IF2211 STRATEGI ALGORITMA
TUGAS BESAR 2 SEMESTER II TAHUN 2021/2022

Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling

bing SLAIMET

13520048 Arik Rayi Arkananta
13520132 Januar Budi Ghifari
13520141 Yoseph Alexander Siregar

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

2021

Bab 1

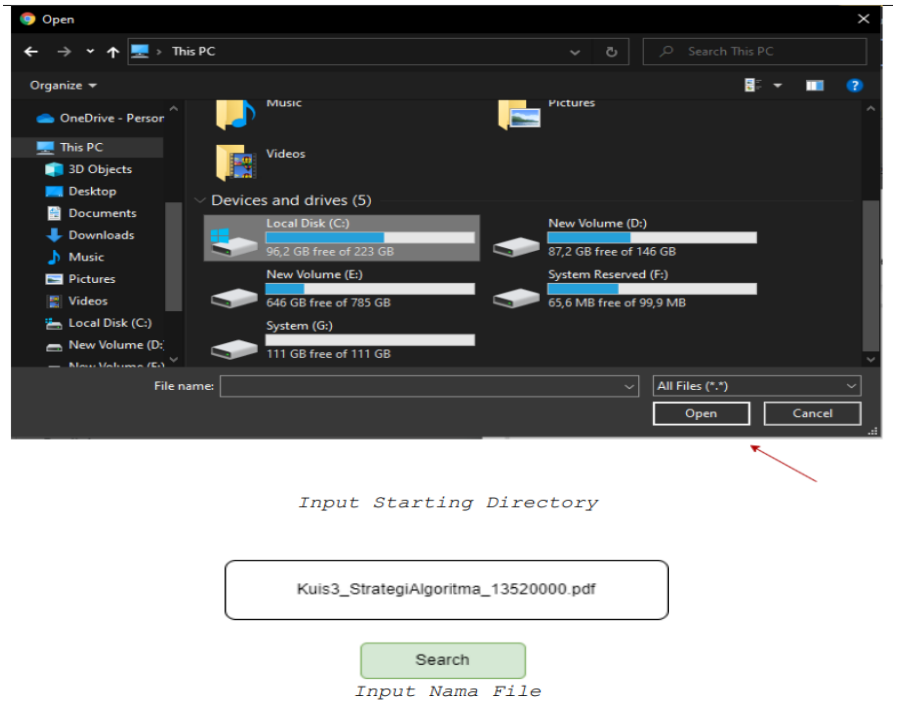
Deskripsi Tugas

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

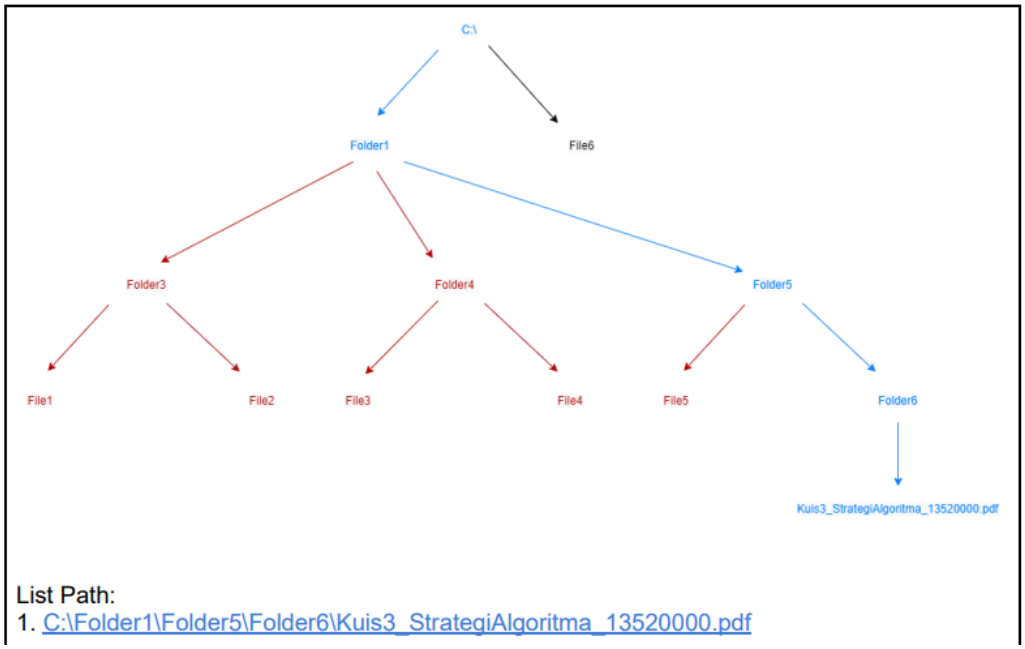
Selain pohon, diminta juga untuk menampilkan list path dari daun-daun yang bersesuaian dengan hasil pencarian. Path tersebut diharuskan memiliki hyperlink menuju folder parent dari file yang dicari, agar file langsung dapat diakses melalui browser atau file explorer. Contoh hal-hal yang dimaksud akan dijelaskan di bawah ini.

Contoh masukan aplikasi:



Gambar 1.1 Contoh Input Program
Sumber : Tugas-Besar-2-IF2211-Strategi-Algoritma-2022

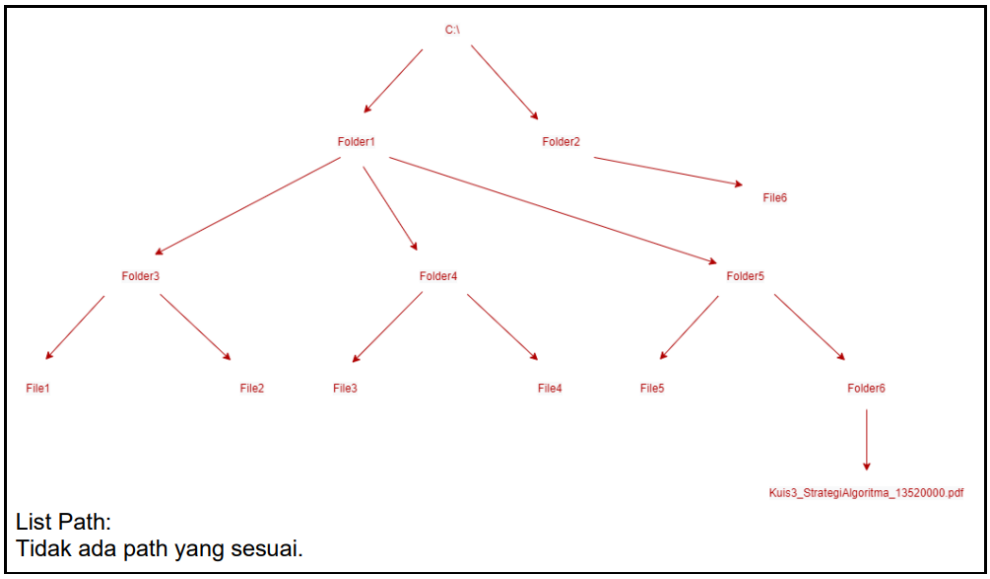
Contoh keluaran aplikasi:



Gambar 1.2 Contoh Output Program
Sumber : Tugas-Besar-2-IF2211-Strategi-Algoritma-2022

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 tetapi belum diperiksa diberi warna hitam. Pada tugas besar ini, rute yang **dilewati** diwarnai dengan warna **merah**, rute menuju **tempat file** diberi warna **hijau**, dan rute yang masuk antrian tetapi **belum diperiksa** diberi warna **hitam.**

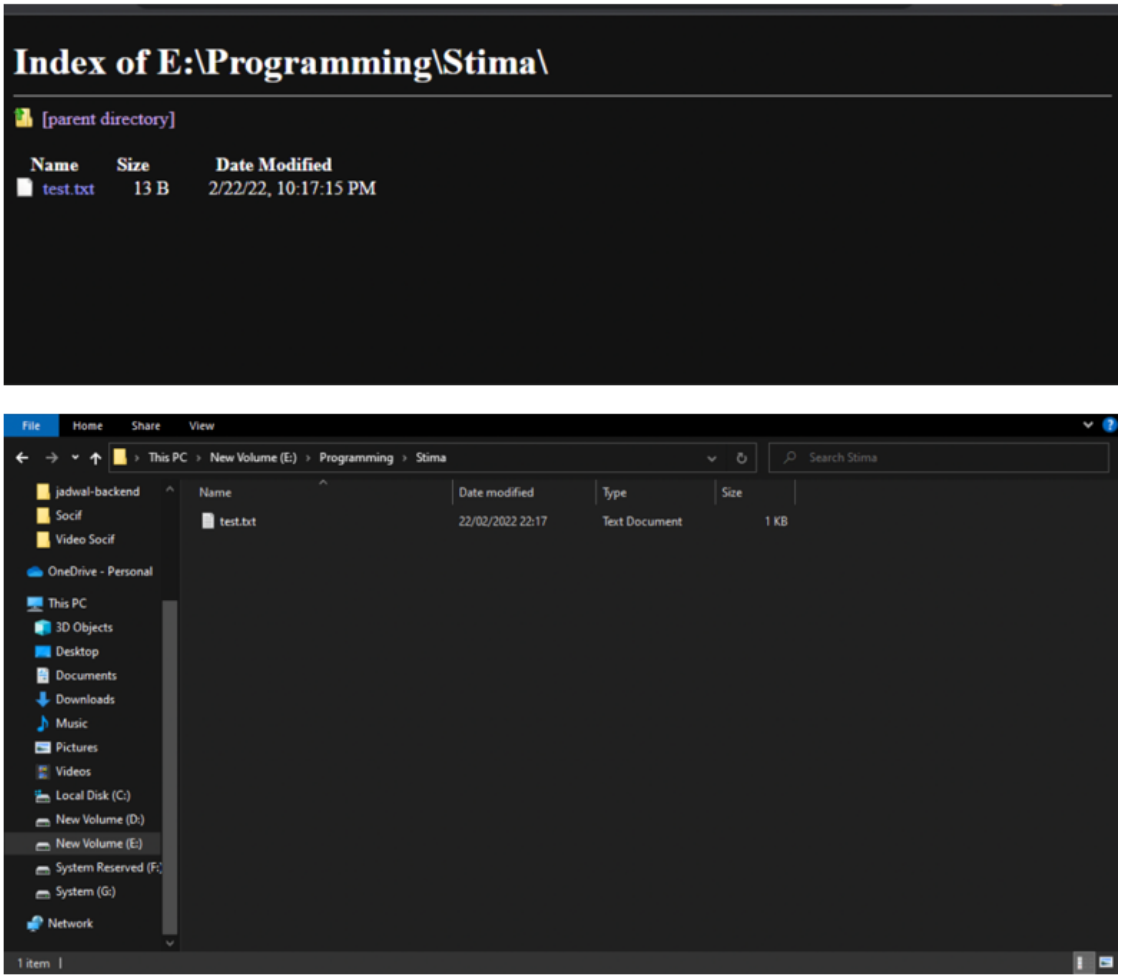


Gambar 1.3 Contoh Output Program jika File Tidak Ditemukan
Sumber : Tugas-Besar-2-IF2211-Strategi-Algoritma-2022

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 1.4 Contoh Apabila Hyperlink di-klik
Sumber : Tugas-Besar-2-IF2211-Strategi-Algoritma-2022

Bab 2

Landasan Teori

I. Dasar Teori

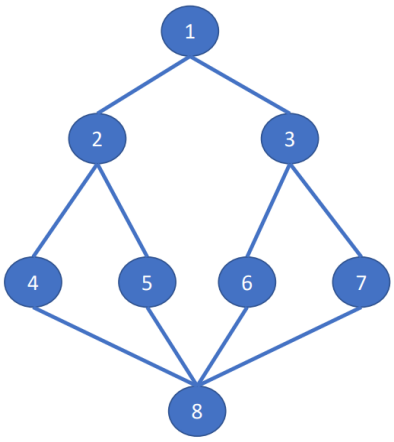
A. Graph Traversal

Graph adalah sebuah data struktur non-linear yang terdiri atas beberapa simpul dan sisi yang menghubuungkannya. Graph Traversal atau juga disebut sebagai Graph Search adalah suatu algoritma atau proses untuk mengunjungi tiap simpul dari graph tersebut secara sistematik.

Salah satu contoh pengaplikasian dari Graph adalah pohon. Dalam tugas besar ini, struktur file dan folder digambarkan sebagai sebuah sebuah pohon berakar. Berdasarkan proses pengunjungannya dalam algortima traversal ini, algoritma dapat dibagi menjadi 2, yaitu BFS (Breadth First Search) dan DFS(Depth First Search). Kedua algoritma ini merupakan penerapan dari *Graph Traversal* yang berbasis pohon.

B. Breadth First Search

BFS atau juga dikenal sebagai pencarian melebar, misalkan terdapat sebuah simpul S dalam sebuah *Graph* G, akan mengunjungi setiap simpul yang bertetangga dengan simpul S tersebut. Setelah mengunjungi seluruh simpul tersebut, proses traversal dilanjutkan dengan mengunjungi seluruh simpul yang bertetangga dengan simpul-simpul yang dikunjungi sebelumnya. Oleh karena itu, dalam pengimplementasiannya, algoritma BFS biasa memanfaatkan struktur data *Queue* atau antiran dalam algoritmanya. Pencarian BFS berakhir apabila seluruh simpul pada *Graph* G sudah dikunjungi.



Gambar 2.1 Contoh *Graph* G

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/

Dalam Graph G diatas, proses pencarian BFS akan memulai dari simpul 1. Simpul 2 dan 3 yang bertetangga dengan simpul 1 akan masuk ke dalam misal *Queue* Q = {2,3), selanjutnya pencarian akan mengunjungi simpul 2 dan simpul 4 dan 5 akan masuk dalam Q = {3,4,5}, selanjutnya mengunjungi simpul 3 dan simpul 6 dan 7 masuk dalam Q = {4,5,6,7} selanjutnya mengunjungi simpul 4 dan simpul 8 masuk dalam Q = {5,6,7,8} dan selanjutnya mengunjungi simpul 5, 6, 7, dan 8 secara berurut. Sehingga urutan akhir dari proses BFS pada *Graph* G ini adalah 1->2->3->4->5->6->7->8.

C. Depth First Search

DFS atau juga dikenal sebagai pencarian mendalam, misalkan terdapat sebuah simpul S dalam sebuah *Graph* G, akan mengunjungi satu simpul yang bertetangga dengan simpul S tersebut dan mengulang proses pencarian DFS tersebut pada simpul tersebut. Ketika sudah mencapai simpul dimana seluruh simpul tetangganya sudah dikunjungi, dilakukan proses *backtrack* atau runut-balik ke simpul sebelumnya yang dikunjungi yang memiliki simpul tetangga yang belum dikunjungi. Proses DFS kembali dilanjutkan pada simpul tersebut. Pencarian DFS akan berakhir apabila tidak lagi terdapat simpul yang dapat dikunjungi dari simpul-simpul yang sudah dikunjungi.

Gambar 2.2 Contoh *Graph* H

Sumber :

https://www.thecrazyprogrammer.com/2017/06/difference-between-bfs-and-dfs.html

Pada *Graph* H diatas, proses DFS dimulai dari simpul A dan mengunjungi simpul B, lalu mengunjungi simpul D, karena sudah tidak terdapat simpul tetangga dari D yang belum dikunjungi maka proses DFS melakukan *backtrack* menuju simpul B, kasus yang tersama terjadi pada simpul B sehingga dilakukan *backtrack* menuju simpul A, selanjutnya dikunjungi simpul C lalu simpul E, pada simpul E juga tidak terdapat simpul tetangga yang belum dikunjungi sehingga dilakukan *backtrack* menuju simpul C dan dilanjutkan mengunjungi simpul F. Sehingga urutan akhir proses DFS adalah A->B->D->C->E->F.

II. C# Desktop Application Development

Dalam pengembangan aplikasi untuk tugas *folder crawling* ini, digunakan IDE Visual Studio, lebih tepatnya Visual Studio .NET untuk pengembangan menggunakan bahasa C#. Dalam pengerjaan tugas ini, digunakan *Window Forms App* template dan *Toolbox* . Windows Forms merupakan sebuah framework UI yang dapat digunakan untuk membangun sebuah aplikasi windows berbasis desktop. Pada windows forms ini, berbagai *utility* atau kegunaan dari UI yang ada dienkapsulasi untuk digunakan dalam pengembangan aplikasi.

Untuk membuat sebuah project baru, pada IDE Visual Studio pilih *Create New Project* dan pilih Windows Forms App(.NET Framework) sebagai template yang akan digunakan. Selanjutnya isi nama dari projek yang dibuat dan *create.* Projek berhasil dibuat dan sudah bisa mulai dikembangkan.

Untuk membuka toolbox, pilih pada bagian *View* lalu *Toolbox.* Pada toolbox terdapat berbagai jenis kontrol yang dapat digunakan dan ditambahkan pada form yang sudah ada.

Bab 3

Analisis Pemecahan Masalah

I. Langkah-langkah Pemecahan Masalah

Untuk menyelesaikan permasalahan ini, kami telah melakukan langkah-langkah di bawah ini:

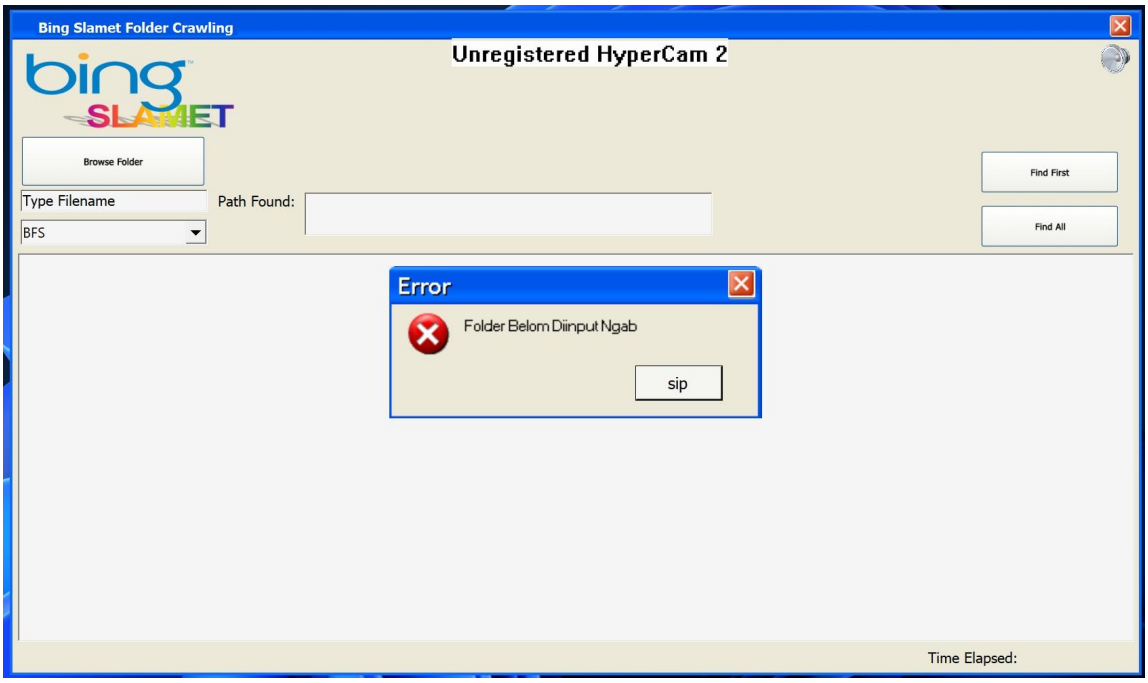
1. Mengidentifikasi permasalahan yang perlu dipecahkan : Aplikasi GUI sederhana dengan fitur Folder Crawling dengan visualisasi tree dengan memanfaatkan Algoritma Traversal BFS dan DFS
2. Mempelajari dan memahami algoritma traversal BFS dan DFS
3. Memahami struktur folder dan file pada sistem operasi Windows
4. Mempelajari bahasa pemrograman C# dan juga Framework .NET dengan Visual Studio khususnya untuk mengakses file dan folder pada Windows, visualisasi graph dengan library MSAgl, dan juga pembuatan GUI berjenis Windows Form App
5. Membuat rancangan algoritma awal untuk pencarian BFS dan DFS
6. Mendesain GUI dengan Figma
7. Merealisasikan program lengkap

II. Proses Mapping Persoalan menjadi elemen-elemen algoritma BFS dan DFS.

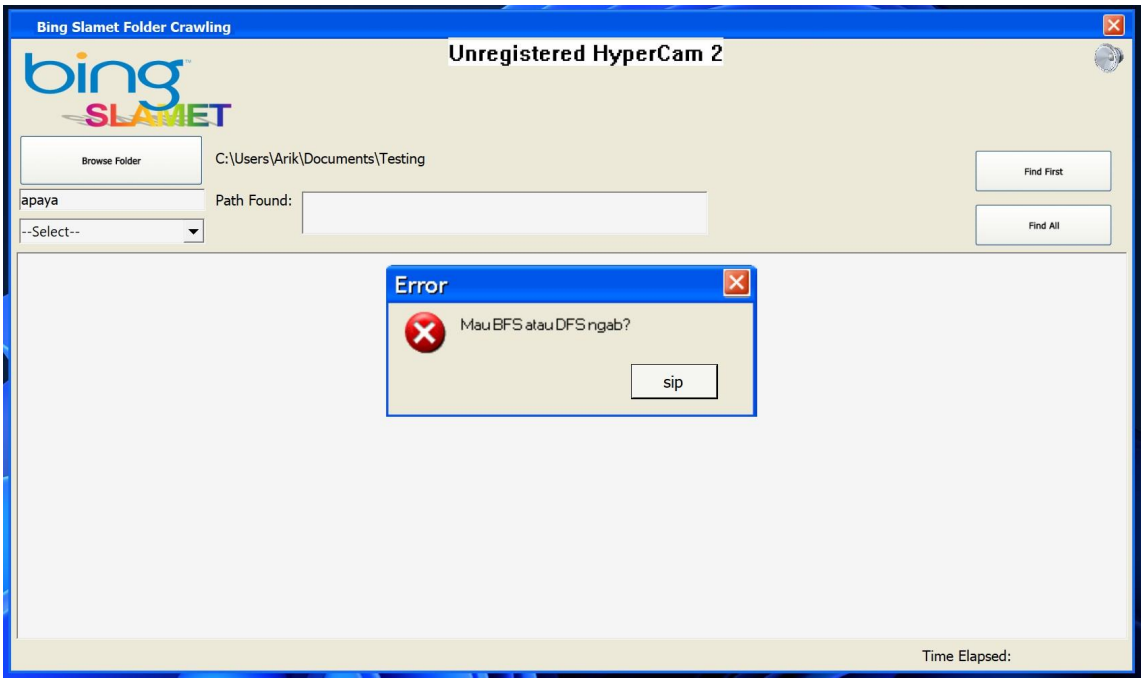
Elemen	Representasi
Simpul Akar	Directory awal pada pencarian
Simpul	Folder-folder dan file-file yang ada di dalam directory awal pencarian
Sisi	Hubungan tiap simpul (parent folder-nya atau child folder-nya)

III. Contoh ilustrasi kasus lain yang berbeda dengan contoh pada spesifikasi tugas

a. Saat Program Dijalankan tapi Lokasi Start Belum Diinput



b. Saat Program Dijalankan tapi Metode Pencarian Belum Diinput



Bab 4

Implementasi dan Pengujian

I. Implementasi program

A. Fungsi pencarian BFS

```
public void bfs_search(int set)
{
    stopwatch.Start();
    var options = new EnumerationOptions()
    {
        IgnoreInaccessible = true,
        AttributesToSkip = FileAttributes.System
    };
    int i = 0;
    bool found = false;
    tree.Add(new dirTree(-999, Path.GetFileName(startdir), startdir,
"Folder", "Not"));
    string parentdir = startdir;
    while (i < tree.Count())
    {
        List<string> temptree = Directory.GetFiles(tree[i].directory,
"*", options).ToList();
        foreach (string a in temptree)
        {
            if ((Path.GetFileName(a) == this.filename))
            {
                tree.Add(new dirTree(i, Path.GetFileName(a), a, "File",
"Found"));
                found = true;
                makeAllFound(tree[tree.Count() - 1]);
                dir_found.Add(a);
            }
            else
            {
                tree.Add(new dirTree(i, Path.GetFileName(a), a, "File",
"Not"));
            }
        }
        if (set == 0 && found)
        {
            return;
        }
        temptree = Directory.GetDirectories(tree[i].directory, "*",
options).ToList();
        foreach (string a in temptree)
        {
            tree.Add(new dirTree(i, Path.GetFileName(a), a, "Folder",
"Not"));
        }
        try
```

```
        {
            i++;
            parentdir = tree[i].directory;
            while (tree[i].type == "File" && i < tree.Count())
            {
                i++;
                parentdir = tree[i].directory;
            }
        }
        catch { break; }
    }
    stopwatch.Stop();
}
```

B. Fungsi pencarian DFS

```
public bool dfs_reccursion(string currDir, int set, bool done)
{
    stopwatch.Start();
    var options = new EnumerationOptions()
    {
        IgnoreInaccessible = true,
        AttributesToSkip = FileAttributes.System
    };
    string[] files = Directory.GetFiles(currDir,"*",options);

    foreach(string content in files){
        if (Path.GetFileName(content) == this.filename)
        {
            tree.Add(new dirTree(findIdbyDir(currDir),
Path.GetFileName(content), content, "File", "Found"));
            done = true;
            makeAllFound(tree[tree.Count() - 1]);
            dir_found.Add(content);
            if (done && set == 0)
            {
                return done;
            }
        }
        else
        {
            tree.Add(new dirTree(findIdbyDir(currDir),
Path.GetFileName(content), content, "File", "Not"));
        }
    }
    string[] Dirs = Directory.GetDirectories(currDir,"*",options);
    foreach (string Folder in Dirs)
    {
        if (done && set == 0)
        {
            tree.Add(new dirTree(findIdbyDir(currDir),
Path.GetFileName(Folder), Folder, "Folder", "Queued"));
        }
```

```
        else
        {
            tree.Add(new dirTree(findIdbyDir(currDir),
Path.GetFileName(Folder), Folder, "Folder", "Not"));
            done = dfs_reccursion(Folder, set, done);
        }
    }
    stopwatch.Stop();
    return done;
}
```

C. Fungsi pembuatan Graf

```
public void makeGraph()
{
    int j = tree.Count() - 1;
    while (j != 0)
    {
        if (findIdbyName(tree[j].name) != j)
        {
            try
            {
                tree[findIdbyName(tree[j].name)].name =
tree[findIdbyName(tree[j].name)].name + " (" +
tree[tree[findIdbyName(tree[j].name)].parent_id].name + ")";
            } catch { }
            tree[j].name += " (" + tree[tree[j].parent_id].name + ")";
        }
        if (tree[j].found == "Found")
        {
            graph.AddEdge(tree[tree[j].parent_id].name,
tree[j].name).Attr.Color = Microsoft.Msagl.Drawing.Color.Blue;
            graph.FindNode(tree[j].name).Attr.FillColor =
Microsoft.Msagl.Drawing.Color.Blue;
            graph.FindNode(tree[tree[j].parent_id].name).Attr.FillColor
= Microsoft.Msagl.Drawing.Color.Blue;
        }
        else if (tree[j].found == "Queued")
        {
            graph.AddEdge(tree[tree[j].parent_id].name,
tree[j].name).Attr.Color = Microsoft.Msagl.Drawing.Color.Black;
        }
        else
        {
            graph.AddEdge(tree[tree[j].parent_id].name,
tree[j].name).Attr.Color = Microsoft.Msagl.Drawing.Color.Red;
            graph.FindNode(tree[j].name).Attr.FillColor =
Microsoft.Msagl.Drawing.Color.Red;
            if
(graph.FindNode(tree[tree[j].parent_id].name).Attr.FillColor !=
Microsoft.Msagl.Drawing.Color.Blue)
            {
```

```
graph.FindNode(tree[tree[j].parent_id].name).Attr.FillColor =
Microsoft.Msagl.Drawing.Color.Red;
                }
            }
            j--;
        }
}
```

II. Struktur data yang digunakan dalam program dan spesifikasi program

A. dirTree

Class dirTree merupakan struktur data buatan yang menyusun Tree di dalam struktur data Search. Fungsi dari struktur data ini adalah untuk menyimpan *path* dan nama tiap objek yang merupakan folder atau file, mengetahui siapa *parent* dari tiap objek sehingga dapat dengan mudah mencari *parent folder* dalam aplikasi program ini, dan mencatat apakah suatu objek file merupakan yang dicari atau bukan.

B. Search

Class search merupakan struktur data buatan dimana dilaksanakan algoritma pencarian BFS dan DFS. Isi dari struktur data ini adalah semua yang dibutuhkan untuk proses pencarian, yaitu Tree yang berfungsi untuk menyimpan tiap folder atau file dalam pencarian, startdir yaitu string *directory* awal pencarian, filename yaitu string nama file yang ingin dicari, dirFound yaitu list path dari semua file yang telah ditemukan (yang namanya sesuai dengan filename), graph yang berfungsi untuk visualisasi, dan stopwatch untuk menghitung waktu berjalannya pencarian.

C. List

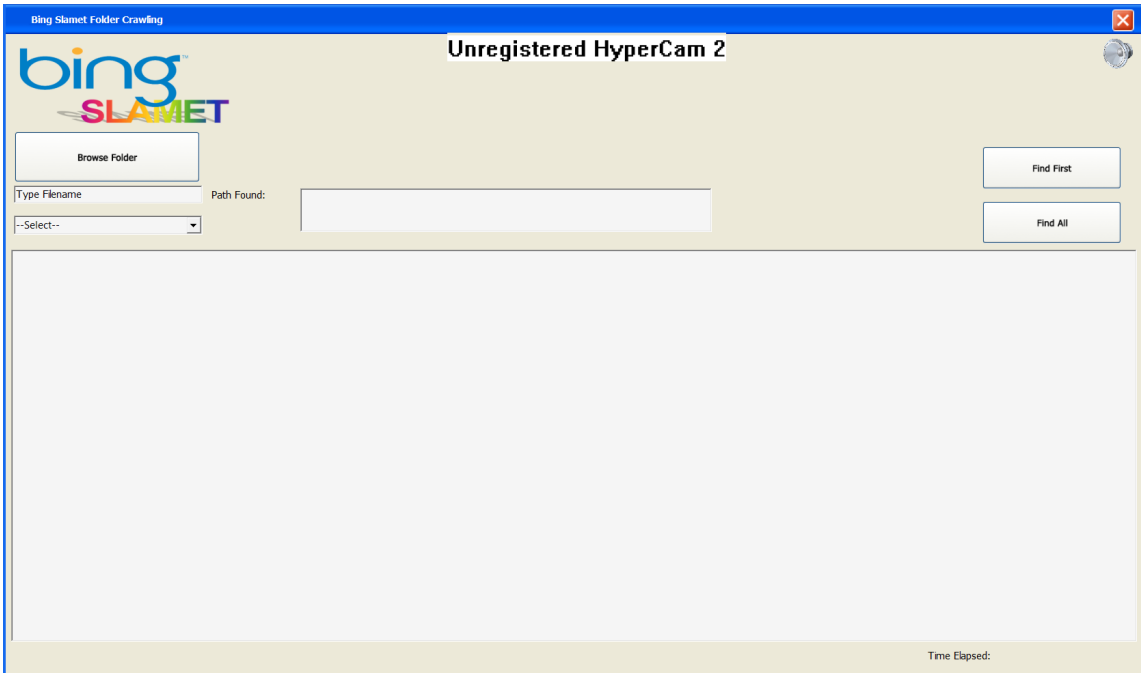
Struktur data ini digunakan di dalam struktur data search sebagai dasar dari Tree yang akan terdiri dari dirTree. Struktur data ini juga digunakan untuk menyusun dirFound pada search yang merupakan kumpulan list path dari semua file yang telah ditemukan (yang namanya sesuai dengan filename).

D. Graph

Graph digunakan untuk membuat visualisasi hasil dari pencarian yang didapat dari Tree yang sudah terbentuk. Graph nantinya akan dibuat dan diberi warna setelah pencarian sudah selesai dan Tree sudah lengkap.

III. Penggunaan program

1. Interface Program



Gambar 1.2 Interface Bing Slamet

Sumber : Arsip Pribadi

Gambar di atas merupakan interface dari aplikasi GUI sederhana yang kami buat untuk memenuhi tugas kali ini. Pada interface program ini, terdapat beberapa hal seperti :

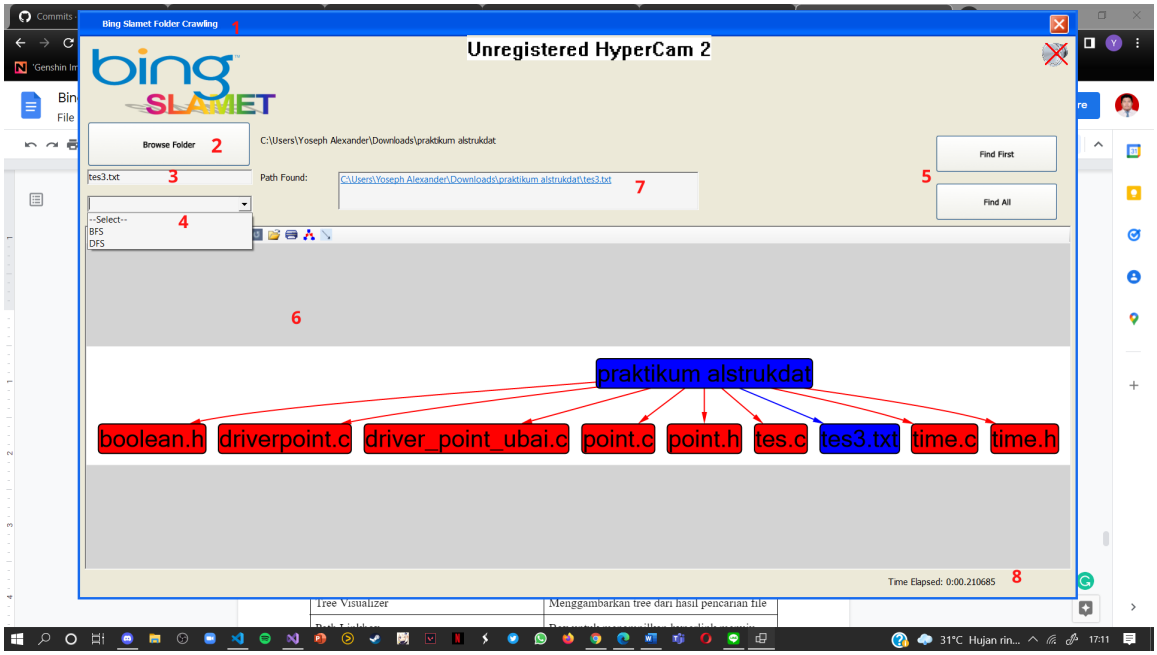
a. Logo Kelompok (Bing Slamet)
b. Tombol On/Off lagu (Background Song)
c. Button Browse Folder (memilih folder root), Find First dan Find All (Find Method)
d. Dropdown BFS dan DFS (Searching Method)
e. Textbox filename (Input Filename)
f. Linkbox untuk Path
g. Panel untuk tampilan tree (Tree Visualizer)
h. Label Time

2. Fitur Program

Fitur	Kegunaan
Browse Folder	Memilih folder yang akan digunakan sebagai akar pencarian
Input Filename	Nama file yang akan dicari dalam program dimulai dari folder yang dipilih sebagai akar
Searching Method	Memilih metode traversal yang akan digunakan dalam pencarian file
Find Method (First / All)	Memilih opsi kemunculan dari file yang dicari
Tree Visualizer	Menggambarkan tree dari hasil pencarian file sesuai pemeriksaan **(Bonus)**
Path Linkbox	Box untuk menampilkan hyperlink menuju file yang dicari
Duration	Menampilkan waktu yang dibutuhkan untuk melakukan pencarian
Song On/Off	Mematikan atau menyalakan lagu

3. Alur Penggunaan Program

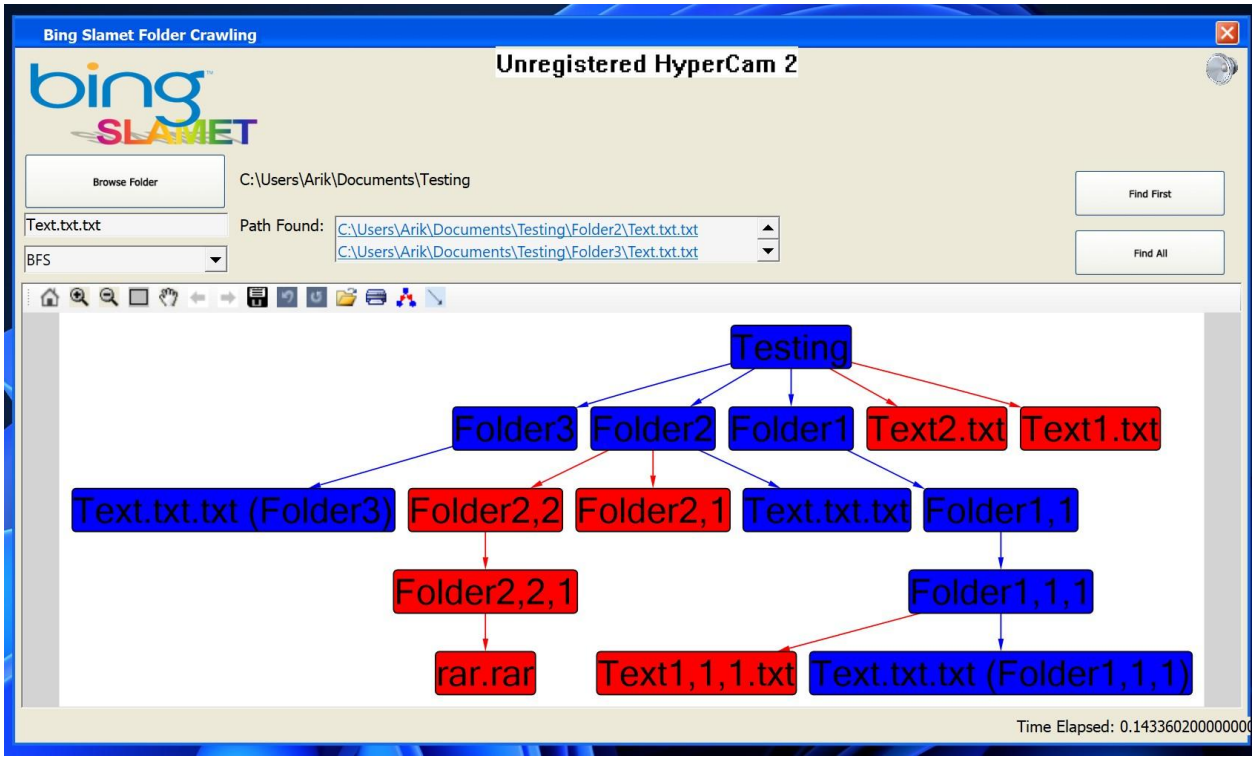
Berikut Alur untuk Penggunaan Aplikasi GUI sederhana yang kami buat :



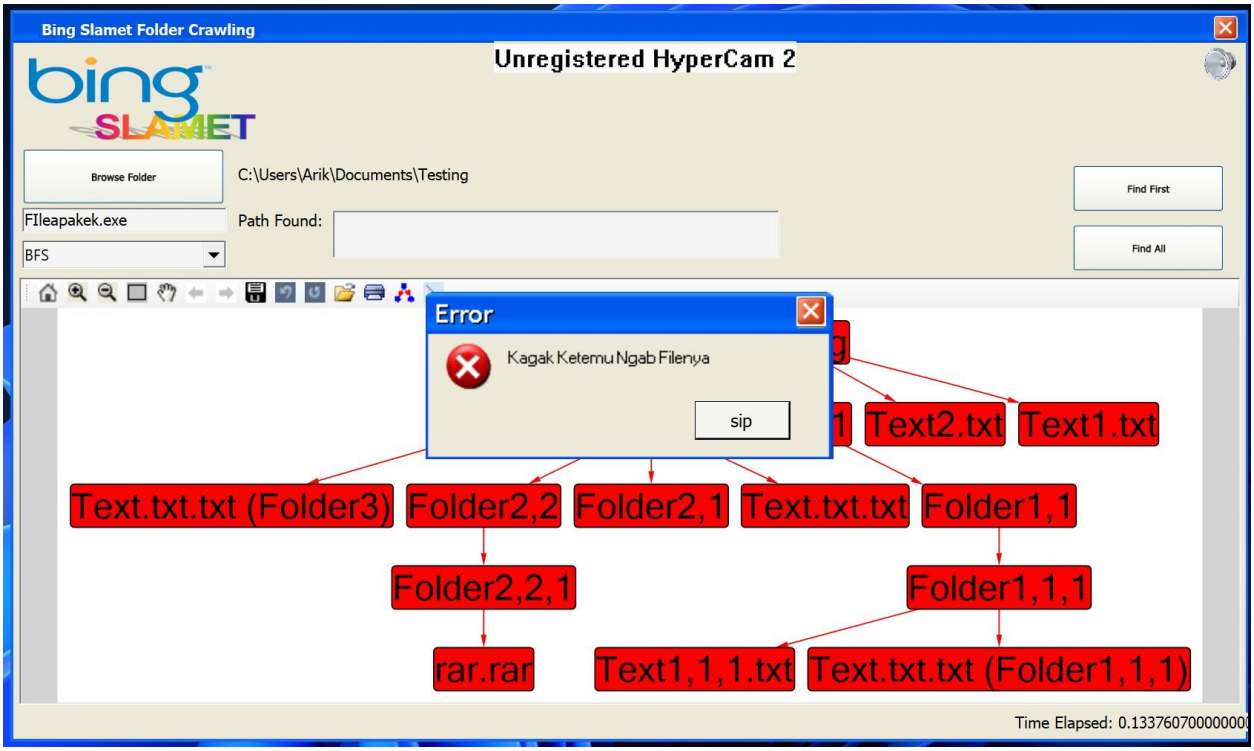
1. Jalankan Program
2. Pilih folder yang akan dijadikan akar pada *Browse Folder*
3. Input nama dari file yang akan dicari
4. Pilih Metode Pencarian yang akan dipilih (BFS / DFS)
5. Pilih Opsi Kemunculan (Find First / Find All)
6. Panel akan menampilkan visualisasi tree dari pencarian file secara bertahap sesuai pemeriksaan **(Bonus)**
7. Linkbox akan menampilkan path menuju file
8. Akan ditampilkan juga waktu dari pencarian file

IV. Hasil pengujian

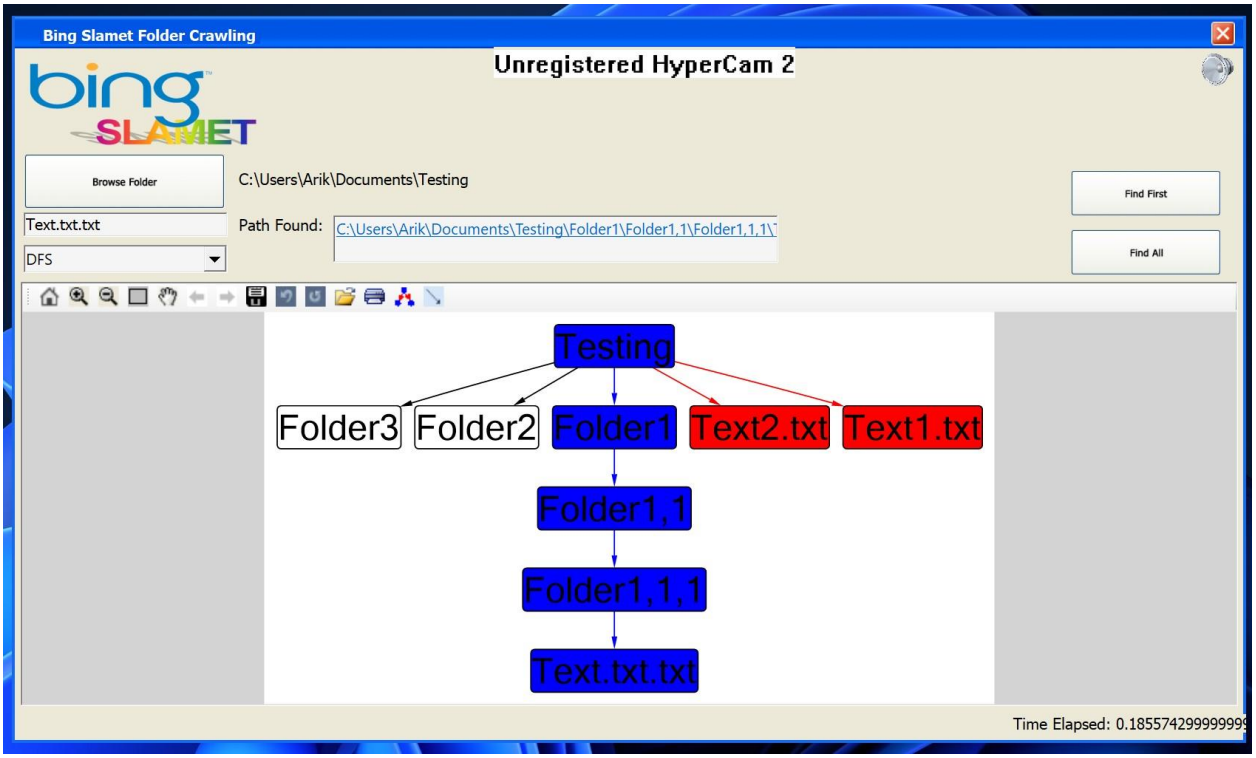
a. Hasil Pengujian BFS Berhasil



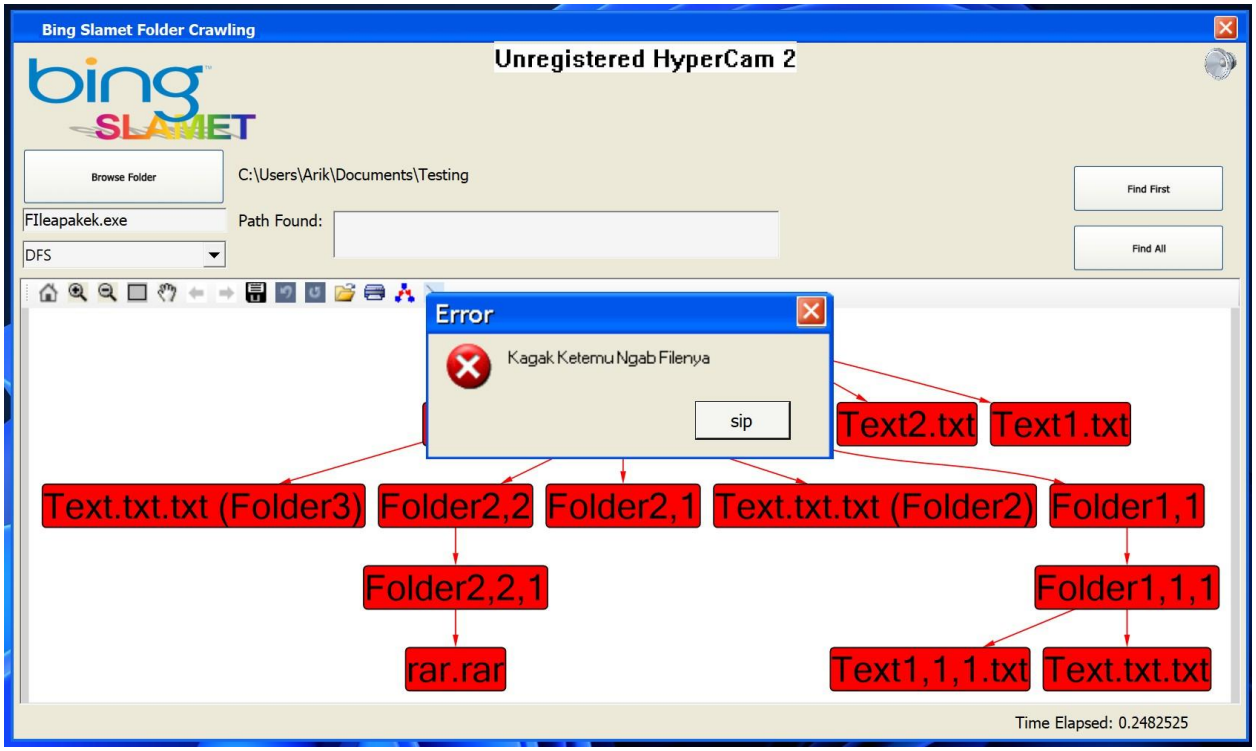
b. Hasil Pengujian BFS Gagal



c. Hasil Pengujian DFS Berhasil



d. Hasil Pengujian DFS Gagal



V. Analisis Desain Solusi

Dari hasil pengujian yang telah kami lakukan pada sub bab sebelum ini, dapat dilihat bahwa pencarian baik menggunakan algoritma BFS maupun DFS berjalan dengan baik dan dapat mengeluarkan hasil pencarian yang sesuai dengan yang dicari. Dapat dilihat juga dari waktu pencarian pada GUI, program berjalan dengan cepat. Namun, jika folder dan file yang dicari sangat banyak, program akan berjalan sedikit lambat, tetapi bukan karena algoritma pencariannya karena kompleksitas waktu untuk algoritma BFS dan DFS sendiri adalah O(n) dengan n adalah jumlah file yang ada di dalam *directory* awal, melainkan karena visualisasi graph-nya yang akan membuat sangat banyak node.

Bab 5

Kesimpulan dan Saran

I. Kesimpulan

Pengimplementasian Folder Crawling dengan menggunakan algoritma Graph Traversal, dapat dilakukan dengan menggunakan dua algoritma, yaitu algoritma BFS(*Breadth-First-Search*) dan DFS(*Depth-First-Search)*.

Dalam pengembangan aplikasi GUI sederhana, juga dapat dimanfaatkan pengembangan menggunakan Visual Studio menggunakan Windows Forms App (bahasa C#, framework .NET). Visualisasi yang digunakan untuk pengimplementasian Folder Crawler adalah menggunakan struktur data Tree.

II. Saran

Saran untuk tugas ke depannya, agar tidak terikat pada suatu OS (*Operating System)* yang spesifik, misal Windows atau MacOS karena bisa menghambat produktivitas untuk mahasiswa dengan OS yang berbeda.

Bab 6

Daftar Pustaka

https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag2.pdf

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

https://www.thecrazyprogrammer.com/2017/06/difference-between-bfs-and-dfs.html

https://www.tutorialspoint.com/graphs-and-its-traversal-algorithms

Link Repository

https://github.com/arikrayi/Tubes2_13520048