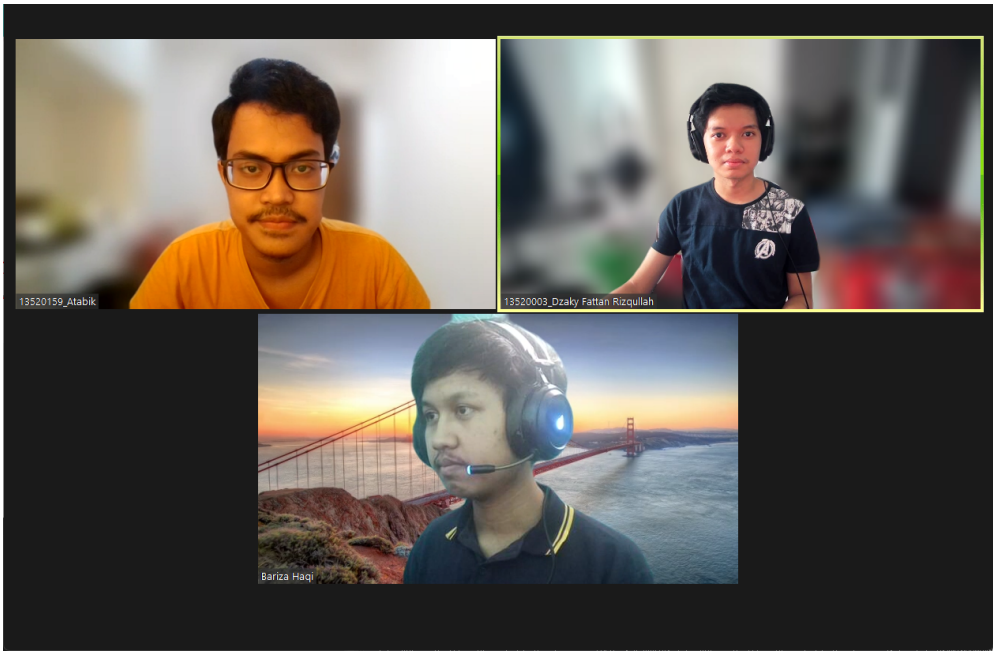
TUGAS BESAR II
PEMANFAATAN ALGORITMA DEPTH-FIRST SEARCH DAN BREADTH-FIRST SEARCH DALAM APLIKASI FOLDER CRAWLING

LAPORAN
Diajukan sebagai salah satu tugas mata kuliah
IF2211 Strategi Algoritma Semester II
Tahun Akademik 2021-2022

Oleh
Kelompok 46 - DeathFromStima

Dzaky Fattan Rizqullah	**13520003**
Bariza Haqi	**13520018**
Atabik Muhammad Azfa Shofi	**13520159**



SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG

2022

DAFTAR ISI

BAB 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. Meskipun demikian, hampir seluruh sistem operasi sudah menyediakan fitur search yang dapat digunakan untuk mencari file yang kita inginkan. 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.

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

Aplikasi yang akan dibangun dibuat berbasis GUI. 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 pencarian file tersebut dengan memberikan keterangan folder/file yang sudah diperiksa, folder/file yang sudah masuk antrian tapi belum diperiksa, dan rute folder beserta 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 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.

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

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

BAB II
LANDASAN TEORI

2.1 Graph Traversal

Graph traversal atau disebut juga algoritma traversal graf merupakan algoritma pada graf dengan mengunjungi simpul dengan cara yang sistematik. Graf di sini merepresentasikan persoalan, dan traversal graf merupakan teknik pencarian solusi pada graf. Dalam proses pencarian solusi, terdapat dua pendekatan, yaitu:

a. Graf statis, yaitu graf yang sudah terbentuk sebelum proses pencarian dilakukan. Graf itu sendiri direpresentasikan sebagai struktur data.
b. Graf dinamis,yaitu graf yang terbentuk saat proses pencarian dilakukan. Graf tidak tersedia sebelum pencarian karena graf dibangun selama pencarian solusi.

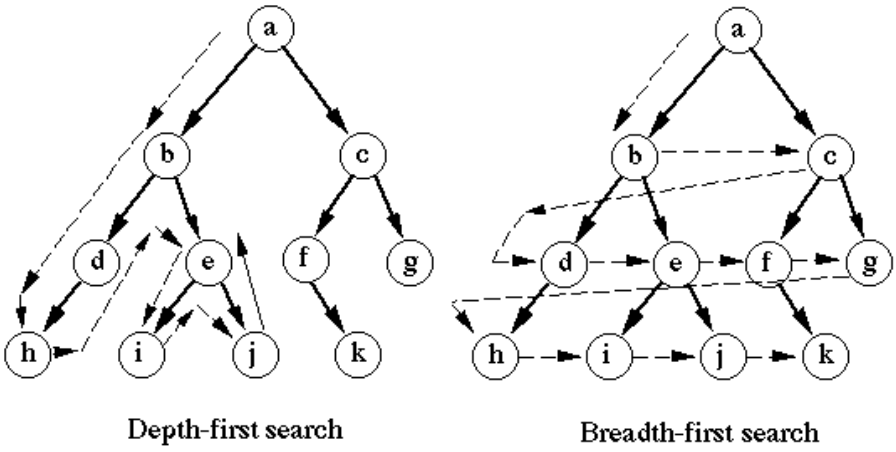
Algoritma pencarian solusi pada graf terbagi menjadi dua jenis, yaitu pencarian tanpa informasi (*uninformed search/blind search*) dan pencarian dengan informasi (*informed search*). Terdapat dua jenis pencarian solusi tanpa informasi yang umum digunakan, yaitu pencarian melebar (*Breadth First Search/BFS*) dan pencarian mendalam (*Depth First Search/DFS*).

Algoritma BFS adalah suatu metode pencarian pada sebuah pohon dengan menelusuri semua simpul tetangga terlebih dahulu sebelum mengunjungi simpul berikutnya, hingga menemukan solusinya. Berikut adalah langkah algoritma BFS dengan traversal dimulai dari simpul v.

1. Kunjungi simpul v
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

Algoritma DFS adalah suatu metode pencarian pada sebuah pohon dengan menelusuri satu cabang sebuah pohon sebelum menuju cabang tetangganya, hingga menemukan solusinya. Berikut adalah langkah algoritma DFS dengan traversal dimulai dari simpul v.

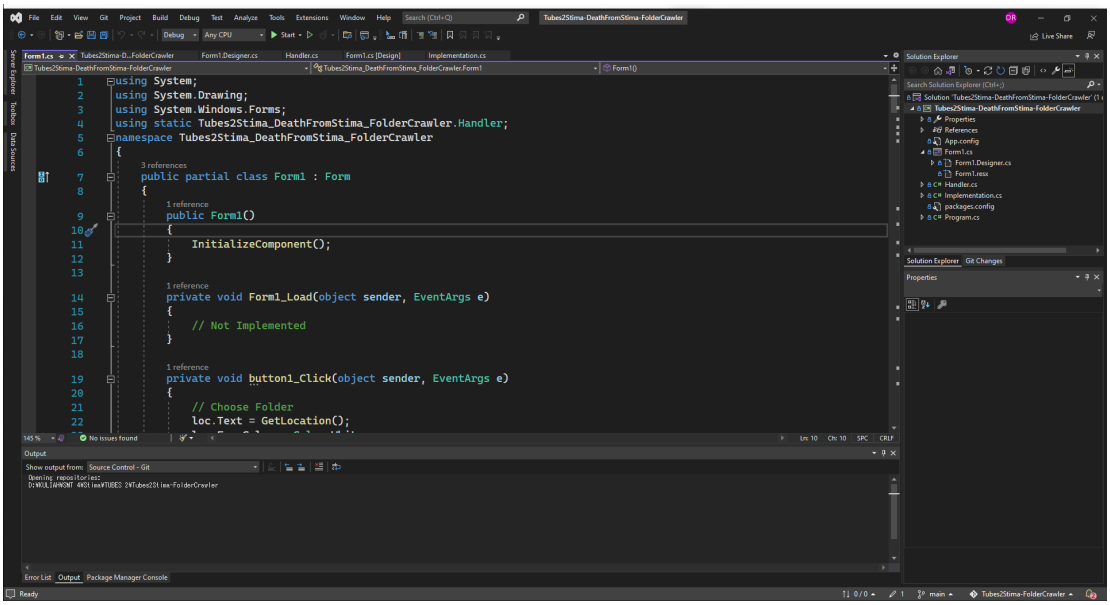
1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v
3. Ulangi DFS mulai dari simpul w
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian di-runut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.



Gambar 1. Ilustrasi perbedaan algoritma pencarian melebar dengan pencarian mendalam.
Sumber: https://medium.com/@tim_ng/bfs-and-dfs-52d3cb642a0e

2.2 C# Desktop Application Development

C# Desktop Application Development adalah sebuah pengembangan aplikasi desktop dengan menggunakan bahasa C#. Untuk melakukan pengembangan tersebut dibutuhkan sebuah perangkat lunak yaitu Microsoft Visual Studio IDE. Pada Microsoft Visual Studio IDE, disediakan berbagai fasilitas untuk membangun sebuah aplikasi desktop dengan bahasa C#. Salah satunya adalah *Windows Forms* atau disingkat *WinForms*. *WinForms* adalah pustaka pemrograman *open-source* yang disediakan *Microsoft* dalam *.NET Framework* yang dapat digunakan untuk membangun aplikasi desktop berbasis *Graphical User Interface* (*GUI*).



Gambar 2. Tampilan antarmuka Microsoft Visual Studio IDE 2022.
Sumber: Dokumen penulis

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Langkah Pemecahan Masalah

Pemecahan masalah pada pemanfaatan algoritma *DFS* dan *BFS* untuk aplikasi *Folder Crawler* ini dimulai dengan memahami persoalan secara menyeluruh dan mengidentifikasi elemen-elemen permasalahan yang dapat di-*mapping* menjadi elemen algoritma *DFS* dan *BFS*. Setelah melakukan proses mapping, Algoritma dapat dirancang dengan bantuan *pseudocode* dengan penyesuaian terhadap elemen-elemen permasalahannya. Selanjutnya program dikode dan diimplemen menjadi sebuah aplikasi berbasis *GUI* dengan memanfaatkan pustaka *WinForms*.

3.2 Proses *Pemetaan* Persoalan

3.2.1 Pemetaan Pencarian 1 File dengan Algoritma BFS dan DFS

Pada Pencarian 1 file akan berhenti melakukan pencarian disaat file yang dicari ditemukan. Berikut ini merupakan pemetaan Folder Crawling pada pencarian 1 file dengan algoritma BFS.

Elemen Algoritma BFS	Pemetaan pada Folder Crawling
Akar	Folder awal yang akan dilakukan pencarian file
Simpul	Sub direktori dari akar yang akan menuju ke lokasi file yang dicari
Daun	File yang dicari
Ruang Status	Seluruh folder yang telah dilalui saat pencarian BFS dan DFS
Ruang Solusi	Semua File yang telah dilalui saat pencarian BFS dan DFS

3.2.2 Pemetaan Pencarian Semua Kemunculan File dengan Algoritma DFS

Pada Pencarian Semua Kemunculan File, semua Sub direktori dari folder yang akan dilakukan pencarian akan diperiksa untuk menemukan semua lokasi file yang dicari. Berikut ini merupakan pemetaan Folder Crawler pada pencarian Semua Kemunculan File dengan Algoritma DFS.

Elemen Algoritma BFS	Pemetaan pada Folder Crawling
Akar	Folder awal yang akan dilakukan pencarian file
Simpul	Sub direktori dari akar yang akan menuju ke lokasi file yang dicari
Daun	Semua File yang dicari
Ruang Status	Seluruh folder yang berada di dalam folder yang dicari
Ruang Solusi	Semua File yang berada di folder yang dilakukan pencarian termasuk di folder dalamnya

3.3 Kemungkinan Kasus Lain

Kemungkinan kasus lain yang mungkin muncul pada pencarian yaitu:

- Mencari pada direktori akar yang isinya kosong
- Mencari pada folder yang memiliki suatu bentuk proteksi sehingga folder tidak dapat diakses begitu saja oleh program

Untuk kemungkinan pertama, program hanya akan menampilkan pohon penyelesaian yang berisi simpul akar saja. Untuk kemungkinan kedua, kasusnya tidak di-*handle* sehingga memungkinkan terjadi error.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma *DFS* dan *BFS*

Implementasi algoritma *DFS* dan *BFS* pada aplikasi "Folder Crawler" ditulis pada file `Implementation.cs` sebagai kelas tersendiri, yang kemudian akan dimanfaatkan oleh `Handler.cs` yang menjalankan program secara keseluruhan. Antarmuka program memanfaatkan *WinForms* yang interaksinya ditulis dalam `Form1.cs`.

Berikut adalah *pseudocode* pemanfaatan algoritma *DFS* dan *BFS* yang sudah disimplifikasi namun tidak mengubah esensi dari algoritmanya.

```
function DFS(input File inputFile, input boolean Multiple) → array of File result
Kamus lokal
subDirs : array of Directory
filesContainer : array of Files
Algoritma

subDirs ← root.GetDirectories() {Recursive function}

if (subDirs ≠ null) then

    for each directory in subDirs

        recPath ← DFS(Multiple)

    endfor

endif

filesContainer ← root.GetFiles() {file checking}

if (filesContainer ≠ null) then

    for each file in filesContainer

        if (file.name = inputFile) then

        result.addElmt(file)

        if (not multiple) then break endif

    endfor

endif
```

function BFS(input File inputFile, input boolean Multiple, input/output array of Directory dirQueue) → array of string resultPath

Kamus Lokal

```
subDirs : array of Directory
filesContainer : array of Files
```

Algoritma

```
{Move to Queue}

subDirs ← root.GetDirectories()

if (subDirs ≠ null) then

    for each directory in subDirs

        dirQueue.addElmt(directory)

    endfor

endif

{file checking}

filesContainer ← root.GetFiles()

if (filesContainer ≠ null) then

    for each file in filesContainer

        if (file.name = inputFile) then

        result.addElmt(file)

        if (not multiple) then break endif

    endfor

endif

{Recursive function}

if (dirQueue ≠ null) then

    recPath ← DFS(Multiple)

endif
```

4.2 Penjelasan Struktur Data

Berikut dilampirkan struktur data program yang berisi objek serta variabel yang dimanfaatkan untuk menjalankan algoritma pencarian yang sudah dijelaskan sebelumnya. Objek yang tertera hanyalah objek-objek yang bersifat fungsional saja.

No.	Nama	Tipe	Deskripsi
Bagian Antarmuka (`Form1.cs Form1.Designer.cs`**)**			
1.	`pic_Result`	`PictureBox`	Menyimpan gambar pohon hasil pencarian beserta informasi propertinya.
2.	`tlp_ResultList`	`TableLayoutPanel`	Menyimpan daftar *hyperlink* hasil pencarian. Bila file tidak ditemukan, akan dituliskan pesan "File tidak ditemukan"
3.	`lbl`	`LinkLabel`	Elemen *hyperlink* di dalam `tlp_ResultList`
4.	`label_TimeSpent`	`label`	Menampilkan tulisan waktu terpakai saat algoritma dijalankan
Bagian *input* dan *output* (`Handler.cs`**)**			
5.	`rootFolder`	`string`	Menyimpan nama folder akar untuk memulai pencarian
6.	`fileName`	`string`	Menyimpan nama file yang dicari
7.	`searchMode`	`string`	Menyimpan algoritma *searching* yang digunakan
8.	`arrResultPath`	`string[]`	Menyimpan larik berisi daftar *string path* menuju file yang dicari. Hanya berisi satu elemen bila tidak menggunakan mode pencarian seluruh kemunculan file.
9.	`picBoxWidth`	`int`	Menyimpan ukuran lebar gambar pohon pencarian yang akan dibentuk
10.	`picBoxHeight`	`int`	Menyimpan ukuran lebar gambar pohon pencarian yang akan dibentuk
11.	`findAllOccurence`	`boolean`	Bernilai benar bila menggunakan mode pencarian seluruh kemunculan file, bernilai salah bila menggunakan mode pencarian kemunculan file pertama kali
Bagian Algoritma (`Implementation.cs`**) -** Kelas `Implementation`			
12.	`root`	`DirectoryInfo (private)`	Menyimpan nama folder akar untuk memulai pencarian
13.	`input`	`string (private)`	Menyimpan nama file yang dicari
14.	`resultPath`	`string[] (private)`	Menyimpan larik berisi daftar *string path* menuju file yang dicari. Hanya berisi satu elemen bila tidak menggunakan mode pencarian seluruh kemunculan file
15	`graphResult`	`Graph`	Objek graf yang nantinya menjadi pohon hasil

			pencarian file
16.	`files`	`FileInfo[]`	Menyimpan daftar nama file sementara untuk di cek kesamaannya dengan file yang sedang dicari
17.	`subDirs`	`DirectoryInfo[]`	Menyimpan daftar nama folder sementara untuk di cek isi dalamnya
18.	`dirQueue`	`DirectoryInfo[]`	**(Khusus *BFS*)** Merupakan antrian yang berisi simpul direktori yang akan dikunjungi
18.	`edgeMap`	`Dictionary<(string,string), Edge>`	**(Khusus *BFS*)** menyimpan *map* berisi sisi antara simpul direktori dengan simpul direktori *parent*-nya
18.	`prevRoot`	`Dictionary<string,string >`	**(Khusus *BFS*)** menyimpan direktori akar sementara untuk

4.3 Tata Cara Penggunaan Program

Program ini dapat digunakan dengan menjalankan file ".exe" pada folder `/bin` ataupun dengan *build* file ".sln" pada folder `/src` dengan menggunakan `Microsoft Visual Studio IDE`. Saat program dibuka, akan muncul sebuah antarmuka dengan beberapa jenis input yang dapat langsung diisi untuk menjalankan fungsinya. Untuk visualnya dapat dilihat pada subbab 4.4.

Fitur-fitur yang terdapat pada program adalah sebagai berikut.

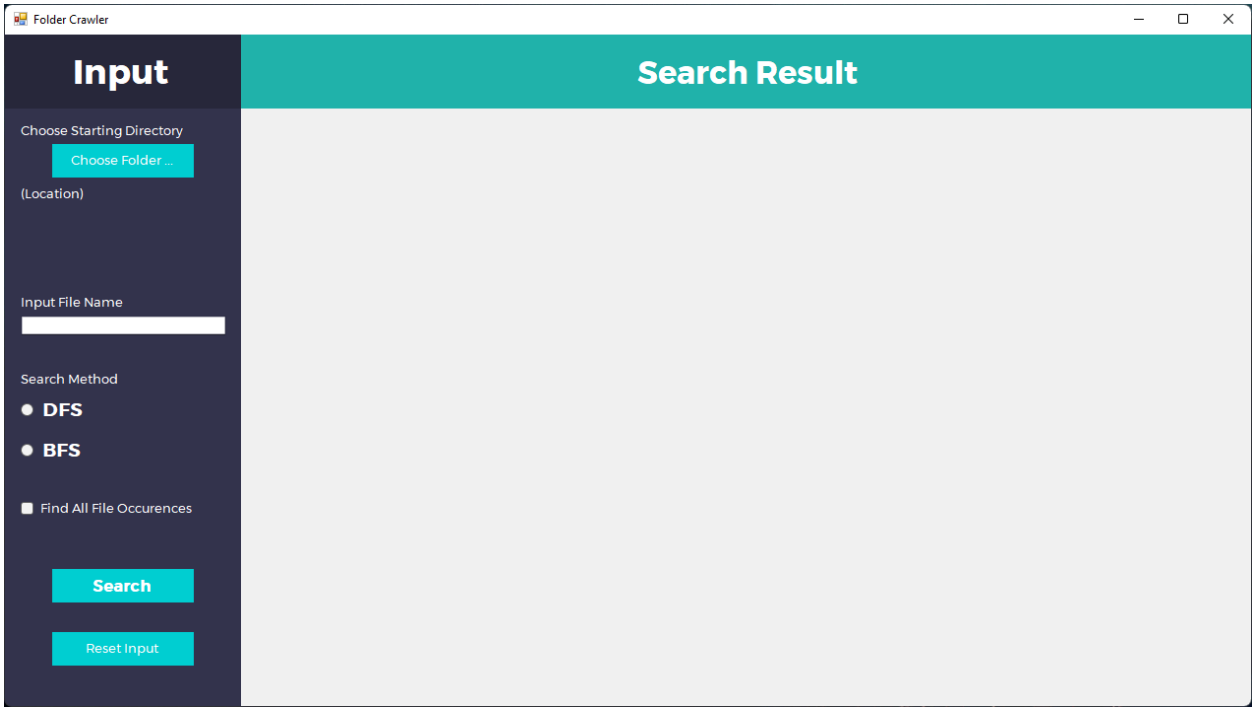
- Input folder sebagai *starting directory*, folder tempat pencarian dimulai. Direktori penuh (direktori yang ditulis dimulai dari volume partisi, misal "D://") juga akan ditampilkan oleh program.
- Input nama file yang akan dicari beserta ekstensinya,
- Input pilihan algoritma pencarian, dalam hal ini *DFS* ataupun *BFS*.
- Kotak centang "*Find All File Occurrences*" yang bila dicentang akan mengaktifkan mode pencarian keseluruhan file yang ada.
- Tombol *Search* yang akan memulai proses pencarian file dan pembentukan pohon pencarian file.
- Tombol *Reset input* yang akan me-*reset* input yang telah dimasukkan.
- *Viewer* untuk menampilkan pohon pencarian beserta waktu eksekusi, dalam milidetik.
- Daftar *hyperlink* menuju folder tempat file berada, yang dapat diklik untuk langsung membuka *Windows Explorer* dan membuka folder tersebut.

Alur program adalah sebagai berikut. Pengguna memasukkan *input* folder tempat pencarian, kemudian memasukkan *input* file yang akan dicari beserta ekstensinya (misal, "`Test.txt`"). Selanjutnya memilih opsi algoritma pencarian, mencentang kotak "*Find All File Occurrences*" bila ingin mencari seluruh keberadaan file, dan mengklik tombol *Search*. Bila seluruh *input* sudah diisi, pengguna dapat menekan tombol *Search* untuk memulai pencarian. Bila tidak semua *input* diisi, program akan menampilkan pesan error yang tertera di bawah input yang belum terisi. Program akan menampilkan pohon pencarian beserta daftar *hyperlink* yang dapat diklik untuk langsung membuka *explorer* pada folder tempat file berada yang dipilih. Bila file tidak ditemukan, akan ditulis "*File not found*".

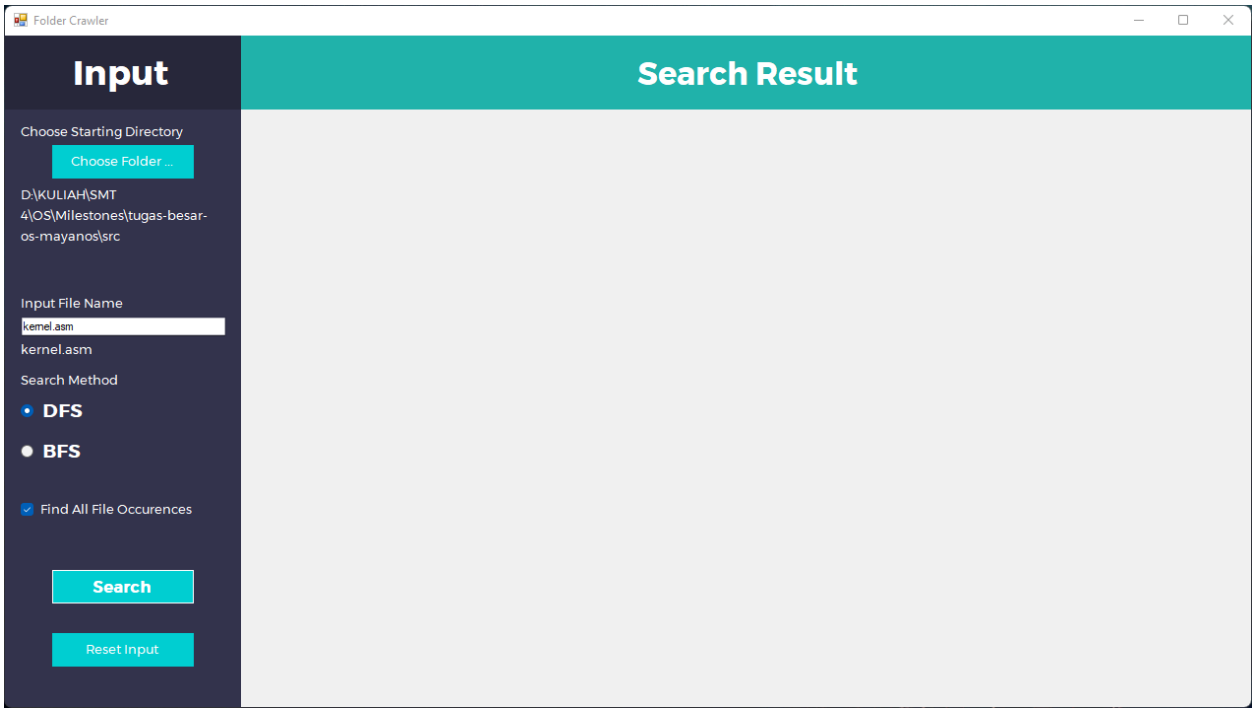
4.4 Hasil Pengujian

Hasil pengujian program dengan beberapa skenario pengujian dilampirkan pada seluruh tangkapan layar di bawah.

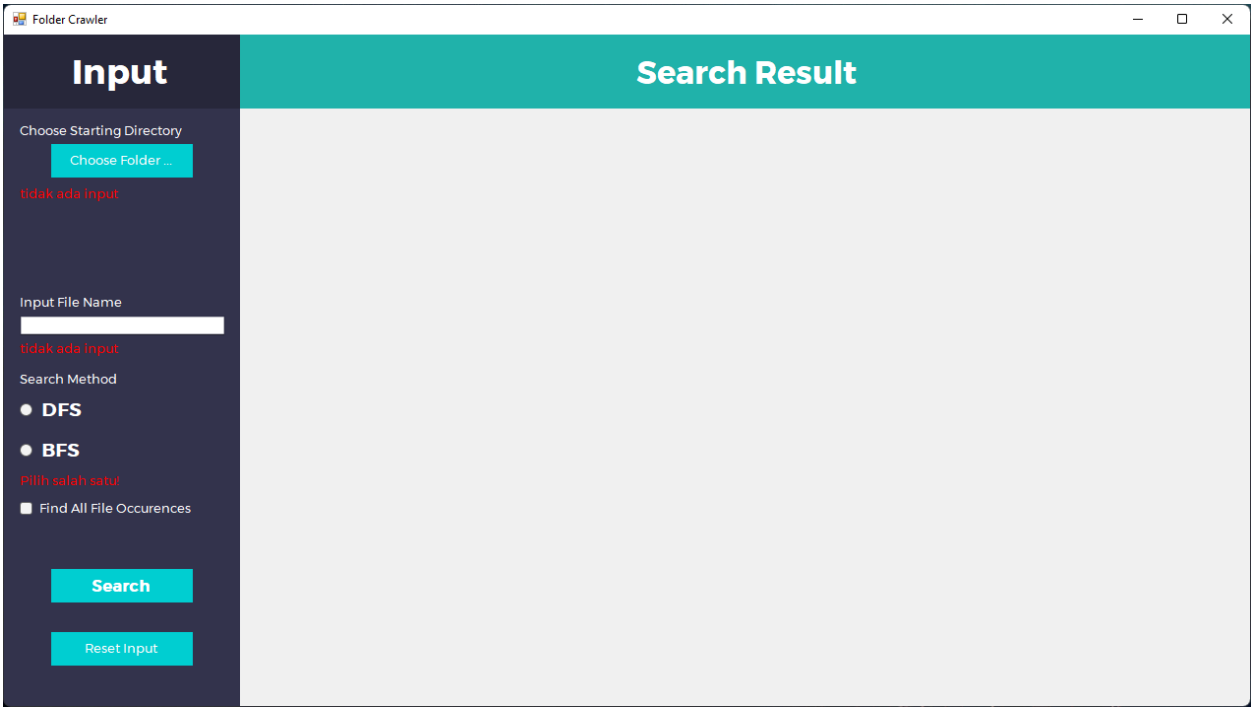
- Antarmuka saat program pertama kali dibuka.



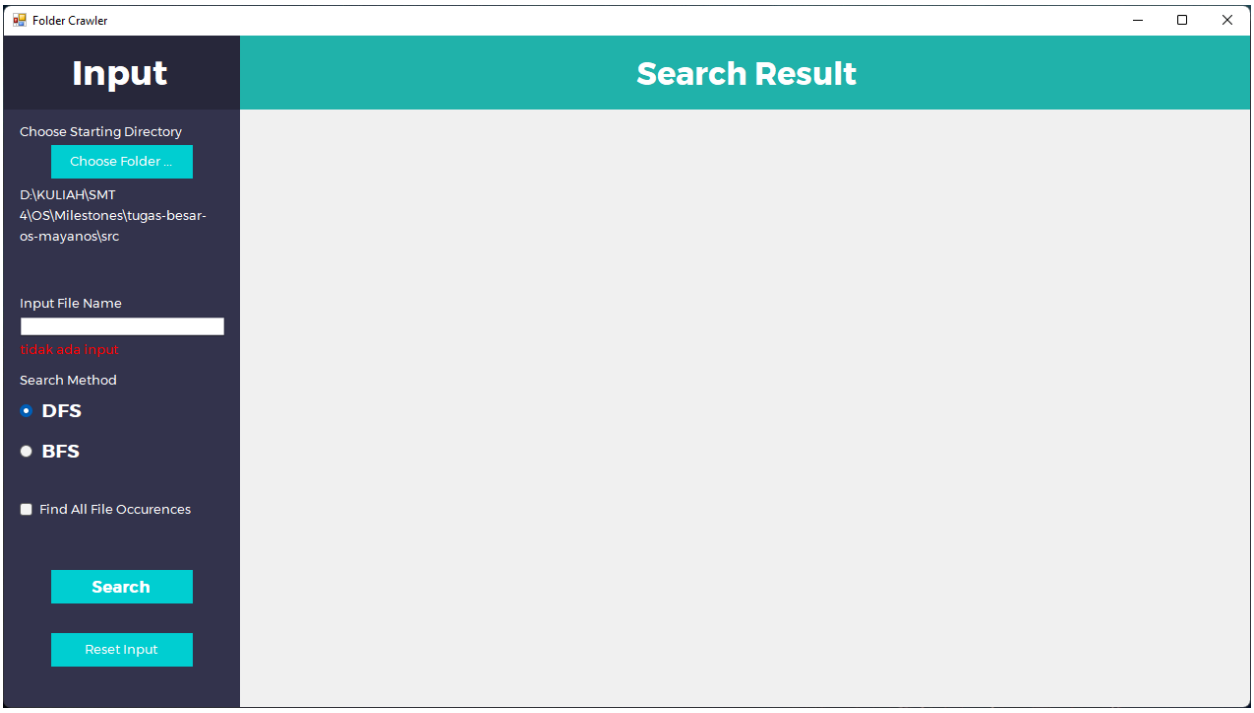
- Contoh input sebelum melakukan pencarian.



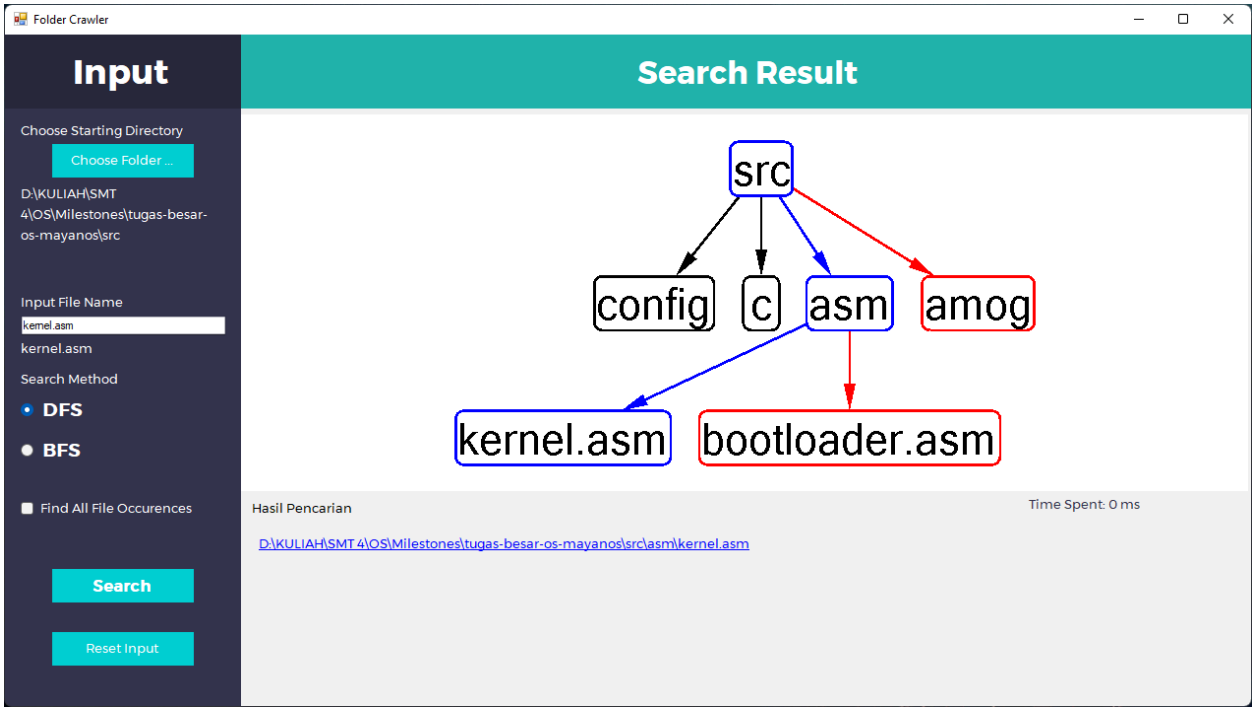
- Contoh *error* karena belum mengisi keseluruhan input yang diperlukan



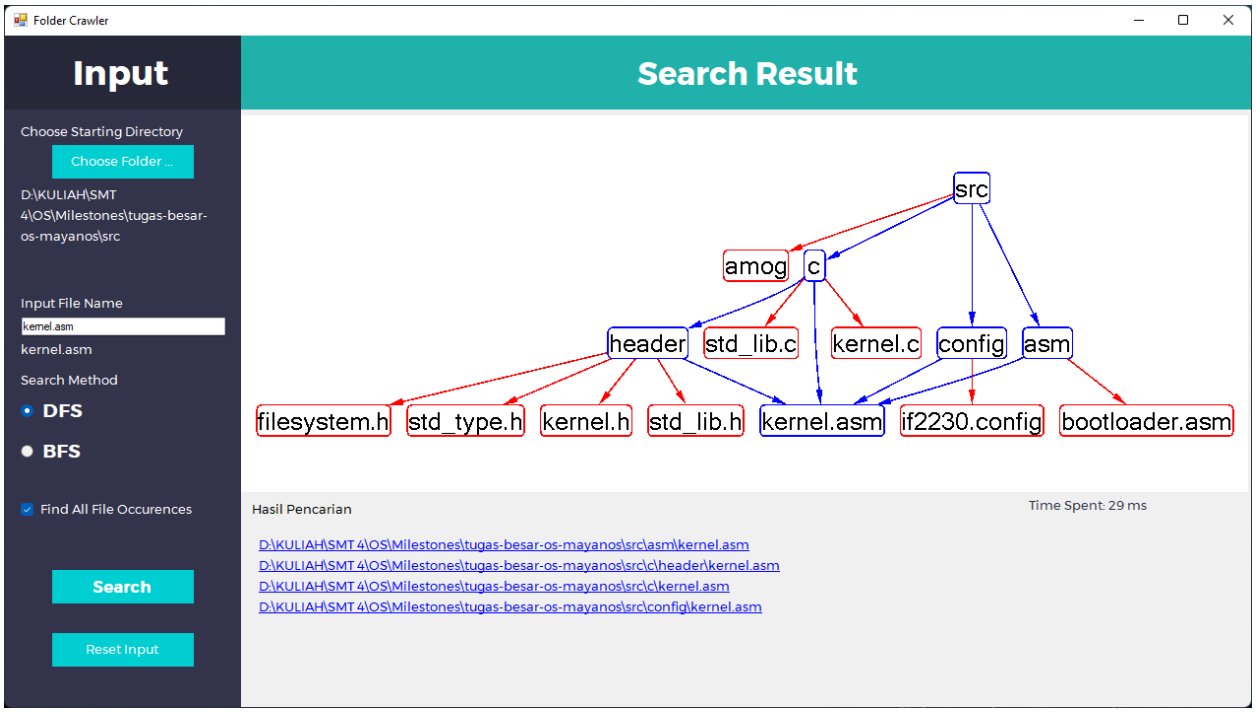
- Contoh *error* karena belum mengisi input nama file



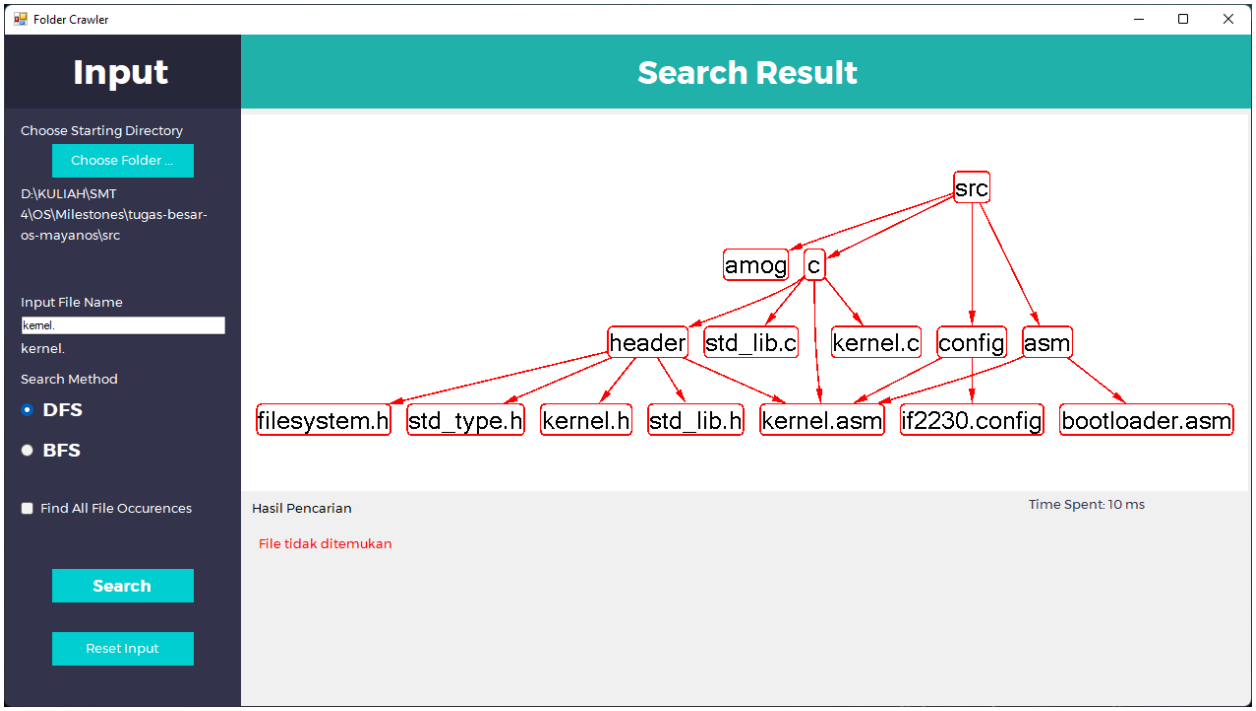
- Hasil pencarian menggunakan DFS, satu file, dan ditemukan.



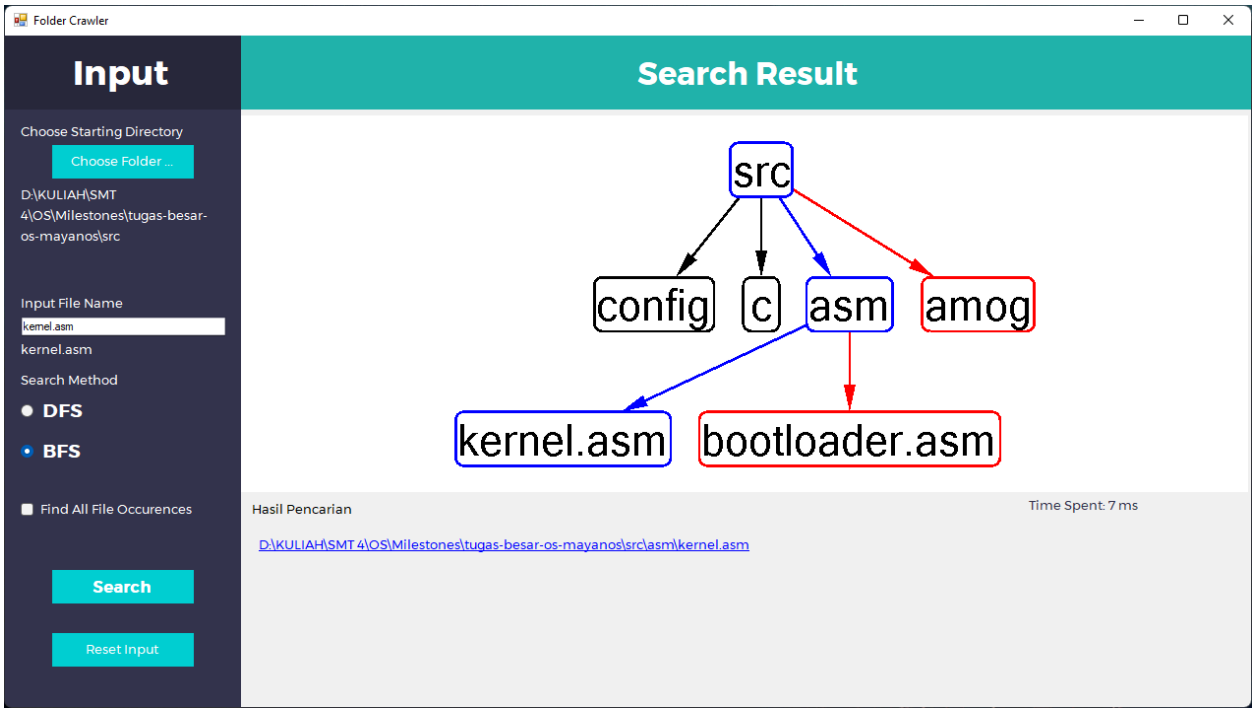
- Hasil pencarian menggunakan DFS, seluruh file, dan ditemukan.



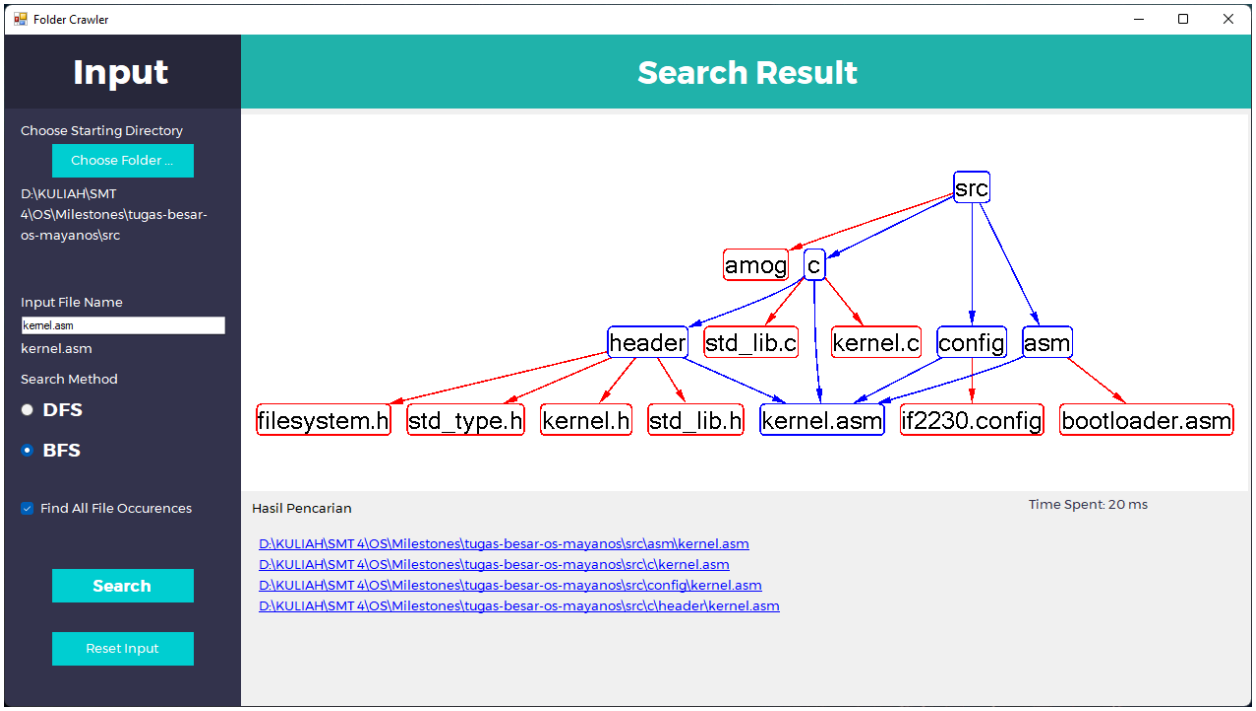
- Hasil pencarian menggunakan DFS, satu file, tidak ditemukan.



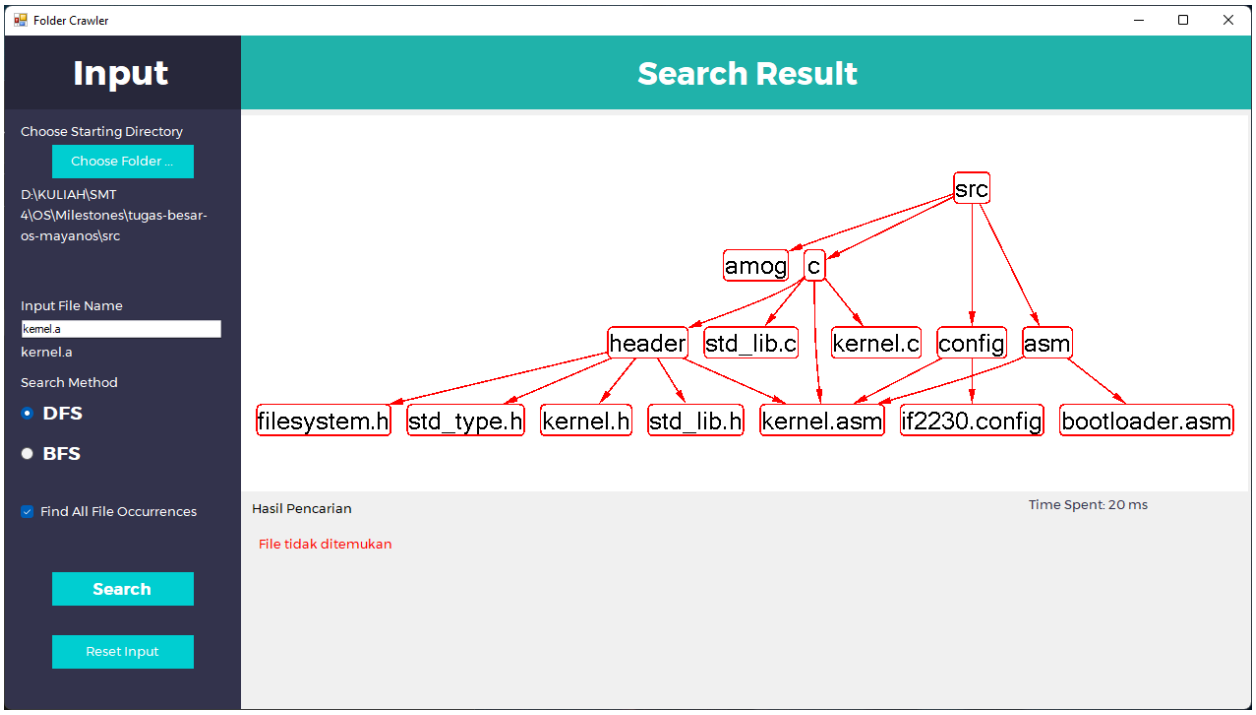
- Hasil pencarian menggunakan BFS, satu file, dan ditemukan.



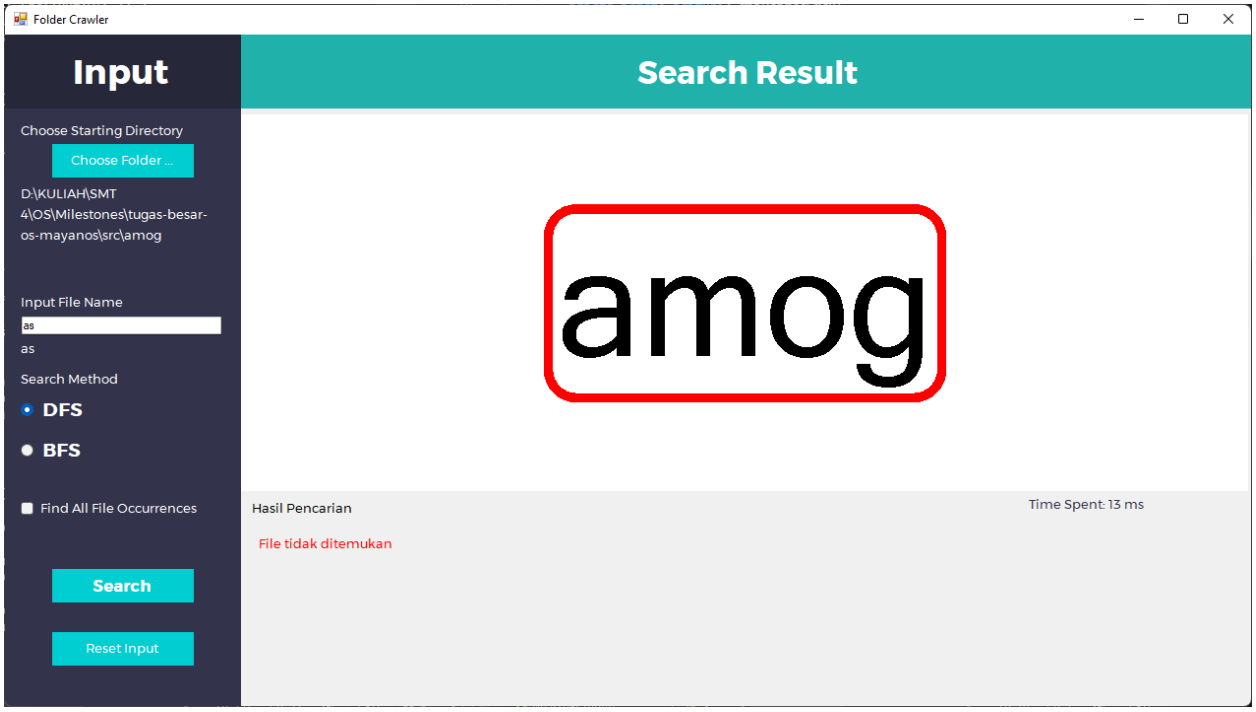
- Hasil pencarian menggunakan BFS, seluruh file, dan ditemukan.



- Hasil pencarian menggunakan BFS, seluruh file, tidak ditemukan.



- Hasil pencarian untuk direktori akar kosong.



4.5 Analisis dari Desain Solusi

Analisis dari desain solusi dilakukan serta menjalankan seluruh skenario penggunaan aplikasi, termasuk melakukan pencarian dengan kedua algoritma. Aplikasi dapat memunculkan pohon pencarian dengan benar, sesuai kaidah algoritma *DFS* dan *BFS*. Aplikasi juga dapat memunculkan pesan *error* dengan benar bila terjadi *error* pada bagian spesifik dari program. Seluruh *hyperlink* yang dihasilkan dapat diklik untuk membuka folder tempat file yang dicari.

Perbedaan pada penggunaan algoritma *DFS* dan *BFS* tidak terlalu terlihat untuk kasus dengan susunan folder yang spesifik. Algoritma *DFS* berjalan lebih lama dari *BFS* untuk kasus susunan folder yang cenderung lebih mendalam, yaitu ketika terdapat banyak sekali aras yang terbentuk dari direktori awal hingga file dan/atau folder paling dalam. Sementara itu, Algoritma *BFS* berjalan lebih lama dari *DFS* untuk kasus susunan folder yang cenderung lebih melebar, ketika ada satu folder yang berisi banyak sekali file dan/atau folder unik.

Tentu saja, Algoritma *DFS* dan *BFS* akan menghasilkan pohon pencarian yang sama untuk kasus pencarian yang tidak ditemukan, serta kasus pencarian untuk susunan folder sedemikian rupa sehingga setiap folder hanya memiliki antara satu folder saja ataupun satu file saja (pohon pencarian yang dihasilkan tidak memiliki cabang sama sekali),

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

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

1. Telah dibuat suatu aplikasi “Folder Crawler” untuk mencari satu / seluruh file di dalam suatu direktori.
2. Aplikasi ini berhasil menjalankan fungsinya dengan memanfaatkan algoritma *DFS* dan *BFS*.
3. Aplikasi juga berhasil menampilkan pohon pencarian beserta *hyperlink* menuju folder tempat file tersebut terletak.

5.2 Saran

Berdasarkan implementasi algoritma pada program dan eksperimen yang telah dilakukan, penulis memberikan beberapa saran sebagai berikut.

1. Memahami kembali konsep paradigma pemrograman berorientasi objek pada C# agar dapat membuat program dengan lebih rapi.
2. Mengimplementasikan modul pembuatan graf yang lebih baik agar dapat menghasilkan pohon pencarian yang lebih bagus.
3. *GUI* dapat diprogram lebih baik lagi agar aplikasi dapat menjalankan fungsinya dengan lebih baik.

LINK REPOSITORY

https://github.com/DzakyFattan/Tubes2Stima-FolderCrawler

DAFTAR PUSTAKA

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

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf

https://www.microsoft.com/en-us/research/project/microsoft-automatic-graph-layout/code-samples/

https://docs.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022