS dengan traversal dimulai dari simpul v.

1. Kunjungi simpul v.
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

Struktur data yang digunakan pada BFS sebagai berikut:

1. Matriks ketetanggaan  $A = [a_{ij}]$  yang berukuran  $n \times n$ ,  $a_{ij} = 1$ , jika simpul i dan simpul j bertetangga,  $a_{ij} = 0$ , jika simpul i dan simpul j tidak bertetangga.
2. Antrian q untuk menyimpan simpul yang telah dikunjungi.
3. Tabel Boolean, diberi nama "dikunjungi" dikunjungi : array[l..n] of boolean dikunjungi[i] = true jika simpul i sudah dikunjungi dikunjungi[i] = false jika simpul i belum dikunjungi

## BFS: Ilustrasi



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

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

## C# Desktop Application Development

Desktop app ini ditulis dengan bahasa C# (C-Sharp) di dalam aplikasi Visual Studio, dengan menggunakan Windows Form. Windows Form atau sering disebut dengan WinForm adalah class library buatan Microsoft yang bersifat GUI atau Graphical User Interface. Windows Form tergabung dalam Framework .NET. Tujuan diciptakan Win Form adalah untuk mempermudah developer dalam membuat suatu aplikasi berbasis desktop. Banyak sekali fitur yang disediakan oleh winform, seperti label, panel dan beberapa fitur lainnya yang menunjang developer dalam UI/UX aplikasi.

## Bab 3 - Analisis Pemecahan Masalah

### Langkah - Langkah Pemecahan Masalah

Langkah yang kami gunakan untuk memecahkan masalah yaitu dengan membuat graf yang berisi data hasil *crawling* dari folder *root* yang dimasukkan pengguna.

#### BFS

Proses BFS dilakukan secara iteratif dengan menggunakan sebuah *queue*. Proses dimulai dengan memasukkan *folder root* sebagai simpul akar graf ke dalam *queue*. Selama *queue* masih ada isinya, akan dilakukan pemrosesan terhadap *folder/file* yang berada di depan *queue*. Pemrosesan untuk *folder* berbeda dengan pemrosesan untuk *file*. Suatu *node* baru dimasukkan ke dalam pohon apabila *node* tersebut sudah diambil dari *queue*.

Apabila elemen depan di *queue* adalah sebuah *folder*, semua *file* diikuti semua *subdirectory* yang berada di dalam *folder* tersebut akan dimasukkan ke dalam *queue* untuk diproses nantinya. *Node folder* ini akan diberi warna merah.

Apabila elemen depan di *queue* adalah sebuah *file*, *file* tersebut akan diperiksa namanya. Apabila nama *file* sama dengan nama yang dicari, semua *node* dan *edge* yang berhubungan dengan *file* tersebut akan diberi warna hijau. Jika tidak, *node* tersebut akan diberi warna merah.

Jika di dalam *queue* masih ada elemen tapi suatu *file* sudah ditemukan dan *checkbox Find All Occurrence* tidak dicek, semua elemen pada *queue* tersebut akan dicek sebagai *node* berwarna hitam yang artinya elemen tersebut belum dicek tetapi proses *searching* sudah selesai.

#### DFS

Proses DFS dilakukan secara iteratif dengan menggunakan sebuah *stack*. Proses dimulai dengan memasukkan *folder root* sebagai simpul akar graf ke dalam *stack*. Selama *stack* masih ada isinya, akan dilakukan pemrosesan terhadap *folder/file* yang berada di depan *queue*. Pemrosesan untuk *folder* berbeda dengan pemrosesan untuk *file*. Suatu *node* baru dimasukkan ke dalam pohon apabila *node* tersebut sudah diambil dari *stack*.

Apabila elemen depan di *stack* adalah sebuah *folder*, semua *file* diikuti semua *subdirectory* yang berada di dalam *folder* tersebut akan dimasukkan ke dalam *stack* untuk diproses nantinya. *Node folder* ini akan diberi warna merah.



Apabila elemen depan di *stack* adalah sebuah *file*, *file* tersebut akan diperiksa namanya. Apabila nama *file* sama dengan nama yang dicari, semua *node* dan *edge* yang berhubungan dengan *file* tersebut akan diberi warna hijau. Jika tidak, *node* tersebut akan diberi warna merah.

Jika di dalam *stack* masih ada elemen tapi suatu *file* sudah ditemukan dan *checkbox Find All Occurrence* tidak dicek, semua elemen pada *stack* tersebut akan dicek sebagai *node* berwarna hitam yang artinya elemen tersebut belum dicek tetapi proses *searching* sudah selesai.

## Proses Mapping Persoalan Menjadi Elemen-Elemen Algoritma BFS dan DFS

Kami melakukan mapping persoalan menjadi sebuah graf (lebih khususnya, sebuah pohon) dengan setiap file atau folder menjadi sebuah simpul dan path ke file atau folder tersebut dari directory sebelumnya menjadi sebuah edge. *Folder* awal sebagai akar dari pohon. Daun atau ujung dari tree dapat berupa file baik yang dicari maupun yang bukan, folder kosong juga dapat menjadi daun dari tree ini.

## Contoh Ilustrasi Kasus Lain

### Ilustrasi Pemecahan Masalah dengan Algoritma BFS

Dalam kasus ini, pencarian akan berhenti ketika sudah ditemukan *file* yang sesuai, yaitu *FindMe.txt*.

Misalkan ada sebuah *directory* yang memiliki struktur sebagai berikut.

```
FindMe/  
|- 1/  
|   |- FindMe.txt  
|- 2/  
|   |- 3  
|       |- 4  
|- NotMe.txt
```

Proses pembentukan *graf* ditunjukkan oleh tabel berikut.

iterasi ke-	Simpul Ekspan	Simpul Hidup
1	FindMe	FindMe/NotMe.txt, FindMe/1, FindMe/2
2	FindMe/NotMe.txt	FindMe/1, FindMe/2
3	FindMe/1	FindMe/2, FindMe/1/FindMe.txt
4	FindMe/2	FindMe/1/FindMe.txt, FindMe/2/3
5	FindMe/1/FindMe.txt	FindMe/2/3

Karena simpul FindMe/2/3 masih ada di dalam *queue*, simpul tersebut diberi warna hitam karena tidak dicek oleh program.

### Ilustrasi Pemecahan Masalah dengan Algoritma DFS

Dalam kasus ini, pencarian akan berhenti ketika sudah ditemukan *file* yang sesuai, yaitu *FindMe.txt*.

Misalkan ada sebuah *directory* yang memiliki struktur sebagai berikut.

FindMe/
- 1/
- FindMe.txt
- 2/
- 3
- 4
- NotMe.txt

Proses pembentukan *graf* ditunjukkan oleh tabel berikut.

iterasi ke-	Simpul Ekspan	Simpul Hidup
1	FindMe	FindMe/NotMe.txt, FindMe/1, FindMe/2
2	FindMe/NotMe.txt	FindMe/1, FindMe/2
3	FindMe/1	FindMe/1/FindMe.txt, FindMe/2
4	FindMe/1/FindMe.txt	FindMe/2

Karena simpul FindMe/2 masih ada di dalam *stack*, simpul tersebut diberi warna hitam karena tidak dicek oleh program.

## Bab 4 - Implementasi dan Pengujian

### Implementasi program

```
using Microsoft.WindowsAPICodePack.Dialogs;
using System.Diagnostics;

namespace FolderCrawlerTest
{
    public partial class Form1 : Form
    {
        class Item
        {
            public string type { get; set; }
            public string path { get; set; }
            public Item(string type, string path)
            {
                this.type = type;
                this.path = path;
            }
        }

        private static bool validFolder = false;
        private static bool foundFile = false;
        private static bool isSearching = false;
        private static Microsoft.Msagl.Drawing.Graph graph = new
Microsoft.Msagl.Drawing.Graph("graph");
        private static List<Label> hyperlinks = new List<Label>();
        private static List<String> foundPaths = new List<String>();
        private static Stopwatch sw = new Stopwatch();
        private void wait(int milliseconds)
        {
            /* Fungsi wait membuat program berada dalam proses "waiting" selama x
milliseconds. */
            /* Credits to StackOverflow */
            var timer1 = new System.Windows.Forms.Timer();
            if (milliseconds == 0 || milliseconds < 0) return;

            timer1.Interval = milliseconds;
            timer1.Enabled = true;
            timer1.Start();

            timer1.Tick += (s, e) =>
            {
                timer1.Enabled = false;
                timer1.Stop();
            }
        }
    }
}
```

```

    };

    while (timer1.Enabled)
    {
        Application.DoEvents();
    }
}

private string GetSafeName(string path)
{
    /* Mengembalikan safe name dari suatu path, misal x/y/z mengembalikan z saja */
    return path.Substring(path.LastIndexOf(Path.DirectorySeparatorChar) + 1);
}

private string GetRootPath(string path)
{
    /* Menghilangkan directory terakhir dari path, misal x/y/z mengembalikan x/y */
    return path.Substring(0, path.LastIndexOf(Path.DirectorySeparatorChar));
}

private void ProcessQueuedItem(Microsoft.Msagl.Drawing.Graph graph, string path)
{
    /* Memproses item yang berada di dalam queue, yaitu menambah nodenya ke graph */
    /* Apabila file sudah ditemukan, artinya node ini tidak dicek tapi ada dalam
    antrian (warna hitam). */
    graph.AddEdge(GetRootPath(path), path);
    Microsoft.Msagl.Drawing.Node r = graph.FindNode(GetRootPath(path));
    Microsoft.Msagl.Drawing.Node c = graph.FindNode(path);
    r.LabelText = GetSafeName(GetRootPath(path));
    c.LabelText = GetSafeName(path);
}

private void ProcessQueuedDirectory(Microsoft.Msagl.Drawing.Graph graph, string
path)
{
    /* Memproses folder yang ada di dalam queue, yaitu menambah nodenya ke graph */
    graph.AddEdge(GetRootPath(path), path).Attr.Color =
Microsoft.Msagl.Drawing.Color.Red;
    Microsoft.Msagl.Drawing.Node r = graph.FindNode(GetRootPath(path));
    Microsoft.Msagl.Drawing.Node c = graph.FindNode(path);
    r.LabelText = GetSafeName(GetRootPath(path));
    c.LabelText = GetSafeName(path);
    c.Attr.FillColor = Microsoft.Msagl.Drawing.Color.MistyRose;
    if (r.Attr.FillColor != Microsoft.Msagl.Drawing.Color.PaleGreen)
    {
        r.Attr.FillColor = Microsoft.Msagl.Drawing.Color.MistyRose;
    }
}

private void ProcessFoundFile(Microsoft.Msagl.Drawing.Graph graph, string path,
List<String> foundPaths)
{

```

```

        /* Memproses file yang ditemukan, yaitu memberikan warna sisi dan node pada
        lintasan dari leaf ke root node */
        graph.AddEdge(GetRootPath(path), path).Attr.Color =
Microsoft.Msagl.Drawing.Color.Green;
        Microsoft.Msagl.Drawing.Node r = graph.FindNode(GetRootPath(path));
        Microsoft.Msagl.Drawing.Node c = graph.FindNode(path);
        r.LabelText = GetSafeName(GetRootPath(path));
        c.LabelText = GetSafeName(path);
        /* Warnai semua node dan edge dari c sampai root */
        Queue<Microsoft.Msagl.Drawing.Node> nodeQueue = new
Queue<Microsoft.Msagl.Drawing.Node>();
        nodeQueue.Enqueue(c);
        while (nodeQueue.Count > 0)
        {
            Microsoft.Msagl.Drawing.Node node = nodeQueue.Dequeue();
            node.Attr.FillColor = Microsoft.Msagl.Drawing.Color.PaleGreen;
            foreach (Microsoft.Msagl.Drawing.Edge edge in node.InEdges)
            {
                edge.Attr.Color = Microsoft.Msagl.Drawing.Color.Green;
                nodeQueue.Enqueue(edge.SourceNode);
            }
        }
        foundFile = true;
        foundPaths.Add(path);
    }
    private void ProcessWrongFile(Microsoft.Msagl.Drawing.Graph graph, string path)
    {
        /* Memproses file yang tidak dicari, yaitu memberinya warna merah */
        graph.AddEdge(GetRootPath(path), path).Attr.Color =
Microsoft.Msagl.Drawing.Color.Red;
        Microsoft.Msagl.Drawing.Node r = graph.FindNode(GetRootPath(path));
        Microsoft.Msagl.Drawing.Node c = graph.FindNode(path);
        r.LabelText = GetSafeName(GetRootPath(path));
        c.LabelText = GetSafeName(path);
        c.Attr.FillColor = Microsoft.Msagl.Drawing.Color.MistyRose;
        if (r.Attr.FillColor != Microsoft.Msagl.Drawing.Color.PaleGreen)
        {
            r.Attr.FillColor = Microsoft.Msagl.Drawing.Color.MistyRose;
        }
    }
    private void PrepareSearch()
    {
        /* Preparasi untuk search, mengeset variabel-variabel ke kondisi yang seharusnya
        */
        isSearching = true;
        clearHyperlinks();
        labelSearchTime.Text = "";
        labelSearchResult.Text = "";
        foundFile = false;
    }

```

```

        foundPaths = new List<String>();
        graph = new Microsoft.Msagl.Drawing.Graph("graph");
        sw = new Stopwatch();
        sw.Start();
    }
    private void EndSearch()
    {
        /* Menunjukkan hasil search dan mereset variabel-variabel yang ada */
        sw.Stop();
        labelSearchTime.Text = $"Search took {sw.Elapsed.TotalSeconds:0.00} seconds.";
        if (foundPaths.Count == 0)
        {
            labelSearchResult.Text = "Path not found!";
        }
        else
        {
            labelSearchResult.Text = $"Found {foundPaths.Count} path(s)!";
            addHyperlinks(foundPaths);
        }
        isSearching = false;
    }
    private void DFS(string root, string filename, List<String> foundPaths)
    {
        /* Proses DFS iteratif menggunakan stack */
        Stack<Item> itemStack = new Stack<Item>();
        itemStack.Push(new Item("FOLDER", root));
        while (itemStack.Count > 0)
        {
            wait(trackBarSpeed.Value);

            Item item = itemStack.Pop();
            if (!checkBoxOccurence.Checked && foundFile)
            {
                ProcessQueuedItem(graph, item.path);
            }
            else
            {
                if (item.type == "FOLDER")
                {
                    /* Proses sebuah directory. */
                    /* Masukkan ke dalam graf dulu, */
                    if (item.path != root)
                    {
                        ProcessQueuedDirectory(graph, item.path);
                    }
                    /* Masukkan subdirectories ke dalam stack */
                    string[] subdirectoryEntries = Directory.GetDirectories(item.path);
                    foreach (string subdir in subdirectoryEntries.Reverse())
                    {

```

```

        itemStack.Push(new Item("FOLDER", subdir));
    }
    /* Masukkan files ke dalam stack */
    string[] fileEntries = Directory.GetFiles(item.path);
    foreach (string file in fileEntries.Reverse())
    {
        itemStack.Push(new Item("FILE", file));
    }
}
else
{
    /* Proses sebuah file. */
    if (GetSafeName(item.path) == filename)
    {
        ProcessFoundFile(graph, item.path, foundPaths);
    }
    else
    {
        ProcessWrongFile(graph, item.path);
    }
}
}

graphViewer.Graph = graph;
}

private void BFS(string root, string filename, List<String> foundPaths)
{
    /* Proses BFS iteratif menggunakan queue */
    Queue<Item> itemQueue = new Queue<Item>();
    itemQueue.Enqueue(new Item("FOLDER", root));
    while (itemQueue.Count > 0)
    {
        wait(trackBarSpeed.Value);

        Item item = itemQueue.Dequeue();
        if (!checkBoxOccurence.Checked && foundFile)
        {
            ProcessQueuedItem(graph, item.path);
        }
        else
        {
            if (item.type == "FOLDER")
            {
                /* Proses sebuah directory. */
                /* Masukkan ke dalam graf dulu, */
                if (item.path != root)
                {

```

```

        ProcessQueuedDirectory(graph, item.path);
    }
    /* Masukkan files ke dalam queue */
    string[] fileEntries = Directory.GetFiles(item.path);
    foreach (string file in fileEntries)
    {
        itemQueue.Enqueue(new Item("FILE", file));
    }
    /* Masukkan subdirectories ke dalam queue */
    string[] subdirectoryEntries = Directory.GetDirectories(item.path);
    foreach (string subdir in subdirectoryEntries)
    {
        itemQueue.Enqueue(new Item("FOLDER", subdir));
    }
}
else
{
    /* Proses sebuah file. */
    if (GetSafeName(item.path) == filename)
    {
        ProcessFoundFile(graph, item.path, foundPaths);
    }
    else
    {
        ProcessWrongFile(graph, item.path);
    }
}
}

graphViewer.Graph = graph;
}

private void addHyperlinks(List<String> paths)
{
    /* Fungsi ini digunakan untuk menambahkan label-label hyperlink yang bisa diklik
    oleh pengguna */
    int Y = 0;
    foreach (String path in paths)
    {
        Label l = new Label();
        Controls.Add(l);
        l.Font = new Font("Open Sans", 8.25F, FontStyle.Underline,
GraphicsUnit.Point);
        l.ForeColor = Color.FromArgb(((int)(((byte)100))), ((int)(((byte)52))),
((int)(((byte)187))));
        l.Location = new Point(26, 359 + Y * 30);
        l.Name = path;
        l.Size = new Size(461, 28);
    }
}

```



```

        l.TabIndex = 23;
        l.Text = path.Substring(textBoxDirectory.Text.Length);
        l.TextAlign = ContentAlignment.MiddleLeft;
        l.Click += new EventHandler(hyperlink_Click);
        l.MouseHover += new EventHandler(hyperlink_Hover);
        l.BringToFront();
        hyperlinks.Add(l);
        Y++;
    }
}

private void clearHyperlinks()
{
    /* Menghapus semua hyperlink yang ada di layar */
    foreach (Label l in hyperlinks)
    {
        this.Controls.Remove(l);
    }
}

private void hyperlink_Click(object sender, EventArgs e)
{
    /* Membuka explorer yang mengarah ke directory sesuai path yang diklik */
    Label l = sender as Label;
    string path = l.Name.Substring(0, l.Name.LastIndexOf('\\'));
    Process.Start(new System.Diagnostics.ProcessStartInfo()
    {
        FileName = path,
        UseShellExecute = true,
        Verb = "open"
    });
}

private void hyperlink_Hover(object sender, EventArgs e)
{
    /* Menunjukkan bahwa suatu hyperlink clickable */
    Label l = sender as Label;
    l.Cursor = Cursors.Hand;
}

public Form1()
{
    InitializeComponent();
    this.Text = "Folder Crawler";
}

private void buttonChooseFolder_Click(object sender, EventArgs e)
{
    CommonOpenFileDialog dialog = new CommonOpenFileDialog();
    dialog.InitialDirectory = Directory.GetCurrentDirectory();
}

```

```

        dialog.IsFolderPicker = true;
        if (dialog.ShowDialog() == CommonFileDialogResult.Ok)
        {
            textBoxDirectory.Text = dialog.FileName;
            validFolder = true;
        }
    }
    private void buttonSearch_Click(object sender, EventArgs e)
    {
        if (isSearching)
        {
            MessageBox.Show("Please wait for the current search to end first!");
            return;
        }
        if (!validFolder)
        {
            MessageBox.Show("Please input a valid directory to search from first!");
            return;
        }
        if (textBoxInputFile.TextLength == 0)
        {
            MessageBox.Show("Please input a filename to search first!");
            return;
        }
        if (!radioDFS.Checked && !radioBFS.Checked)
        {
            MessageBox.Show("Please select a crawling method first!");
            return;
        }
        if (radioBFS.Checked)
        {
            PrepareSearch();
            BFS(textBoxDirectory.Text, textBoxInputFile.Text, foundPaths);
            EndSearch();
            return;
        }
        if (radioDFS.Checked)
        {
            PrepareSearch();
            DFS(textBoxDirectory.Text, textBoxInputFile.Text, foundPaths);
            EndSearch();
            return;
        }
    }

    private void trackBarSpeed_Scroll(object sender, EventArgs e)
    {
        labelCrawlSpeed.Text = Convert.ToString(trackBarSpeed.Value) + " ms";
    }

```

```
}  
}
```

## Penjelasan Struktur Data yang Digunakan

Semua file dan folder akan disimpan dalam sebuah class Item yang terdiri dari type dan name. Type di sini merupakan penanda apakah item bertipe file atau folder. Pada DFS, Item akan disimpan ke dalam stack (Last In First Out). Hal ini akan mensimulasikan traversal graf yang akan melakukan pencarian melalui folder terdalam terlebih dahulu hingga file ditemukan. Pada BFS, Item akan disimpan ke dalam queue (First In First Out). Hal ini akan mensimulasikan traversal graf yang melakukan pencarian semua file dan folder pada depth yang rendah terlebih dahulu. Sedangkan hyperlink akan disimpan dalam sebuah list of string yang akan selalu berubah sesuai dengan kondisi pencarian.

## Tata Cara Penggunaan Program

### Cara Menjalankan Program

Untuk menjalankan program ini cukup mudah, hanya perlu run FolderCrawlerTest.exe yang terdapat pada folder bin.

### Cara Menggunakan Program

Terdapat lima input yang harus diisi untuk menggunakan program ini, yaitu :

- Folder Path

User dapat memilih folder path mana yang akan digunakan sebagai root dari pencarian. Saat akan mengisi field tersebut akan terbuka file explorer local dari device tersebut, dapat dipilih folder seperti pada umumnya.


- File Name

User dapat mengetikkan nama file yang akan dicari secara manual pada field yang tersedia di sebelah kanan field untuk folder path.

- Pemilihan DFS/BFS

User dapat checklist salah satu box DFS atau BFS untuk memilih algoritma apa yang akan dipakai dalam searching. Hal ini akan memengaruhi urutan pencarian dan visualisasi.

- Check Box Find All Occurence



User dapat checklist box ini jika ingin mencari semua file bernama sesuai dengan input pada folder root yang sudah kita input. Jika tidak dichecklist, maka hanya file pertama yang dioutput pathnya dan dilakukan visualisasinya.

- Slider Crawl Speed

User dapat slide ke kiri ataupun kanan slider ini. Slider ini berguna untuk mengubah kecepatan visualisasi.

Terdapat dua output dari program ini, yaitu :

- Hyperlink

Hyperlink ini adalah path dari root folder hingga file yang dicari. Hyperlink ini bisa terdapat lebih dari satu jika dilakukan find all occurrences. Hyperlink juga dapat diklik dan akan redirect membuka folder tersebut.

- Visualisasi

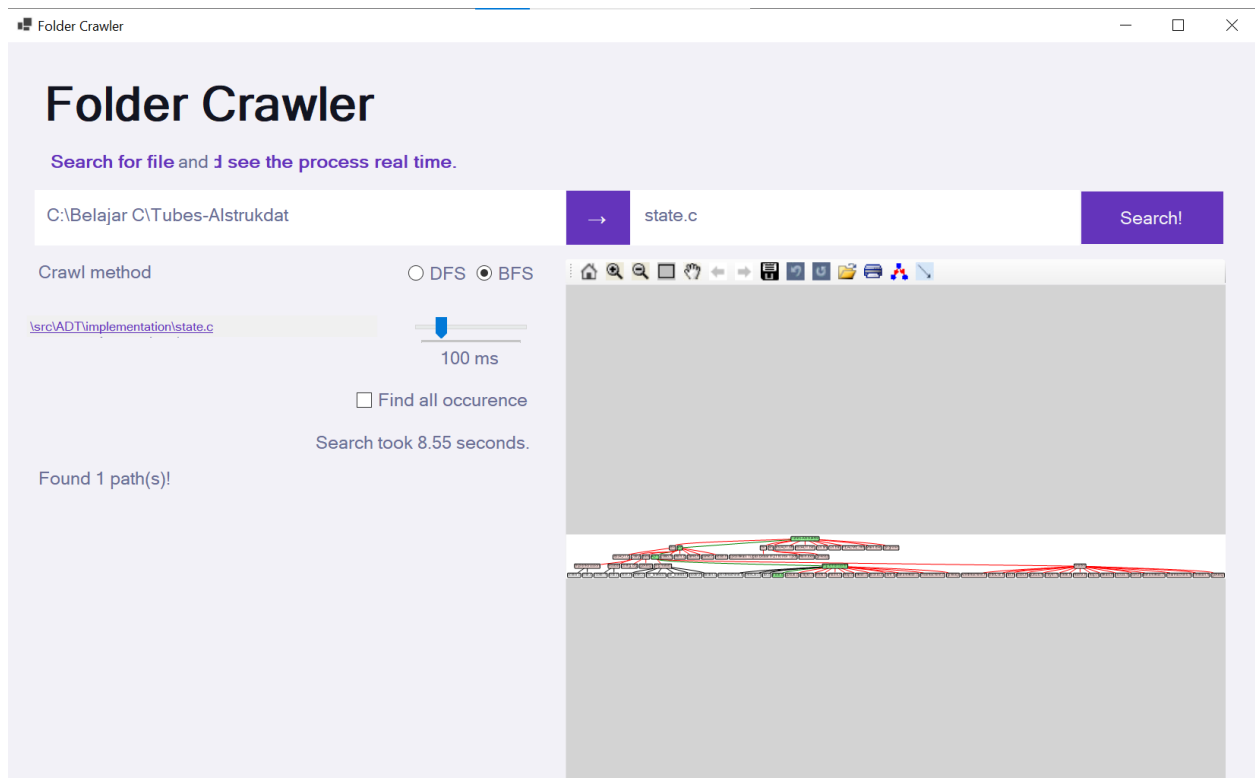
Visualisasi ini berupa gambar yang melambangkan file dan folder yang dikunjungi. Urutan yang muncul pada visualisasi tergantung pada algoritma yang digunakan yaitu DFS/BFS. Warna merah artinya file/folder sudah dikunjungi tetapi tidak mengarah ke solusi. Warna hijau artinya file/folder mengarah ke solusi. Tanpa warna artinya file atau folder belum dicek.

## Pengujian

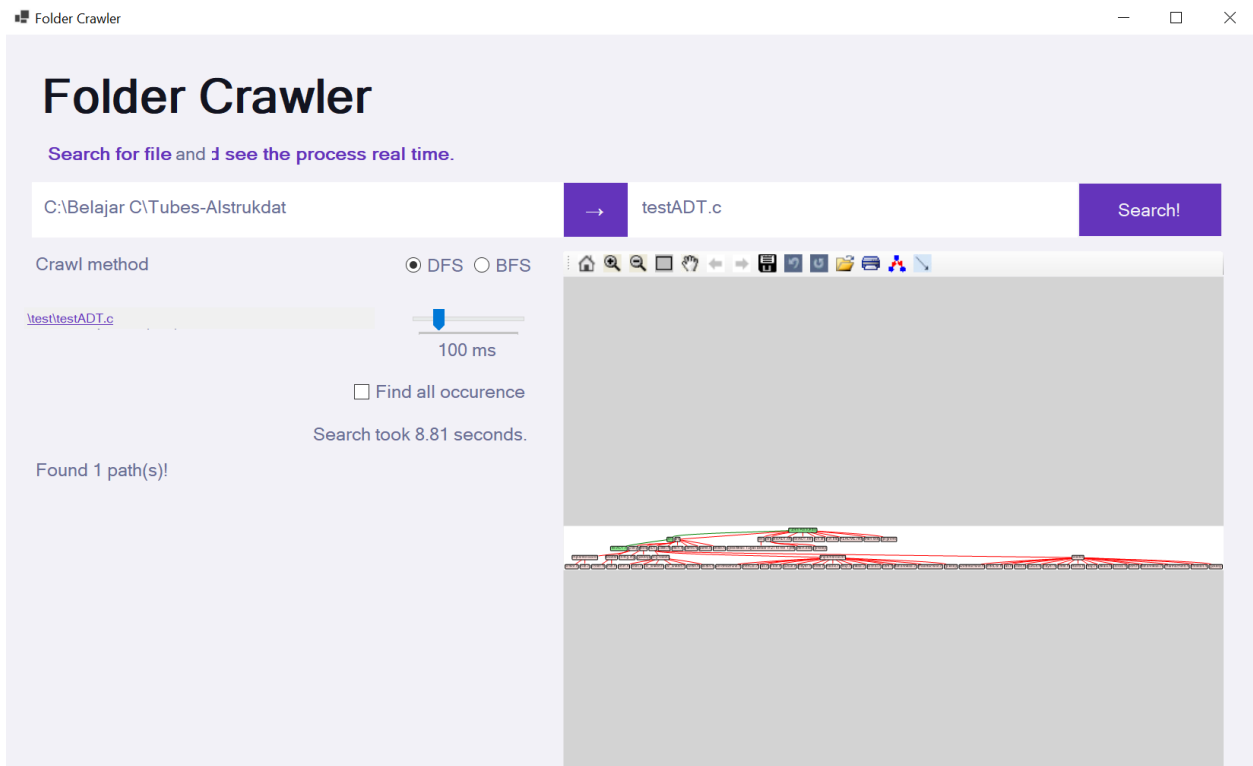
### Perbandingan DFS dengan BFS

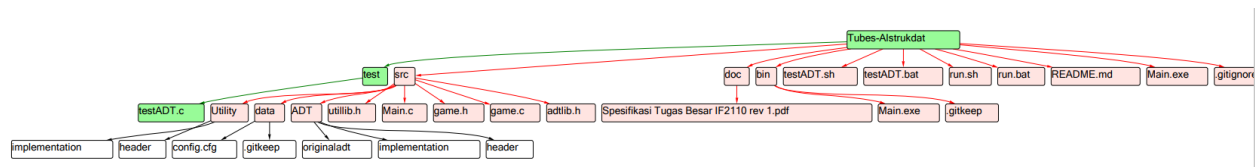
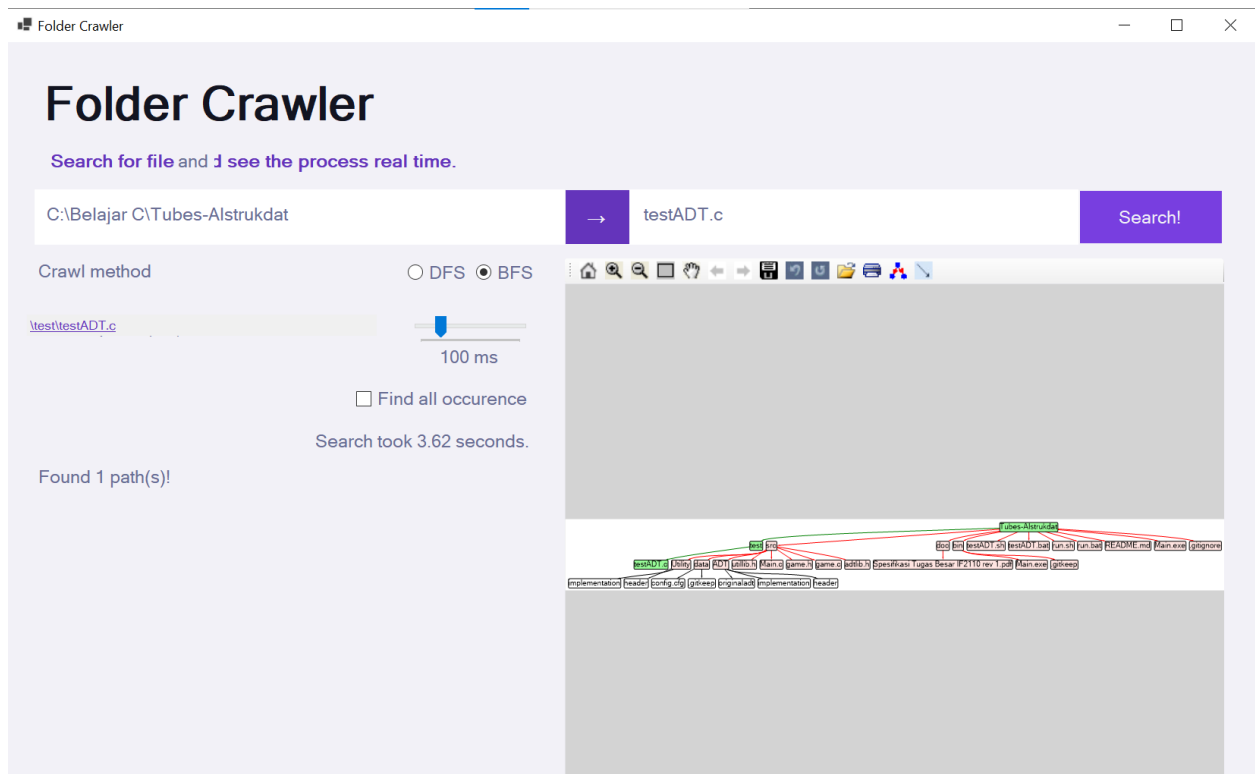
Mencari File state.c (Perbedaan tidak signifikan)





Mencari file TestADT.c (BFS lebih baik)

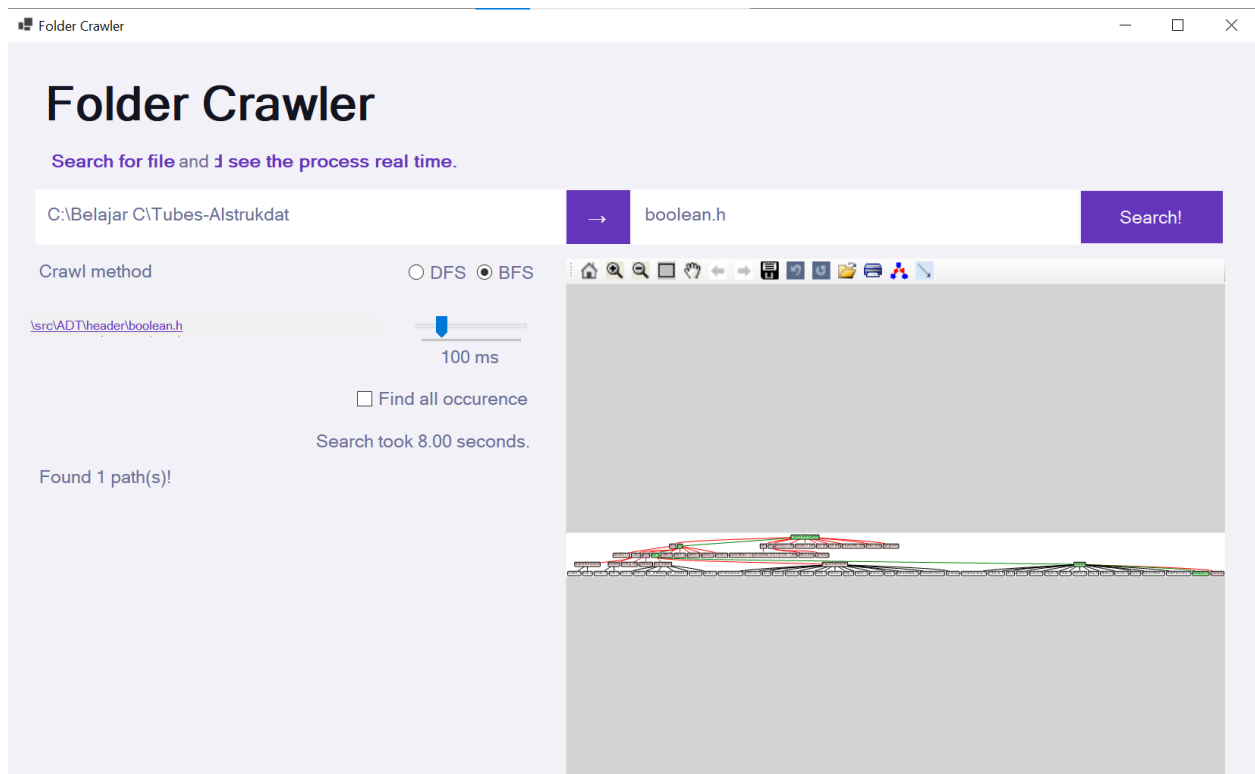




Mencari File boolean.h (DFS lebih baik)



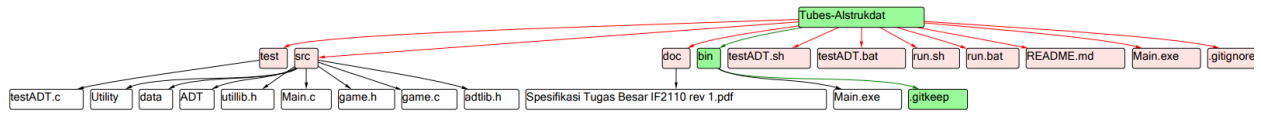




Perbandingan Find All Occurence Dengan Pencarian Biasa

Mencari file .gitkeep





Folder Crawler

Search for file and see the process real time.

C:\Belajar C\Tubes-Alstrukdat → .gitkeep Search!

Crawl method: ☐ DFS ☒ BFS

Find all occurrence ☐

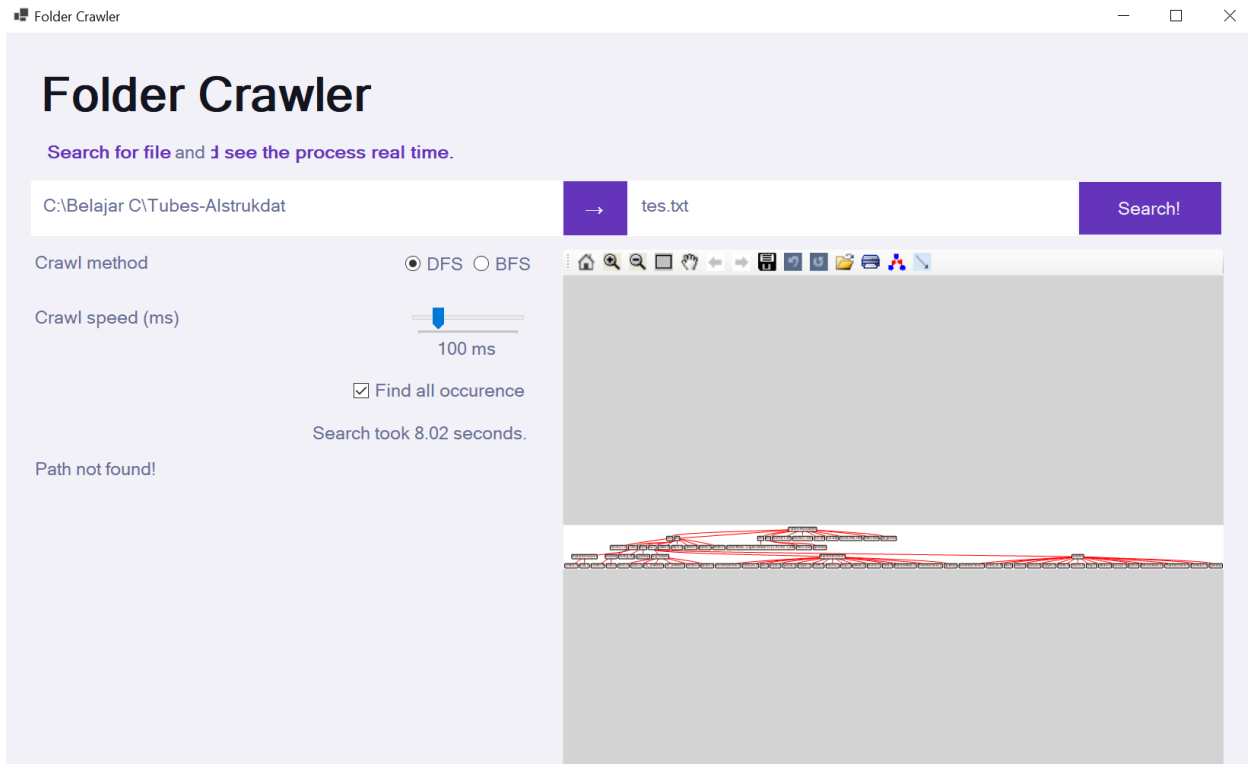
Search took 7.98 seconds.

Found 4 path(s)!

[\bin\.gitkeep](#)  
[\src\data\.gitkeep](#)  
[\src\ADT\header\.gitkeep](#)  
[\src\ADT\implementation\.gitkeep](#)




Mencari file yang tidak ada



## Analisis

Dari hasil percobaan berbagai test case yang dilakukan, selalu ditemukan solusi dari permasalahan baik menggunakan algoritma DFS maupun BFS. Kedua algoritma tidak dapat dibandingkan dari segi kecepatan karena waktu untuk menemukan solusinya rata-rata hanya sepersekian detik. Akan tetapi dengan melihat kompleksitas algoritma yang dimiliki oleh kedua algoritma tersebut, terdapat perbedaan kompleksitas tergantung dari test case yang diberikan. Jika pada test case yang diberikan, titik tujuan pada representasi graf memiliki kedalaman yang tinggi relatif terhadap titik mulai, kompleksitas dari algoritma BFS cenderung lebih besar daripada kompleksitas algoritma DFS. Sebaliknya, jika pada test case yang diberikan, titik tujuan pada representasi graf memiliki jarak yang pendek relatif terhadap titik mulai dan lebih banyak folder didalamnya, maka kompleksitas dari algoritma DFS cenderung lebih besar daripada kompleksitas algoritma BFS. Hal ini berarti algoritma DFS lebih efektif apabila titik tujuan yang dicari persoalan memiliki kedalaman yang tinggi relatif terhadap titik mulai dan tidak terlalu banyak folder.



Sedangkan algoritma BFS lebih efektif apabila titik tujuan yang dicari memiliki lebih banyak folder di dalam folder root dan kedalamannya relatif lebih pendek terhadap titik mulai.

## Bab 5 - Kesimpulan dan Saran

### Kesimpulan

Algoritma DFS dan BFS dapat digunakan untuk menyelesaikan permasalahan berbasis graf statis. Meskipun kedua algoritma tidak selalu menghasilkan solusi tercepat atau terpendek, keduanya akan selalu memberikan solusi terhadap permasalahan. Algoritma DFS dan BFS memiliki keunggulan masing-masing tergantung dari posisi titik awal dan titik akhir. Jika graf digambarkan secara bertingkat dimulai dari titik awal, algoritma DFS lebih efektif jika titik yang dicari berjarak jauh dengan titik awal dan folder yang ada lebih sedikit, sedangkan algoritma BFS lebih efektif jika titik yang dicari berjarak dekat dengan titik awal dan folder yang ada lebih banyak.

### Saran

Penulis menyadari bahwa masih terdapat banyak kekurangan dalam pembuatan laporan ini. Penulis berharap bahwa persoalan dapat dijelajahi lebih lanjut agar dapat ditemukan solusi yang lebih baik dan efisien untuk menyelesaikan permasalahan soal.



## Link Repository

[https://github.com/ubaidalih/Tubes2\\_13520056](https://github.com/ubaidalih/Tubes2_13520056)

## Link Video

<https://youtu.be/8n9eSWD9ZUY>





## Daftar Pustaka

Tim Dosen IF221 Strategi Algoritma. 2022. Slide Perkuliahan BFS dan DFS Bagian 1 dan Bagian 2.

Microsoft. 2022. Dokumentasi .NET yang diakses dari laman  
<https://docs.microsoft.com/en-us/dotnet/?view=net-6.0>

Microsoft. 2022. Library MSAGL yang diakses dari laman  
<https://github.com/microsoft/automatic-graph-layout>.