LAPORAN TUGAS BESAR II

Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling

Laporan dibuat untuk memenuhi salah satu tugas besar mata kuliah

IF2211 Strategi Algoritma

Disusun oleh:

Kelompok C# C# di dinding

M. Garebaldhie Er Rahman	**13520029**
I Gede Arya Raditya P.	**13520036**
Aira Thalca Avila Putra	**13520101**

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

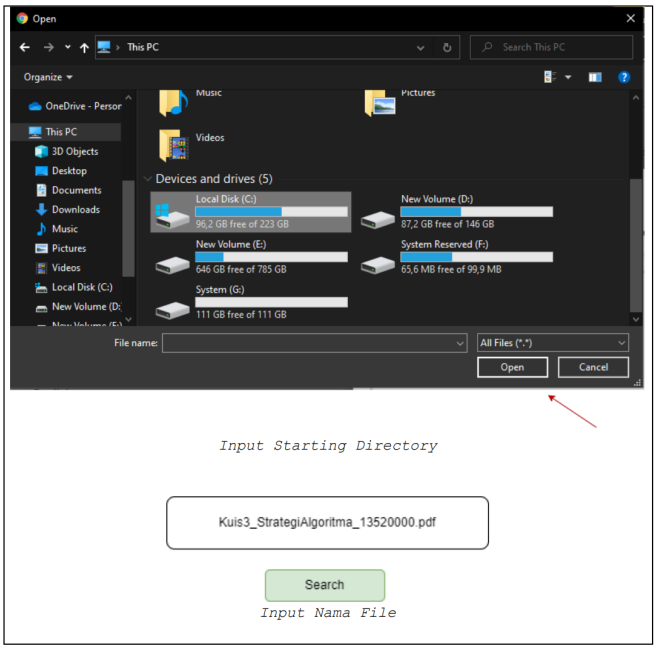
Deskripsi Tugas

1.1 Deskripsi Tugas

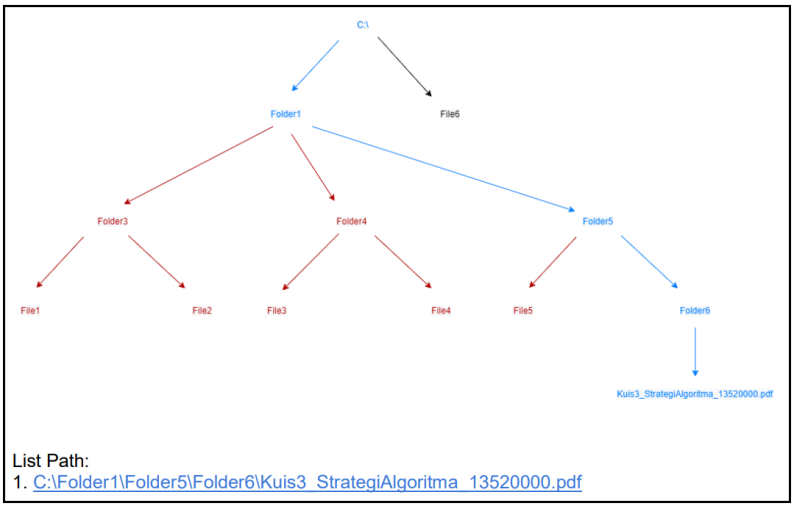
Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari file explorer pada sistem operasi, yang pada tugas ini disebut dengan Folder Crawling. Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), Anda dapat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

Berikut adalah contoh input dan output program,



Gambar 1.1.1 Contoh input program



Gambar 1.2.1 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.

1.2 Spesifikasi Tugas

1.2.1 Spesifikasi GUI

1. Program dapat menerima input folder dan query nama file.
2. Program dapat memilih untuk menampilkan satu hasil saja atau menemukan semua file yang memiliki nama file sama persis dengan input query.
3. Program dapat memilih algoritma yang digunakan.
4. Program dapat menampilkan pohon hasil pencarian file tersebut dengan memberikan keterangan folder/file yang sudah diperiksa, folder/file yang sudah masuk antrian tapi belum diperiksa, dan rute folder serta file yang merupakan rute hasil pertemuan.
5. (Bonus) Program dapat menampilkan progress pembentukan pohon dengan menambahkan node/simpul sesuai dengan pemeriksaan folder/file yang sedang berlangsung.
6. Program dapat menampilkan hasil pencarian berupa rute/path (bisa lebih dari satu jika memilih menemukan semua file) serta durasi waktu algoritma.
7. GUI dapat dibuat sekreatif mungkin asalkan memuat 5 (6 jika mengerjakan bonus) spesifikasi di atas.

1.2.2 Spesifikasi Wajib

1. Buatlah program dalam bahasa C# untuk melakukan penelusuran Folder Crawling sehingga diperoleh hasil pencarian file yang diinginkan. Penelusuran harus memanfaatkan algoritma BFS dan DFS.
2. Awalnya program menerima sebuah input folder pada direktori yang ada dan nama file yang akan dicari oleh program.
3. Terdapat dua pilihan pencarian, yaitu:
    a. Mencari 1 file saja, program akan memberhentikan pencarian ketika sudah menemukan file yang memiliki nama sama persis dengan input nama file.
    b. Mencari semua kemunculan file pada folder root, program akan berhenti ketika sudah memeriksa semua file yang terdapat pada folder root dan program akan menampilkan daftar semua rute file yang memiliki nama sama persis dengan input nama file.

4. Program kemudian dapat menampilkan visualisasi pohon pencarian file berdasarkan informasi direktori dari folder yang di-input. Pohon hasil pencarian file ini memiliki root adalah folder yang di-input dan setiap daunnya adalah file yang ada di folder root tersebut. Setiap folder/file direpresentasikan sebagai sebuah node atau simpul pada pohon. Cabang pada pohon menggambarkan folder/file yang terdapat di folder parent-nya. Visualisasi pohon juga harus disertai dengan keterangan node yang sudah diperiksa, node yang sudah masuk antrian tapi belum diperiksa, dan node yang bagian dari rute hasil penemuan.

   Proses visualisasi ini boleh memanfaatkan pustaka atau kakas yang tersedia. Sebagai referensi, salah satu kakas yang tersedia untuk melakukan visualisasi adalah MSAGL (https://github.com/microsoft/automatic-graph-layout) Berikut ini adalah panduan singkat terkait penggunaan MSAGL oleh tim asisten yang dapat diakses pada: https://docs.google.com/document/d/1XhFSpHU028Gaf7YxkmdbluLkQgVl3MY6gt1tL30LA/edit?usp=sharing.

5. Program juga dapat menyediakan hyperlink pada setiap hasil rute yang ditemukan. Hyperlink ini akan membuka folder parent dari file yang ditemukan. Folder hasil hyperlink dapat dibuka dengan browser atau file explorer.

6. Mahasiswa tidak diperkenankan untuk melihat atau menyalin library lain yang mungkin tersedia bebas terkait dengan pemanfaatan BFS dan DFS. Tapi untuk algoritma lainnya seperti string matching dan akses directory, diperbolehkan menggunakan library jika ada.

BAB II

Landasan Teori

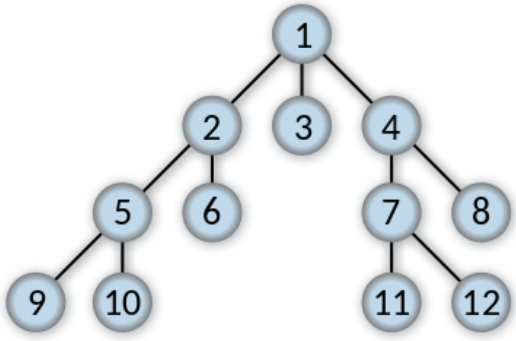
2.1 Graph Traversal

Graph Traversal adalah teknik untuk mengunjungi setiap node dari graf. Teknik ini juga digunakan untuk menghitung urutan simpul dalam proses traverse. Graph Traversal mengunjungi semua node mulai dari satu node yang terhubung satu sama lain tanpa masuk ke loop.

Pada dasarnya dalam graf mungkin terjadi beberapa waktu pengunjung dapat mengunjungi satu node lebih dari sekali sehingga dapat menyebabkan terjadinya *infinite loop*. Jadi, untuk menghindar dari kondisi *infinite loop* ini, graf traversal mencatat setiap simpul. Seperti jika pengunjung mengunjungi vertex maka nilainya akan menjadi nol, jika tidak maka satu.

2.2 Breadth First Search (BFS)

Algoritma Breadth First Search (BFS) adalah algoritma pencarian melebar yang dilakukan dengan mengunjungi node pada level n terlebih dahulu sebelum mengunjungi node-node pada level n+1. Algoritma BFS berbeda dengan DFS dan biasanya lebih lambat dibandingkan DFS.



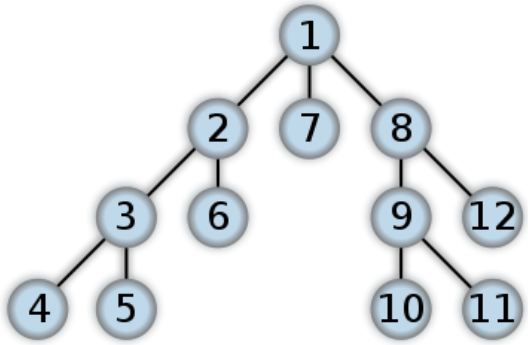
Gambar 2.2.1 Illustrasi graph menggunakan BFS

Cara kerja algoritma Breadth First Search yaitu masukkan simpul ujung ke dalam sebuah antrian kemudian ambil simpul dari awal antrian. Lakukan pengecekan apakah simpul awal merupakan solusi. Jika simpul merupakan solusi pencarian selesai dan hasil dikembalikan. Jika simpul bukan merupakan solusi, masukan seluruh simpul yang bertetangga dengan simpul

tersebut. Apabila semua simpul sudah dicek dan antrian kosong, pencarian selesai dengan mengembalikan hasil solusi tidak ditemukan. Pencarian diulang dari simpul awal antrian.

2.3 Depth First Search (DFS)

Algoritma Depth First Search (DFS) adalah algoritma pencarian mendalam yang dimulai dari node awal dilanjutkan dengan hanya mengunjungi node anak paling kiri pada tingkat selanjutnya.



Gambar 2.3.1 Illustrasi graph menggunakan DFS

Cara kerja algoritma Depth First Search yaitu masukan masukan node akar kedalam sebuah tumpukan. Kemudian ambil simpul pertama pada level paling atas, jika simpul merupakan solusi pencarian selesai dan hasil dikembalikan. Jika simpul bukan merupakan solusi, masukan seluruh simpul yang bertetangga dengan simpul tersebut ke dalam tumpukan. Apabila semua simpul sudah dicek dan antrian kosong, pencarian selesai dengan mengembalikan hasil solusi tidak ditemukan. Pencarian diulang dari simpul awal antrian.

2.4 Pengembangan Aplikasi Desktop

2.4.1 Bahasa Pemrograman C#

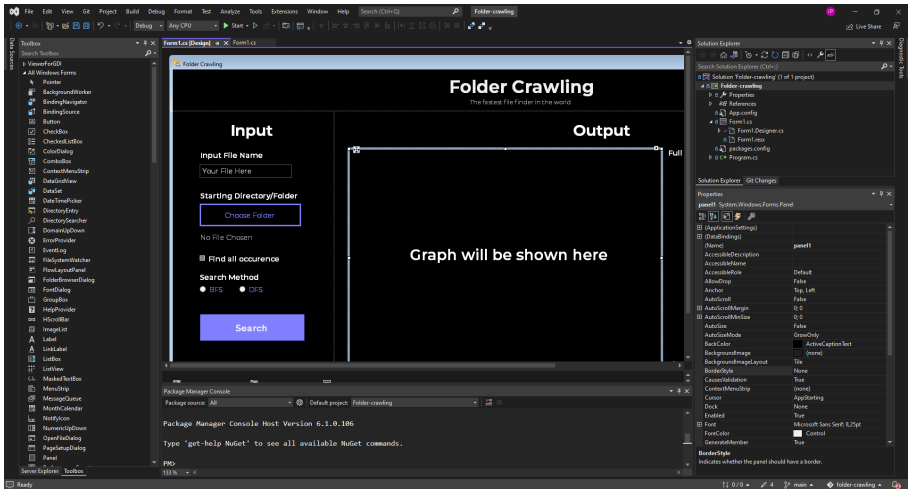
C# atau dibaca C Sharp merupakan sebuah bahasa pemrograman berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti Java, Delphi, Visual Basic, dan lain-lain dengan beberapa penyederhanaan. Menurut standar ECMA-334 C# Language Specification, nama C# terdiri atas sebuah huruf Latin C (U+0043) yang diikuti oleh tanda pagar yang menandakan angka # (U+0023).

Beberapa tujuan dari pembentukan bahasa C# ini diantaranya adalah,

- Dibentuk sebagai bahasa pemrograman yang bersifat bahasa pemrograman general-purpose, berorientasi objek, modern, dan sederhana.
- Mengembangkan komponen perangkat lunak yang mampu mengambil keuntungan dari lingkungan terdistribusi.
- Supaya cocok digunakan untuk menulis program aplikasi baik dalam sistem klien-server (*hosted system*) maupun sistem terbenam (*embedded system*), mulai dari perangkat lunak yang sangat besar yang menggunakan sistem operasi yang canggih hingga kepada perangkat lunak yang sangat kecil yang memiliki fungsi-fungsi terdedikasi.

2.4.2 .NET Framework

.NET Framework atau dibaca dot net framework adalah perangkat lunak kerangka kerja yang dibentuk oleh Microsoft untuk memfasilitasi segala kebutuhan aplikasi yang sedang berjalan di Windows. Software ini menyediakan dua komponen utama: Common Language Runtime (CLR), untuk menangani aplikasi yang sedang berjalan, dan Framework Class Library (FCL), pustaka kode terkelola untuk melakukan pemrograman pada aplikasi.



Gambar 2.4.2.1 Tampilan antarmuka visual studio code

Dari kedua komponen tersebut (CLR & FCL), .NET Framework menyediakan berbagai layanan untuk aplikasi, seperti pengelolaan memori, penggunaan Common Type System (CTS) untuk membaca dan menggunakan tipe data, memiliki sifat ready-to-use karena librarynya luas, memiliki area pengembangan lebih spesifik seperti ASP.NET (untuk aplikasi web) dan ADO.NET (untuk data akses), serta menggunakan kode perantara Common Intermediate Language (CIL) guna proses pemrograman dalam satu bahasa bisa diakses oleh bahasa lain supaya programmer bisa lebih fokus membuat aplikasi menggunakan bahasa yang mereka pilih.

BAB III

Analisis Pemecahan Masalah

3.1 Langkah - Langkah Pemecahan Masalah

Dalam menyelesaikan permasalahan ini, telah didekomposisi beberapa permasalahan sebagai langkah-langkah berikut ini:

1. Memahami dan mempelajari struktur folder pada windows
2. Memahami konsep dari algoritma traversal DFS dan BFS
3. Mempelajari syntax bahasa pemrograman C# dan .NET Framework khususnya dalam mengakses folder dan file, membuat graph dalam C#, dan menyambungkan MSAGL dengan graph yang telah dibuat lo
4. Membentuk UI design menggunakan figma
5. Membuat Graphical User Interface (GUI) menggunakan .NET Framework pada Ms Visual Studio
6. Memetakan setiap input menjadi elemen-elemen pada DFS dan BFS
7. Mengimplementasikan algoritma BFS dan DFS pada pencarian file di setiap elemen
8. Membuat visualisasi graf dengan MSAGL
9. Menghubungkan seluruh program dengan GUI yang sudah dibuat

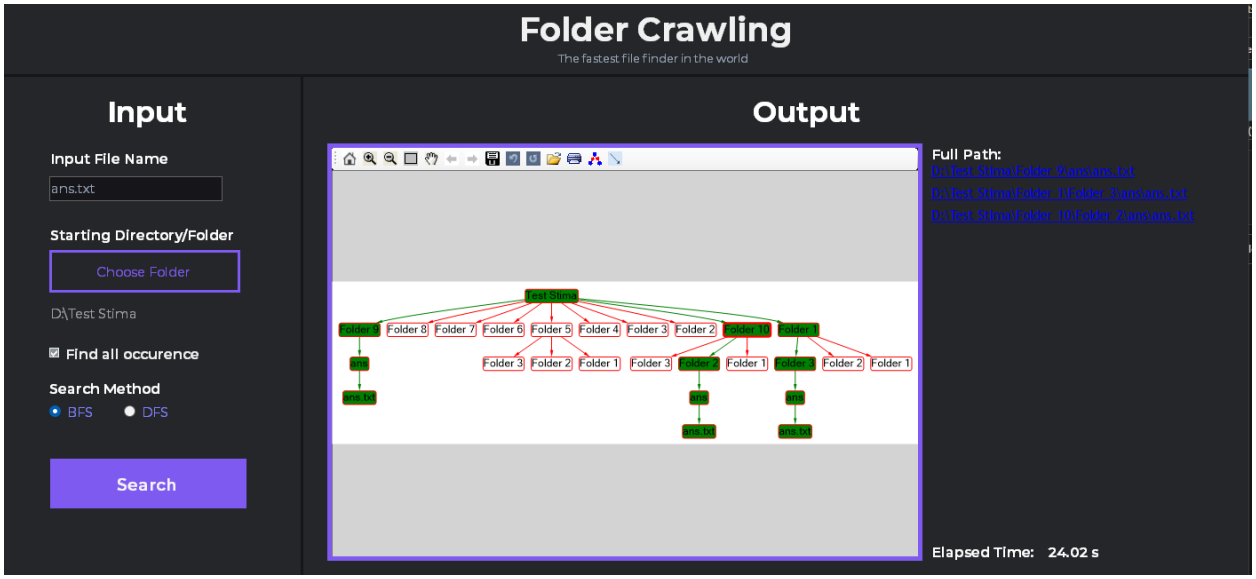
3.2 Mapping Persoalan Menjadi Elemen Algoritma BFS dan DFS

Pada kasus pencarian ini graf yang digunakan bertipe dinamis karena pada awal mula pencarian graf kosong dan graf akan dibangun secara bertahap selama pencarian berlangsung.
Berikut adalah beberapa elemennya:

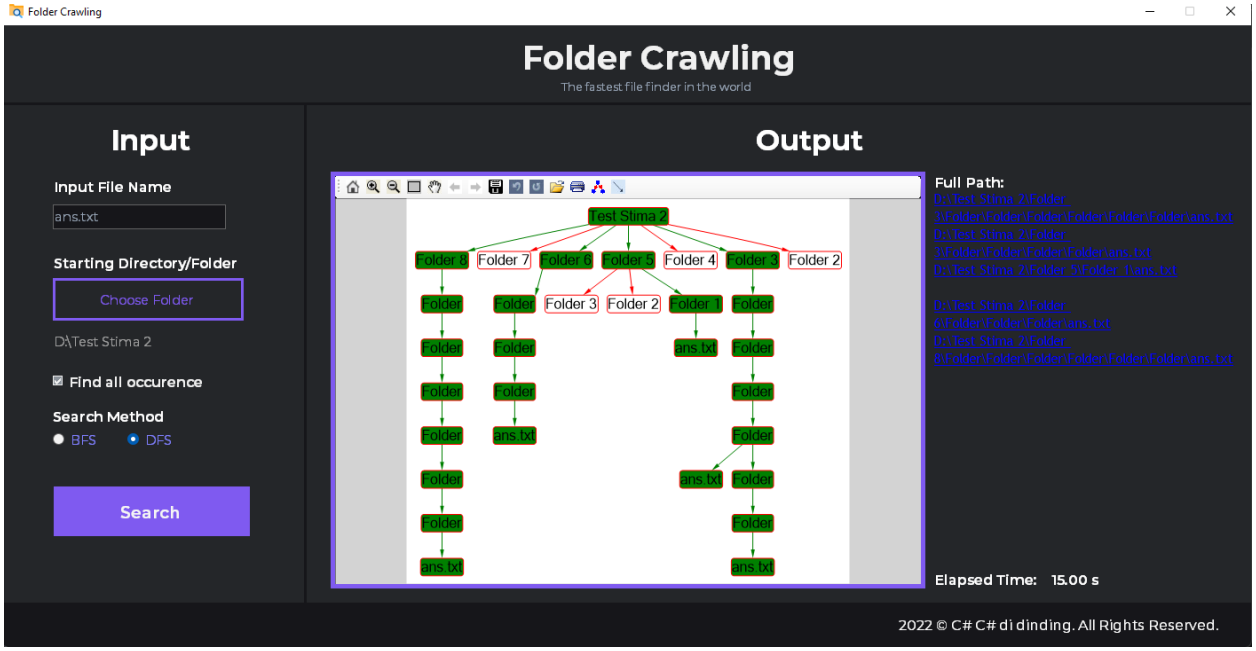
Tabel 3.2.1 Elemen pada BFS/DFS

Nodes	Folder dan file yang berada pada lokasi pencarian
Edges	Subfolder dan subfiles yang ada pada suatu directory
Goal State	Target file yang ingin ditemukan

3.3 Ilustrasi Kasus Lain



Gambar 3.3.1 Illustrasi kasus lain ketika folder melebar sebanyak 10 buah



Gambar 3.3.2 Illustrasi kasus ketika terdapat jawaban pada kedalaman yang banyak

BAB IV

Implementasi dan Pengujian

4.1 Implementasi Program (Pseudocode)

4.1.1 Implementasi BFS

```
public void BFS_method(string root, string target, Graph listDirectory, List<string> listAns, bool
allOcurence, ref List<string> fullPath)
        {
            panel1.Controls.Clear();
            //if not found throw exc
            if (!Directory.Exists(root))
            {
                throw new Exception("No directory exists! Please insert correct folder");
            }
            Queue<string> queueDir = new Queue<string>();
            queueDir.Enqueue(root);
            while (queueDir.Count > 0)
            {
                string dir = queueDir.Dequeue();
                if (dir != root)
                {
                    Node now = listDirectory.FindNode(dir);
                    //color the curNode, edge, and sourceNode
                    foreach (Edge inEdge in now.InEdges)
                    {
                        wait(500);
                        inEdge.Attr.Color = Color.Red;
                        now.Attr.Color = Color.Red;
                        inEdge.SourceNode.Attr.Color = Color.Red;
                        show(viewer, listDirectory);
                    }
                }
                //Get the info on subDir, files, and the dir itself
                DirectoryInfo dirInfo = new DirectoryInfo(dir);
                string[] subDir, files;
                subDir = Directory.GetDirectories(dir);
                files = Directory.GetFiles(dir);

                //iterate all file in current Directory and process (addNode, addEdge, color the node and edge)
                foreach (string file in files)
                {
                    wait(500);
                    FileInfo fileInfo = new FileInfo(file);
                    ChangeLabel(listDirectory, dir, file);
                    if (fileInfo.Exists && fileInfo.Name == target)
```

```
            {
                string s = fileInfo.FullName;
                fullPath.Add(s);
                listAns.Add(file);
                if (!allOcurence)
                {
                    return;
                }
            }
            show(viewer, listDirectory);
        }

        //iterate all subdirectories in current Directory and process (addNode, addEdge)
        foreach (string curDir in subDir)
        {
            wait(500);
            DirectoryInfo directoryInfo = new DirectoryInfo(curDir);
            listDirectory.AddEdge(dir, curDir);
            listDirectory.FindNode(dir).LabelText = dirInfo.Name;
            listDirectory.FindNode(curDir).LabelText = directoryInfo.Name;
            queueDir.Enqueue(curDir);
            show(viewer, listDirectory);
        }
    }
}
```

4.1.2 Implementasi DFS

```
public void DFS_method(string root, string target, Graph listDirectory, ref bool search, List<string>
listAns, bool allOcurence, ref List<string> fullPath)
    {
        //if not found throw exc
        if (!Directory.Exists(root))
        {
            throw new Exception("Folder didn't exists! Please choose correct folder");
        }

        //if the target has been found and searching for one file only, break.
        if (search && !allOcurence)
        {
            return;
        }

        string[] subDir;
        subDir = Directory.GetDirectories(root);

        //traverse deeper to the root folder
        foreach (string dir in subDir)
```

```
{
    DirectoryInfo directoryInfo = new DirectoryInfo(dir);
    DirectoryInfo rootInfo = new DirectoryInfo(root);
    listDirectory.AddEdge(root, dir);
    listDirectory.FindNode(root).LabelText = rootInfo.Name;
    listDirectory.FindNode(dir).LabelText = directoryInfo.Name;
}
show(viewer, listDirectory);
foreach (string dir in subDir)
{
    if (search && !allOcurence)
    {
        return;
    }
    Node now = listDirectory.FindNode(dir);
    //color the curNode, edge, and sourceNode
    foreach (Edge inEdge in now.InEdges)
    {
        wait();
        inEdge.Attr.Color = Color.Red;
        now.Attr.Color = Color.Red;
        inEdge.SourceNode.Attr.Color = Color.Red;
    }
    show(viewer, listDirectory);
    DFS_method(dir, target, listDirectory, ref search, listAns, allOcurence, ref fullPath);
}

string[] files;
files = Directory.GetFiles(root);
foreach (string file in files)
{
    wait();
    if (search && !allOcurence)
    {
        return;
    }
    //get the detail of the file
    FileInfo fileInfo = new FileInfo(file);
    ChangeLabel(listDirectory, root, file);
    show(viewer, listDirectory);
    if (fileInfo.Exists && fileInfo.Name == target)
    {
        string s = fileInfo.FullName;
        fullPath.Add(s);
        listAns.Add(file);
        if (!allOcurence)
        {
            search = true;
            break;
        }
```

```
            }
        }
    }
```

Implementasi ChangeLabel & Coloring

```
public static void ChangeLabel(Graph listDirectory, string src, string target)
    {
        FileInfo _src = new FileInfo(src);
        FileInfo _target = new FileInfo(target);
        listDirectory.AddEdge(src, target).Attr.Color = Color.Red;
        listDirectory.FindNode(src).LabelText = _src.Name;
        listDirectory.FindNode(src).Attr.Color = Color.Red;
        listDirectory.FindNode(target).LabelText = _target.Name;
        listDirectory.FindNode(target).Attr.Color = Color.Red;
    }

    public void graphColoring(Graph listDirectory, string src, List<string> listAns)
    {
        //if no ans found don't bother
        if (listAns.Count == 0)
        {
            return;
        }
        //traverse from ans to root
        Node dir;
        foreach (string ans in listAns)
        {
            string traverse = ans;
            while (src != traverse)
            {
                dir = listDirectory.FindNode(traverse);
                dir.Attr.FillColor = Color.Green;
                foreach (Edge inEdge in dir.InEdges)
                {
                    inEdge.Attr.Color = Color.Green;
                    dir = inEdge.SourceNode;
                }
                traverse = dir.Id;
            }
            dir = listDirectory.FindNode(src);
            dir.Attr.FillColor = Color.Green;
            show(viewer, listDirectory);
        }
    }
```

4.2 Penjelasan Struktur Data

Terdapat beberapa struktur data yang digunakan pada program *folder crawling ini*

1. **Graph**
   Graph digunakan untuk membentuk ruang pencarian ketika program menelusuri folder. Prosess pewarnaan solusi juga menggunakan struktur data ini untuk mencari node dan edges yang menuju solusi/target yang dicari.
2. **List**
   Struktur data ini lebih dikenal dengan Array. Kita menggunakan List bertipe string untuk menyimpan hasil solusi pencarian serta fullPath dari solusi tersebut. Digunakan pula array yang menyimpan list directory dari suatu folder serta list file dari suatu directory.
3. **Queue**
   Struktur data ini digunakan untuk menyimpan antrian penelusuran dengan metode BFS.

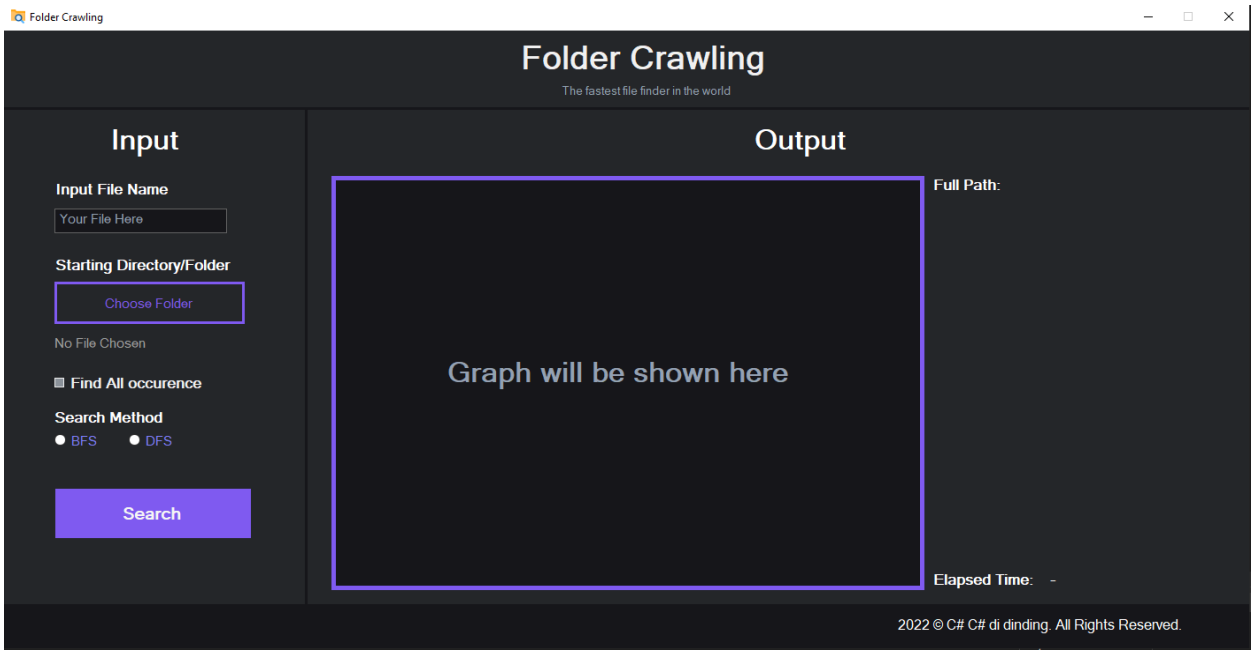
4.3 Penjelasan Tata Cara Penggunaan Program

Untuk menggunakan program ini diperlukan beberapa requirement sebagai berikut,

- Memiliki .NET setidaknya v4.8
- Komputer berjalan dengan sistem operasi Windows

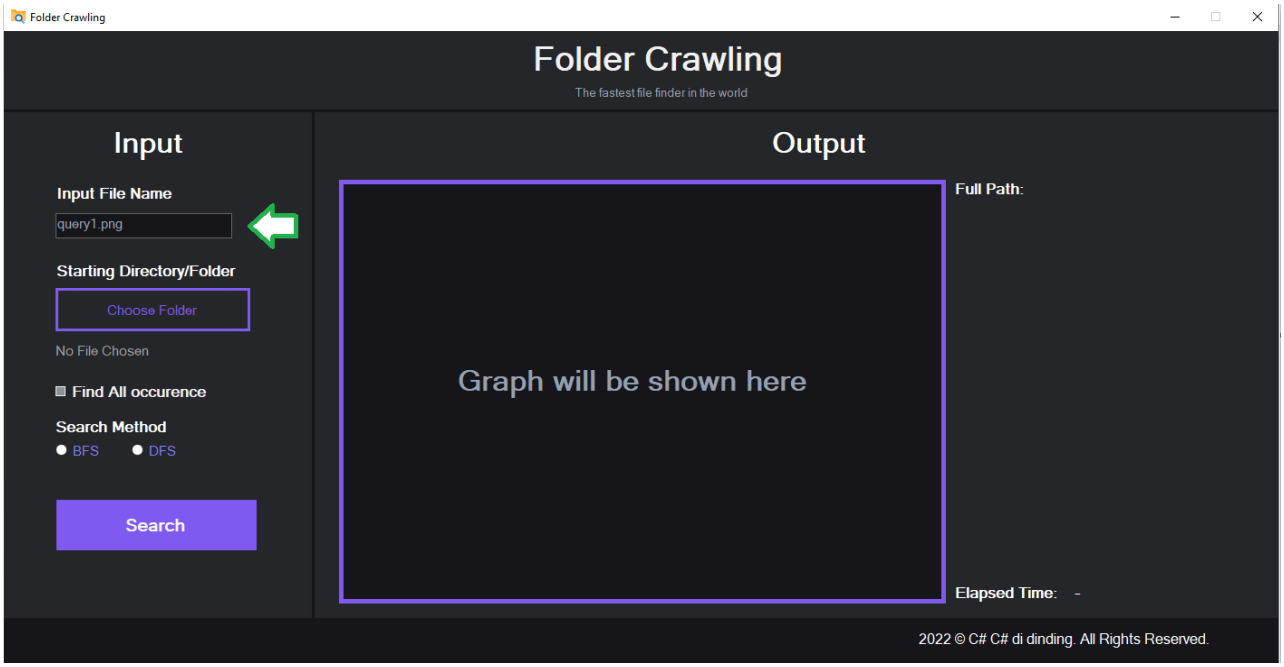
Setelah requirement dipenuhi maka jalankan step - step berikut:

1. Setelah mendownload program, letakkan di tempat yang diinginkan
2. Jalankan program lalu akan muncul tampilan seperti ini



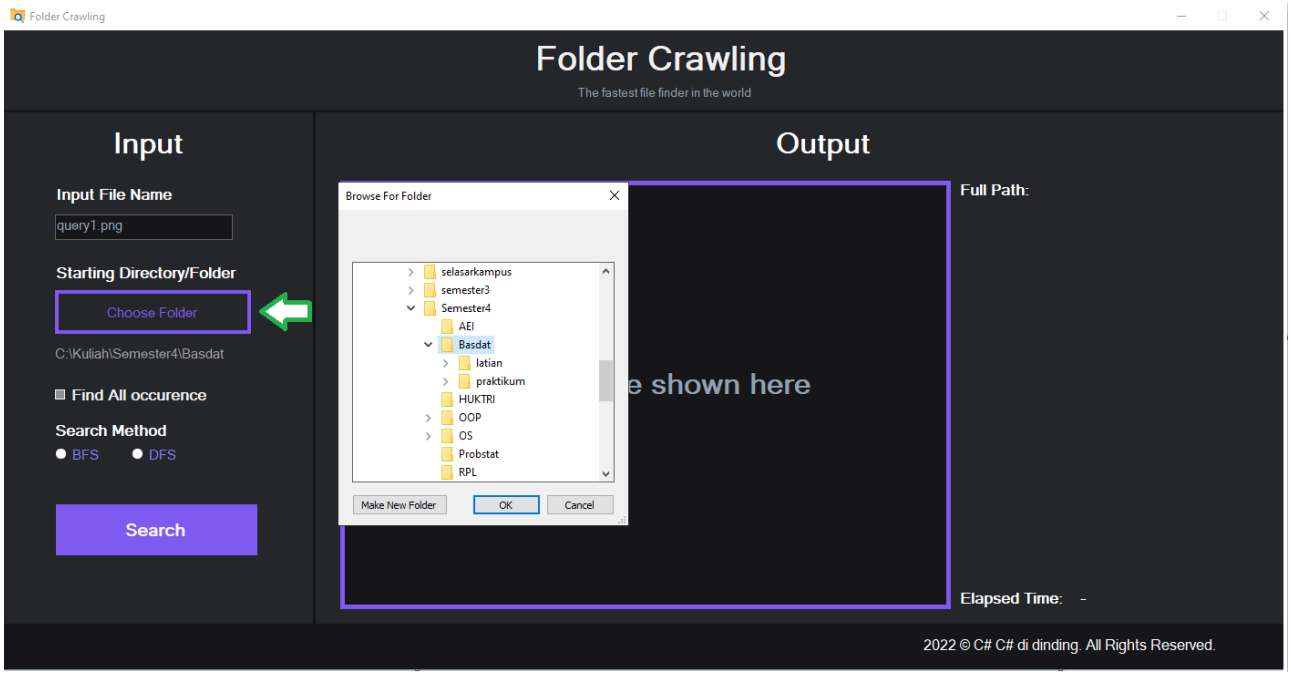
Gambar 4.3.1 Tampilan saat menjalankan program

3. Masukan nama file pada kolom ini



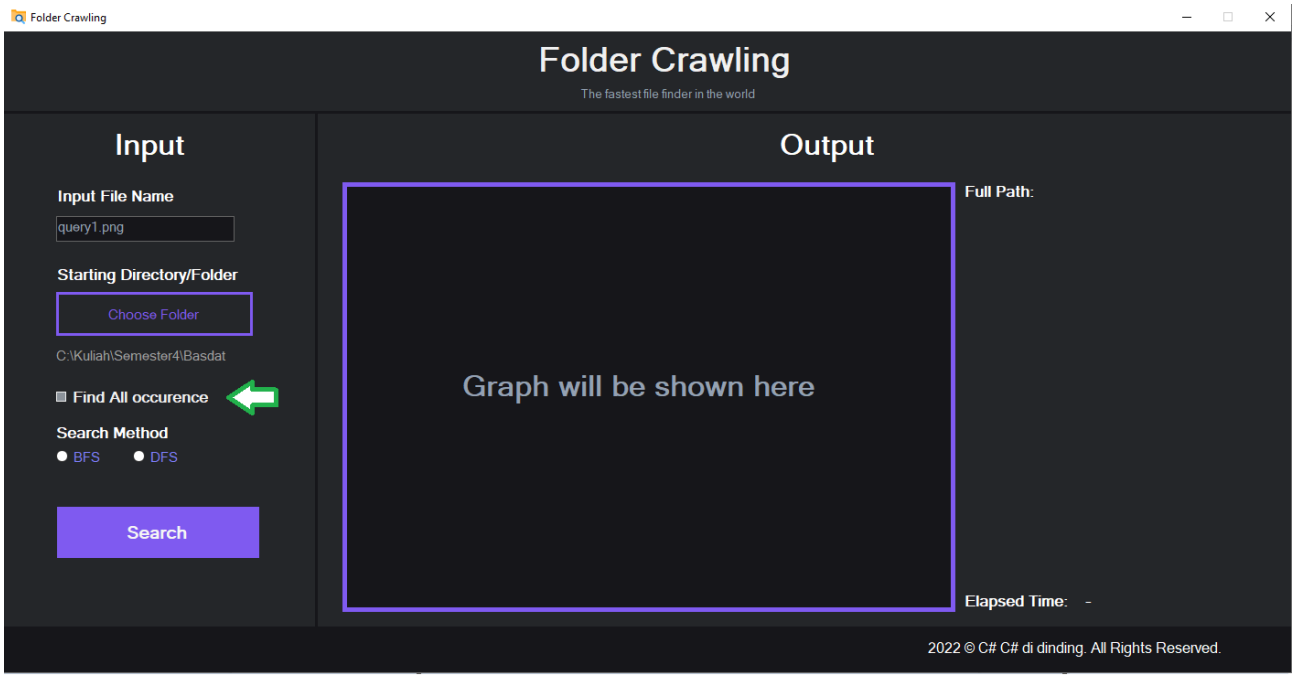
Gambar 4.3.2 Tampilan saat memasukan nama file

4. Pilih lokasi folder pencarian



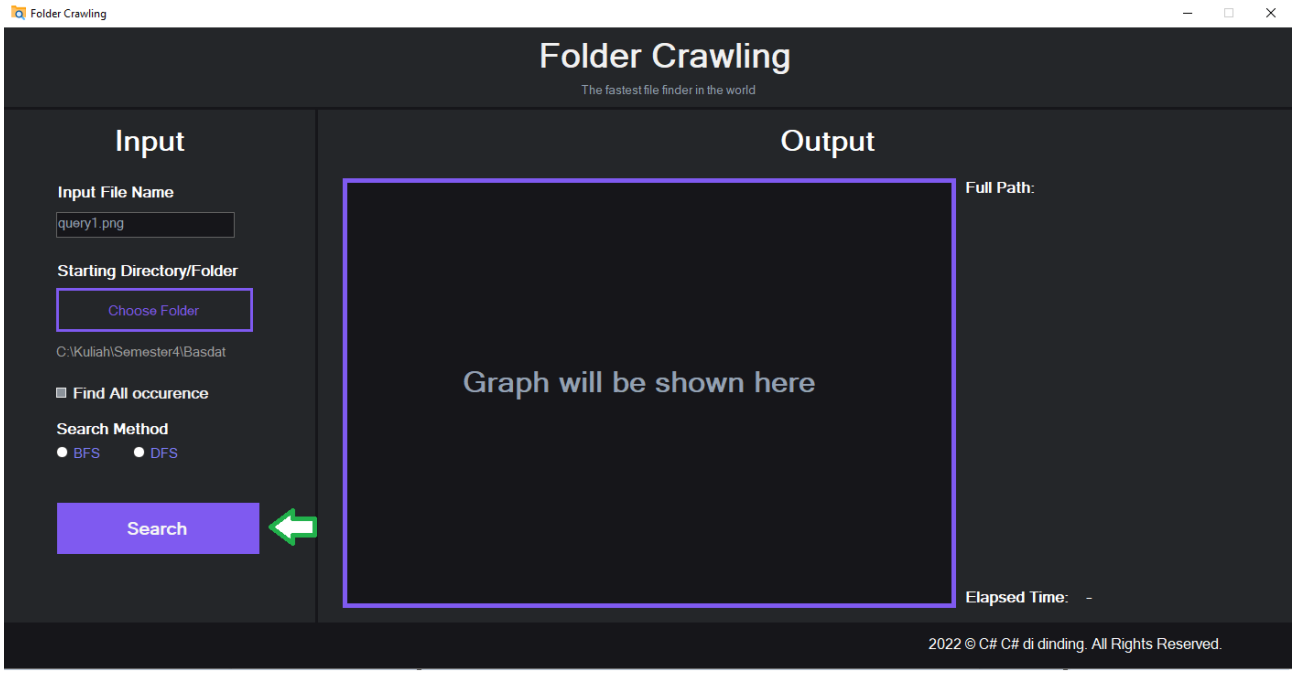
Gambar 4.3.3 Tampilan saat memilih folder

5. Apabila ingin menemukan lebih dari 1 file check *findAllOccurence* agar dapat menemukan lebih dari 1 file



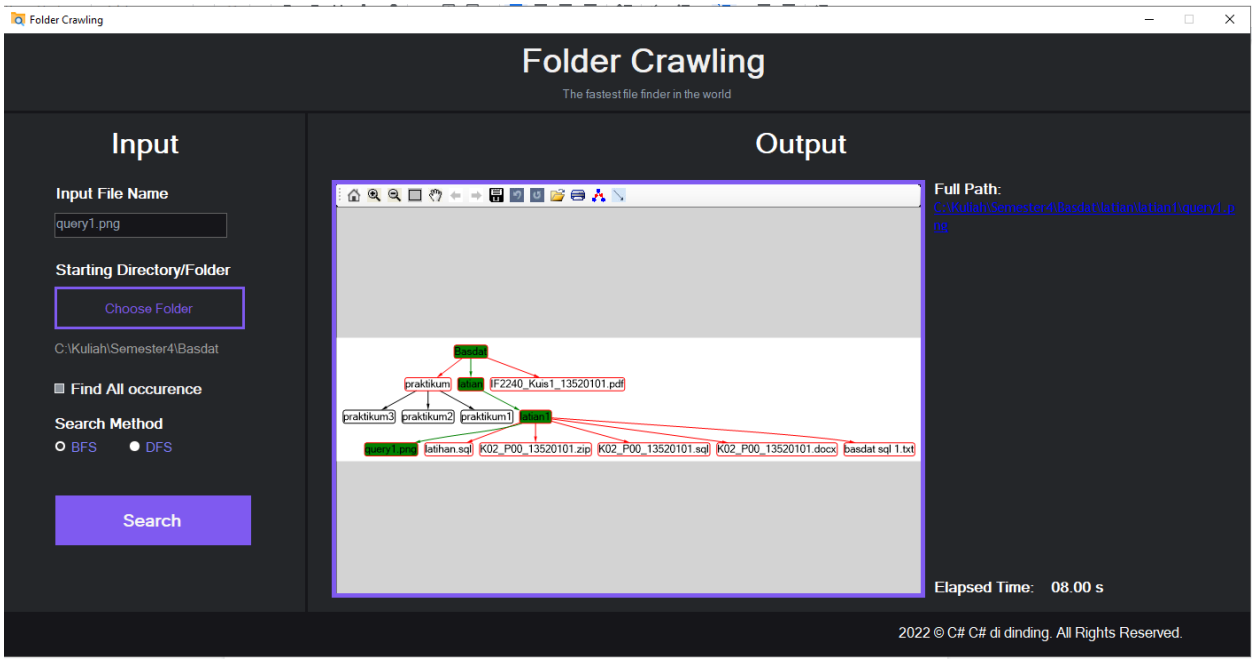
Gambar 4.3.4 Tampilan saat mencentang *findAllOccurence*

6. Tekan tombol search pada bagian bawah



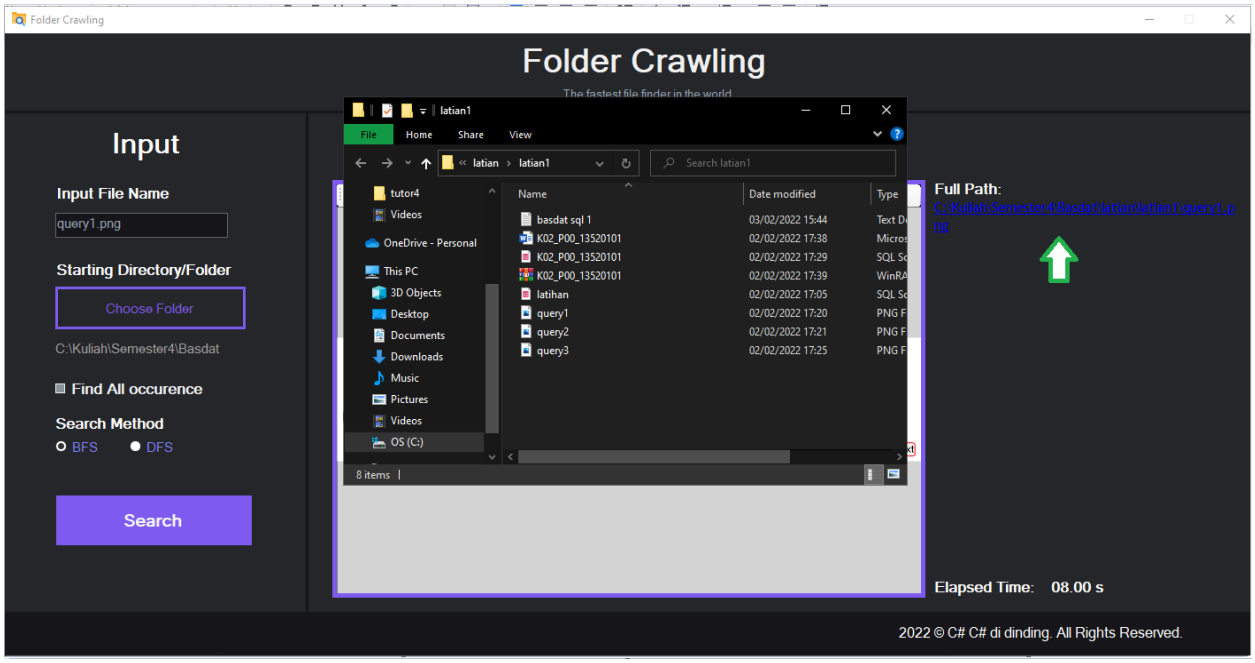
Gambar 4.3.5 Tampilan highlight tombol search

7. Ta-da Aplikasi ini sudah berjalan dan simulasi pencarian akan ditampilkan pada bagian kanan serta path dari solusi pencarian.



Gambar 4.3.6 Tampilan saat pencarian selesai dilakukan

8. Apabila ingin membuka folder yang berisi solusi file tersebut, tekan link yang berada pada bagian kanan lalu akan muncul folder yang diinginkan.

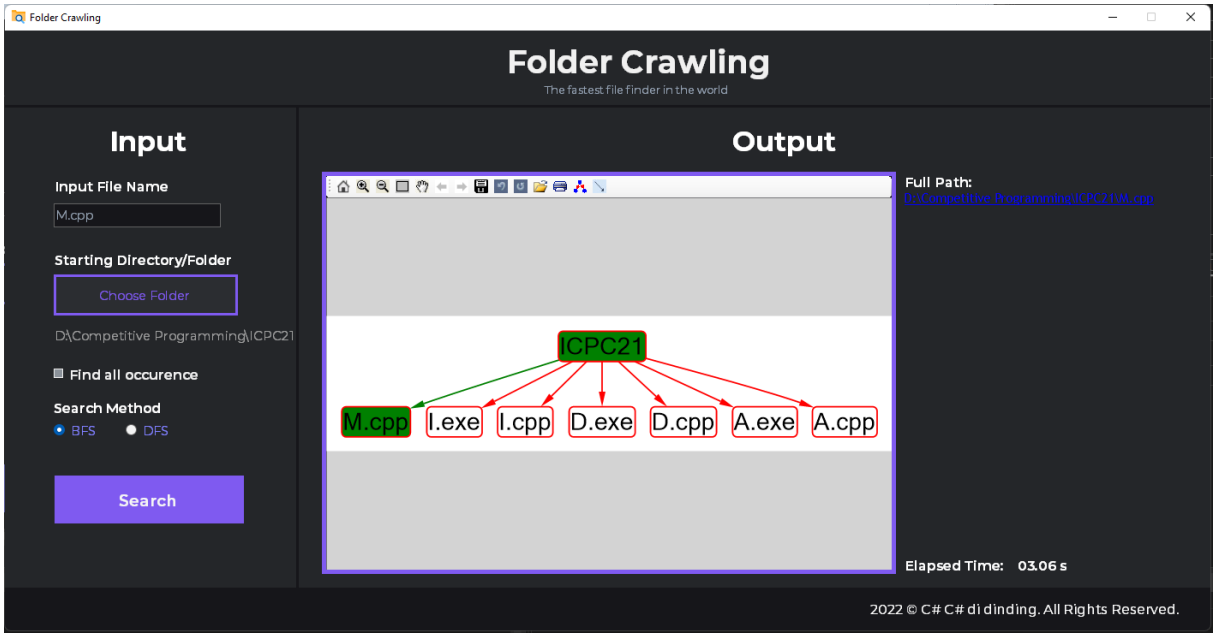


Gambar 4.3.7 Tampilan saat menekan hyperlink folder

4.4 Hasil Pengujian

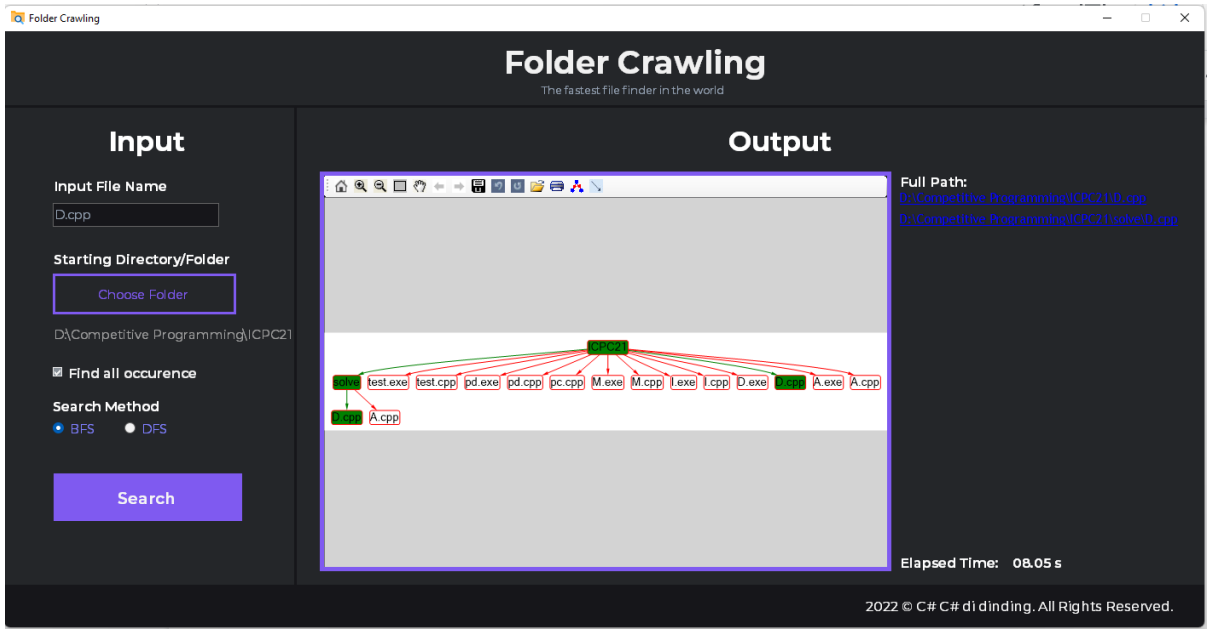
Berikut adalah beberapa hasil pengujian kami pada aplikasi folder crawling ini. Pengujian dibagi menjadi 4 yaitu BFS, DFS, serta BFS dan DFS juga namun dengan Find all occurence.

4.4.1 BFS



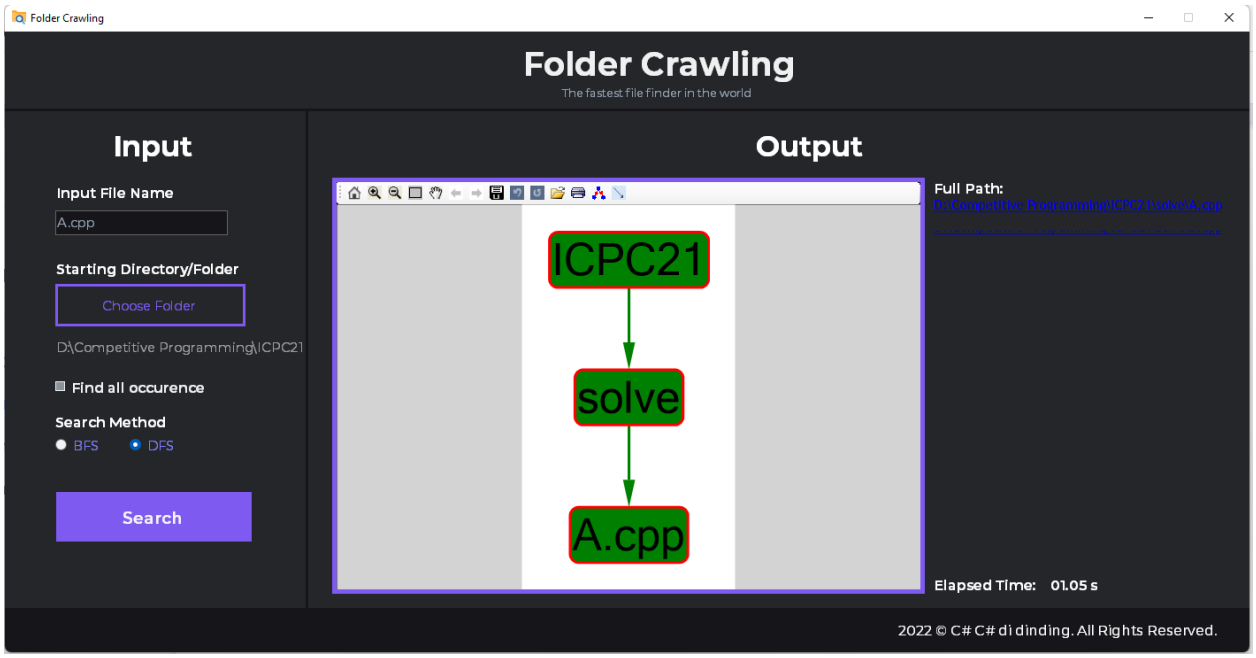
Gambar 4.4.1.1 Tampilan pencarian BFS tanpa all occurence

4.4.2 BFS & Find all occurence



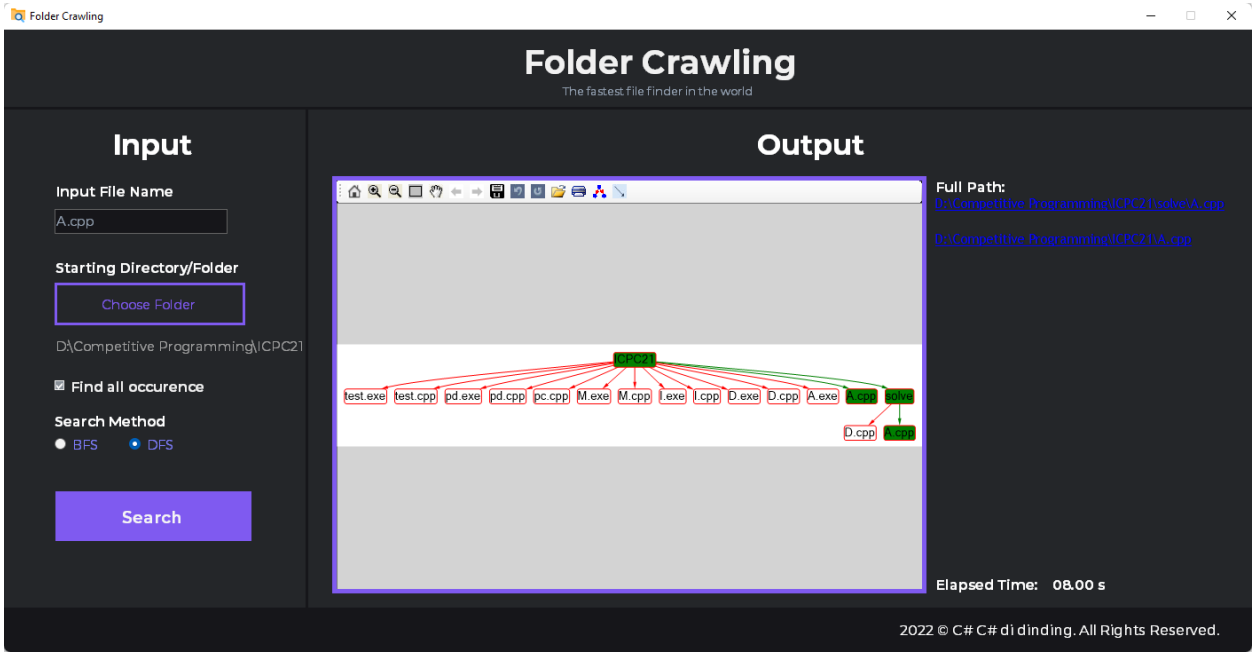
Gambar 4.4.2.1 Tampilan pencarian BFS dengan all occurence

4.4.3 DFS



Gambar 4.4.3.1 Tampilan pencarian DFS tanpa all occurence

4.4.4 DFS & Find all occurence



Gambar 4.4.4.1 Tampilan pencarian DFS dengan all occurence

4.5 Analisis Desain Solusi Algoritma BFS dan DFS

Dari hasil pengujian yang sudah dilakukan, algoritma DFS dan BFS sudah menjalankan tugasnya dengan baik dan memberikan hasil yang sesuai entah ketika diminta mencari semua file atau hanya salah satu saja. Algoritma DFS dan BFS yang kami lakukan juga berjalan dengan cukup cepat dan memiliki kompleksitas O(n) dengan n merupakan jumlah file keseluruhan yang terdapat pada folder yang hendak dicari. Namun agar memenuhi bonus, kami sengaja untuk memperlihat langkah-langkah pencarian dengan jeda 0.5 detik setiap langkahnya sehingga dapat terbaca dengan jelas langkah demi langkahnya. Sebenarnya jika tidak diberikan jeda, program dapat berjalan dibawah 0.1 detik untuk jumlah file lebih kecil dari 10e7 file.

BAB V

Kesimpulan dan Saran

5.1 Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma semester 2 2021/2022 berjudul "Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling", kami berhasil membuat Aplikasi Desktop yang dapat melakukan pencarian file dari sebuah folder atau directory dengan metode DFS dan BFS. Pencarian file ini digunakan untuk mencari file dari folder spesifik yang sudah ditentukan, serta dapat memilih opsi untuk mencari hanya satu file saja atau semua file yang sesuai.

Dari pengerjaan tugas besar ini, kami mendapatkan beberapa kesimpulan, yaitu:

- Algoritma BFS memakan lebih banyak memori karena menyimpan semua semua node pada pohon sedangkan DFS tidak melakukan penyimpanan memori.
- Rata-rata waktu yang diperlukan algoritma DFS lebih cepat dibandingkan rata-rata waktu yang diperlukan algoritma BFS dalam mencari file, hal ini disebabkan karena algoritma BFS mengiterasi tiap level n terlebih dahulu sebelum menemukan solusi pada level n-1.

5.2 Saran

Saran kami adalah menggunakan *tools* dan *stack* yang mudah dijalankan di seluruh sistem operasi karena pengguna sistem operasi non-Windows cukup kesulitan dalam tugas besar kali ini. Selain itu, dokumentasi MSAGL yang disediakan kurang jelas sehingga kami kesulitan memakai MSAGL.

LINK PENTING

Link Figma : TUBES 2 STIMA – Figma

Link Repository Github : IloveNooodles/folder-crawling: Tugas Besar 2 Strategi Algoritma: Folder Crawling (github.com)

Link Youtube: https://youtu.be/DlnykBm4duk

DAFTAR PUSTAKA

- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf (Diakses 21 Maret 2022, 21:34)
- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf (Diakses 21 Maret 2022, 23:41)