LAPORAN TUGAS BESAR 1
IF2211 STRATEGI ALGORITMA
PENGAPLIKASIAN ALGORITMA BFS DAN DFS DALAM IMPLEMENTASI *FOLDER CRAWLING*

oleh

Ahmad Romy Zahran	13520009
Firizky Ardiansyah	13520095
Muhammad Fahmi Irfan	13520152

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

2022

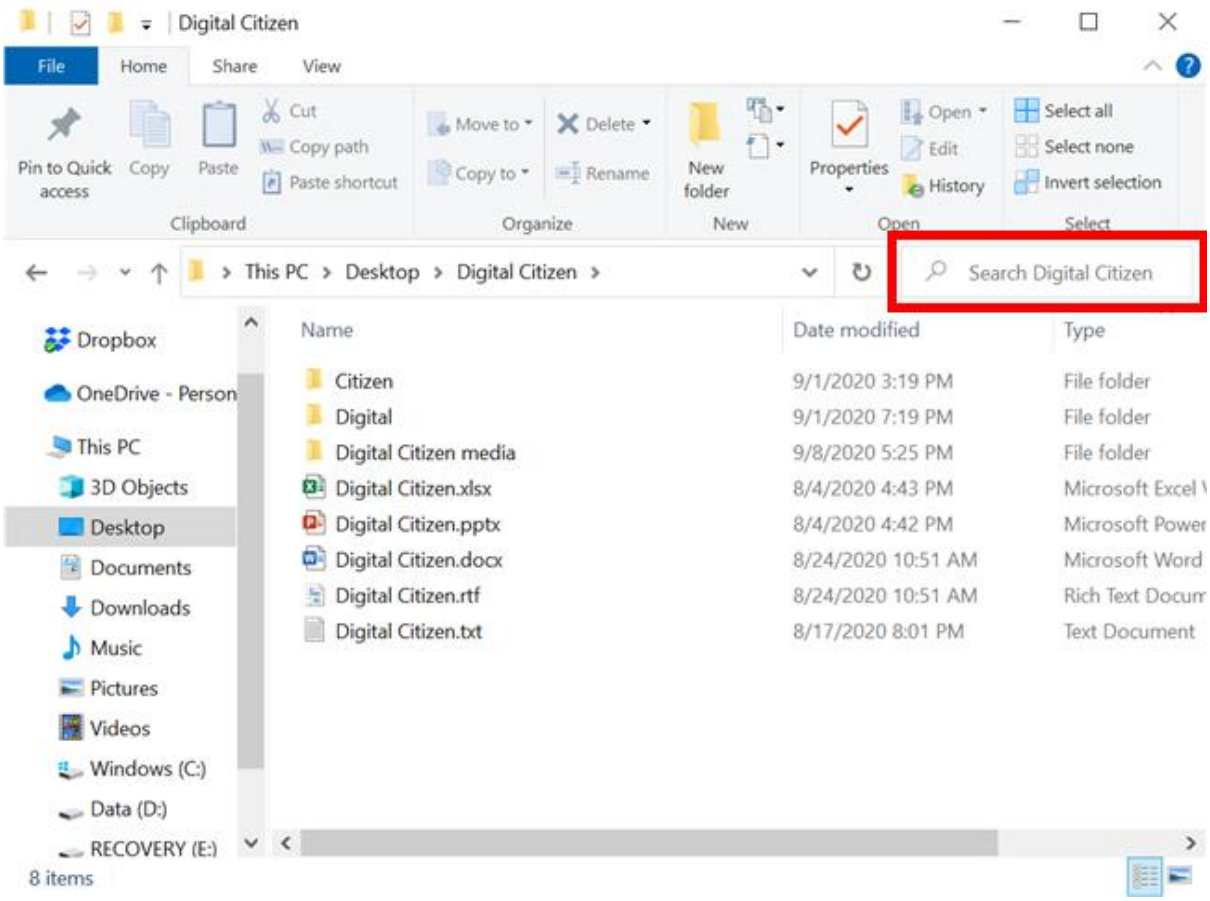
DAFTAR ISI

DAFTAR GAMBAR

BAB 1
DESKRIPSI MASALAH

1.1 Latar Belakang

Pada saat kita ingin mencari *file* spesifik yang tersimpan pada komputer kita, seringkali task tersebut membutuhkan waktu yang lama apabila kita melakukannya secara manual. Bukan saja harus membuka beberapa folder hingga dapat mencapai *directory* yang diinginkan, kita bahkan dapat lupa di mana kita meletakkan *file* tersebut. Sebagai akibatnya, kita harus membuka berbagai folder secara satu persatu hingga kita menemukan *file* yang diinginkan. Hal ini pastinya akan sangat memakan waktu dan energi.



Gambar 1.1.1 Fitur Search pada Windows 10 File Explorer[3]

Meskipun demikian, kita tidak perlu cemas dalam menghadapi persoalan tersebut sekarang. Pasalnya, hampir seluruh sistem operasi sudah menyediakan fitur *search* yang dapat digunakan untuk mencari *file* yang kita inginkan. Kita cukup memasukkan *query* atau kata kunci pada kotak pencarian, dan komputer akan mencarikan seluruh *file* pada suatu starting *directory* (hingga seluruh *children*-nya) yang berkorespondensi terhadap *query* yang kita masukkan.

Fitur ini diimplementasikan dengan teknik *folder crawling*, di mana mesin komputer akan mulai mencari *file* yang sesuai dengan *query* mulai dari *starting directory* hingga seluruh *children* dari *starting directory* tersebut sampai satu *file* pertama/seluruh *file* ditemukan atau tidak ada *file* yang ditemukan. Algoritma yang dapat dipilih untuk melakukan crawling tersebut pun dapat bermacammacam dan setiap algoritma akan memiliki teknik dan konsekuensinya sendiri. Oleh karena itu, penting agar komputer memilih algoritma yang tepat sehingga hasil yang diinginkan dapat ditemukan dalam waktu yang singkat.

1.2 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.

BAB 2
LANDASAN TEORI

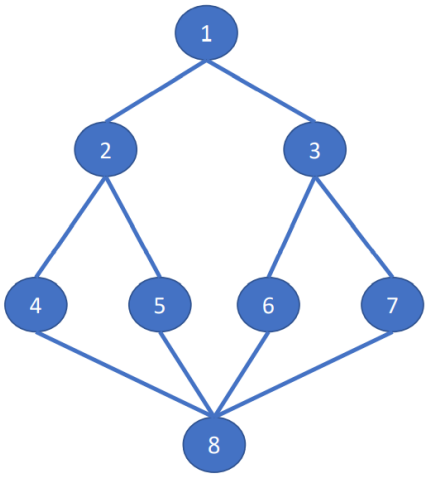
2.1 Dasar Teori

Graf adalah sebuah representasi objek-objek diskrit yang bisa menampilkan hubungan antara objek-objek tersebut. Graf merepresentasikan objek diskrit sebagai simpul-simpul graf dan hubungan antar objek-objek tersebut dihubungkan dengan sisi. Representasi graf memungkinkan analisis terhadap objek yang saling terhubung lebih mudah dimanipulasi. Karena itu, graf banyak digunakan dalam merepresentasikan banyak sekali permasalahan.

Salah satu jenis graf yang paling umum adalah pohon. Pohon adalah graf terhubung yang tidak mengandung sirkuit. Artinya, pohon tidak memiliki lintasan yang mulai dari sebuah simpul dan berakhir pada simpul yang sama. Perlu diperhatikan bahwa sebuah simpul pada graf pohon dapat dipilih sebagai *root* atau simpul akar. Simpul akar merupakan simpul yang dijadikan tolok ukur simpul lainnya dalam menentukan *parent* dan *child* masing-masing simpul. Simpul akar berada pada kedalaman terkecil, sehingga hanya memiliki *child* dan tidak memiliki *parent*. Adapun simpul daun adalah simpul dengan kedalaman terbesar, sehingga hanya memiliki *parent* dan tidak memiliki *child*. Simpul yang bukan merupakan simpul akar maupun simpul daun disebut simpul dalam.

Seringkali sebuah teknik dalam mengunjungi simpul-simpul dalam graf diperlukan dalam menyelesaikan permsalahan yang direpresentasikan dalam graf. Algoritma traversal graf adalah sebuah teknik sistematis dalam mengunjungi simpul-simpul yang ada dalam graf melalui lintasan yang tersedia. Algoritma traversal yang paling sering digunakan diantaranya pencarian melebar (*Breadth First Search, BFS*) dan pencarian mendalam (*Depth First Search, DFS*).

Algoritma BFS pada graf secara umum langkah pertama yang dilakukan adalah pemilihan simpul awal untuk diproses. Pada graf pohon, simpul yang dipilih merupakan simpul akar dari graf pohon. Proses traversal dimulai dengan mengunjungi semua simpul yang bertetangga dengan simpul yang dipilih. Simpul-simpul ini urutannya ditentukan berdasarkan prinsip *first in first out* (FIFO), sehingga struktur data dari algoritma BFS secara umum berupa struktur data antrean. Simpul yang bertetangga dengan simpul yang sedang diproses masuk ke dalam antrean, kemudian simpul selanjutnya yang akan diproses adalah simpul yang masuk ke dalam antrean paling awal. Simpul-simpul yang sudah dikunjungi akan diberi status, sehingga pada proses traversal selanjutnya, tidak ada simpul yang dikunjungi dua kali.



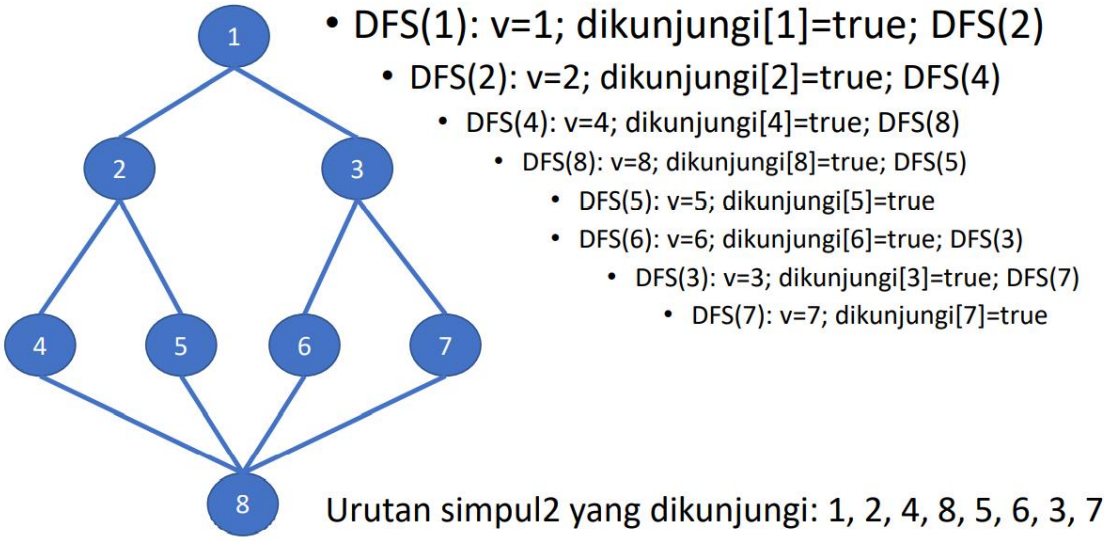
Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi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

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

Gambar 2.1.1 Ilustrasi Algortima BFS[1]

Karakteristik BFS yang menerapkan FIFO memungkinkan proses pengunjungan simpul didasarkan pada jarak sebuah simpul terhadap simpul akar. Semakin dekat sebuah simpul dengan simpul akar, semakin cepat simpul tersebut akan diproses. Sehingga urutan traversal algoritma ini melebar dari kedalaman terendah ke kedalaman lebih besar. Untuk itulah, algoritma ini disebut pencarian melebar.

Adapun algoritma traversal DFS, sama seperti BFS, langkah pertama yang dilakukan adalah pemilihan sebuah simpul untuk diproses paling awal. Berbeda dengan BFS, langkah selanjutnya adalah memilih salah satu simpul yang bertetangga dengan simpul yang sedang diproses dan simpul tersebut belum pernah diproses sebelumnya. Simpul baru ini dijadikan simpul ekspan baru, dengan kata lain, cara yang sama dilakukan pada simpul yang baru. Saat sebuah simpul tidak memiliki tetangga lain yang bisa diproses atau diekspan, pencarian di runut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul yang belum dikunjungi. Algoritma traversal berakhir ketika tidak ada lagi simpul yang belum dikunjungi.



Gambar 2.1.2 Ilustrasi Algoritma DFS[1]

Karakteristik mendasar dari DFS adalah urutan pencarian simpul dari mulai simpul tertentu kemudian menelusur simpul-simpul hingga ke bagian terdalam graf, kemudian dirunut balik ke simpul yang bisa ditelusur lebih dalam. Karenanya, algoritma DFS disebut juga sebagai pencarian mendalam.

2.2 Pengenalan Bahasa C# dan Kerangka Kerja yang Digunakan

Perangkat lunak yang digunakan dalam pengembangan aplikasi yang dibuat pada implementasi BFS dan DFS adalah visual studio. Visual studio merupakan perangkat lunak yang digunakan untuk mengembangkan aplikasi, salah satunya adalah pengembangan aplikasi *desktop*.

Bahasa C# (baca: C-sharp) merupakan salah satu bahasa pemrograman yang disediakan visual studio dalam mengembangkan aplikasi *desktop*. Visual studio menyediakan kerangka kerja yang memungkinkan pengembangan aplikasi menggunakan bahasa C# jauh lebih mudah. Salah satu kerangka kerja yang disediakan visual studio adalah WinForms. Winforms merupakan kerangka kerja *user interface* yang disediakan visual studio yang memungkinkan pengembang untuk menggunakan banyak *tools* dalam pengembangan aplikasi terutama *user interface*. WinForms diantaranya menyediakan *tools* untuk memanipulasi objek-objek seperti *controls*, *graphic*, dan fitur lainnya. Kerangka kerja WinForms disediakan oleh sebuah platform atau perangkat lunak kerangka kerja yang disebut sebagai .NET Framework.

Gambar 2.2.1 Ilustrasi Penggunaan Visual Studio dan WinForms

Kaitannya dengan pengembangan aplikasi, bahasa C# secara umum merupakan bahasa pemrograman yang berbasis *object-oriented programming*, sehingga pengembangan menggunakan bahasa ini cukup mudah untuk diintegrasikan dengan antarmuka.

Antarmuka yang disediakan WinForms, selain itu, memungkinkan pengembang untuk melakukan design kerangka aplikasi hanya dengan teknik *drag and drop* dengan kontrol yang beragam. Di samping itu, WinForms juga melakukan proses *auto generate* entitas-entitas grafis yang dipakai pengembang. Pengembang juga dapat memberikan kontrol secara fleksibel pada elemen-elemen yang digunakan.

.NET Framework, sebagai platform yang menjalankan kerangka WinForms, juga memberikan banyak sekali pustaka yang dapat digunakan. Salah satunya adalah pustaka *auto generate* graf yang disebut sebagai MSAGL (Microsft Automatic Graph Layout). Pustaka ini, bersama-sama dengan WinForms bisa digunakan dalam pengembangan aplikasi yang membutuhkan visualisasi dari sebuah graf.

BAB 3
ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Secara umum persoalan *folder crawling* diselesaikan dengan merepresentasikan folder dan file dan keterhubungannya dengan graf. Graf yang terbentuk pada akhirnya akan terdiri dari simpul yang berupa direktori dan sisi yang merepresentasikan bahwa sebuah folder berada di dalam folder yang lain. Dengan kata lain, simpul yang saling bertetangga didefinisikan sebagai simpul-simpul yang salah berada di dalam simpul yang lain.

Dapat diperhatikan bahwa tidak mungkin terdapat sirkuit pada graf representasi struktur sebuah direktori, sehingga representasi graf yang dihasilkan adalah representasi graf pohon. Simpul akar akan dipilih melalui input yang diberikan pengguna, graf akan terbentuk dari simpul akar ke simpul daun (yaitu folder yang tidak memiliki file atau sebuah file).

Folder Crawling meminta untuk penelusuran simpul yang memiliki nama tertentu. Sehingga atribut simpul pada graf representasi adalah nama dari folder/file yang direpresentasikan simpul tersebut dan cara menelusurinya adalah dengan melakukan proses traversal simpul-simpul pada struktur direktori.

Untuk mencari sebuah file/folder dengan nama tertentu, kita cukup melakukan traversal hingga simpul yang diproses memiliki atribut yang sesuai dengan nama yang file/folder yang dicari. Setiap langkah traversal, perlu disimpan status lintasan yang sudah dibentuk dari simpul akar hingga simpul yang sedang diproses. Ketika atribut yang sesuai dicapai, lintasan yang berupa status pencarian dikembalikan untuk diberikan kepada pengguna.

Adapun untuk mencari semua file/folder dengan atribut tertentu, dilakukan dengan cara yang sama, tetapi traversal graf baru dihentikan ketika semua simpul selesai diperiksa. Lintasan disimpan pada sebuah array dan ketika proses traversal berakhir, semua lintasan yang merupakan solusi dikembalikan untuk diberikan kepada pengguna.

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

Pertama, input folder dijadikan root dan dibuat graf pohon yang menyatakan struktur direktori. Pada graf pohon yang terbuat, folder dapat memiliki child berupa folder dan/atau file sedangkan file tidak dapat memiliki child. Setiap simpul memiliki informasi nama folder/file yang tidak harus unik namun harus unik dalam parent yang sama sehingga setiap simpul

memiliki path yang unik secara keseluruhan. Graf dapat direpresentasikan dalam adjacency list antara path yang satu dengan yang lainnya.

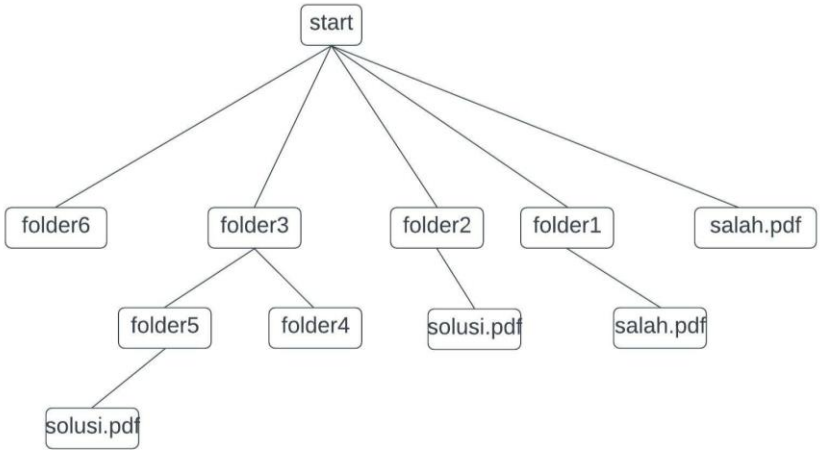
Penelusuran folder crawling dilakukan dengan pengunjungan setiap simpul menggunakan algoritma BFS/DFS dengan root sebagai simpul awal. Untuk query nama file (dan nama folder kalau mau) sama saja dengan melakukan string matching nama file (/folder) pada setiap simpul. Pengunjungan simpul dihentikan begitu ditemukan file (/folder) pertama yang cocok bila dipilih opsi pencarian 1 file(/folder). Bila dipilih opsi mencari seluruh file (/folder), pengunjungan simpul baru dihentikan setelah semua simpul dikunjungi. Setiap solusi dimasukkan pada list path solusi.

Algoritma BFS beserta persoalan visualisasi yang lebih rinci adalah sebagai berikut. Dibuat queue simpul yang ingin diekspansi (diimplementasikan dengan queue of string) dengan isi awal simpul root. Simpul yang ingin diekspansi diambil dari depan queue, simpul ekspansi ini diwarnai biru tua dan proses dijeda sebentar. Bila simpul ekspansi cocok dalam string matching, path dimasukkan ke path solusi dan pencarian dilanjutkan bila dipilih opsi mencari semua. Bila simpul ekspansi berupa folder, satu per satu Child dari simpul ekspansi dimasukkan ke queue dan diwarnai biru muda yang menandakan simpul sedang waiting di queue dan simpul ekspansi diwarnai hijau muda artinya simpul termasuk rute solusi atau masih mungkin menjadi rute solusi (karena punya child yang belum diproses). Bila simpul ekspansi berupa file, tidak ada child yang dapat diekspansi dan rute tersebut dapat dimatikan dengan mewarnai merah simpul ekspansi hingga ancestor (leluhur) yang tidak memiliki child yang belum diproses. Pewarnaan merah tersebut membutuhkan banyak child yang belum diproses, hal ini diinisialisasi saat simpul terpilih untuk diekspansi dan diperbarui setiap child selesai diproses.

Algoritma DFS beserta persoalan visualisasi yang lebih rinci adalah sebagai berikut. Hijau tua menandakan simpul sedang diproses, hijau muda menandakan simpul masuk rute solusi atau child-nya sedang diproses, merah menandakan simpul bukan solusi dan sudah diproses, terakhir putih menandakan simpul belum diproses. Kunjungi simpul root warnai dengan hijau tua dan dijeda, lakukan string matching dan penghentian sesuai opsi. Bila lanjut, kunjungi satu child bila ada dan lakukan DFS di sana. Proses pengunjungan dari parent ke child ini dapat disebut advance, parent diwarnai hijau muda dan child diwarnai hijau tua lalu dijeda. Ketika suatu child selesai diproses, dilakukan backtrack ke parent. Bila child rute solusi, child diwarnai hijau muda dan parent diwarnai hijau tua. Bila child bukan rute solusi, child diwarnai merah dan parent diwarnai hijau tua.

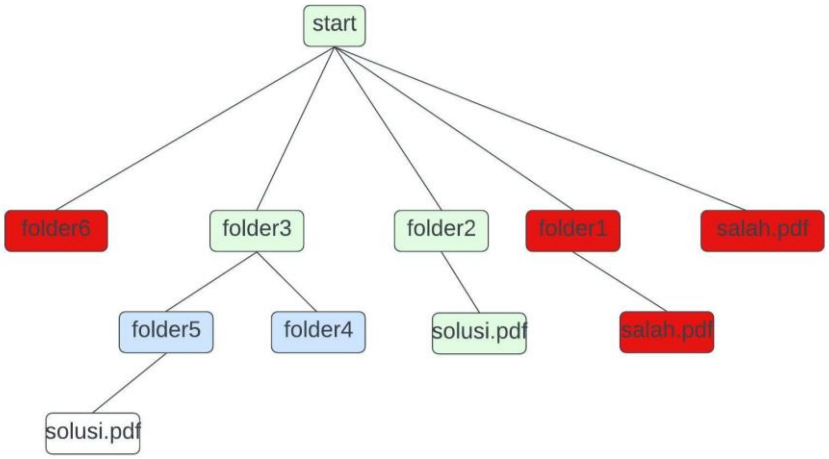
3.3 Contoh Ilustrasi Kasus Lain yang Berbeda dengan Contoh pada Spesifikasi Tugas

Misalkan dipilih folder C:\start sebagai input folder dengan struktur direktori sebagai berikut dan akan dicari query file solusi.pdf.



Gambar 3.3.1 Ilustrasi Struktur Direktori

Penelusuran folder crawling dengan algoritma BFS dan opsi pencarian 1 file menghasilkan visualisasi sebagai berikut.



Gambar 3.3.2 Contoh Visualisasi dengan Algoritma BFS

List path:

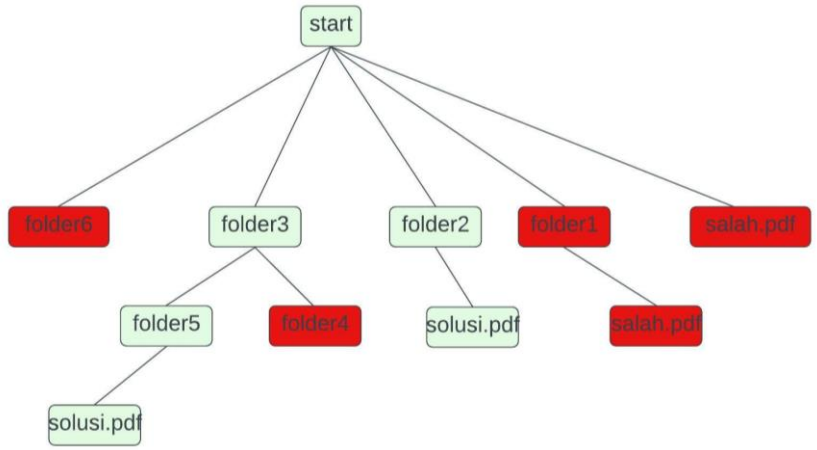
1. C:\start\folder2\solusi.pdf

Tabel 3.3.1 Contoh Proses Algoritma BFS

Simpul ekspansi	Antrian sebelum ekspansi	Pewarnaan
[start]	[start]	salah.pdf, folder1, folder2, folder3, folder6 diwarnai biru

		muda dan dimasukkan antrian. Start diwarnai hijau muda.
[start\salah.pdf]	[start\salah.pdf], [start\folder1], [start\folder2], [start\folder3], [start\folder6]	Salah.pdf diwarnai merah.
[start\folder1]	[start\folder1], [start\folder2], [start\folder3], [start\folder6]	salah.pdf diwarnai biru muda dan dimasukkan antrian. folder1 diwarnai hijau muda.
[start\folder2]	[start\folder2], [start\folder3], [start\folder6], [start\folder1\salah.pdf]	Solusi.pdf diwarnai biru muda dan dimasukkan antrian. folder2 diwarnai hijau muda.
[start\folder3]	[start\folder3], [start\folder6], [start\folder1\salah.pdf], [start\folder2\solusi.pdf]	Folder4, folder5 diwarnai biru muda dan dimasukkan antrian. folder 3 diwarnai hijau muda.
[start\folder6]	[start\folder6], [start\folder1\salah.pdf], [start\folder2\solusi.pdf], [start\folder3\folder4], [start\folder3\folder5]	folder6 diwarnai merah.
[start\folder1\salah.pdf]	[start\folder1\salah.pdf], [start\folder2\solusi.pdf], [start\folder3\folder4], [start\folder3\folder5]	salah.pdf dan folder1 diwarnai merah.
[start\folder2\solusi.pdf]	[start\folder2\solusi.pdf], [start\folder3\folder4], [start\folder3\folder5]	solusi.pdf diwarnai hijau muda. Tambahkan path solusi. Pencarian dihentikan.

Selanjutnya bila dilakukan penelusuran folder crawling dengan algoritma DFS dan opsi pencarian semua file menghasilkan visualisasi sebagai berikut.



Gambar 3.3.3 Contoh Visualisasi dengan Algoritma DFS

List path:

1. C:\start\folder2\solusi.pdf

2. C:\start\folder3\solusi.pdf

Path pencarian DFS adalah sebagai berikut: start → salah.pdf → start → folder1 → salah.pdf → folder1 → start → folder2 → solusi.pdf (tambah list path) → folder2 → start → folder3 → folder4 → folder3 → folder5 → solusi.pdf (tambah list path) → folder5 → folder3 → start → folder6 → start.

BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

Berikut adalah pseudokode dari program ini.

```
Procedure DFS (input tree : Graph, input/output path : array of string, input pathName : map string to string,  input to : string, input all : boolean)
  graph <- tree
  curPath <- pathName["0"]
  root <- "0"
  if (not processDFS(root, to, graph, curPath, all)) then
    processGraph(graph) // visualisasi graf
    path.Clear()
    path.Add("Tidak ditemukan")
  else
    processGraph(graph) // visualisasi graf


Function processDFS(input pathName : map string to string, input p : string, input to : string, input g : Graph, input path : string, input all : boolean) -> boolean
  tmp <- path
  ret <- false
  if (pathName[p] = to) then
    path.Add(path)
    ret <- true
  foreach(e in FindNode(p).OutEdges)
    string q <- e.Target
    path <- path + "\\"
```

```
            path <- path + pathName[q]
            advance(g,e)
            if (processDFS(pathName, p, to, g, path, all) then
              ret <- true
              if(not all) then
                -> true
            else
              path <- tmp
          -> ret
```

Procedure BFS (input/output path : array of string, input tree : Graph, input pathName : map from string to string, input idPath : map from string to string, input to : string, input all : boolean)

```
          path.Clear()
          graph <- tree
          idLifeChild <- map from string to int
          rootId <- "0"
          qu <- Queue of string
          qu.enqueue(rootId)
          while(qu.Count > 0) then
            u <- qu.Dequeue()
            sol <- false
            if (pathName[u] = to) then
              sol <- true
              path.Add(idPath[u])
            if (not all) break
          if (graph.FindNode(u).OutEdges.Count() > 0) then
```

```
        idLifeChild[u] <- graph.FindNode(u).OutEdges.Count()
        foreach e in graph.FindNode(u).OutEdges
          v <- e.Target
          qu.Enqueue(v)
      else
        if not sol then
          while (u != rootId and pathName[u] != to) then
            e <- graph.FindNode(u).InEdges.First()
            u <- e.Source
            idLifeChild[u] <- idLifeChild[u] - 1
            if (idLifeChild[u] != 0 ) then break
          processGraph(graph) // visualisasi graf
```

Gambar 4.1.1 Algoritma Utama

4.2 Penjelasan Struktur Data yang Digunakan

Berikut atribut dan method pada class Graph.

- Atribut
  - tree: Microsoft.Msagl.Drawing.Graph. Struktur data graf yang dapat langsung ditampilkan. Atribut dan method yang digunakan sebagai berikut.
    - method FindNode() untuk mencari simpul dari id
    - atribut FillColor untuk mewarnai simpul
    - atribut LabelText untuk label dari simpul
    - method AddNode() untuk menambah simpul.
    - method AddEdge() untuk menambah *edge*.
    - atribut OutEdges() untuk kumpulan *out edge* yang dimiliki suatu simpul.
    - atribut InEdges() untuk kumpulan *in edge* yang dimiliki suatu simpul.
  - pathId, pathName, idPath: Dictionary<string, string>. pathId memetakan *path* ke id pada tree. pathName memetakan id ke nama folder/file sekaligus LabelText dari simpul. idPath memetakan id ke *path*.
  - path: List<string>. Untuk menyimpan *path* solusi.

- form: Form1. Form yang akan diperbarui visualisasinya. Digunakan method processGraph() untuk memperbarui visualisasi pada form.

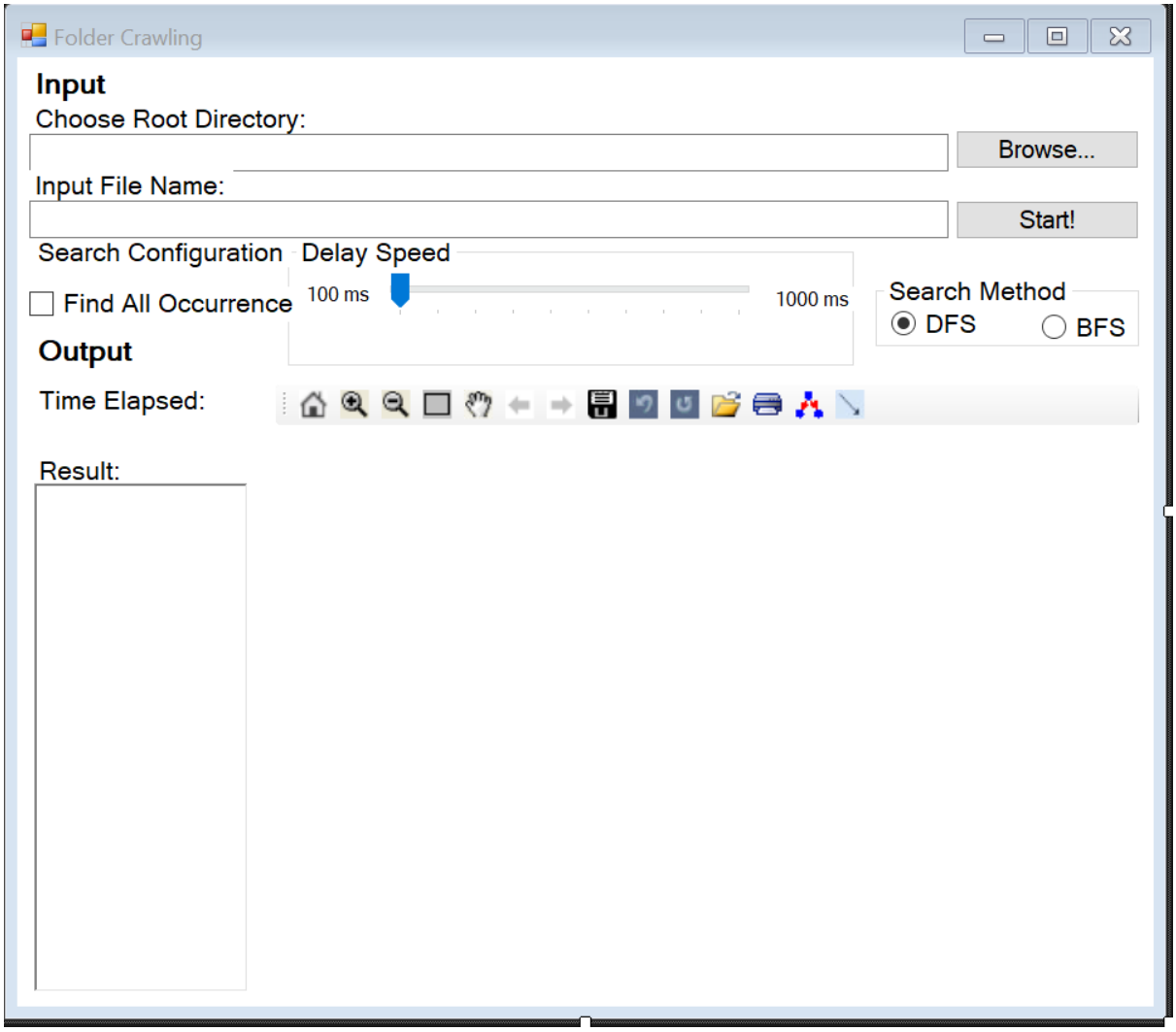
- Method
  - Graph(Form1 form, string root, Dictionary<string, List<string>> adjList): membuat Graph dari *adjacency list* adjList dan *root* root. Atribut tree diisi sesuai adjList dan root. Objek Graph dihubungkan dengan form sebagai form visualisasinya.
  - getTree(): getter tree.
  - getPath(): getter path.
  - DFS(string to, bool all): implementasi DFS untuk mencari file/folder to dengan opsi pencarian semua bila all bernilai true.
  - processDFS(string p, string to, Microsoft.Msagl.Drawing.Graph g, string path, bool all): fungsi rekursif yang digunakan di DFS() juga.
  - advance(g,e), backtrack(g,e,solution), finish(g,finalNode,solution): helper untuk pewarnaan, visualisasi dan jeda.
  - BFS(string to, bool all): implementasi BFS untuk mencari file/folder to dengan opsi pencarian semua bila all bernilai true. Digunakan Queue<string> untuk antrian id dengan method Enqueue() dan Dequeue(). Digunakan pula Dictionary<string, int> idLifeChild untuk menyimpan banyak *child* yang belum selesai diproses dari simpul id.
  - setNodeColor(g, node, color): helper untuk pewarnaan simpul, visualisasi, dan jeda. Digunakan System.Drawing.Color dan method fromName untuk mendapat Color dari string color.

Program dapat dijalankan pada windows dengan menjalankan Folder Crawling.exe di bin/debug. Program dibuat dengan teknologi dan pustaka:

- Bahasa C#
- .NET Framework
- Visual Studio
- MSAGL

4.3 Penjelasan Tata Cara Penggunaan Program

Program dapat dijalankan pada windows dengan menjalankan Folder Crawling.exe di bin/debug. Ketika dijalankan akan muncul antarmuka program sebagai berikut.

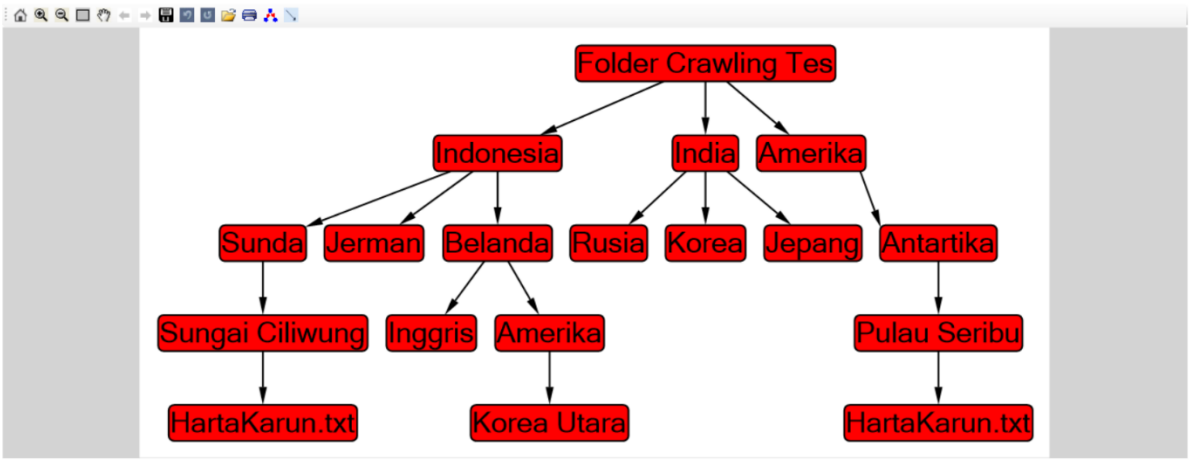


Gambar 4.3.1 Antarmuka Program

Pengguna dapat mengisi input folder dengan mengetik pada textbox ataupun mengklik Browse untuk membuka input folder. Pengguna juga dapat memasukkan query nama file/folder pada textbox. Checkbox Find All occurence dicentang pengguna bila ingin mencari semua file/folder yang cocok. Groupbox DFS/BFS untuk menentukan jenis algoritma penelusuran. Groupbox Delay Speed untuk waktu jeda saat visualisasi. Setelah pilihan sudah sesuai, pengguna dapat menekan tombol search untuk memulai pencarian. Gviewer dari MSAGL akan menampilkan visualisasi progress pencarian, label time elapsed menampilkan waktu yang dibutuhkan untuk pencarian, Rich Text Box Result akan menampilkan daftar path file/folder solusi dan bertindak sebagai hyperlink yang dapat dibuka.

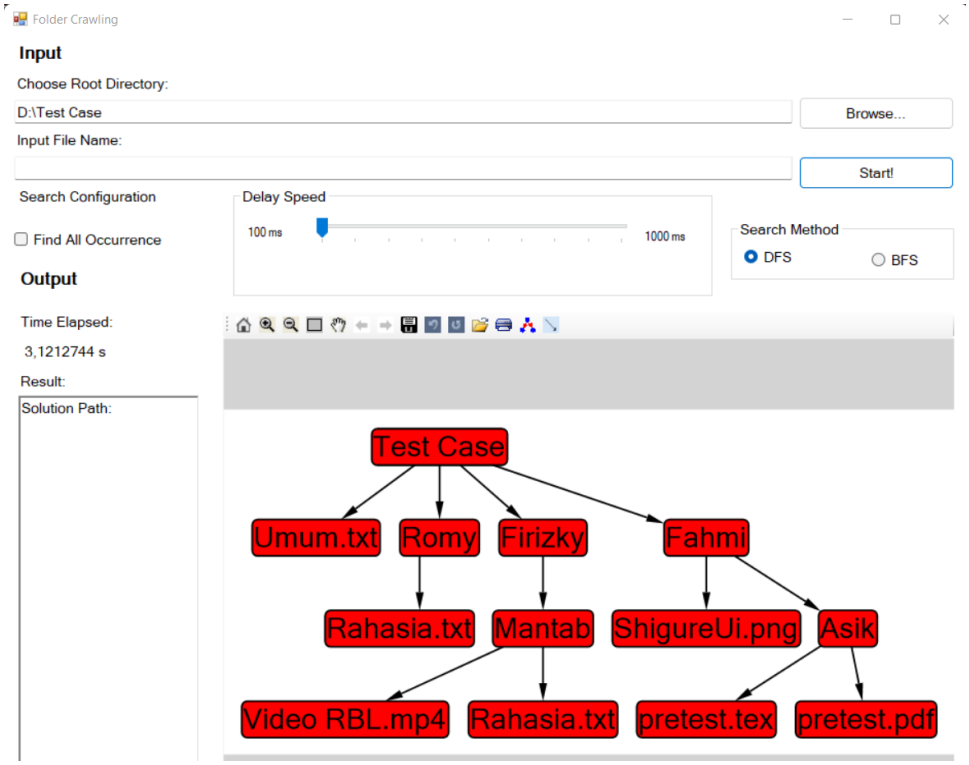
4.4 Hasil Pengujian

Untuk kasus uji pertama dan kedua, struktur folder yang digunakan adalah sebagai berikut.



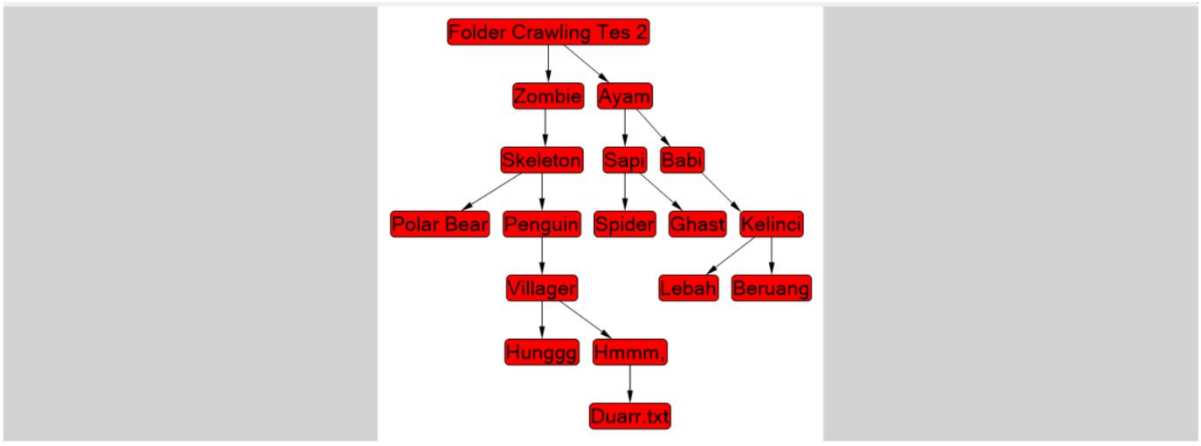
Gambar 4.4.1 Struktur Direktori Kasus Uji Pertama dan Kedua

Untuk kasus uji ketiga, struktur direktori yang digunakan adalah sebagai berikut.



Gambar 4.4.2 Struktur Direktori untuk Kasus Uji Ketiga

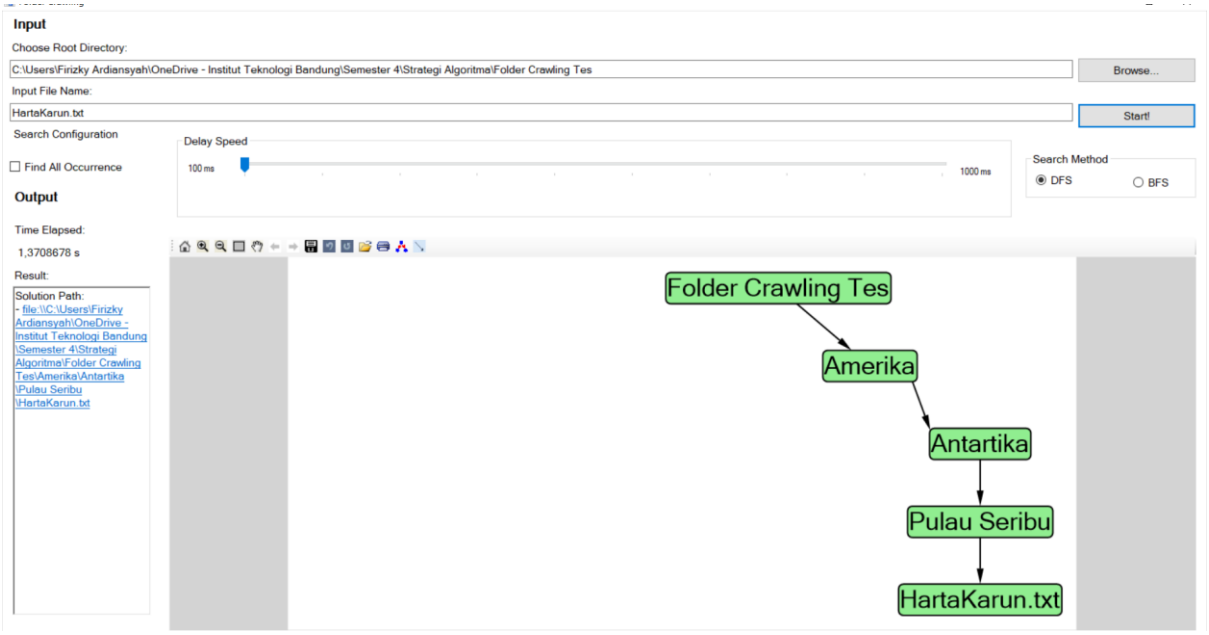
Adapun untuk kasus uji keempat dan kelima, struktur direktori yang digunakan adalah sebagai berikut.



Gambar 4.4.3 Struktur Direktori Kasus Uji Keempat dan Kelima

4.4.1 Pengujian Algoritma DFS Satu File

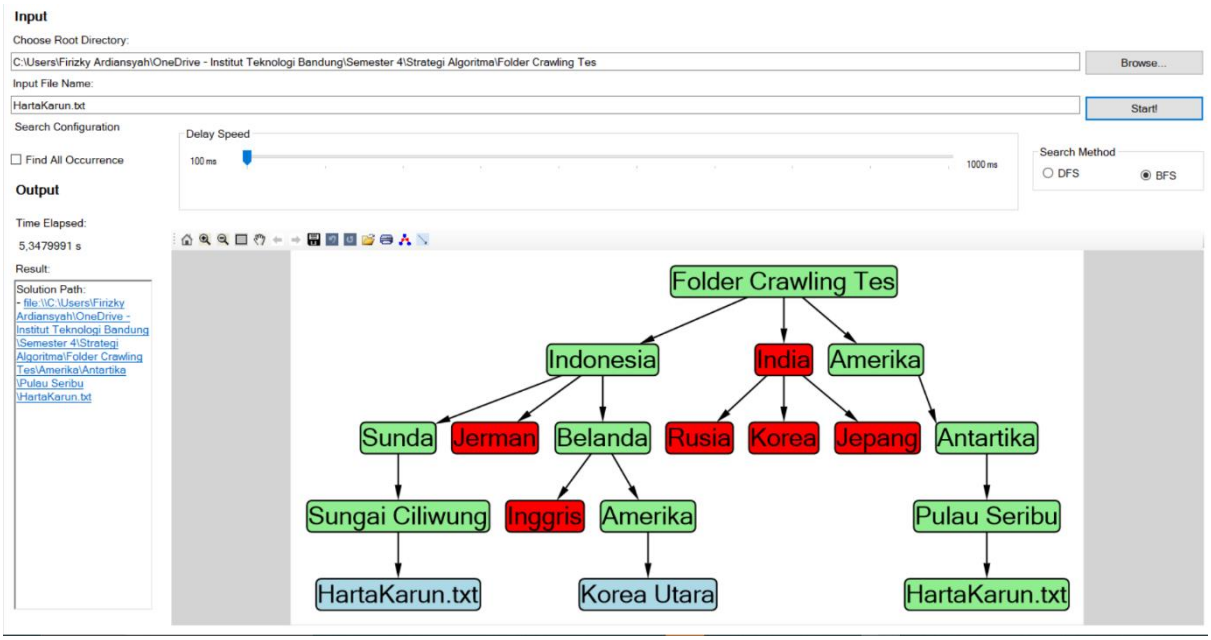
File yang dicari adalah HartaKarun.txt, berikut adalah hasil pencariannya menggunakan algoritma DFS.



Gambar 4.4.4 Kasus Uji 1

4.4.2 Pengujian Algortima BFS Satu File

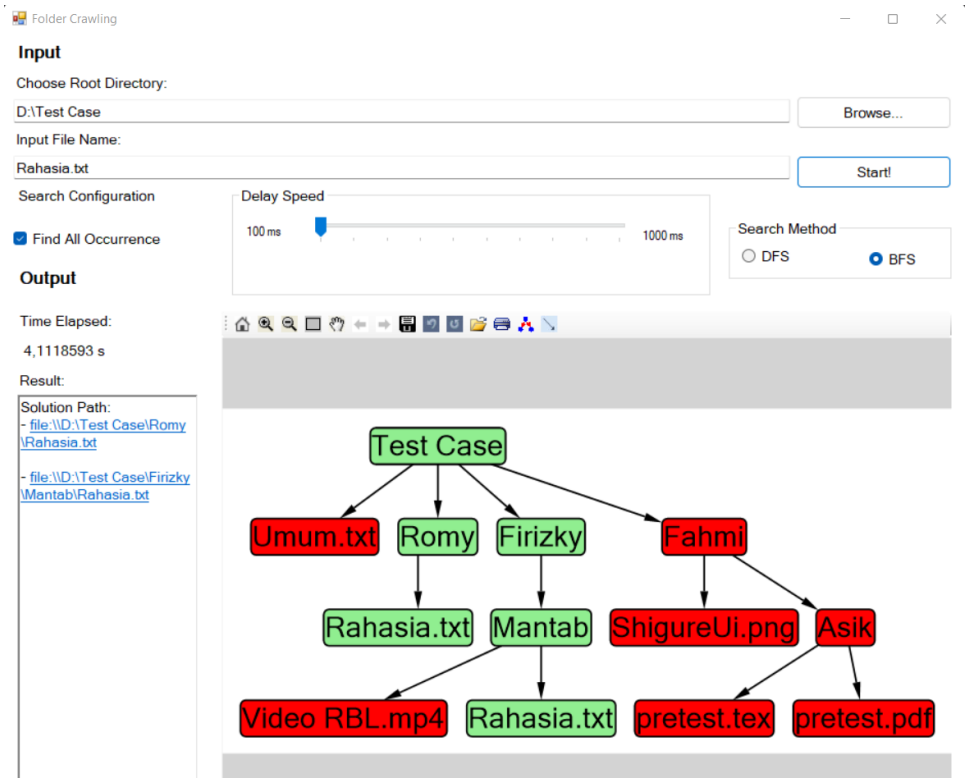
Menggunakan kasus uji yang sama dalam pencarian HartaKarun.txt pada kasus uji sebelumnya, berikut adalah hasil traversal algoritma BFS pada kasus uji tersebut.



Gambar 4.4.5 Kasus Uji 2

4.4.3 Pengujian Algortima BFS Banyak File

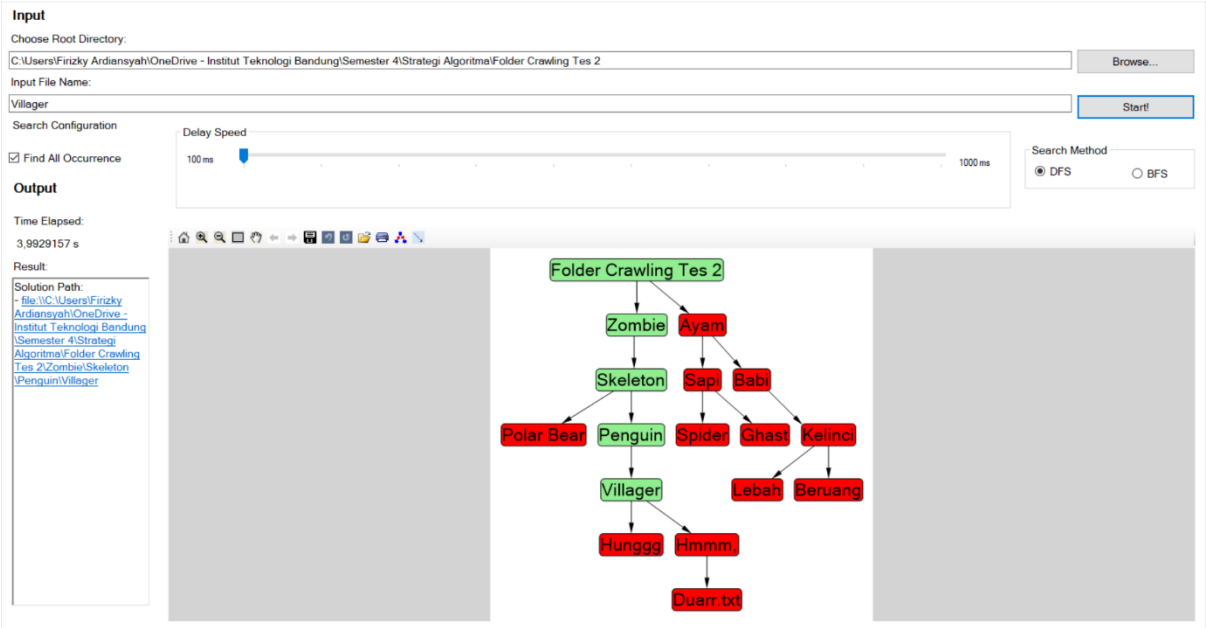
File yang akan dicari ialah file Rahasia.txt di dalam direktori Test Case. Terdapat lebih dari satu file bernama Rahasia.txt, sehingga jika Find All Occurrence dipilih, program akan menghasilkan luaran berikut.



Gambar 4.4.6 Kasus Uji 3

4.4.4 Pengujian Algoritma DFS pada Folder

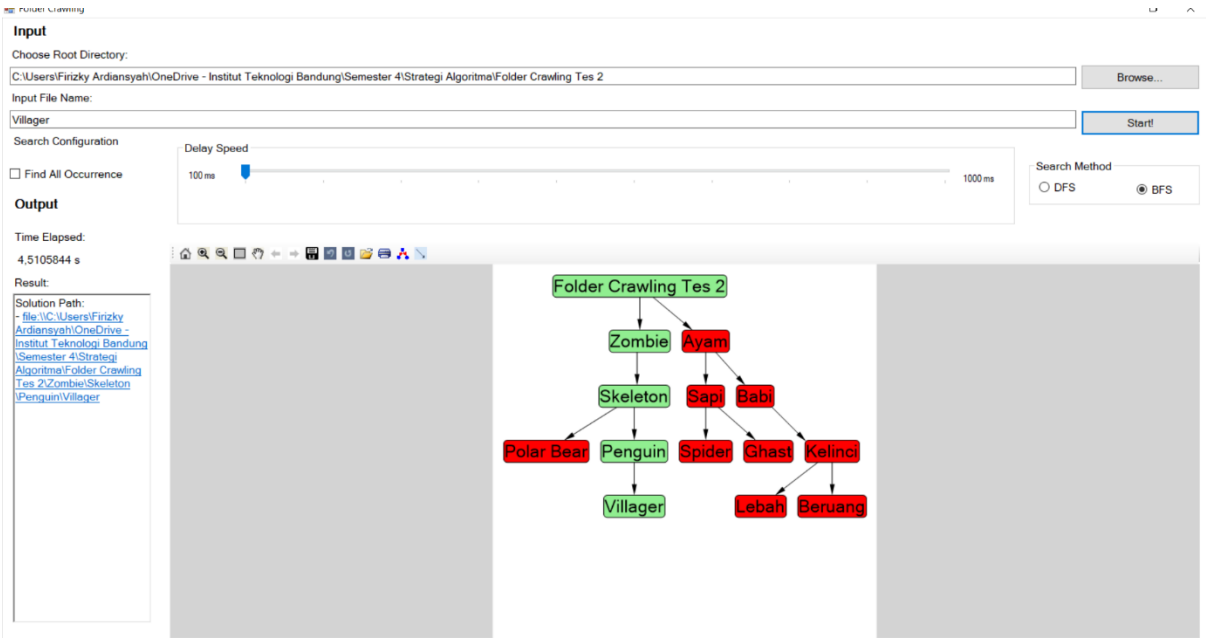
Pada kasus uji ini, folder yang dicari adalah folder Villager. Konfigurasi pencarian ditetapkan sebagai pencarian semua kemunculan. Berikut adalah hasil pengujiannya.



Gambar 4.4.7 Kasus Uji 4

4.4.5 Pengujian Algoritma BFS pada Folder

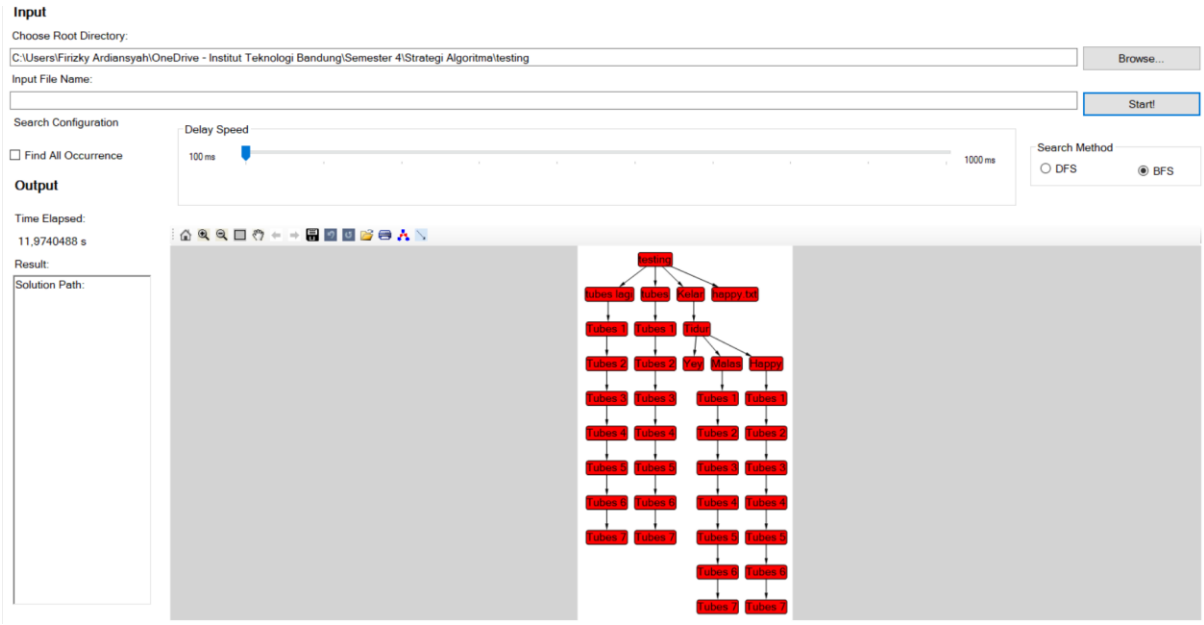
Menggunakan struktur yang sama seperti sebelumnya, luaran hasil algoritma BFS adalah sebagai berikut. Konfigurasi pencarian diset hanya satu folder saja yang dicari (agar terlihat perbedaannya).



Gambar 4.4.8 Kasus Uji 5

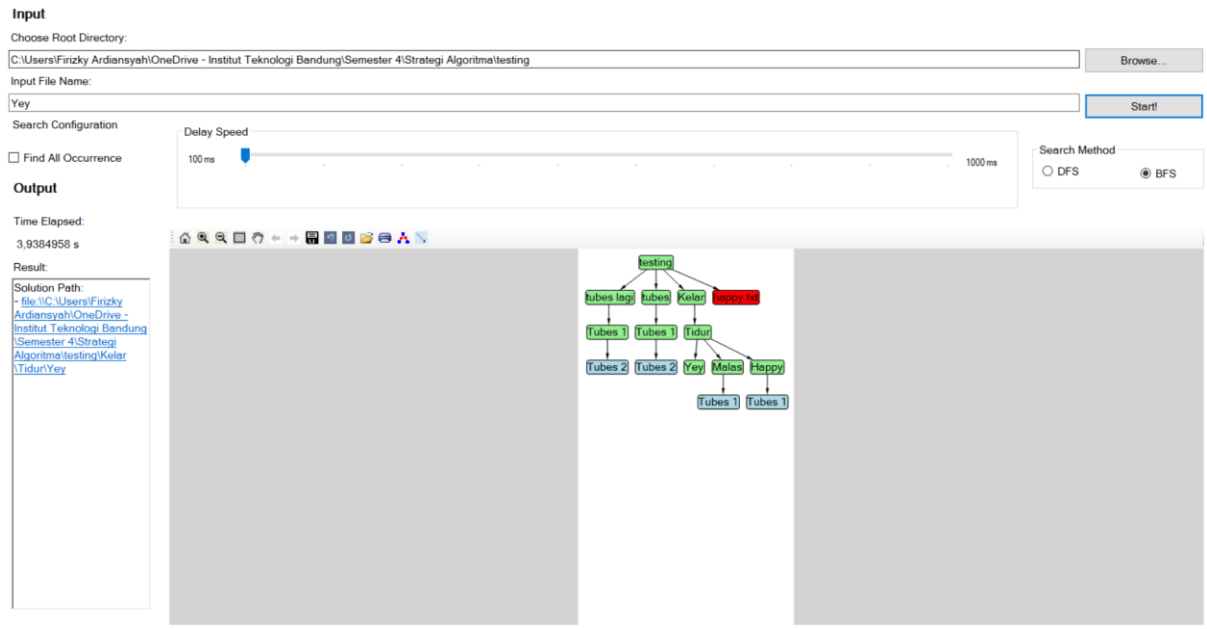
4.5 Analisis Desain Solusi

Penggunaan algortima BFS dan DFS keduanya memiliki kekurangan dan kelebihan masing-masing dan tidak secara absolut bisa ditentukan yang mana yang lebih baik. Contohnya adalah ketika meninjau kasus berikut.



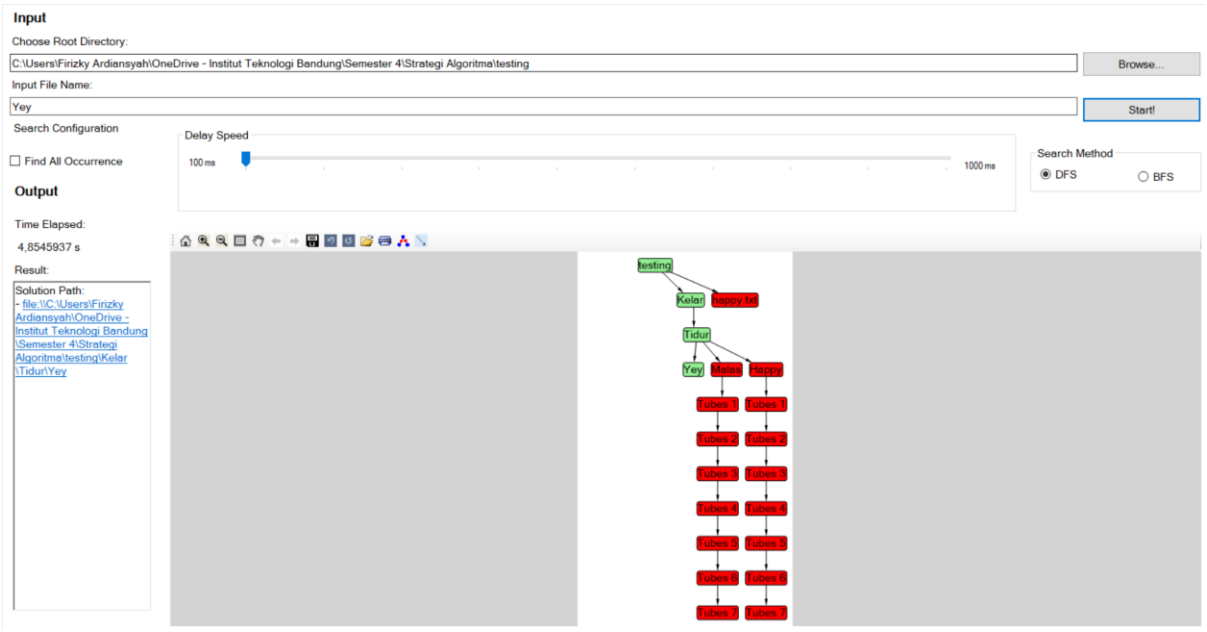
Gambar 4.5.1 Contoh Kasus

Jika menggunakan BFS untuk mencari folder Cepat, traversal akan dilakukan lebih cepat dibandingkan DFS, berikut adalah hasil pencariannya.



Gambar 4.5.2 Pengujian Algoritma BFS pada Analisis Kasus Pertama

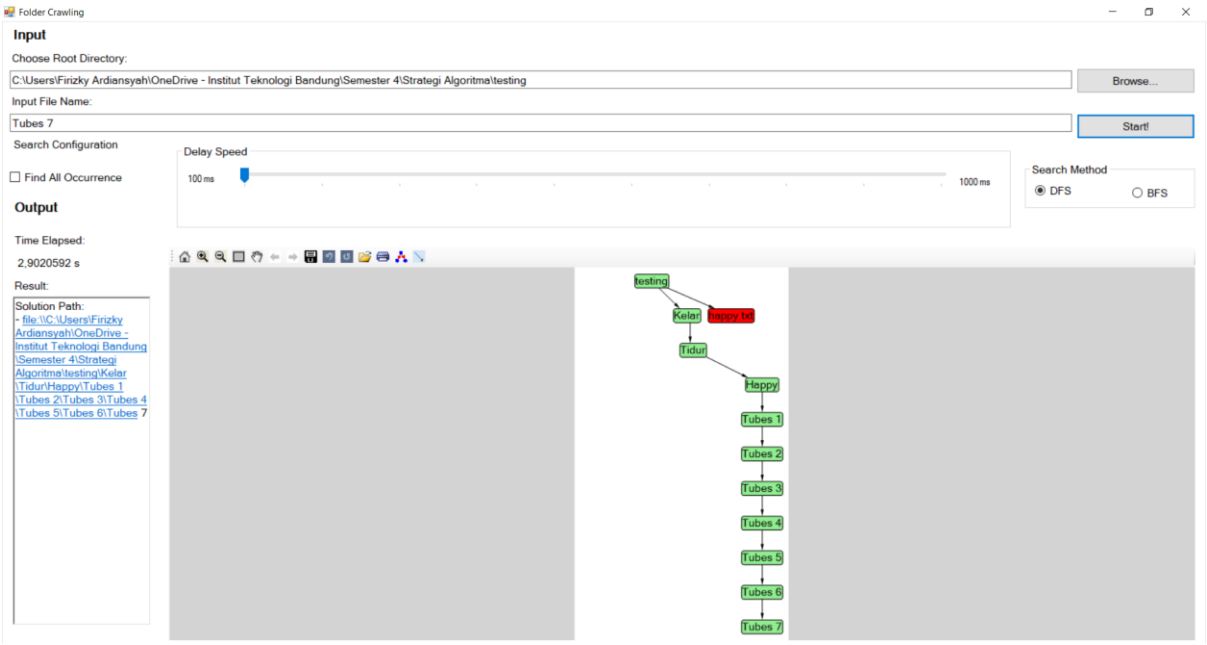
Sedangkan jika menggunakan DFS, hasil traversal yang diperoleh adalah sebagai berikut.



Gambar 4.5.3 Pengujian Algoritma DFS pada Analisis Kasus Pertama

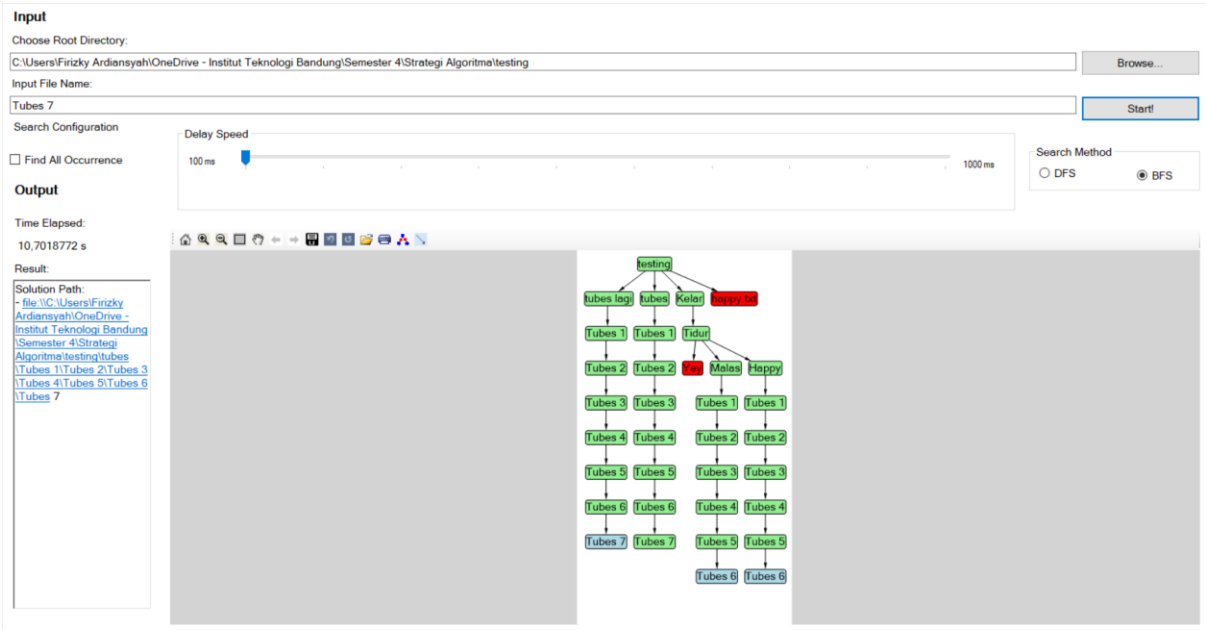
Meskipun perbedaan waktu yang dibutuhkan sedikit, langkah yang ditraversal pada algoritma BFS lebih sedikit dibandingkan algoritma BFS. Pada kasus lain, misalnya ketika ingin mencari folder Tubes 7 dengan kasus yang sama, menggunakan DFS akan lebih cepat.

Berikut adalah penggunaan algoritma DFS dalam mencari folder Tubes 7



Gambar 4.5.4 Pengujian Algoritma DFS pada Analisis Kasus Kedua

Adapun dengan algoritma BFS adalah sebagai berikut.



Gambar 4.5.5 Pengujian Algoritma BFS pada Analisis Kasus Kedua

Terlihat perbedaan besar antara kedua algoritma. Algoritma DFS dalam kasus ini lebih efisien dibanding algoritma BFS. Dengan demikian, penentuan algoritma yang lebih efektif dalam menyelesaikan permasalahan bergantung pada bagaimana permasalahan didefinisikan dan dengan meninjau kembali karakteristik masing-masing algoritma. Untuk BFS pada umumnya akan lebih efisien untuk mencari simpul yang lebih dekat dengan simpul akar, sebab

karakteristiknya yaitu melakukan traversal dengan urutan berdasarkan kedalamannya. Adapun untuk DFS, akan lebih efektif untuk pencarian simpul yang berada jauh dari akar, tetapi simpul *ancestor*-nya ditraversal paling awal.

4.6 *Source Code*

Link github: https://github.com/firizky29/stima-folder-crawling

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Permasalahan *Folder Crawling* dapat dipecahkan dan disederhanakan dalam representasi graf. Dengan representasi graf, bisa dilakukan algoritma traversal atau penelusuran sistematik untuk mencari lokasi relatif sebuah direktori terhadap direktori akar. BFS dan DFS adalah contoh algoritma traversal yang berjalan baik dalam pemecahan permasalahan ini. Visualisasi yang didukung oleh kerangka kerja WinForms juga memudahkan pengguna untuk memahami proses penelusuran dan urutan penelusuran simpul-simpul yang dicari. Algoritma ini juga cukup efisien dan pencarian solusi dijamin didapatkan (hanya terdapat dua kemungkinan, yaitu, tidak ditemukan direktori beratribut tertentu atau ditemukan lintasan solusi pada graf). Algoritma dalam visualisasi graf juga pada implementasinya menggunakan konsep *threading* yang membuat pengembangan aplikasi lebih menarik.

5.2 Saran

Aktualisasi algoritma DFS dan BFS terutama dalam visualisasi pembentukan pohon struktur direktori bisa dikembangkan lebih lanjut. Algoritma diselesaikan oleh *compiler* secara asinkronus menggunakan konsep *mutual exclusion* dan konsep lainnya yang relevan. Optimalisasi lebih lanjut dapat dilakukan dalam memanipulasi konsep tersebut.

Disamping itu, entitas status yang disimpan dalam algoritma traversal graf diimplementasikan dengan menyimpan seluruh *state* graf pada saat itu. Sehingga untuk pencarian yang lebih kompleks, memungkinkan untuk proses transisi antar status lebih lambat. Salah satu caranya adalah menggunakan kerangka kerja antarmuka yang memungkinkan untuk proses *update* struktur graf secara sinkron dan real time, sehingga graf tidak perlu sepenuhnya disimpan di dalam status traversal.

DAFTAR PUSTAKA

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf diakses pada 24 Maret 2022.

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf diakses pada 24 Maret 2022.

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf diakses pada 24 Maret 2022.

[4] www.digitalcitizen.life/wp-content/uploads/2020/10/explorer_search_10.png diakses pada 25 Maret 2022.