TUGAS BESAR II

PENGAPLIKASIAN ALGORITMA BFS DAN DFS DALAM IMPLEMENTASI *FOLDER CRAWLING*"

LAPORAN

Diajukan sebagai salah satu tugas mata kuliah

IF2211 Strategi Algoritma pada Semester II

Tahun Akademik 2021-2022

Oleh

Amar Fadil	**13520103**
Owen Christian Wijaya	**13520124**
Fachry Dennis Heraldi	**13520139**

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

DAFTAR ISI

BAB I

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*.

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.

Program yang dibuat harus memenuhi spesifikasi wajib sebagai berikut:

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/1XhFSpHU028Gaf7YxkmdbluLkQgVl3MY6gt1tPL30LA/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 Traversal Graf

Traversal graf adalah proses mengunjungi setiap simpul dalam suatu graf secara sistematik dengan tujuan mencari jalur yang dapat dilalui dari simpul asal sampai simpul tujuan. Banyak pendekatan yang mengasumsikan graf yang ditelusuri adalah graf terhubung yang merupakan representasi dari persoalan tertentu. Contoh persoalan pada graf terhubung adalah *social graph* yang menunjukkan keterhubungan seseorang dengan orang lain dalam suatu jaringan media sosial. Jika ingin diketahui berapa lama keterhubungan dua orang pada suatu jaringan sosial, maka dilakukan traversal graf untuk mencari solusi dari persoalan tersebut.

Terdapat dua algoritma traversal graf klasik, yaitu *Breadth First Search* (BFS) dan *Depth First Search* (DFS). Kedua algoritma traversal graf tersebut merupakan bagian dari algoritma pencarian solusi tanpa informasi (*uninformed*/*blind search*). Dalam proses pencarian solusi, terdapat dua pendekatan untuk merepresentasikan graf: graf statis, graf yang sudah terbentuk sebelum proses pencarian dilakukan; graf dinamis, graf yang terbentuk saat proses pencarian dilakukan.

2.2 *Breadth First Search* (BFS)

Breadth First Search (BFS) atau pencarian melebar adalah teknik pencarian berbasis simpul yang diawali dari simpul awal kemudian mengunjungi simpul yang bertetangga/sejajar terlebih dahulu. Struktur data yang digunakan pada BFS adalah antrian (*queue*). Algoritma untuk BFS adalah sebagai berikut.

1. Memulai traversal dengan mengunjungi simpul *v*.
2. Setelah mengunjungi simpul *v*, kunjungi semua simpul yang bertetangga dengan simpul *v*. Simpul yang telah dikunjungi disimpan pada antrian *q*.
3. Selanjutnya, kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang sudah dikunjungi, demikian seterusnya.

2.3 *Depth First Search* (DFS)

Depth First Search (DFS) atau pencarian mendalam adalah teknik pencarian berbasis sisi yang diawali dari simpul awal kemudian menelusuri jalur simpul yang bersisian hingga simpul yang paling akhir. Stuktur data yang digunakan pada DFS adalah tumpukan (*stack*). Algoritma untuk DFS adalah sebagai berikut.

1. Memulai traversal dengan mengunjungi simpul *v*.
2. Setelah mengunjungi simpul *v*, kunjungi simpul *w* yang bertetangga dengan simpul *v*.
3. Ulangi pencarian mendalam dimulai 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 akan berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

2.4 C# Desktop Application Development

Bahasa C# merupakan sebuah bahasa pemrograman *general-purpose* berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework dalam pengembangan aplikasi *desktop* maupun *mobile*. Bahasa C# dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa pemrograman berorientasi objek lainnya seperti Java. Dalam pengembangan aplikasi *desktop* dalam bahasa C#, pengembang dapat menggunakan IDE Visual Studio yang terintegrasi dengan .NET Framework.

Pengembangan aplikasi *desktop* menggunakan kerangka .NET Framework melalui Visual Studio cukup interaktif, terutama dalam pembuatan antarmuka/GUI dengan Windows Forms Application, pengembang dapat menyusun tampilan antarmuka secara langsung. Penggunaan kerangka .NET Framework melalui Visual Studio juga mudah karena sistemnya *build and launch*. Aplikasi dapat langsung dicoba ketika dibangun dan diluncurkan melalui Visual Studio.

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

Pemecahan masalah dimulai setelah memahami spesifikasi aplikasi dengan menganalisis dan memodelkan permasalahan ke dalam bentuk graf. Graf yang dibentuk dari persoalan ini adalah graf dinamis, terhubung, berarah, dan tidak membentuk sirkuit (mirip dengan sifat Pohon). Pada graf ini, simpul mewakili suatu direktori, sedangkan sisi menunjukkan konten dari direktori tersebut dengan direktori induk menunjuk kepada direktori anak. Graf ini juga bersifat dinamis karena jumlah simpul tetangga bertambah seiring pengunjungan tiap simpul.

Setelah memodelkan permasalahan ke dalam bentuk graf, langkah berikutnya adalah mendesain algoritma traversal graf secara BFS (pencarian melebar) dan DFS (pencarian mendalam) sesuai dengan teori yang telah dijelaskan pada Bab II. Seiring dengan membuat algoritma, ditentukan pula struktur data yang akan digunakan. Selanjutnya, mendesain tampilan antarmuka pengguna (GUI) agar pengguna dapat nyaman berinteraksi dengan aplikasi. Tahapan desain ini dilakukan secara berulang sesuai spesifikasi kebutuhan aplikasi.

Algoritma dan struktur data yang telah didesain selanjutnya diimplementasi dalam bentuk kumpulan kelas dengan konsep pemrograman berorientasi objek dalam bahasa C# menggunakan .NET Framework dan IDE Visual Studio. Implementasi dan pengembangan aplikasi dilakukan secara modular. Secara umum, program aplikasi dipecah menjadi dua buah proyek, yaitu Proyek GUI untuk antarmuka aplikasi dan Proyek Lib untuk pustaka aplikasi. Proyek GUI dibangun menggunakan Windows Forms Application. Sementara itu, Proyek Lib berisi algoritma dan struktur data dari aplikasi.

Setelah melakukan implementasi, selanjutnya aplikasi dibangun dan diluncurkan. Aplikasi akan diuji terlebih dahulu kebenarannya. Aplikasi harus dipastikan mengandung seluruh input dan output yang terdapat pada spesifikasi. Seiring melakukan pengujian pada aplikasi, *bug* yang ditemukan pada aplikasi harus diperbaiki. Tahapan ini dilakukan secara berulang hingga aplikasi bebas dari *bug* dan aplikasi siap untuk dirilis.

3.2 Proses *Mapping* Persoalan

Secara umum tahapan proses penyelesaian persoalan dengan memanfaatkan BFS dan DFS adalah sebagai berikut. Persoalan akan diselesaikan dengan melakukan pencarian. Proses pencarian akan membentuk pohon dinamis sebagai representasi ruang status pencarian. Simpul-simpul pada pohon yang dibentuk akan dikunjungi dan diperiksa. Jika pada saat pemeriksaan suatu simpul yang sudah dibangkitkan pada pohon ternyata sama dengan simpul tujuan atau *goal* yang dicari, maka pencarian dapat dihentikan. Jika yang diinginkan adalah memunculkan semua kemungkinan solusi maka pencarian tetap dilanjutkan hingga tidak ada lagi solusi yang dapat dibangkitkan. Solusi dari persoalannya adalah jalur (*path*) dari akar pohon hingga daun yang berupa simpul tujuan. Pohon dinamis yang dibentuk selama proses pencarian solusi merepresentasikan ruang status pencarian. Representasi tersebut dirincikan pada Tabel 3.2.1 berikut.

Tabel 3.2.1 Representasi Pohon Dinamis sebagai Ruang Status Pencarian

Elemen	Deskripsi
Pohon Ruang Status (State Space Tree)	Pohon yang dibangkitkan selama proses pencarian berlangsung. Pada persoalan *folder crawling*, pohon ruang status yang terbentuk adalah pohon direktori yang menunjukkan hubungan antar direktori induk dengan direktori anaknya.
Problem State	Simpul-simpul pada pohon yang layak membentuk solusi. Pada persoalan *folder crawling*, simpul yang layar membentuk solusi adalah direkori yang dapat mengarahkan kepada simpul tujuan
Solution/Goal State	Daun pada pohon sebagai simpul yang dikunjungi paling akhir. Daun juga dapat berupa simpul yang paling akhir dikunjungi namun bukan merupakan solusi karena tidak ada aksi yang dapat dilakukan lagi setelah mencapai simpul tersebut. Pada persoalan *folder crawling*, daun adalah *file* dan yang menjadi solusinya adalah *file* yang ingin ditemukan.

Initial State	Akar pada pohon sebagai simpul yang dikunjungi paling awal. Pada persoalan *folder crawling*, akar adalah direktori awal untuk memulai pencarian (*starting directory*).
Cabang (Branch)	Cabang pada pohon yang merupakan operator sebagai aksi atau langkah yang mungkin dilakukan dari suatu keadaan (*state*). Pada persoalan *folder crawling*, cabang yang terbentuk menunjukkan konten dari suatu direktori.
State Space	Himpunan semua simpul pada pohon. Pada persoalan *folder crawling*, himpunan simpul pada pohon yang terbentuk adalah *directory path* yang terbentuk selama pencarian berlangsung.
Solution Space	Himpunan dari status solusi. Pada persoalan *folder crawling*, himpunan dari status solusi pada pohon yang terbentuk adalah *direcotry path* yang mengarah kepada *file* yang ingin ditemukan.

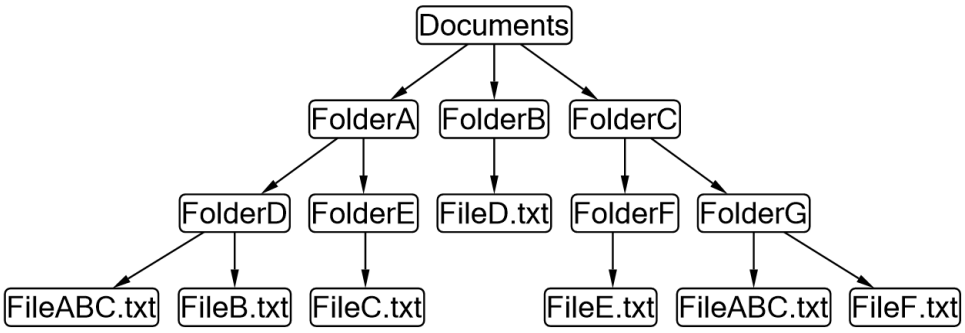
Penerapan BFS untuk penyelesaian persoalan dengan membentuk pohon ruang status pencarian solusi adalah sebagai berikut. Mula-mula, Inisialisasi dengan status awal sebagai akar. Selanjutnya, akan dibangkitkan semua simpul pada level *d*, sebelum simpul-simpul pada level $d+1$. Semua simpul pada suatu level dibangkitkan karena pada level sebelumnya belum ditemukan solusi dari persoalan yang akan diselesaikan. Setelah semua simpul pada suatu level dibangkitkan, diperiksa satu-persatu adakah yang sudah membentuk solusi atau sudah sampai simpul solusi. Jika sudah sampai dan tidak ingin dicari kemungkinan solusi lain, maka level berikutnya tidak dibangkitkan, namun jika masih ingin mencari kemungkinan solusi lain maka simpul-simpul pada level kedalaman berikutnya juga ikut diperiksa.

Penerapan DFS untuk penyelesaian persoalan dengan membentuk pohon ruang status pencarian solusi adalah sebagai berikut. Mula-mula inisialisasi dengan status awal sebagai akar. Selanjutnya, akan dibangkitkan satu simpul pada level *d*. Jika simpul tersebut bukan merupakan simpul solusi, lanjutkan dengan pembangkitan satu simpul di level $d+1$, hingga tidak ada lagi simpul yang bisa dibangkitkan pada kedalaman berikutnya. Pembangkitan simpul dilanjutkan dengan simpul ke-2 pada kedalaman terakhir. Jika tidak ada lagi simpul

yang bisa dibangkitkan pada kedalaman terakhir, maka *backtrack* atau kembali lagi ke kedalaman sebelumnya untuk membangkitkan simpul berikutnya pada kedalaman tersebut dan dicoba lagi untuk membangkitkan simpul pada kedalaman berikutnya hingga ditemukan suatu simpul solusi.

3.3 Ilustrasi Kasus

Misalkan pengguna ingin mengetahui langkah *folder crawling* untuk menemukan *file* FileABC.txt yang pencariannya dimulai dari folder Documents. Struktur pohon direktori dari Documents adalah sebagai berikut.



Gambar 3.3.1 Struktur Pohon Direktori Documents

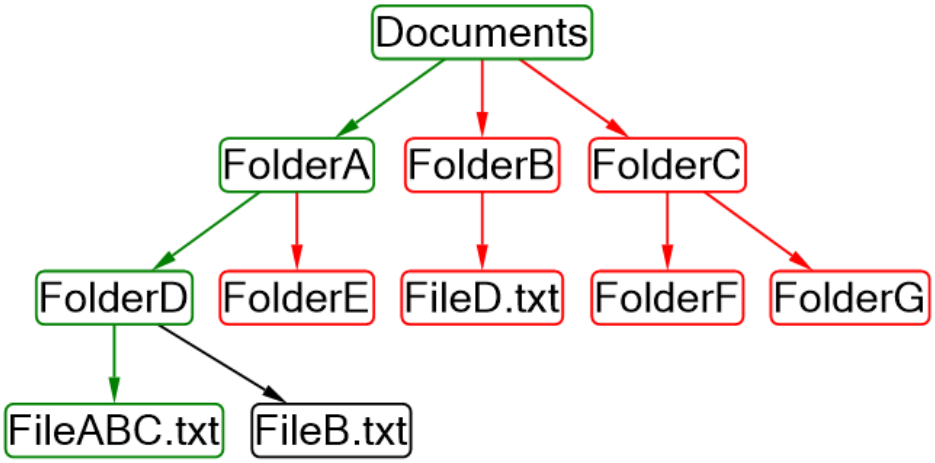
Persoalan kemudian dianalisis menjadi representasi pohon dinamis. Berdasarkan narasi, didapatkan representasi pohon dinamis sebagai berikut.

Elemen	Representasi
Pohon Ruang Status (State Space Tree)	Pohon direktori yang terbentuk dari Documents
Problem State	Documents, FolderA, FolderB, FolderC, FolderD, FolderE, FolderF, FolderG
Solution/Goal State	Daun: FileABC.txt, FileB.txt, FileC.txt, FileD.txt, FileE.txt, FileABC.txt, FileF.txt. Solution: FileABC.txt

Initial State	Documents sebagai akar dari pohon direktori
Cabang (Branch)	Cabang-cabang yang dibentuk oleh simpul (problem state) mengindikasikan konten dari tiap folder
State Space	Jalur/*path* yang terbentuk adalah: 1. Documents – FolderA – FolderD – FileABC.txt 2. Documents – FolderA – FolderD – FileB.txt 3. Documents – FolderA – FolderE – FileC.txt 4. Documents – FolderB – FileD.txt 5. Documents – FolderC – FolderF – FileE.txt 6. Documents – FolderC – FolderG – FileABC.txt 7. Documents – FolderC – FolderG – FileF.txt
Solution Space	Jalur/*path* yang menjadi solusi adalah: 1. Documents – FolderA – FolderD – FileABC.txt 2. Documents – FolderC – FolderG – FileABC.txt

Pencarian file dilakukan memanfaatkan BFS dan DFS. Terdapat dua skenario pencarian yaitu pencarian file yang pertama kali ditemukan dan pencarian semua keberadaan file yang ingin ditemukan. Pada tiap skenario, akan diberikan gambar pohon pencarian untuk memberikan ilustrasi jalur pencarian yang dilakukan. Simpul dan sisi diberi warna merah untuk menunjukkan jalur tersebut telah dikunjungi, diberi warna hijau untuk menunjukkan solusi, dan diberi warna hitam untuk menunjukkan jalur yang masuk antrian tetapi tidak dikunjungi. Penjelasan skenario tersebut adalah sebagai berikut.

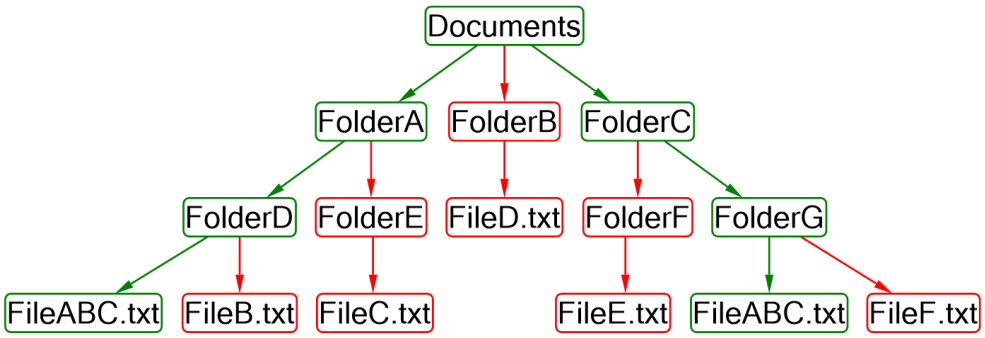
1. BFS Skenario 1: Pencarian file yang pertama kali ditemukan dengan Metode BFS



Gambar 3.3.2 Pohon Pencarian dengan BFS Skenario 1

Pada skenario ini, pencarian dilakukan dengan mengunjungi semua simpul pada setiap aras secara bertahap. Dapat terlihat pada Gambar 3.3.2, simpul dan sisi pada level-0 hingga level-2 telah dikunjungi semua. Ketika aras-3 dikunjungi, file ditemukan dan pencarian dihentikan.

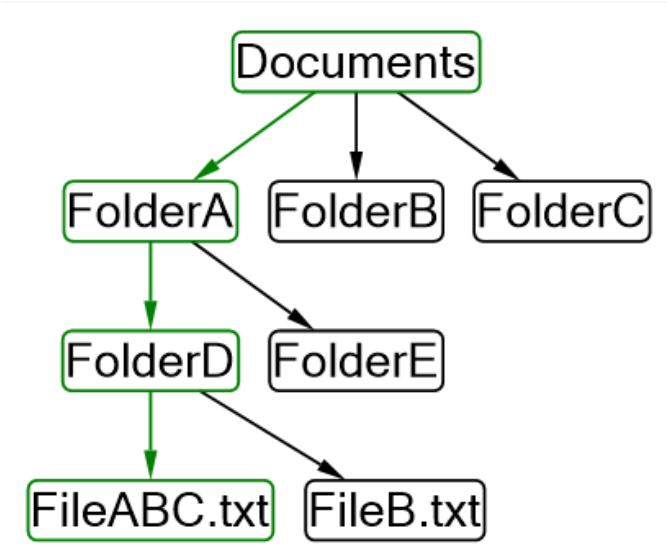
2. BFS Skenario 2: Pencarian semua keberadaan file yang ingin ditemukan dengan Metode BFS



Gambar 3.3.3 Pohon Pencarian dengan BFS Skenario 2

Pada skenario ini, semua simpul dikunjungi pada tiap aras. Dapat terlihat pada Gambar 3.3.3, file yang dicari ditemukan pada aras-3. Ditemukan terdapat file dapat ditemukan pada dua jalur yang berbeda.

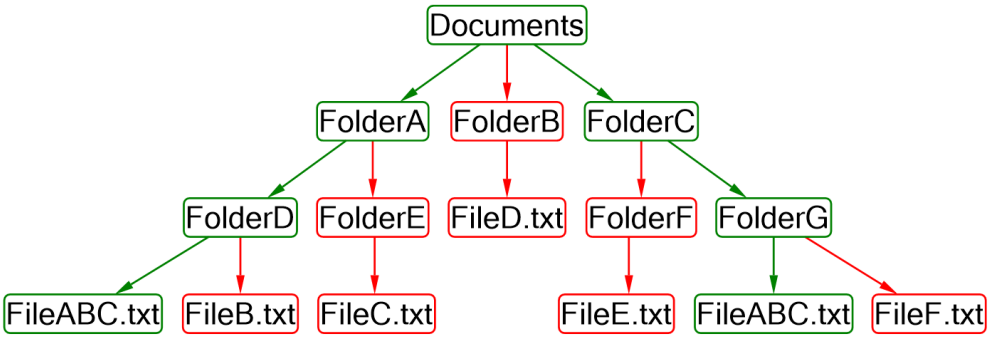
3. DFS Skenario 1: Pencarian file yang pertama kali ditemukan dengan Metode DFS



Gambar 3.3.4 Pohon Pencarian dengan DFS Skenario 1

Pada skenario ini, berbeda dengan BFS, pada DFS suatu simpul dikunjungi dan anak dari simpul dikunjungi hingga mencapai daun. Dapat terlihat pada Gambar 3.3.4, penelurusan simpul lebih sedikit dibandingkan dengan BFS karena pencarian dilakukan berdasarkan kedalaaman pohon.

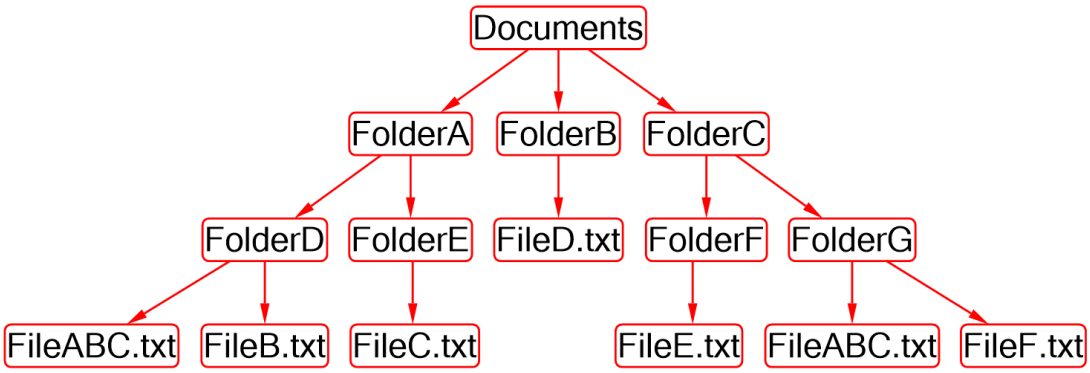
1. DFS Skenario 2: Pencarian semua keberadaan file yang ingin ditemukan dengan Metode DFS



Gambar 3.3.5 Pohon Pencarian dengan DFS Skenario 2

Pada skenario ini, semua simpul dikunjungi dengan menelusuri hingga pada kedalaman dari pohon. Pohon pencarian yang terbentuk terlihat serupa dengan BFS Skenario 2, namun terdapat perbedaan pada prosesnya. Ditemukan terdapat file dapat ditemukan pada dua jalur yang berbeda.

Adapun alternatif kasus yang mungkin terjadi yaitu ketika file yang ingin dicari tidak ditemukan. Pada pencarian baik menggunakan metode BFS dan DFS membentuk pohon pencarian sebagai berikut.



Gambar 3.3.6 Pohon Pencarian Kasus File Tidak Ditemukan

Tampak pada Gambar 3.3.6, simpul dan sisi pada pohon pencarian diwarnai dengan warna merah mengindikasikan seluruh jalur telah dikunjungi namun file yang dicari tidak ditemukan.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

Program diimplementasikan menggunakan *framework .NET 6.0* dan bahasa pengembangan C#. Tampilan depan program dikembangkan menggunakan *IDE Visual Studio 2022* berbasis WinForms dengan tambahan *package* MaterialSkin untuk menambah estetika aplikasi**.** Animasi penggambaran menggunakan *class graphViewer* dari *library* Microsoft Automatic Graph Layout (MSAGL).

Terdapat dua buah *library* dalam program, yaitu *library* GUI untuk mengatur antarmuka program dan *library* Lib yang mengandung algoritma BFS dan DFS yang digunakan. Penjelasan mengenai *library* GUI adalah sebagai berikut.

1. MainForm.cs

   MainForm bertanggung jawab sebagai penghubung interaksi antara *back-end* dengan *front-end* dari program. Implementasi *front-end* (properti label, *button*, *checkbox*, dan lain-lain) dari program terletak pada MainForm.Designer.cs. *Pseudocode* dari MainForm adalah sebagai berikut.

```
class MainForm : MaterialForm
    /* Attribute */
    // Inisialisasi atribut yang bertanggung jawab dalam
       visualisasi pohon pencarian
    DirectoryDrawer Drawer

    // Inisialisasi atribut yang bertanggung jawab untuk UI
    MaterialSkinManager SkinManager

    /* Default Constructor */
    // menginisiasi objek yang bertanggung jawab untuk UI
    public MainForm()

    /* Kelompok Method Event Listener */

    // interaksi dengan tombol 'CHOOSE FOLDER'
    void DirButton_Click(object, EventArgs)

    // interaksi dengan text box file name
    void FileInput_KeyDown(object, KeyEventArgs)

    // interaksi dengan checkbox all occurences
    void OccurenceCheckBox_CheckedChanged(object, EventArgs)

    // interaksi dengan radio button BFS
```

```
    void BFSButton_CheckedChanged(object, EventArgs)

    // interaksi dengan radio button DFS
    void DFSButton_CheckedChanged(object, EventArgs)

    // interaksi dengan tombol 'SEARCH'
    void SearchButton_Click(object sender, EventArgs)

    // interaksi dengan switch theme
    void ThemeSwitch_CheckedChanged(object, EventArgs)

    // interaksi dengan slider delay speed
    void DelaySpeed_ValueChanged(object, int)

    /* Kelompok Method Updater */
    // penghubung class DirectoryTraversal dengan MainForm
    // interaksi dengan label status
    void UpdateStatus(string)

    // interaksi dengan label path hasil pencarian
    void UpdateLink(string)
```

2. DirectoryDrawer.cs

DirectoryDrawer bertanggung jawab dalam visualisasi atau penggambaran pohon pencarian. *Pseudocode* dari DirectoryDrawer adalah sebagai berikut.

```
class DirectoryDrawer
      /* Attribute */
      // Atribut yang menerima hasil interaksi program dengan pengguna
      string Path = "" // menyimpan starting directory
      bool IsRunning = false // menyimpan informasi keberjalanan program
      string FileName = "" // menyimpan informasi file yang ingin ditemukan
      bool AllOccurences = false // menyimpan informasi skenario pencarian
      Algorithm Algorithm // menyimpan informasi metode BFS/DFS
      int DrawDelay // menyimpan informasi delay penggambaran pohon pencarian

      // Atribut yang bertanggung jawab dalam proses traversal graf
      List<string> foundPaths // menyimpan path file yang dicari
      Graph graph // menyimpan graf/pohon pencarian
      bool isFound = false // menyimpan informasi ditemukan/tidaknya file

      // worker memungkinkan operasi berjalan pada thread yang berbeda
      BackgroundWorker worker
      // traverser bertanggung jawab terhadap algoritma BFS/DFS
      DirectoryTraversal Traverser
      // bertanggung jawab untuk menampilkan graf
      GViewer graphViewer
      GViewer GraphViewer
      // menyimpan informasi sisi graf
      Dictionary<string, Edge> idToEdges
      // bertanggung jawab dalam kalkulasi waktu eksekusi
      Stopwatch sw
      // atribut yang bertanggung jawab untuk memperbarui Label Status dan Link
      Action<string> UpdateStatus
```

```
        Action<string> UpdateLink

        /* Default Constructor */
        // menginisiasi atribut-atribut yang dimiliki oleh objek worker,
        // traverser, dan graphViewer
        DirectoryDrawer()

        /* Method */
        void DrawTraverse() // mengeksekusi worker, proses pencarian dimulai

        void Traverse() // inisiasi graf traversal

        void CreateGraph() // inisiasi graf
        void DrawNodeGraph(string, string) // menggambar simpul graf
        void DrawEdgeGraph(string, string) // menggambar sisi graf
        void DrawFoundGraph(string) // menggambar path file yang dicari

        // Method yang bertanggung jawab dengan objek worker
        void Worker_ProgressChanged(object, ProgressChangedEventArgs)
        void Worker_DoWork(object, DoWorkEventArgs)
        void Worker_CalculateTime(object, RunWorkerCompletedEventArgs)

        // Method yang bertanggung jawab dengan objek traverser
        void OnFile(FileInfo)
        void OnFound(FileInfo)
        void OnDirectory(DirectoryInfo)
```

3. Program.cs

Program adalah kelas yang pertama kali dijalankan oleh aplikasi. Pseudocode dari Program adalah sebagai berikut.

```
class Program
    void Main()
         // Program utama, fungsi yang pertama kali dijalankan oleh aplikasi
        ApplicationConfiguration.Initialize()
        Application.Run(new MainForm())
```

Dalam *library* Lib terdapat bagian Algorithms, yang mengatur implementasi *class* untuk pemakaian algoritma di form. File TraversalAlgorithm.cs adalah *base class* yang digunakan untuk implementasi *class* BFS dan DFS. Kedua algoritma memanfaatkan penggunaan *background worker* untuk melakukan penggambaran graf pada *graphViewer*. Penjelasan mengenai *pseudocode* program adalah sebagai berikut.

1. TraversalAlgorithm.cs

```
class TraversalAlgorithm
    // Atribut yang bertanggung jawab dalam proses penggambaran pohon pencarian
    Action<FileInfo> OnFile
    Action<FileInfo> OnFound
    Action<DirectoryInfo> OnDirectory
```

```
    string FileName
    int DrawDelay
    bool IsFound
    bool AllOccurences

interface ITraversable
    // Method yang akan diimplementasikan pada kelas BFS dan kelas DFS
    void Traverse(string DirPath, string FileName, bool AllOccurance)
```

2. BFS.cs

   Algoritma BFS diimpementasikan sebagai berikut:

   1) Inisialisasi struktur data *queue* dan *dictionary*. *Queue* digunakan untuk mengantrikan *folder* yang akan diperiksa, sedangkan *dictionary* digunakan untuk menandai nama-nama file/folder yang telah diperiksa
   2) Melakukan pencarian *file* dan *folder* pada *folder root.* Pencarian mengutamakan file dengan melakukan iterasi nama file dalam folder tersebut.
      a. Apabila direktori *file* belum ditemukan dalam *dictionary*, tambahkan direktori *file* tersebut ke dalam *dictionary*
      b. Fungsi akan melakukan penggambaran *node* dan edge baru dari *folder root* ke file.
      c. Apabila nama *file* sesuai dengan nama *file* yang ingin dicari, fungsi akan melakukan invokasi sehingga *background worker* akan melakukan penggambaran *node* hijau (tanda bahwa *file* berhasil ditemukan)
   3) Apabila pengguna ingin mencari semua file dengan nama yang diinginkan dalamf older tersebut, pencarian akan berlanjut dengan meng-*enqueue* nama direktori yang ada dalam folder ke *queue*. Selanjutnya, program akan memperbarui nama *folder* root ke nama direktori di *queue* dan pencarian akan dilakukan dengan nama direktori sebagai *folder root*.
   4) Pencarian dilakukan terus-menerus hingga *queue* kosong (dengan kata lain, semua direktori sudah diperiksa). Apabila pengguna ingin mencari satu kemunculan saja, pencarian berhenti saat nama file ditemukan pertama kali.

Pseudocode dari algoritma BFS adalah sebagai berikut.

```
    class BFS : TraversalAlgorithm, ITraversable
        Queue<string> Q
        Dictionary<string, bool> VisitedNodes
        BFS(int DrawDelay = 25)
            this.DrawDelay = DrawDelay
```

```
        void TraverseBFS(string DirPath)
            Q.Clear()
            VisitedNodes.Clear()
            VisitedNodes[DirPath] = true
            Q.Enqueue(DirPath)

            while (Q.Count > 0)
                string DeqNode = Q.Dequeue()
                DirectoryInfo DirMain = new(DeqNode)

                // Bangkitkan simpul ekspan
                foreach (DirectoryInfo Directory in
DirMain.EnumerateDirectories().Reverse())
                    if (!VisitedNodes.ContainsKey(Directory.FullName))
                        OnDirectory?.Invoke(Directory)
                        Thread.Sleep(DrawDelay)

                foreach (FileInfo File in DirMain.EnumerateFiles().Reverse())
                    if (!VisitedNodes.ContainsKey(File.FullName))
                        OnFile?.Invoke(File)
                        Thread.Sleep(DrawDelay)

                // Visit ke setiap file dan directory
                foreach (FileInfo File in DirMain.EnumerateFiles())
                    if (!VisitedNodes.ContainsKey(File.FullName))
                        OnVisitedFile?.Invoke(File)
                        if (File.Name == FileName)
                            OnFound?.Invoke(File)
                            if (!AllOccurences)
                                Q.Clear()
                                VisitedNodes.Clear()
                                return
                        Thread.Sleep(DrawDelay)

                foreach (DirectoryInfo Directory in DirMain.EnumerateDirectories())
                    if (!VisitedNodes.ContainsKey(Directory.FullName))
                        Q.Enqueue(Directory.FullName)
                        OnVisitedDirectory.Invoke(Directory)
                        Thread.Sleep(DrawDelay)

        void Traverse(string DirPath, string FileName, bool AllOccurences)
            this.FileName = FileName
            this.AllOccurences = AllOccurences
            TraverseBFS(DirPath)
```

3. DFS.cs

   Algoritma DFS diimplementasikan sebagai berikut:

   1) Pencarian *file* pada *folder root*, dan penggambaran *node* dan *edge* baru dari *folder root* ke file

2) Pencarian dilanjutkan dengan melakukan pengecekan direktori. Untuk tiap direktori, pencarian akan dilakukan secara rekursif dengan memanggil fungsi DFS lagi di dalamnya dengan direktori tersebut sebagai basis pencarian.
3) Pencarian akan dilakukan terus-menerus hingga semua fungsi rekursif sudah selesai dilakukan. Apabila pengguna ingin mencari satu kemunculan saja, pencarian berhenti saat nama file ditemukan pertama kali.

Pseudocode dari algoritma DFS adalah sebagai berikut.

```
class DFS : TraversalAlgorithm, ITraversable
        DFS(int DrawDelay = 25)
            this.DrawDelay = DrawDelay
            IsFound = false

        void TraverseDFS(string DirPath)
            // Jika sudah ketemu dan tak cek alloccurence, skip smua backtrack.
            if (!AllOccurences && IsFound)
                return
            DirectoryInfo DirMain = new(DirPath)

            // Bangkitkan simpul hidup
            foreach (DirectoryInfo Directory in
DirMain.EnumerateDirectories().Reverse())
                OnDirectory.Invoke(Directory)
                Thread.Sleep(DrawDelay)
            foreach (FileInfo File in DirMain.EnumerateFiles().Reverse())
                OnFile?.Invoke(File)
                Thread.Sleep(DrawDelay)

            // Visit semua node simpul hidup
            foreach (FileInfo File in DirMain.EnumerateFiles())
                OnVisitedFile.Invoke(File)
                if (File.Name == FileName)
                    IsFound = true
                    OnFound.Invoke(File)
                    if (!AllOccurences)
                        return

                Thread.Sleep(DrawDelay)

            foreach (DirectoryInfo Directory in DirMain.EnumerateDirectories())
                // Bangkitkan simpul ekspan
                OnVisitedDirectory.Invoke(Directory)
                TraverseDFS(Directory.FullName)
                // Jika sudah ketemu dan tak cek alloccurence, skip checking
setelahnya.
                if (!AllOccurences && IsFound)
                    return
                Thread.Sleep(DrawDelay


         void Traverse(string DirPath, string FileName, bool AllOccurences)
            this.FileName = FileName
            this.AllOccurences = AllOccurences
            IsFound = false
            TraverseDFS(DirPath)
```

4.2 Struktur Data

Pada implementasi program, adapun struktur data yang digunakan dan yang perlu diperhatikan adalah sebagai berikut.

1. Struktur Data Queue

```
class BFS : TraversalAlgorithm, ITraversable
    // Inisialisasi Queue
    Queue<string> Q
    ...
```

Struktur data queue digunakan pada implementasi algoritma BFS. Queue digunakan untuk menyimpan urutan daftar direktori tetangga yang akan dikunjungi berikutnya.

2. Struktur Data Stack

```
class DFS : TraversalAlgorithm, ITraversable
     void TraverseDFS(string DirPath)
          . . .
          if (!AllOccurences && IsFound) // basis
               return
          . . .
          foreach (DirectoryInfo Directory in DirMain.EnumerateDirectories())
               // Bangkitkan simpul ekspan
               OnVisitedDirectory.Invoke(Directory)
               TraverseDFS(Directory.FullName) // rekurens
```

Struktur data stack tidak secara eksplisit diimplementasikan pada program, melainkan secara implisit dengan melihat perilaku pemanggilan fungsi pada implementasi algoritma DFS. Fungsi dipanggil secara rekursif membentuk *call stacks* pada program. Penerapan stack ini dimaksudkan agar ketika suatu direktori diakses, maka konten dari direktori tersebut akan diakses lagi alih-alih mengakses direktori tetangga. Pengaksesan direktori dilakukan secara rekursif hingga menuju suatu file atau tidak ada lagi direktori yang dapat diakses.

3. Struktur Data List

```
class DirectoryDrawer
      . . .
      // Atribut yang bertanggung jawab dalam proses traversal graf
      List<string> foundPaths // menyimpan path file yang dicari
      . . .
```

Struktur data list digunakan pada proses penggambaran pohon pencarian. Ketika file yang dicari ditemukan, maka directory path yang menunjuk pada file tersebut akan disimpan pada list yang nantinya digunakan dalam penggambaran pohon pencarian.

4. Struktur Data Dictionary

```
class DirectoryDrawer
    . . .
    // menyimpan informasi simpul dan sisi pohon pencarian
    Dictionary<string, Edge> idToEdges
    . . .
```

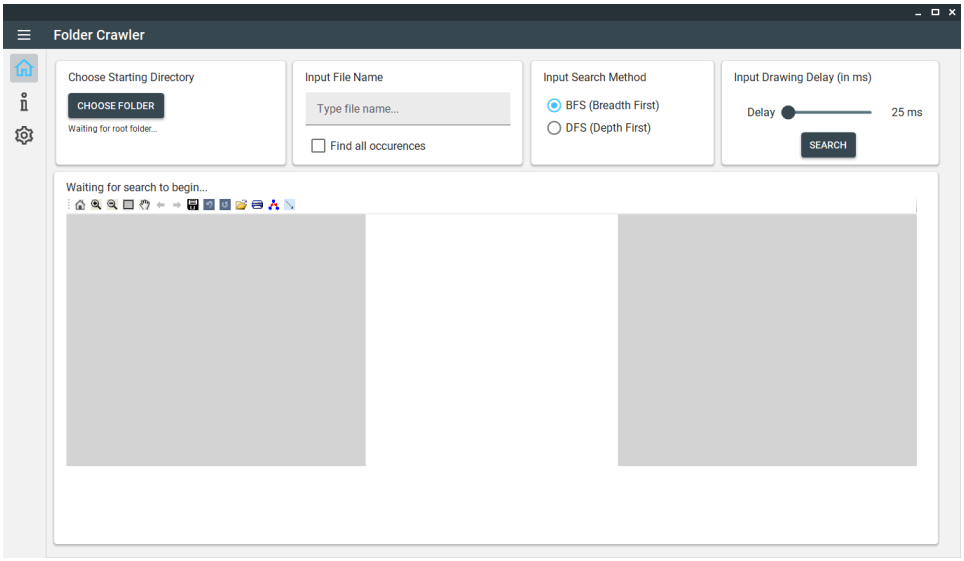
Struktur data dictionary digunakan untuk menyimpan informasi simpul dan sisi untuk menggambar pohon pencarian. Informasi ini nantinya akan dikirim dan diolah pada metode yang bertanggung jawab dalam visualisasi graf pada saat penggambaran pohon pencarian.

```
class BFS : TraversalAlgorithm, ITraversable
    . . .
    // Inisialisasi Dictionary untuk menyimpan informasi simpul
    // yang telah dikunjungi
    Dictionary<string, bool> VisitedNodes
    . . .
```

Struktur data dictionary digunakan juga digunakan pada algoritma BFS. Penggunaan struktur data ini adalah untuk menyimpan informasi simpul-simpul yang telah dikunjungi. Informasi pengunjungan simpul ini membantu struktur data queue dalam menentukan simpul yang akan dikunjungi berikutnya.

4.3 Tata Cara Penggunaan Program

Untuk memulai menggunakan program, pengguna terlebih dahulu menjalankan program yang bernama FilEdge.exe hingga didapatkan tampilan antarmuka aplikasi sebagai berikut.



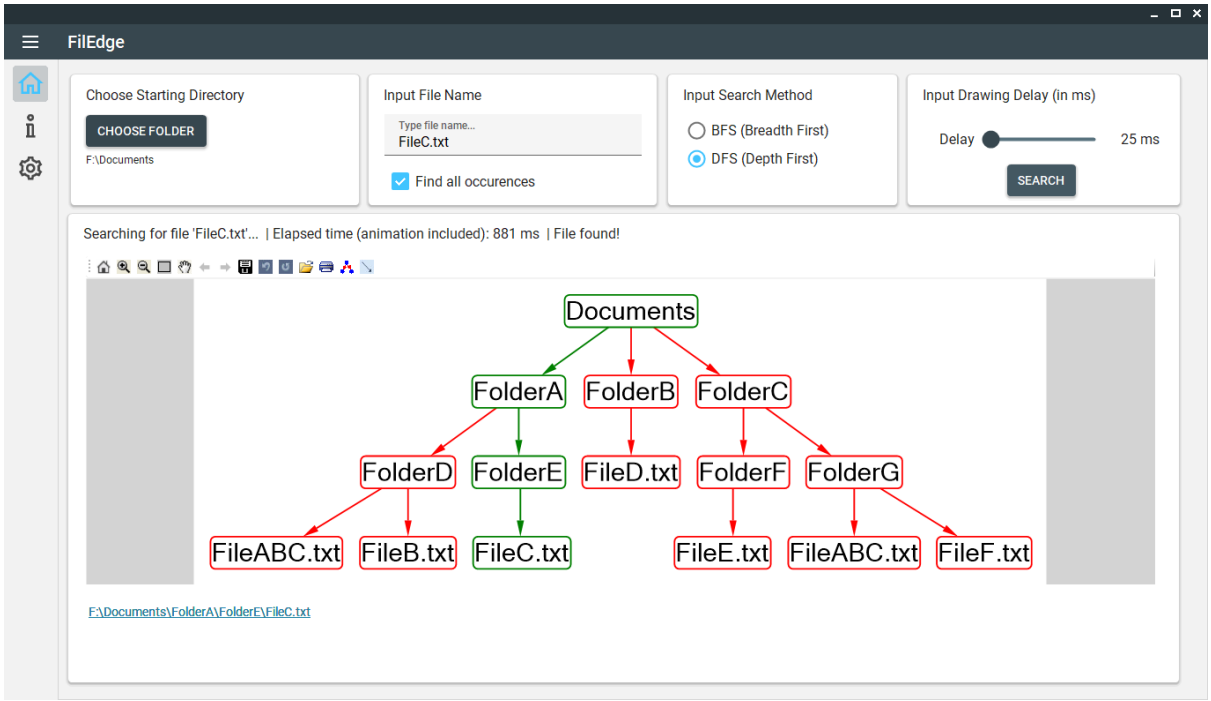
Gambar 4.3.1 Tampilan Antarmuka Aplikasi

Setelah berhasil menjalankan aplikasi, berikut ini adalah petunjuk dan langkah-langkah penggunaan program.

1. Mula-mula, pada bagian ‘Choose Starting Directory’, pengguna memberikan masukan direktori awal (*starting directory*) sebagai simpul awal pencarian dengan menekan tombol ‘CHOOSE FOLDER’.
2. Langkah selanjutnya, pada bagian ‘Input File Name’, pengguna memberikan masukan nama file yang ingin ditemukan lengkap dengan ekstensinya sebagai simpul tujuan dengan mengisi kotak yang diberi petunjuk ‘Type file name…’
3. Di bawah kotak pengisian nama file, pengguna memilih apakah ingin mencari semua keberadaan file atau hanya file yang ditemukan pertama kali. Jika ingin mencari semua keberadaan file, dapat mencentang box ‘Find all occurences’.
4. Langkah selajutnya, pada bagian ‘Input Search Method’, pengguna memilih metode yang digunakan untuk melakukan pencarian, yaitu apakah menggunakan metode BFS atau DFS. Secara default program akan memilih metode BFS sebagai metode pencarian.
5. Langkah selanjutnya, pada bagian ‘Input Drawing Delay’, pengguna memberikan masukan *delay speed* penggambaran pohon pencarian dengan menggeser *slider*. Secara default program akan memberikan *delay* penggambaran sebesar 25ms.
6. Langkah terakhir, di bawah slider Delay, pengguna menekan tombol search untuk memulai pencarian.

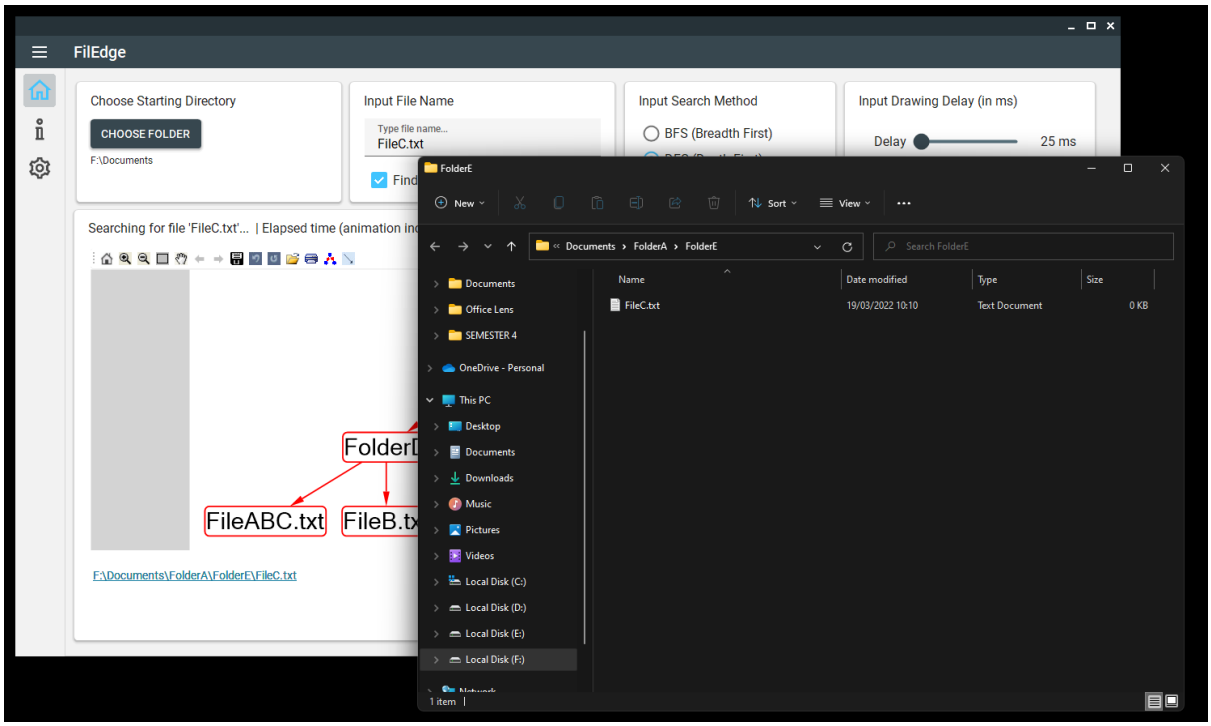
7. Program akan menampilkan pohon pencarian selama pencarian berlangsung hingga selesai.
8. Pencarian selesai ketika file yang dicari berhasil ditemukan atau seluruh file telah diakses namun file tidak ditemukan.
9. Program akan menampilkan lama waktu eksekusi dan memberikan *hyperlink path* dari file yang ingin ditemukan.

Berikut ini adalah tampilan dari aplikasi setelah melakukan pencarian.



Gambar 4.3.2 Tampilan Aplikasi Setelah Melakukan Pencarian

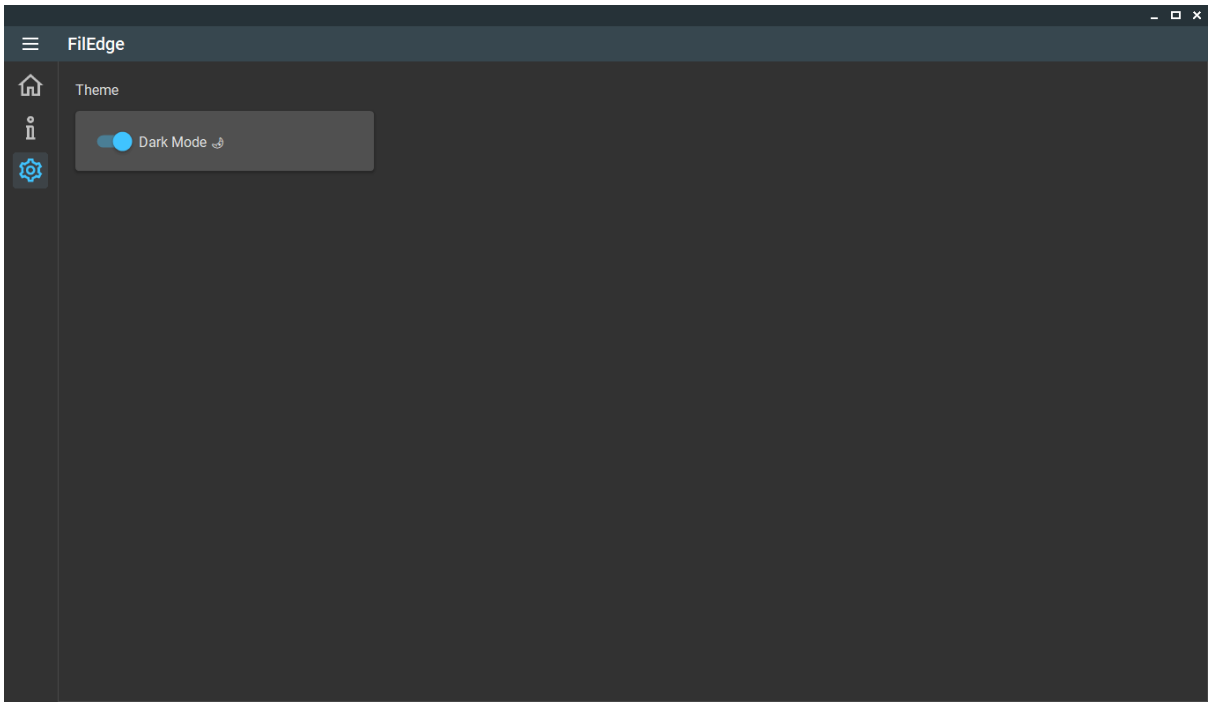
Ketika *hyperlink path* dari file yang ingin dicari diklik, maka akan muncul File Explorer dimana file tersebut berada.



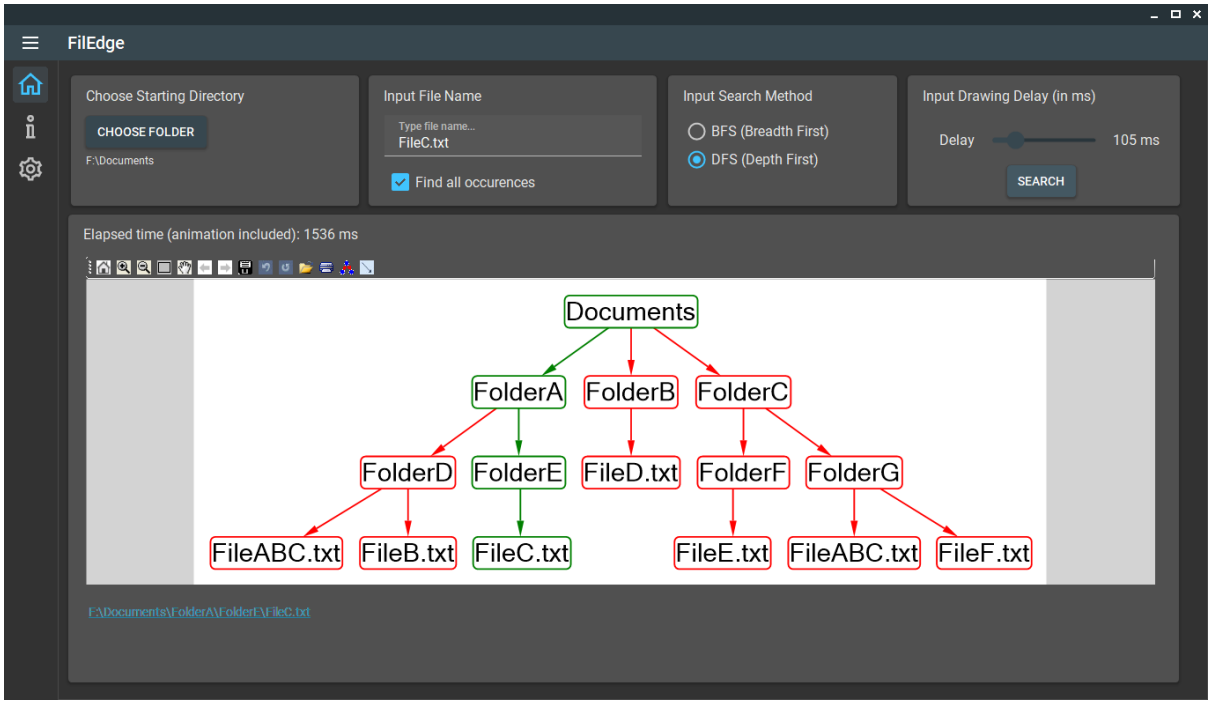
Gambar 4.3.3 Hyperlink diklik

Opsional:

Pengguna dapat mengganti tema aplikasi menjadi ‘Dark’ dengan mengakses halaman Settings dari drawer yang berada di paling kiri dari aplikasi.



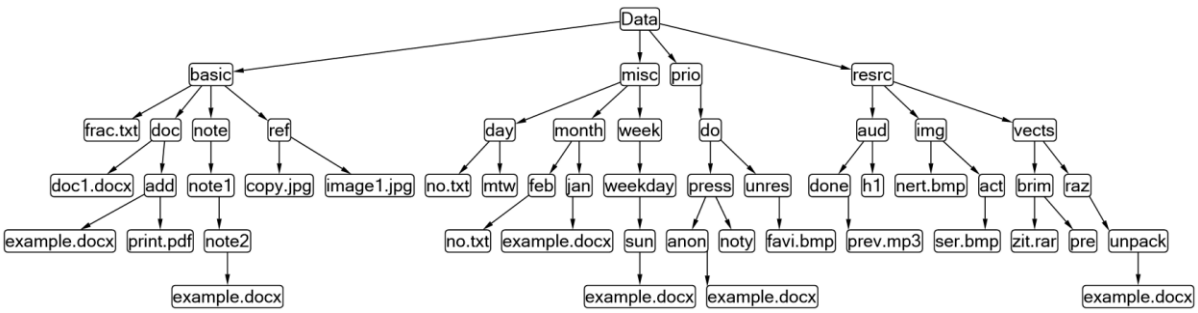
Gambar 4.3.4 Tampilan Halaman Settings



Gambar 4.3.5 Tampilan Aplikasi dengan Tema Dark

4.4 Hasil Pengujian

Untuk menguji kebenaran program, perlu dilakukan pengujian beberapa kemungkinan kasus. Adapun pohon direktori yang akan digunakan pada pengujian kasus adalah sebagai berikut.

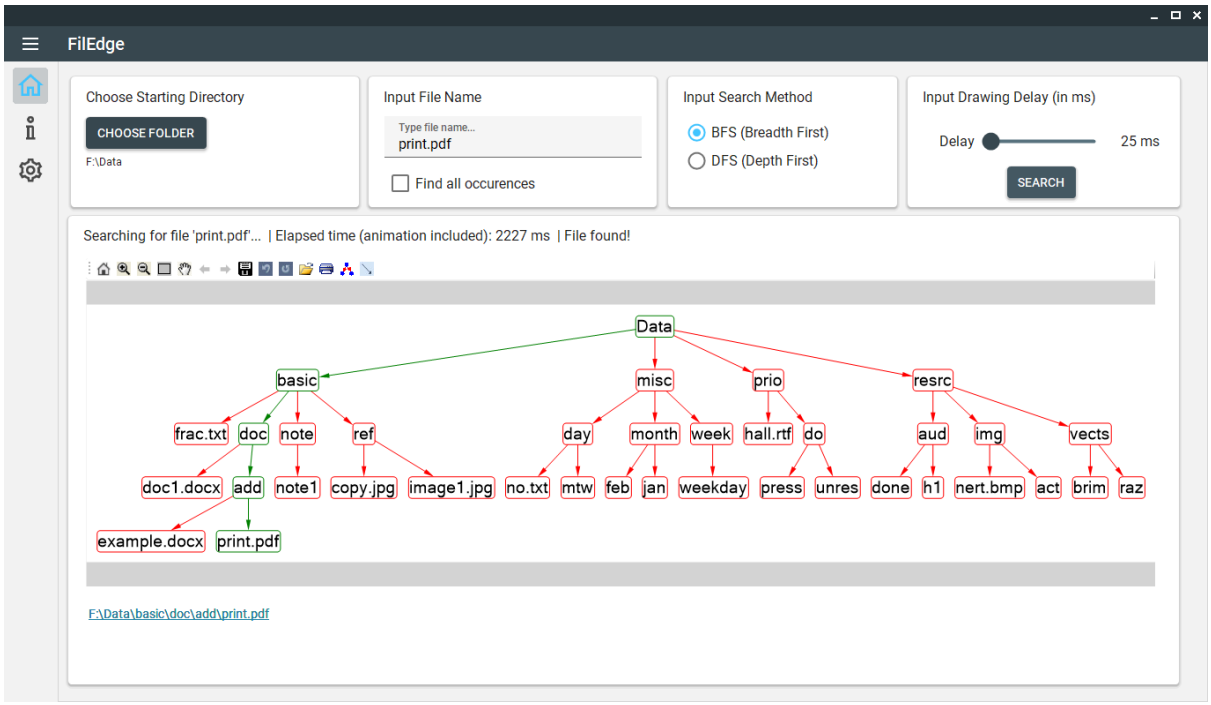


Gambar 4.4.1 Pohon Direktori untuk Pengujian Kasus

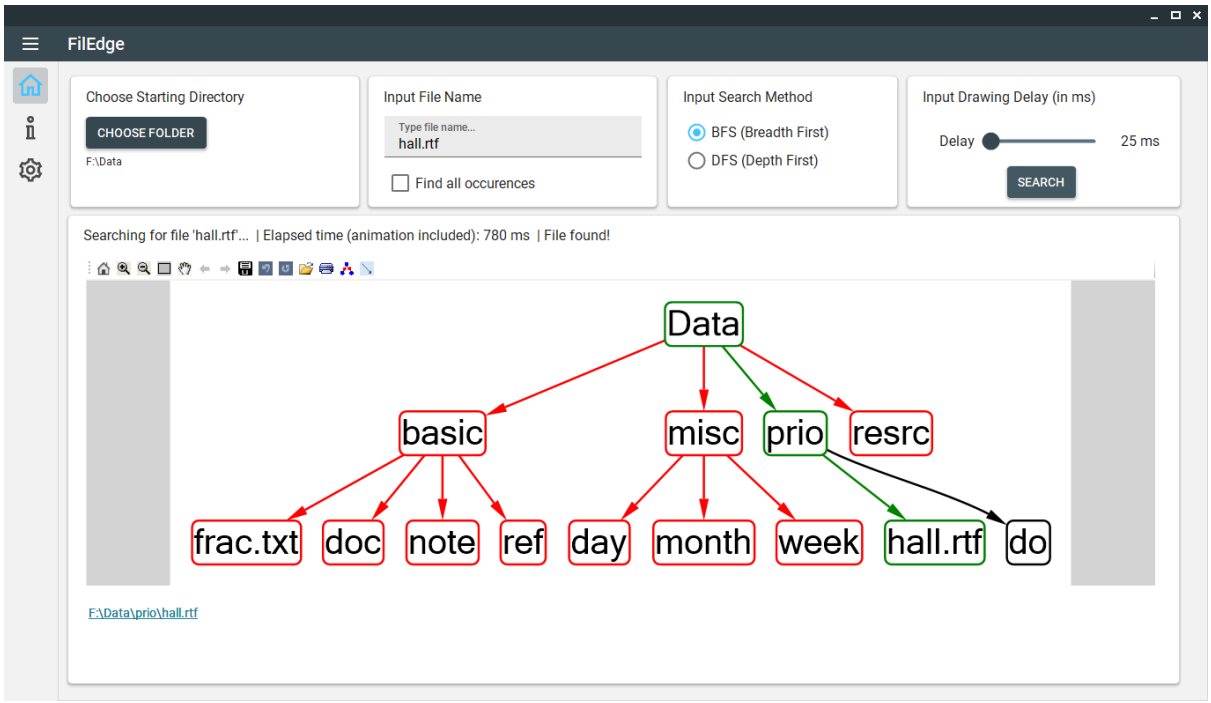
1. Kasus Uji 1: Mencari file yang pertama kali ditemukan

Akan dicari file yang bernama ‘print.pdf dan ‘hall.rtf’

Menggunakan metode BFS, didapatkan hasil sebagai berikut.

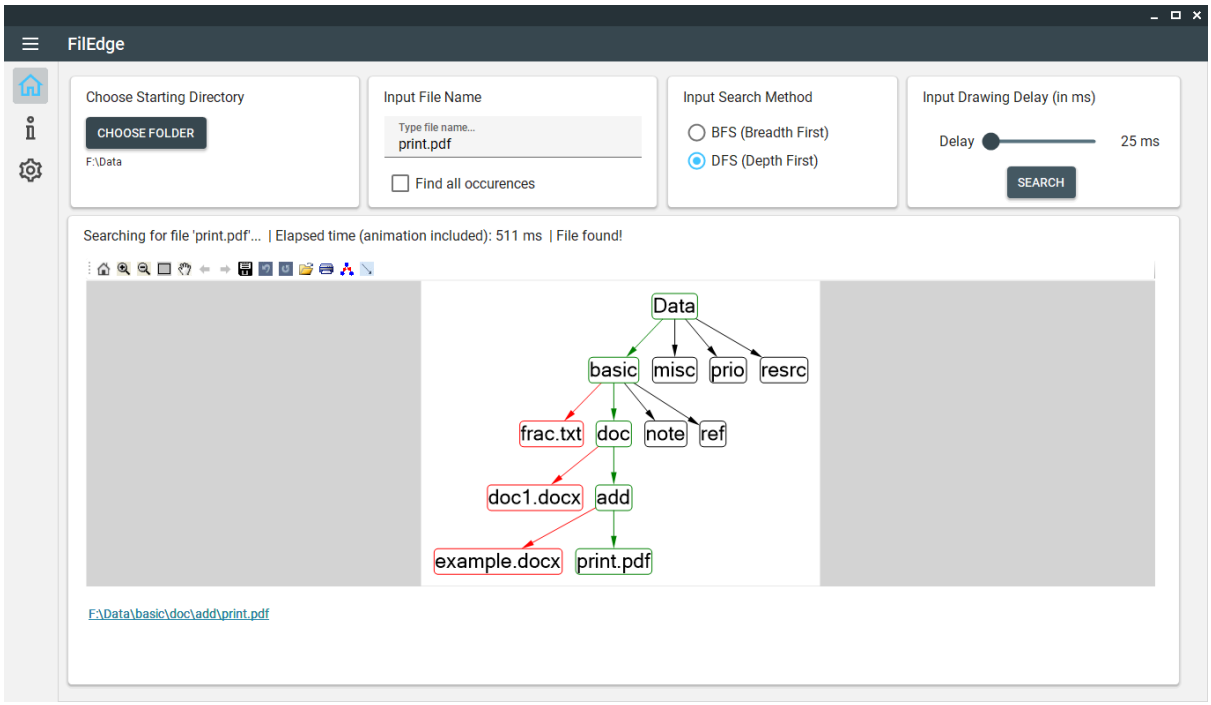


Gambar 4.4.2 Pencarian file 'print.pdf' dengan BFS

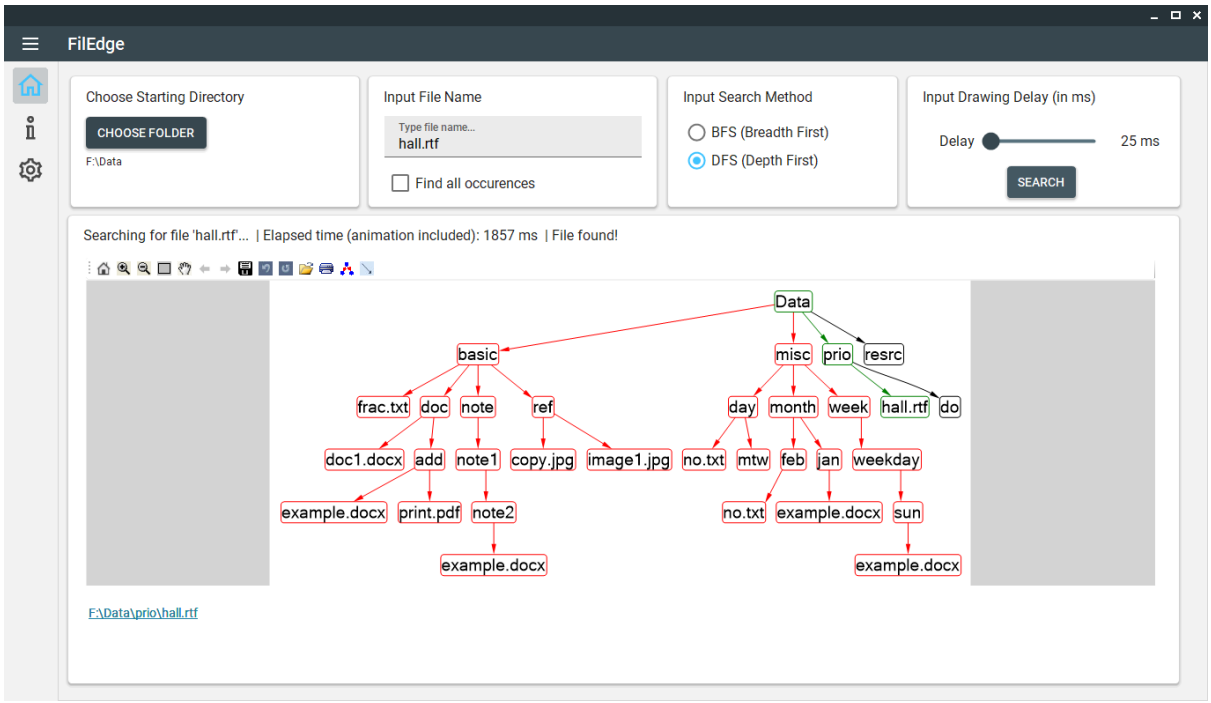


Gambar 4.4.3 Pencarian file hall.rtf' dengan BFS

Menggunakan metode DFS, didapatkan hasil sebagai berikut.



Gambar 4.4.4 Pencarian file 'print.pdf' dengan Metode DFS



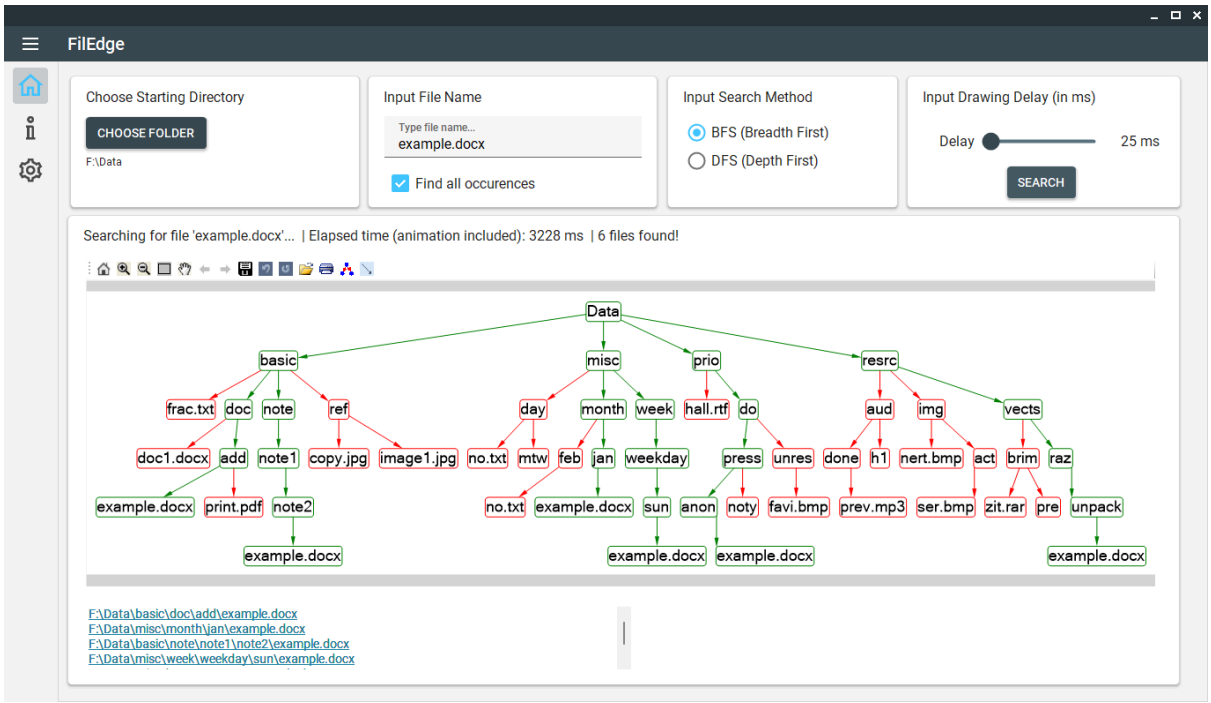
Gambar 4.4.5 Pencarian file 'hall.rtf' dengan Metode DFS

Berdasarkan hasil tersebut, ketika mencari file 'print.pdf', metode DFS lebih cepat menemukan file dan pohon pencarian yang dihasilkan juga lebih ringkas dibandingkan dengan metode BFS. Sementara itu, ketika mencari file 'hall.rtf', metode BFS lebih cepat menemukan file dan pohon pencarian yang dihasilkan juga lebih ringkas dibandingkan dengan metode BFS.

2. Kasus Uji 2: Mencari seluruh keberadaan dari suatu file

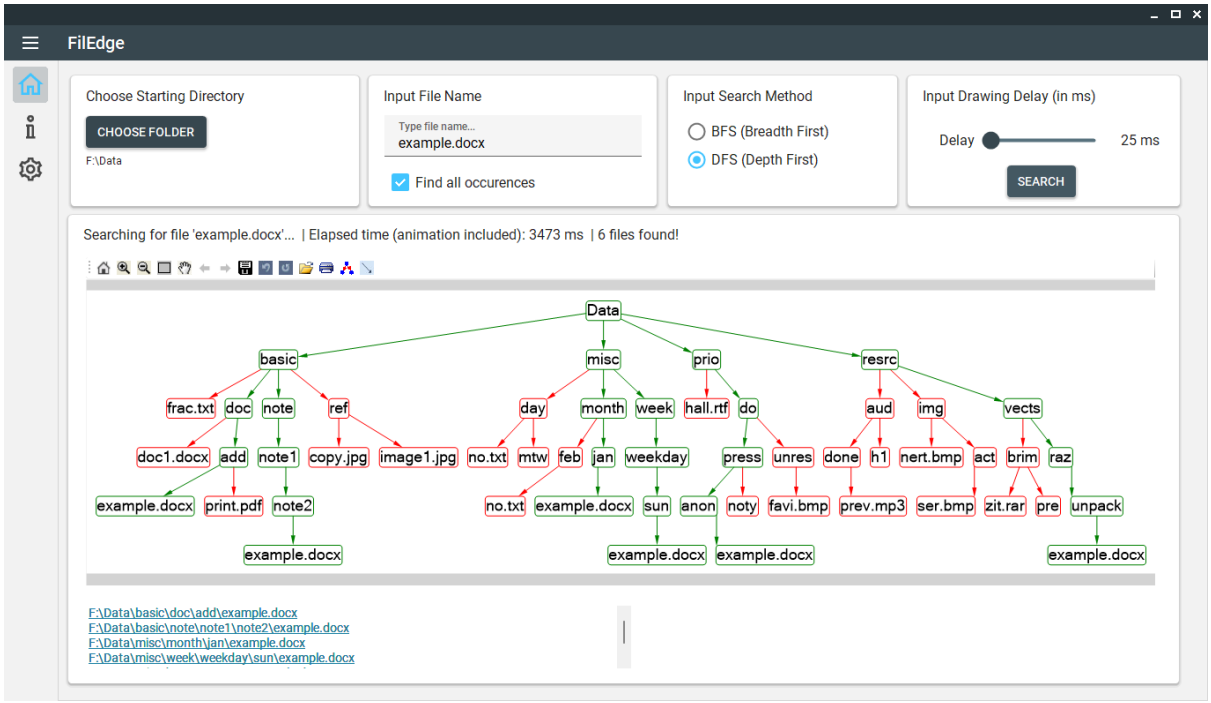
Akan dicari file bernama ‘example.docx’ dan pencarian akan dilakukan untuk semua keberadaaan file tersebut.

Menggunakan metode BFS, didapatkan hasil sebagai berikut.



Gambar 4.4.6 Pencarian file ‘example.docx’ dengan Metode BFS

Menggunakan metode DFS, didapatkan hasil sebagai berikut.



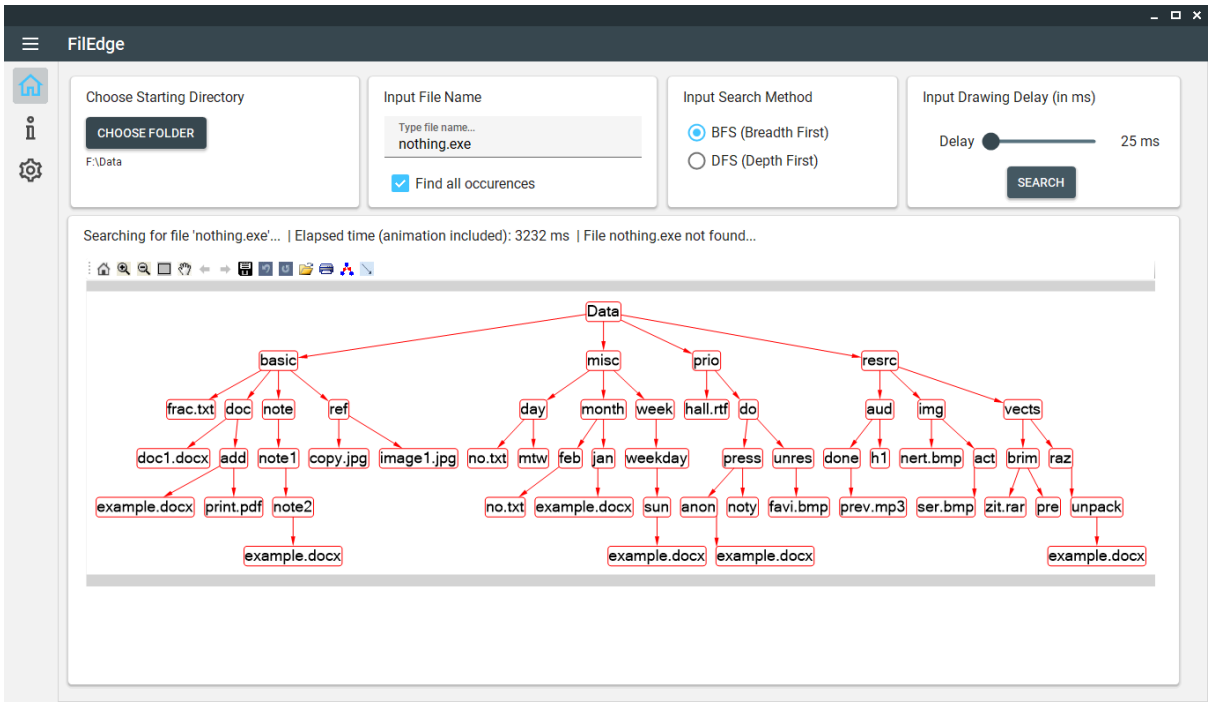
Gambar 4.4.7 Pencarian file 'example.docx' dengan Metode DFS

Berdasarkan hasil tersebut, didapatkan kesimpulan pencarian baik menggunakan metode BFS maupun DFS tampak sama dan lama waktu eksekusi juga tidak beda jauh.

3. Kasus Uji 3: Mencari file yang tidak ada pada direktori manapun

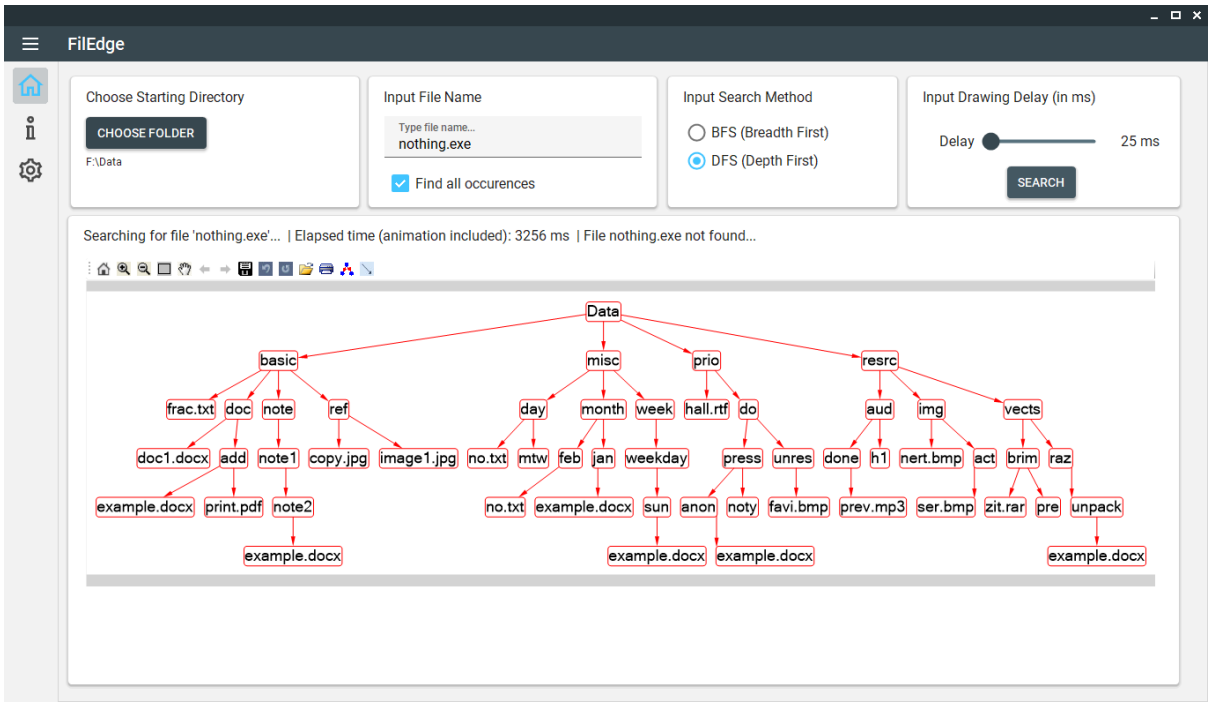
Akan dicari file bernama 'nothing.exe'

Menggunakan metode BFS, didapatkan hasil sebagai berikut.



Gambar 4.4.8 Pencarian file 'nothing.exe' dengan Metode BFS

Menggunakan metode DFS, didapatkan hasil sebagai berikut.



Gambar 4.4.9 Pencarian file 'nothing.exe' dengan Metode DFS

Berdasarkan hasil tersebut, didapatkan kesimpulan kedua pencarian baik menggunakan metode BFS maupun DFS tampak sama dan pencarian dilakukan hingga semua file ditelusuri namun tidak ada file yang ditemukan.

4.5 Analisis dari Desain Solusi Algoritma

Berdasarkan desain solusi BFS dan DFS yang diimplementasikan pada setiap pengujian yang dilakukan, didapatkan kesimpulan sebagai berikut.

1. Pencarian file ditemukan pertama kali yang lokasinya perlu pengaksesan direktori secara berlapis (jauh dari direktori awal) akan lebih baik menggunakan **metode DFS**. Hal ini dikarenakan pencarian dilakukan secara mendalam untuk setiap pengaksesan direktori hingga mencapai kedalaman dari pohon.
2. Pencarian file ditemukan pertama kali yang lokasinya hanya perlu sedikit pengaksesan direktori (dekat dari direktori awal) akan lebih baik menggunakan **metode BFS**. Hal ini dikarenakan pencarian dilakukan secara melebar. Pengecekan dilakukan dengan mengecek semua file pada satu level baru lanjut pada level berikutnya.
3. Pencarian semua keberadaan file menggunakan metode BFS dan DFS akan menghasilkan pohon pencarian yang sama dan waktu eksekusi yang tidak berbeda jauh.

Hal ini dikarenakan semua direktori dikunjungi dan yang membedakan hanya dari urutan pengunjungan saja.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan implementasi program dan eksperimen yang telah dilakukan, didapatkan kesimpulan sebagai berikut.

1. Telah dibangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari *file explorer* pada sistem operasi yang disebut dengan *Folder Crawling*.
2. *Folder Crawling* dapat dilakukan dengan memanfaatkan algoritma *Breadth First Search* (BFS) dan *Depth First Search* (DFS).
3. Dengan menggunakan aplikasi GUI sederhana yang telah dibangun, dapat ditelusuri folder-folder yang ada pada direktori untuk mendapatkan direktori yang diinginkan, serta dapat ditampilan pula visualisasi hasil dari pencarian folder tersebut dalam bentuk pohon.

5.2 Saran

Berdasarkan implementasi program dan eksperimen yang telah dilakukan, adapun saran dari penulis sebagai berikut.

1. Alokasikan waktu sebaik mungkin dalam pengerjaan tugas besar ini karena eksplorasi strategi yang baik akan memakan waktu yang banyak akibat *trial and error*.
2. Pelajari C# Desktop Application Development dengan membaca dokumentasi yang cukup banyak beredar di Internet.

LINK REPOSITORY DAN VIDEO

Repository GitHub

https://github.com/clumsyyyy/TubesStima2

Video Demo

https://www.youtube.com/watch?v=mXFE6QEy3v0

DAFTAR PUSTAKA

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf
(Diakses pada tanggal 20 Maret 2022)

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf
(Diakses pada tanggal 20 Maret 2022)