LAPORAN TUGAS BESAR II

IF2211 STRATEGI ALGORITMA

Pengaplikasian Algoritma BFS dan DFS dalam Implementasi *Folder Crawling*

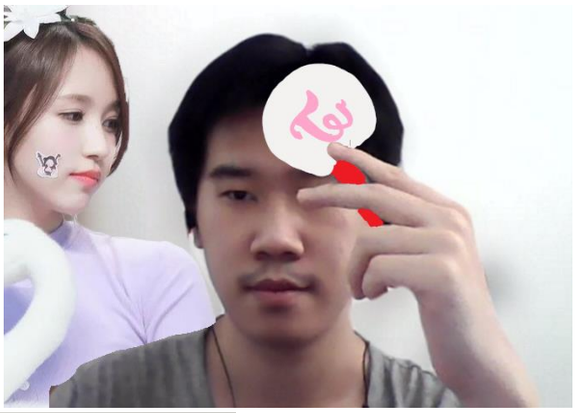
Disusun oleh :

Kelompok 07 – BreatheFirstBreadthLater

13520006 – Vionie Novencia Thanggestyo

13520092 – Vieri Mansyl

13520131 – Steven



PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

DAFTAR ISI

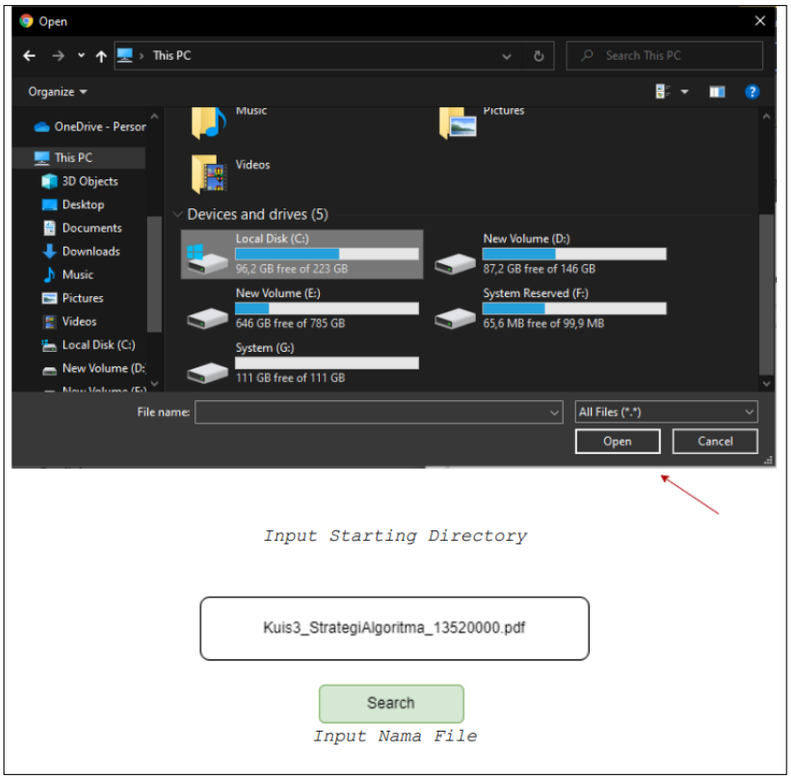
I. DESKRIPSI TUGAS

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.

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.

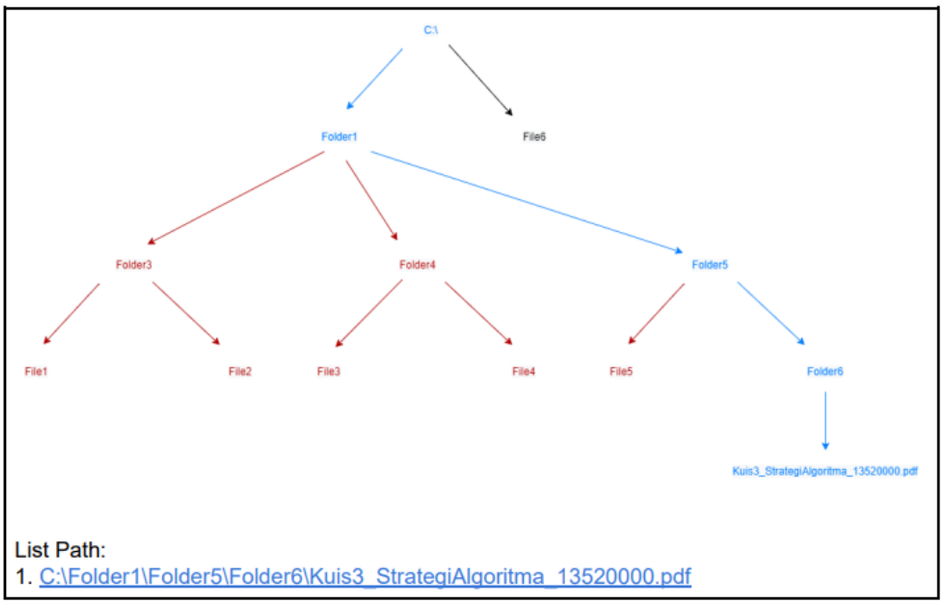
Fitur ini diimplementasikan dengan teknik folder crawling, di mana mesin komputer akan mulai mencari file yang sesuai dengan query mulai dari starting directory hingga seluruh children dari starting directory tersebut sampai satu file pertama/seluruh file ditemukan atau tidak ada file yang ditemukan. Algoritma yang dapat dipilih untuk melakukan crawling tersebut pun dapat bermacam-macam 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.

Dalam tugas besar ini, Kami diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari file explorer pada sistem operasi, yang pada tugas ini disebut dengan *Folder Crawling*. Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), program dapat menelusuri folder-folder yang ada pada direktori untuk mendapatkan direktori yang diinginkan. Kami diminta untuk memvisualisasikan hasil dari pencarian folder tersebut dalam bentuk pohon. Selain pohon, program diminta untuk dapat 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.



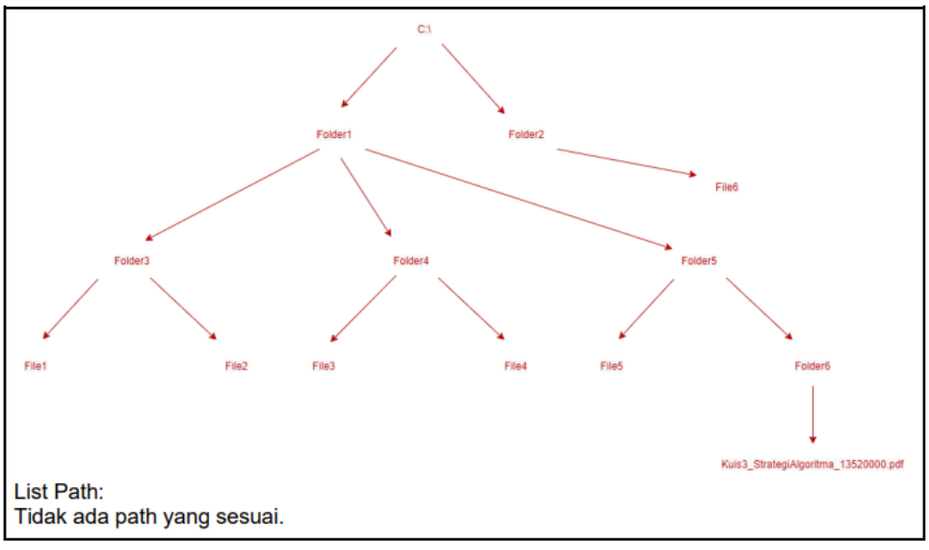
Gambar 1. Contoh tampilan input program

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 dibawah, 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.



Gambar 2. Contoh output program

Jika file yang ingin dicari pengguna tidak ada pada direktori file, misalnya saat pengguna mencari Kuis3Probstat.pdf, maka path pencarian DFS adalah sebagai berikut: C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3_StrategiAlgoritma_13520000.pdf → Folder6 → Folder5 → Folder1 → C:\ → Folder2 → File6. Pada gambar di bawah, semua simpul dan cabang berwarna merah yang menandakan seluruh direktori sudah selesai diperiksa semua namun tidak ada yang mengarah ke tempat file berada.



Gambar 3. Contoh output program jika file tidak ditemukan

Gambar 4. Contoh Ketika hyperlink di-klik

II. LANDASAN TEORI

Algoritma *graph traversal* (*graph search*) adalah algoritma dimana dilakukan pengunjungan pada simpul - simpul pada graf , yang merupakan representasi dari persoalan , secara sistematik untuk menemukan solusi yang diinginkan. Terdapat 2 jenis permasalahan dalam menggunakan algoritma *graph traversal* untuk mencari solusi, yaitu pada permasalahan yang tidak memberikan informasi (*uninformed*, disebut dengan *blind search*) , serta permasalahan yang memberikan informasi (*informed*). Dalam penyelesaiannya, pencarian solusi pada graf dapat menggunakan dua jenis pendekatan , yaitu dengan :

1. graf statis : graf yang telah terbentuk sebelum proses pencarian dilakukan
2. graf dinamis : graf yang terbentuk selama dilakukan pencarian

Dalam suatu graf, algoritma graph traversal bisa diimplementasikan dengan dua metode pencarian yang dikenal dengan pencarian Breadth First Search (BFS) serta pencarian Depth First Search (DFS).

Algoritma pencarian BFS merupakan algoritma yang mengunjungi suatu simpul beserta simpul tetangganya yang belum dikunjungi sebelumnya terlebih dahulu sebelum mengunjungi simpul anaknya. Diperlukan 3 komponen dalam mengimplementasikan algoritma BFS, yaitu :

1. Matriks ketetanggaan A = [aij] yang berukuran n × n,
   a. a_{ij}= 1, jika simpul i dan simpul j bertetangga
   b. a_{ij}= 0, jika simpul i dan simpul j tidak bertetangga.
2. Antrian (*queue*) q untuk menyimpan simpul yang telah dikunjungi.
3. Tabel Boolean, diberi nama “dikunjungi”
   dikunjungi : array[1..n] of boolean
   dikunjungi[i] = true jika simpul i sudah dikunjungi
   dikunjungi[i] = false jika simpul i belum dikunjungi

Berikut merupakan algoritma BFS secara umum.

```
Procedure BFS (input v: integer)
{Traversal graf dengan alforitma pencarian BFS.
  Masukan  : v adalah simpul awal kunjungan
  Keluaran : semua simpul yang dikunjungi dicetak ke layar}

DEKLARASI
       w : integer
       q : antrian
       procedure BuatAntrian (input/output q : antrian)
       {membuat antrian kosong, kepala(q) diisi 0}
       procedure MasukAntrian (input/output q : antrian, input v : integer)
       {memasukkan v ke dalam antrian q pada posisi belakang}
       procedure HapusAntrian (input/output q : antrian, output v : integer)
       {menghapus v dari kepala antrian q}
       function AntrianKosong (input a : antrian) → Boolean
       {true jika antrian q kosong, false jika sebaliknya}

ALGORITMA
       BuatAntrian(q)       {buat antrian kosong}
       write(v)             {cetak simpul awal yang dikunjungi}
       dikunjungi(v)←true   {simpul v telah dikunjungi, tandai dengan true}
       MasukAntrian(q,v)    {masukkan simpula awal kunjungan ke dalam antrian}

       {kunjungi semua simpul graf selama antrian belum kosong}
       While not AntrianKosong(q) do
              HapusAnttrian(q,v)  {simpul v telah dikunjungi, hapus dari antrian}
              for tiap simpul w yang bertetangga dengan simpul v do
                     If not dikunjungi(w) then
                            write(w)
                            MasukAntrian(w) ← true
                     endif
              endfor
       endwhile
       {AntrianKosong(q)}
```

Berbeda dengan Algoritma BFS, algoritma DFS akan mengunjungi simpul anak terlebih dahulu ketika berada pada simpul awal sebelum mengunjungi simpul tetangganya (akan terjadi *backtrack* ketika ingin mengunjungi simpul tetangga). Algoritma DFS juga memerlukan ketiga komponen seperti pada algoritma BFS, namun tidak menggunakan antrian (*queue*), melainkan menggunakan tumpukan (*stack*).

Berikut merupakan algoritma DFS secara umum.

```
Procedure DFS (input v: integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS.
  Masukan  : v adalah simpul awal kunjungan
  Keluaran : semua simpul yang dikunjungi dicetak ke layar}

DEKLARASI
       w : integer

ALGORITMA
       write(v)
       dikunjungi[v] ← true
       for w ← 1 to n do
              if A[v,w] = 1 then         {simpul v dan simpul w bertetangga}
                     if not dikunjungi[w] then
                            DFS(w)
                     endif
              endif
       endfor
```

Pada pencarian solusi pada pendekatan graf dinamis, hasil dari pencarian – baik menggunakan algoritma BFS maupun DFS – akan membentuk suatu graf yang merepresentasikan jalur pencarian dari akar graf (status awal / *initial state*) menuju daun graf (status hasil / *goal state*). Berlaku pada kedua algoritma BFS dan DFS, representasi dari graf dinamis terdiri dari :

1. Pohon ruang status (state space tree)
2. Simpul: problem state (layak membentuk solusi)
    a. Akar: *initial state*
    b. Daun: *solution/goal state*
3. Cabang: operator/langkah dalam persoalan
4. Ruang status (state space): himpunan semua simpul
5. Ruang solusi: himpunan status solusi

III. ANALISIS PEMECAHAN MASALAH

3.1. Langkah Pemecahan Masalah

Penyelesaian permasalahan Folder Crawling diimplementasikan di dalam file MainWindow.xaml.cs beserta beberapa film pembantu, yaitu sebagai berikut.

a. DirectoryTree.cs : untuk menginisialisasi suatu node dan mencari anak node tersebut
b. FolderPicker.cs : untuk menampilkan interface dari file explorer
c. Helper.cs : untuk mengambil path dari parent node ataupun path dari child node

Terdapat dua pendekatan dalam penyelesaian permasalahn Folder Crawling, yaitu dengan algoritma BFS dan algoritma DFS.

a. Penyelesaian dengan algoritma BFS

Pada algoritma BFS, digunakan struktur data Queue untuk mencatat pembacaan node selama keberjalanan algoritma. Awalnya, dicatat terlebih dahulu root node ke dalam queue. Setiap node yang tercatat akan ditandai 'telah dikunjungi' sehingga kedepannya node tersebut tidak akan di-cek kembali. Selanjutnya, selama queue tidak kosong (yang menandakan masih ada node yang belum dikunjungi) , maka akan dilakukan pengecekan pada anak node tersebut. Apabila tidak ditemukan file yang dicari, maka dilanjutkan pencarian. Seluruh child node yang terbaca dari node dimasukkan pada queue secara terurut. Selanjutnya, dilakukan pembacaan dan pengecekan yang sama pada node 'terdepan' dari queue seperti pada pembacaan dan pengecekan terhadap root node sebelumnya. Ketika ditemukan file yang ingin dicari oleh pengguna, maka program akan mencatat path dari file tersebut , yang mana file akan terbaca sebagai salah satu node dari graf, lalu pencarian akan dinyatakan selesai, terkecuali apabila pengguna ingin mencari seluruh file yang sama pada direktori tersebut. Apabila pengguna ingin mencari seluruh file yang sama, maka pencarian akan dilakukan sampai queue kosong (menunjukkan bahwa tidak ada node yang belum di-cek).

b. Penyelesaian dengan algoritma DFS

Secara umum, proses pencarian dan pengecekan pada algoritma DFS mirip dengan proses pencarian serta pengecekan dari algoritma BFS. Perbedaannya ialah terletak dari penggunaan pencatatan node yang akan dibaca. Berbeda dengan algoritma BFS yang mengimplementasikan struktur data queue, algoritma DFS mengimplementasikan pencatatan dengan menggunakan struktur data stack. Hal ini bertujuan agar proses pencarian dan pengecekan selalu dilakukan pada child node sampai dengan node daun. Apabila pada node terdalam tidak ditemukan file yang dicari, maka akan dilakukan *backtracking* , yaitu melakukan pengecekan pada node tetangga untuk tiap parent node. Proses ini akan dijalankan sampai ditemukan file yang ingin dicari. Apabila pengguna ingin mencari seluruh file yang sama, maka pencarian akan dilakukan sampai stack kosong (menunjukkan bahwa tidak ada node yang belum di-cek).

Hasil dari kedua algoritma ialah berupa path dari file-file yang dicari oleh pengguna. Untuk menyesuaikan dengan spesifikasi tugas besar ini, tidak hanya menampilakan *hyperlink* menuju direktori yang menyimpan file yang dicari, program akan menampilkan graf pohon yang merupakan representasi dari pencarian dengan kedua algoritma di atas. Setiap node juga akan diwarnai sesuai dengan spesifikasi tugas besar.

3.2. Proses Mapping Persoalan menjadi Elemen-Elemen Algoritma BFS Dan DFS

Persoalan Folder Crawling merupakan salah satu persoalan dengan pendekatan graf dinamis. Maka dari itu, Elemen-elemen untuk algoritma BFS serta algoritma DFS direpresentasikan sebagai berikut.

Elemen-elemen Algoritma	Algoritma BFS	Algoritma DFS
Pohon ruang status	Graf pohon yang terbentuk dari simpul-simpul yang telah tercatat pada ruang status	
Simpul	Akar : path direktori awal yang dibuka oleh user Daun : file yang ingin dicari oleh user	
Cabang	*Enqueue* child node yang terbaca dari parent node ke dalam *queue*	*push* child node yang terbaca dari parent node ke dalam *stack*
Ruang status	Simpul-simpul yang telah tercatat di dalam *queue*	Simpul-simpul yang telah tercatat di dalam *stack*

Ruang solusi	Himpunan seluruh path direktori menuju file yang dicari (untuk kasus pengguna ingin mencari seluruh file di dalam direktori) atau salah satu path direktori menuju file yang dicari pengguna

3.3. Contoh Ilustrasi Kasus Lain yang berbeda dengan Contoh pada Spesifikasi Tugas

Ilustrasi Kasus Dengan Exhaustive Search

Pertama-tama, lakukan enumerasi semua kemungkinan path yang terbentuk. Selanjutnya, lakukan evaluasi terhadap semua path yang terbentuk pada langkah pertama lalu simpan hasil yang memenuhi (merupakan file yang dicari oleh pengguna). Setelah itu, lakukan pengecekan input pengguna. Apabila pengguna hanya menginginkan satu saja, outputkan satu saja. Akan tetapi, apabila pengguna Find All Occurence, maka tampilkan semua hasilnya.

IV. IMPLEMENTASI dan PENGUJIAN

4.1. Implementasi program (pseudocode)

Berikut merupakan pseudocode dari program utama yang diimplementasikan pada file MainWindow.xaml.cs.

- **public void** ClearTextBlock

Berfungsi untuk membersihkan kotak hyperlinked path.

- **public void** SearchBFS

Berfungsi untuk melakukan pencarian file dengan menggunakan algoritma Breadth First Search

- **public void** SearchDFS

Berfungsi untuk melakukan pencarian file dengan menggunakan algoritma Depth First Search

- **private void** DrawTree

Berfungsi untuk meng-construct dan men-display pohon sesuai dengan spefisikasi tugas besar 2 IF2211 yakni memberikan warna biru pada jalur yang benar (menuju file yang dicari), warna merah pada jalur yang salah (bukan menuju file yang dicari), maupun warna hitam (tidak dilakukan pengecekan).

- **public void** AddHyperlink

Berfungsi untuk menampilkan lokasi-lokasi dari file ingin dicari pada text block dalam bentuk hyperlink menuju file explorer di mana file tersebut berada.

Berikut pseudocode dari program utama untuk menyelesaikan persoalan folder crawling.

```
ALGORITMA OnSearchButton

ClearTextBlock() {Membersihkan text block}

if (folderDir == "") then {Apabila pengguna tidak memilih direktori folder}
    output ("Please Choose a folder first")
endif

if (FileToSearch == "") then {Apabila pengguna tidak memasukkan nama file}
    output ("Please choose a file to search for")
endif

{ Apabila pengguna tidak memilih diantara BFS atau DFS}
if (RadioDFS.IsChecked == false) and (RadioBFS.IsChecked == false) then
    output ("Please choose either BFS or DFS before proceeding")
endif

Stopwatch.Start() { Memulai penghitungan waktu eksekusi program}

{Menghasilkan true apabila pengguna mencentang checkbox Find All Occurrence dan
menghasilkan false apabila sebaliknya.}

boolean findAll
FindAll = (FindAll.IsChecked == true)

if (RadioBFS.IsChecked == true) then
    SearchBFS()
else                              {RadioDFS.IsChecked == true}
    SearchDFS()
endif

DrawTree()                {Menggambar pohon pencarian file}
AddHyperlink()            {Mendisplay hyperlink file pencarian apabila ada}

{Program telah selesai melakukan pencarian pada file}
Stopwatch.Stop()          {Memberhentikan penghitungan waktu}

displayToTextBlock(waktu_eksekusi_program) {format waktu: 00:00:00.00}
```

4.2. Penjelasan Struktur data

Pada implementasinya, program menggunakan beberapa struktur data untuk mengakomodasi keberjalanan program, yaitu sebagai berikut.

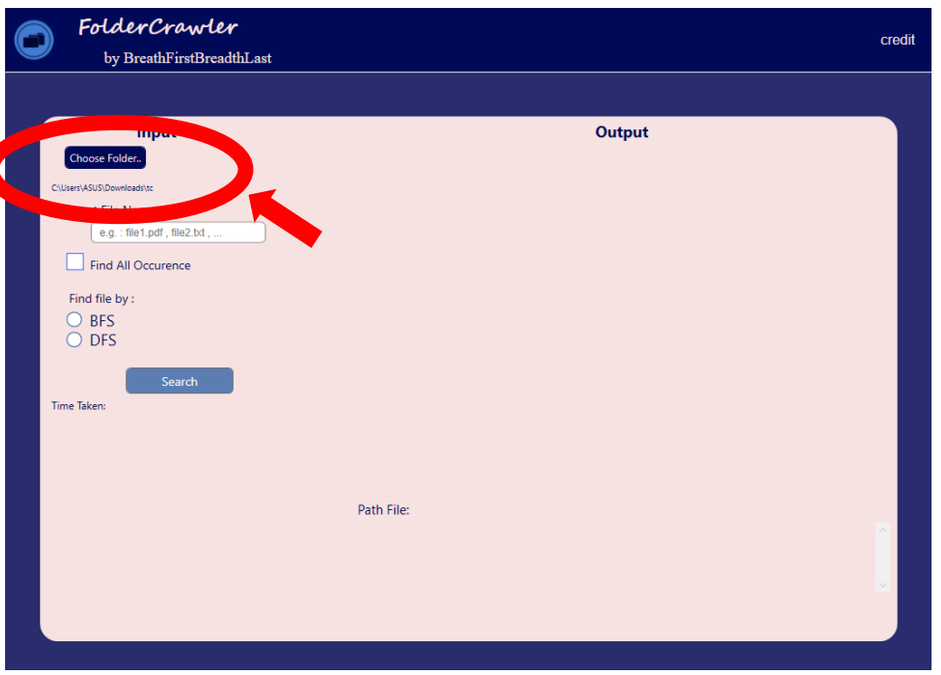
- List of string : digunakan untuk menampung seluruh path dari file-file yang dicari oleh pengguna
- DirectoryTree : kelas yang merepresentasi dari graf tree dan nantinya digunakan untuk menampilkan hasil tree yang terbentuk
- Queue of string : digunakan untuk menampung simpul-simpul yang akan di-cek oleh program. Struktur data queue diimplementasikan di dalam algoritma BFS
- Stack of string : digunakan untuk menampung simpul-simpul yang akan di-cek oleh program. Struktur data stack diimplementasikan di dalam algoritma DFS
- Stopwatch : digunakan untuk mencatat waktu eksekusi program dimulai dari pengecekan direktori awal sampai program selesai.

4.3. Tata cara penggunaan program

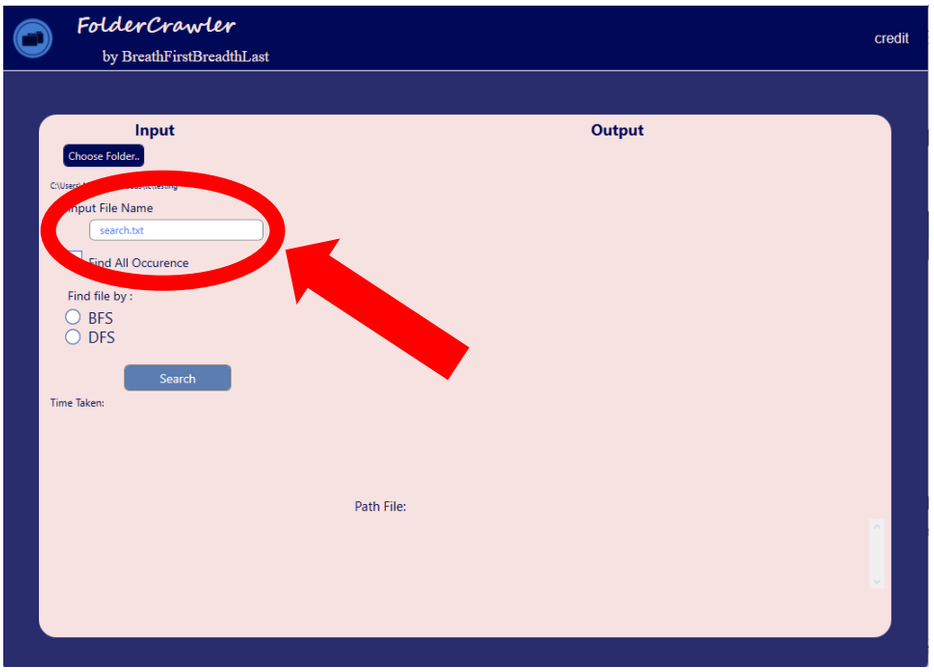
Tampilan awal dari perangkat lunak ialah sebagai berikut.

Tata cara penggunaan perangkat lunak ialah sebagai berikut.

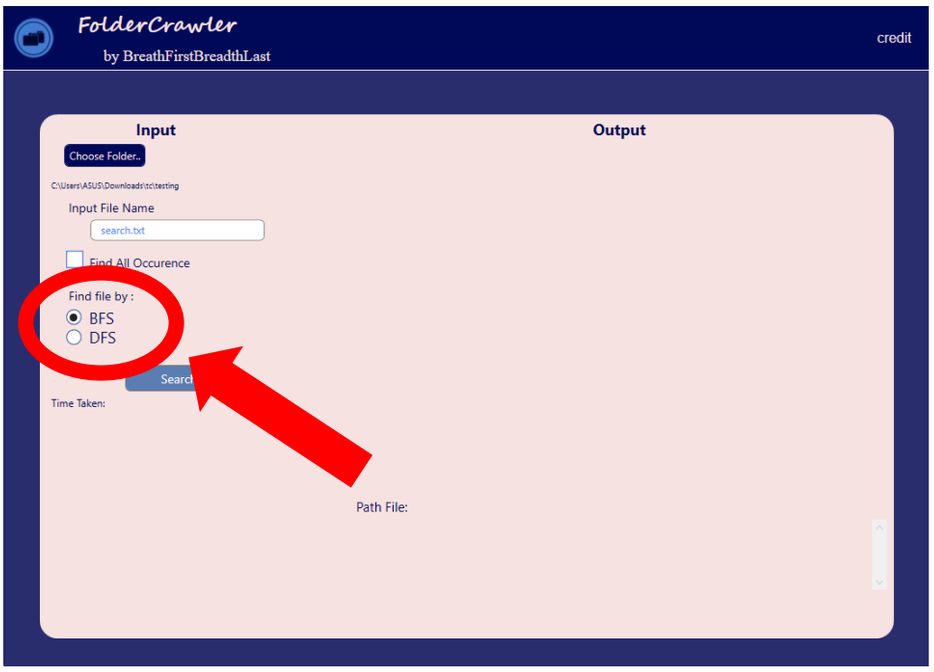
1. pilih folder pada tombol choose folder



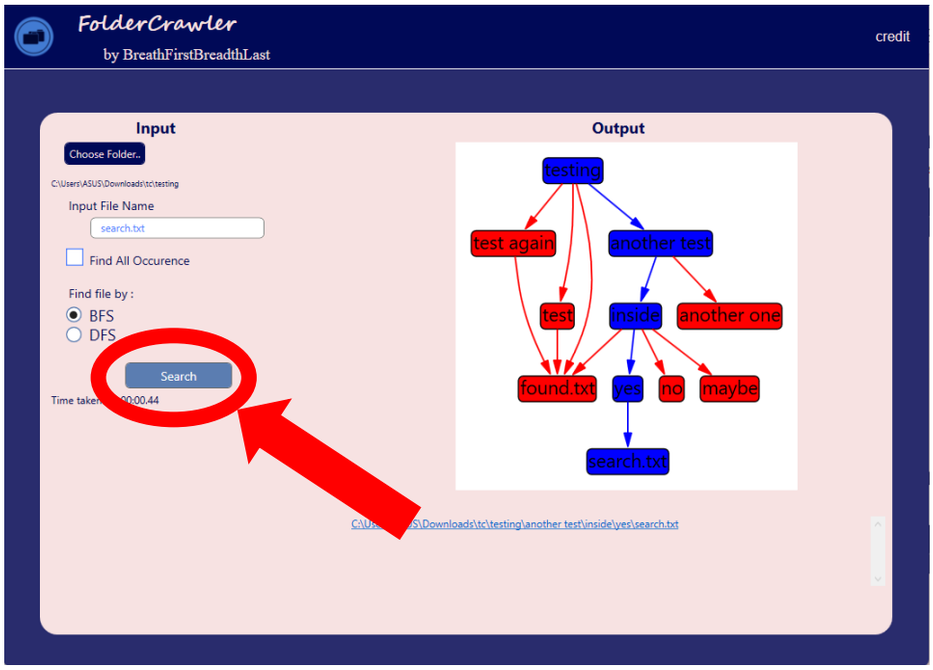
2. masukkan nama file yang ingin dicari pada textbox yang tersedia



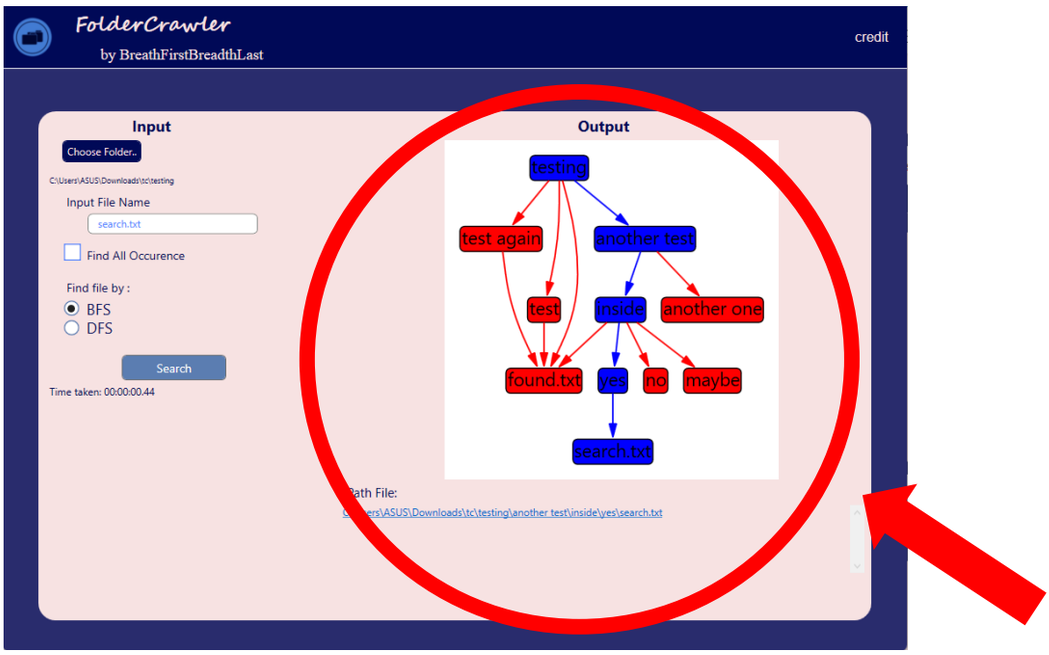
3. pilih metode pencarian yang diinginkan (BFS atau DFS)



4. tekan tombol search



5. Hasil akan ditampilkan pada bagian output.



4.4. Pengujian Program

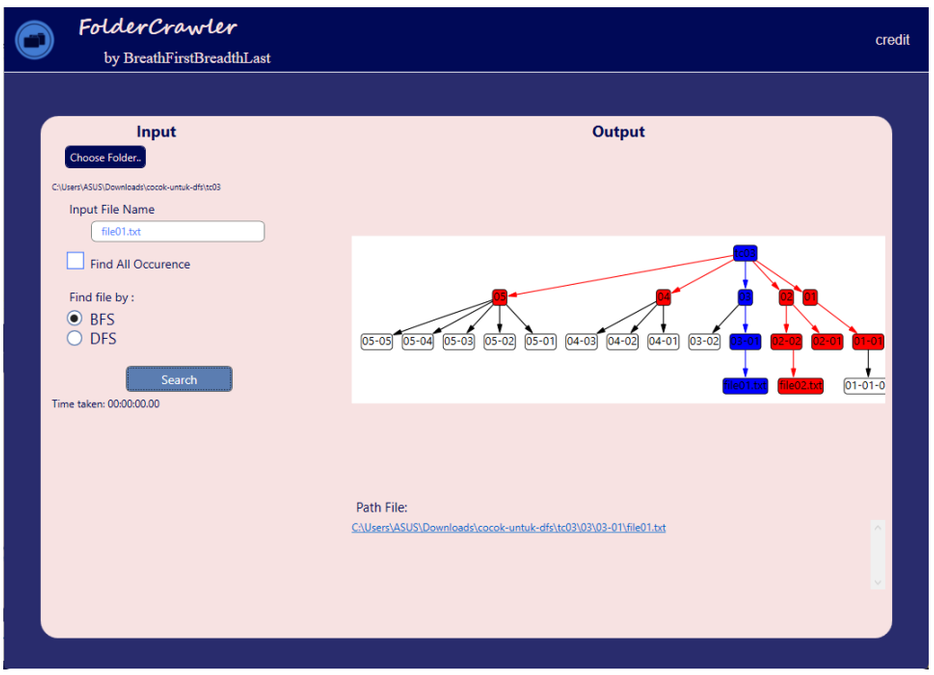
Struktur folder direktori yang akan diuji ialah sebagai berikut.

```
parentNode
        01
                01-01
                        01-01-01
                                01-01-01-01
                                        01-01-01-01-01
                                                file01.txt
        02
                02-01
                02-02
                        file02.txt
        03
                03-01
                        file01.txt
                03-02
        04
                04-01
                04-02
                        04-02-01
                        04-02-02
```

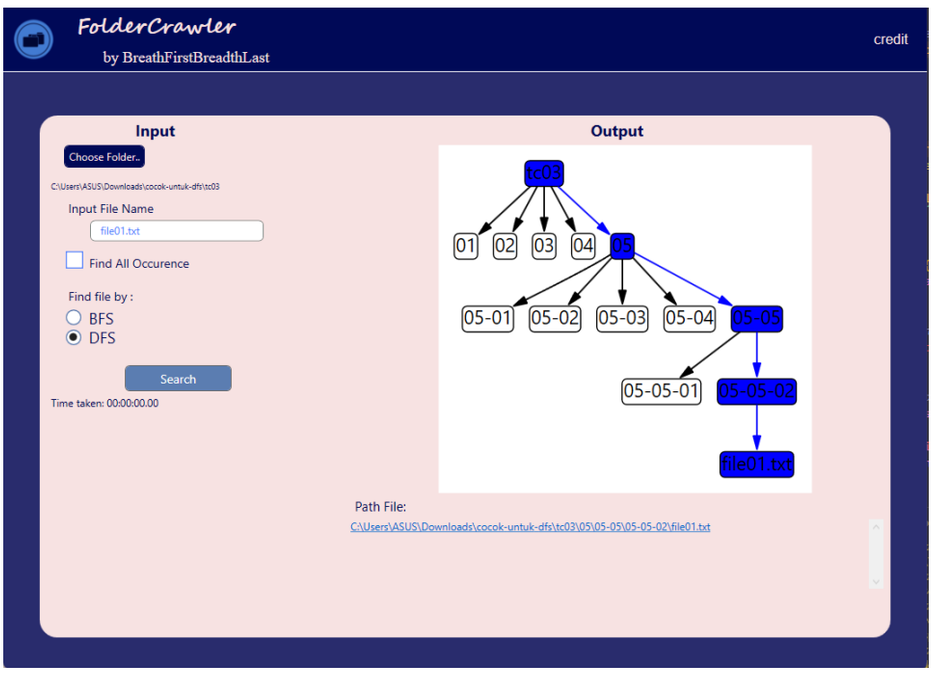
```
                        file02.txt
            04-03
                  file03.txt
      05
            05-01
            05-02
            05-03
            05-04
            05-05
                  05-05-01
                  05-05-02
                  file01.txt
```

a. Test case 1 – mencari file01.txt

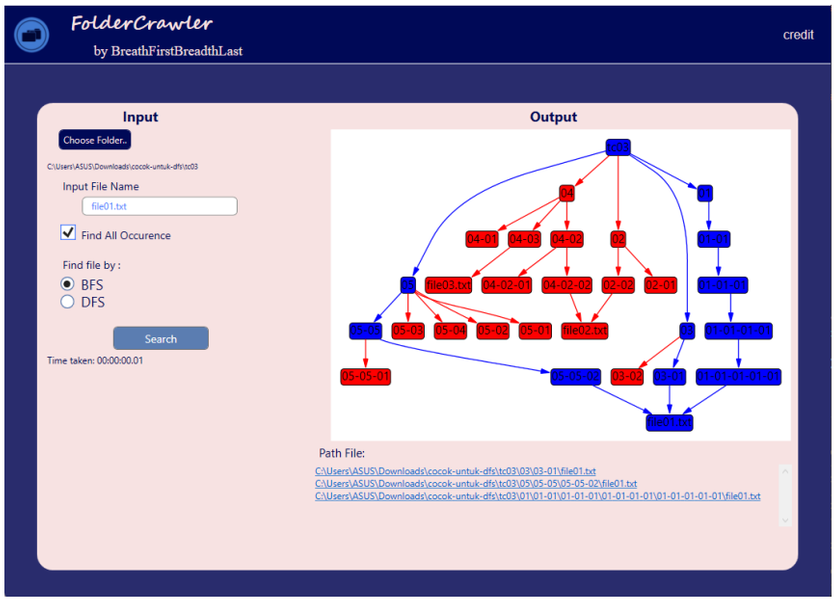
Hasil menggunakan metode **BFS tanpa** centang **find all occurence**:



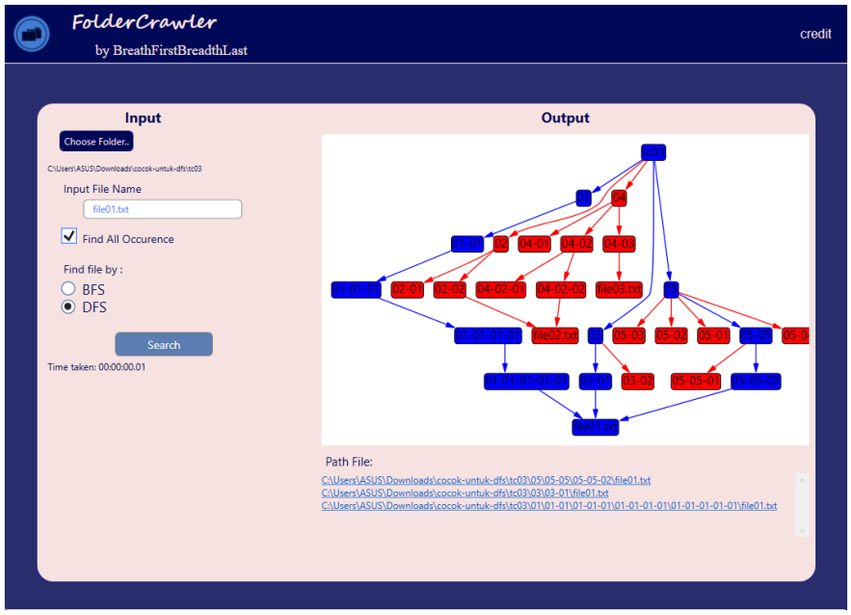
Hasil menggunakan metode **DFS tanpa** centang **find all occurence**:



Hasil menggunakan metode **BFS** dengan centang **find all occurence**:

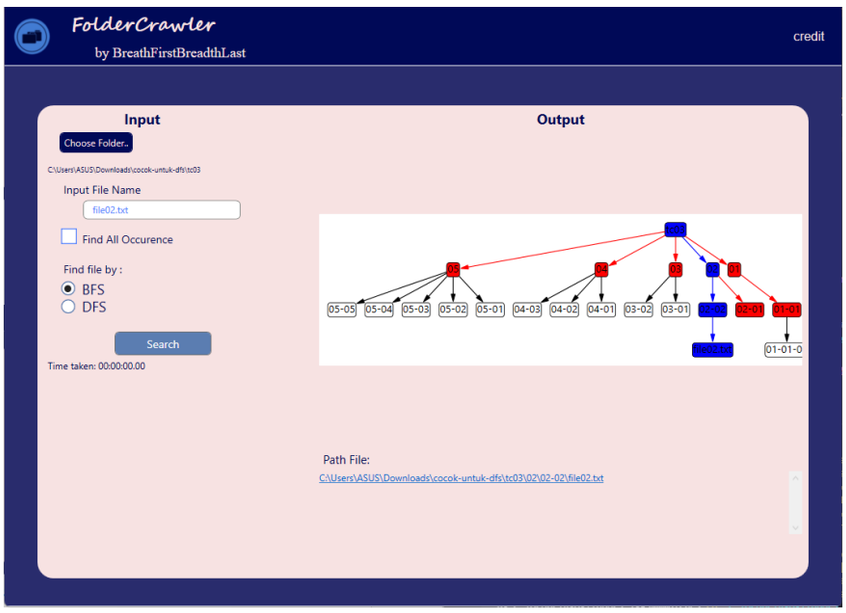


Hasil menggunakan metode **DFS** dengan centang **find all occurence**:

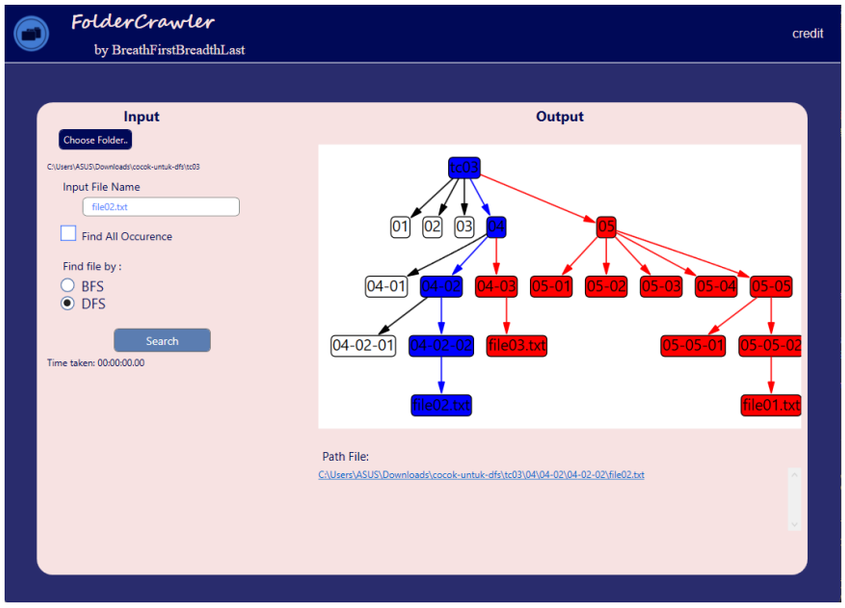


b. Test case 02 – mencari file02.txt

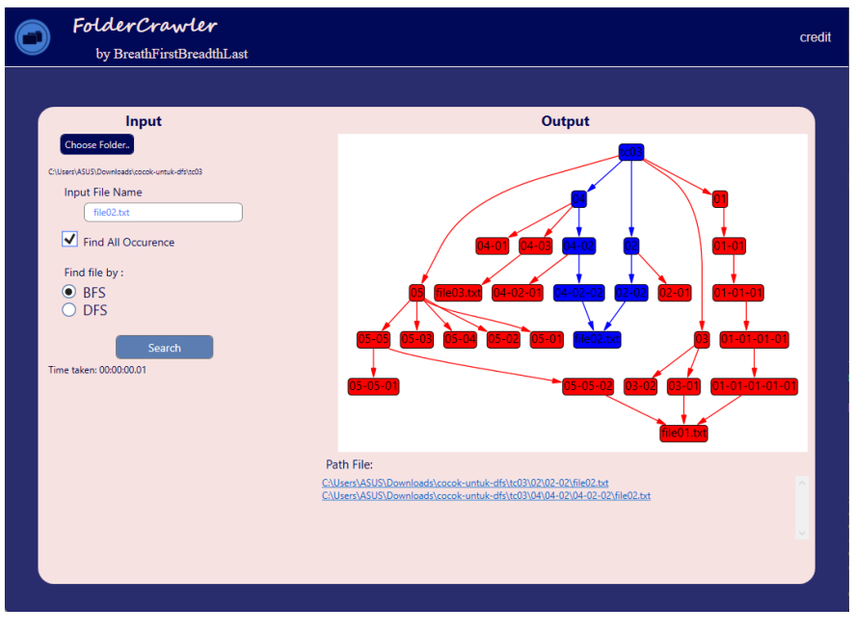
Hasil menggunakan metode **BFS tanpa** centang **find all occurence**:



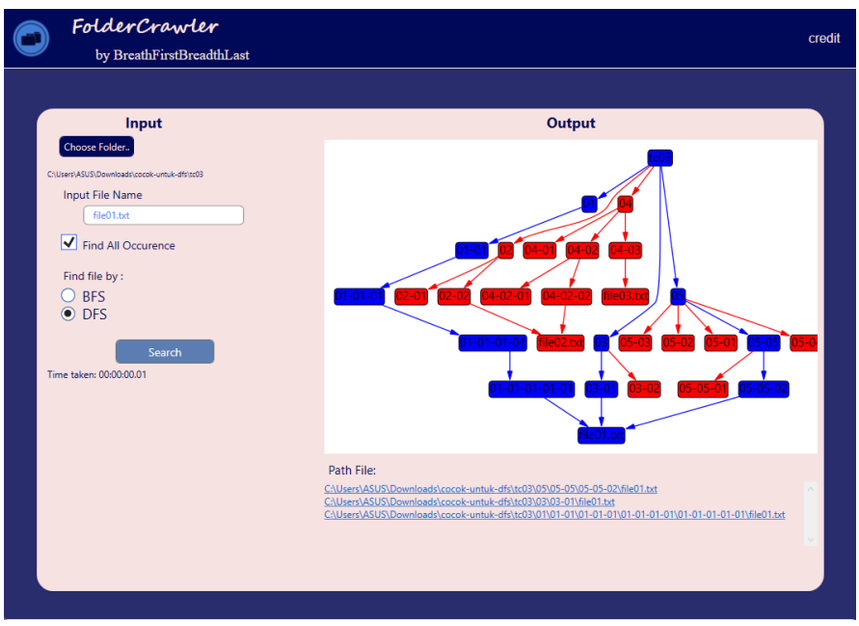
Hasil menggunakan metode **DFS tanpa** centang **find all occurence**:



Hasil menggunakan metode **BFS** dengan centang **find all occurence**:

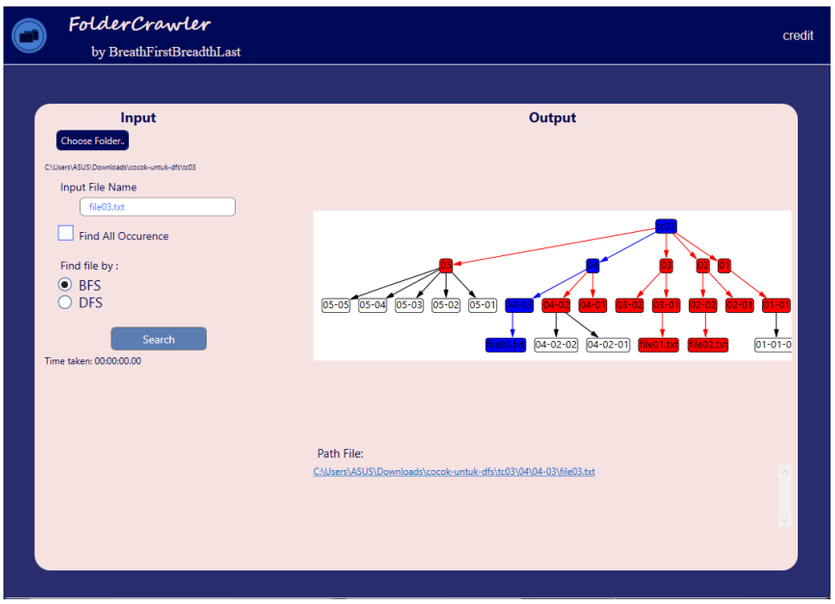


Hasil menggunakan metode **DFS** dengan centang **find all occurence**:

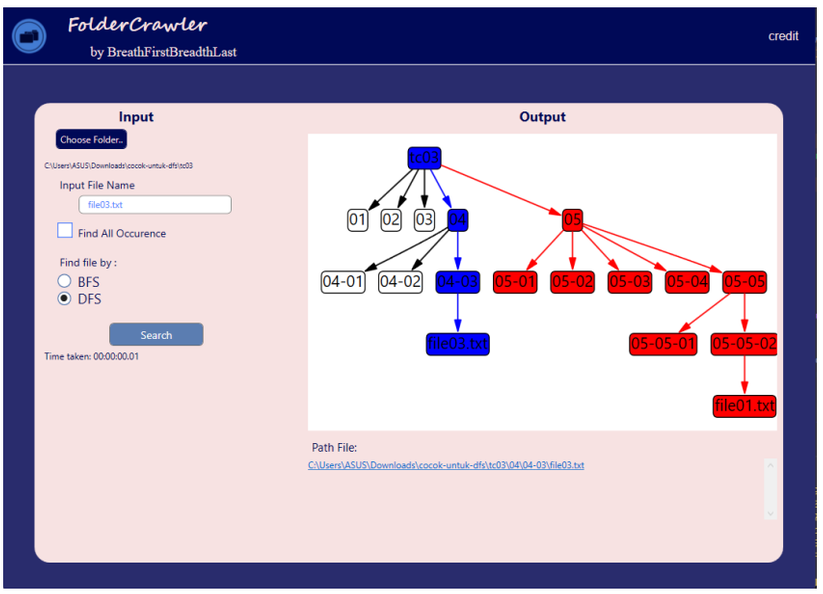


c. Test case 03 – mencari file03.txt

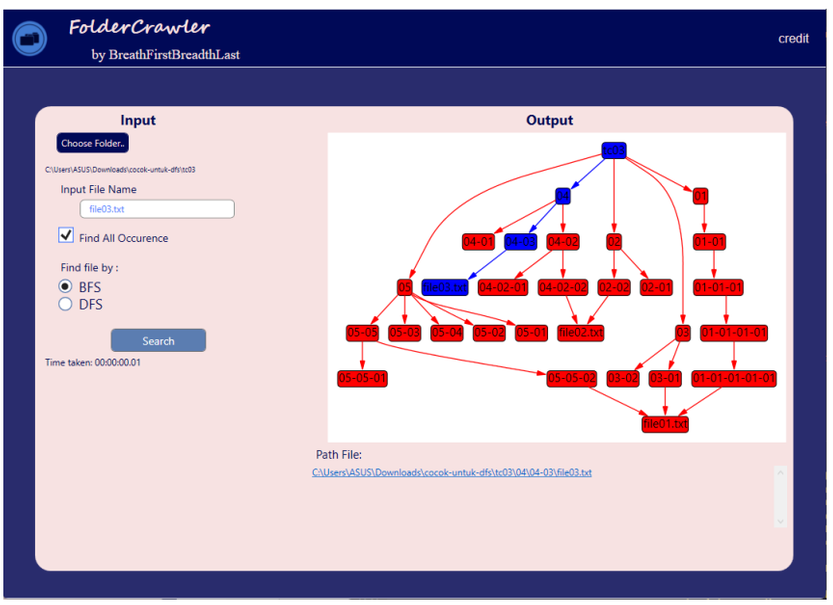
Hasil menggunakan metode **BFS tanpa** centang **find all occurence**:



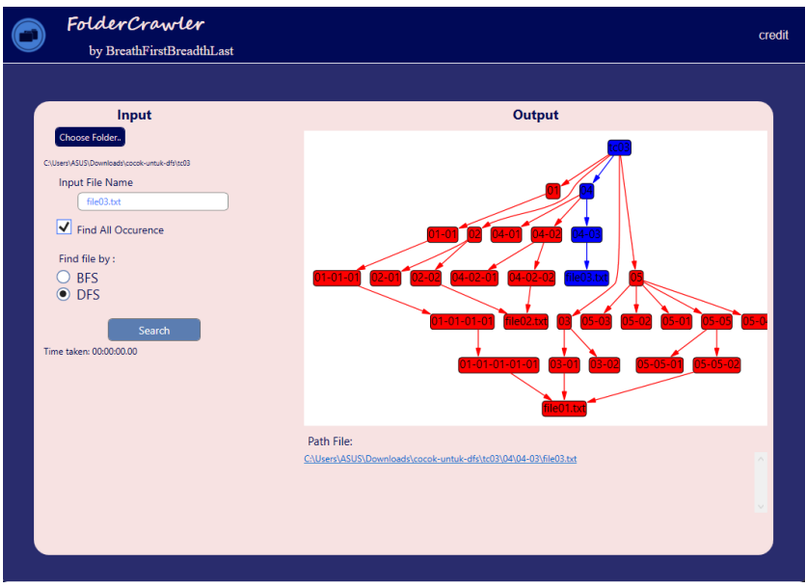
Hasil menggunakan metode **DFS tanpa** centang **find all occurence**:



Hasil menggunakan metode **BFS** dengan centang **find all occurence**:

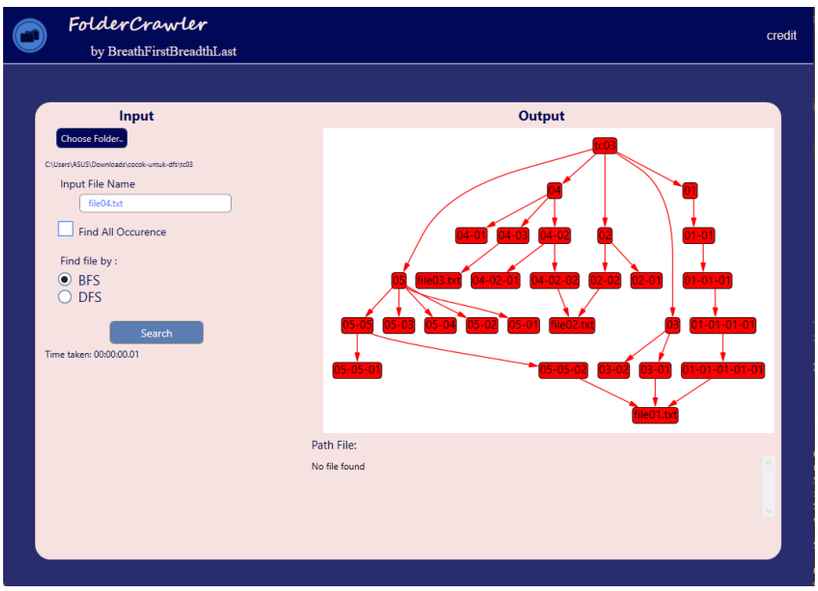


Hasil menggunakan metode **DFS** dengan centang **find all occurence**:

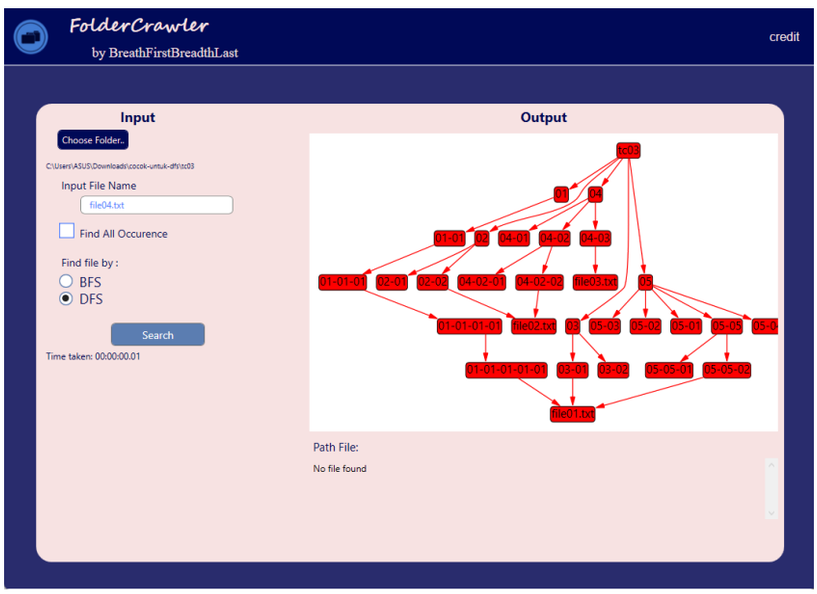


d. Test case 04 – mencari file04.txt

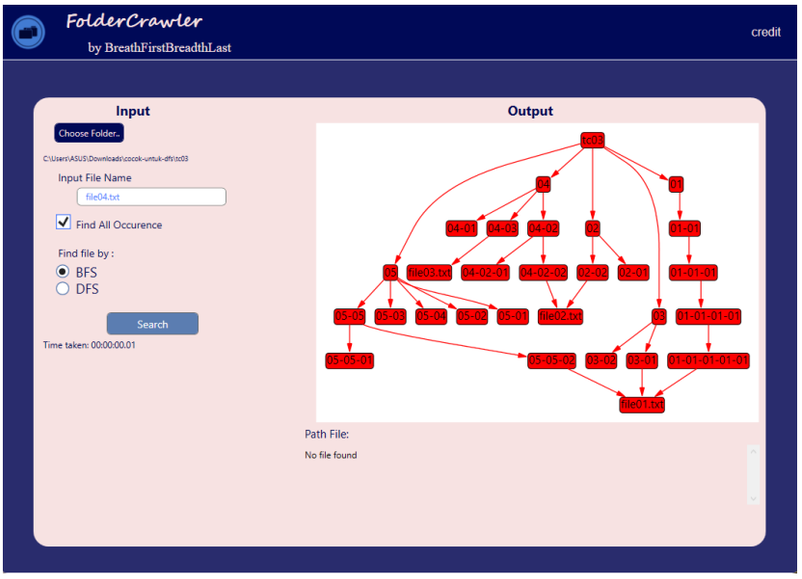
Hasil menggunakan metode **BFS tanpa** centang **find all occurence**:



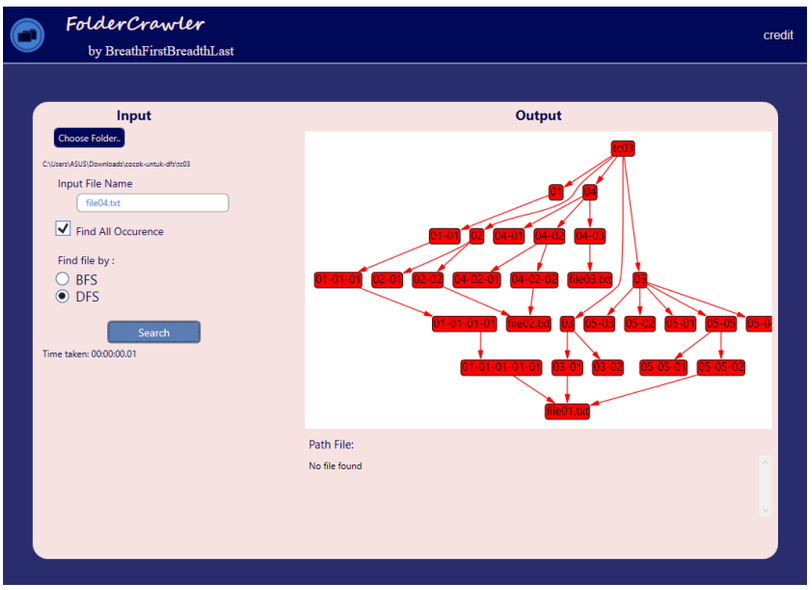
Hasil menggunakan metode **DFS tanpa** centang **find all occurence**:



Hasil menggunakan metode **BFS** dengan centang **find all occurence**:



Hasil menggunakan metode **DFS** dengan centang **find all occurence**:



4.5. Analisis dari desain solusi algoritma BFS dan DFS terhadap pengujian

Dari hasil pengujian di atas, dapat terlihat perbedaan efektivitas dan efisiensi pencarian solusi antara algoritma BFS dengan algoritma DFS. Berikut perbedaan yang dapat terlihat di antara implementasi algoritma BFS dengan algoritma DFS.

Algoritma BFS	Algoritma DFS
Kelebihan	
Pada pencarian dimana file yang ingin dicari terletak pada kedalaman yang cukup ‘dangkal’ terhadap path direktori awal, pencarian akan lebih cepat dikarenakan hanya dilakukan pengecekan pada setiap kedalaman sampai dengan kedalaman dimana file tersebut berada.	Pada pencarian dimana file yang ingin dicari terletak pada kedalaman tinggi, pencarian akan lebih cepat dikarenakan tidak perlu melakukan pengecekan pada tiap folder tetangga.
Kekurangan	
Tidak menjadi *best practice* ketika file yang ingin dicari terletak pada direktori yang sangat dalam.	Tidak menjadi *best practice* ketika ketika file yang ingin dicari terletak pada direktori yang memiliki tetangga dan ‘anak’ folder yang banyak.

Dari analisis desain solusi yang digunakan, kedua algoritma BFS dan DFS memberikan keuntungan dan kerugian masing-masing sehingga tidak dapat dikatakan mana yang lebih baik dan mana yang lebih buruk. Oleh karena itu, BFS dan DFS dapat dikatakan sebagai algoritma yang *equal* bergantung pada permasalahan yang disediakan.

V. KESIMPULAN dan SARAN

5.1. Kesimpulan

Melalui Tugas Besar ini, kami berhasil menyelesaikan persoalan Folder Crawling dengan menggunakan algoritma *Breadth First Search* (BFS) dan algoritma *Depth First Search* (DFS). Dikarenakan penyelesaian pada persoalan Folder Crawling merupakan penyelesaian dengan pendekatan graf dinamis, efisiensi dari algoritma BFS serta algoritma DFS tidak dapat ditebak. Algoritma BFS akan lebih baik performansinya ketika direktori folder yang ada tersebar secara merata dengan kedalaman yang dangkal, sedangkan algoritma DFS akan lebih baik performansinya ketika direktori folder memiliki kedalaman yang tinggi. Efisiensi dari algoritma BFS akan berkurang apabila menghadapi kasus *best case* bagi algoritma DFS. Sebaliknya pula, efisiensi dari algoritma DFS akan berkurang apabila menghadapi kasus *best case* bagi algoritma BFS.

5.2. Saran

Untuk penelitian selanjutnya, dapat dilakukan pengembangan terhadap User Interface sehingga tampilan perangkat lunak menjadi lebih interaktif dan menarik bagi User.

DAFTAR PUSTAKA

- Munir, Rinaldi. Breadth First Search (BFS) dan Depth Firs Search (DFS) (Bagian 1) (Versi baru 2021)

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

- XAML Documentation. Microsoft.

https://docs.microsoft.com/en-us/dotnet/desktop/wpf/advanced/xaml-in-wpf?view=netframeworkdesktop-4.8

- C# Documentation. Microsoft.

https://docs.microsoft.com/en-us/dotnet/csharp/

- Spesifikasi Tugas Besar 2 : Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling.

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Tugas-Besar-2-IF2211-Strategi-Algoritma-2022.pdf

TAUTAN

Link video Demo Youtube :

https://youtu.be/t-JhDHqqEU4

Link Repository :

https://github.com/VionieNovencia/Tubes2_13520006